

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению лабораторных работ по дисциплине
«Структуры и алгоритмы компьютерной обработки данных»
для студентов направления подготовки
02.03.01 Математика и компьютерные науки

Лабораторная работа № 1 « Методы сортировки »

Цель работы: ознакомление с алгоритмами сортировки линейных и нелинейных структур и методикой оценки эффективности алгоритмов.

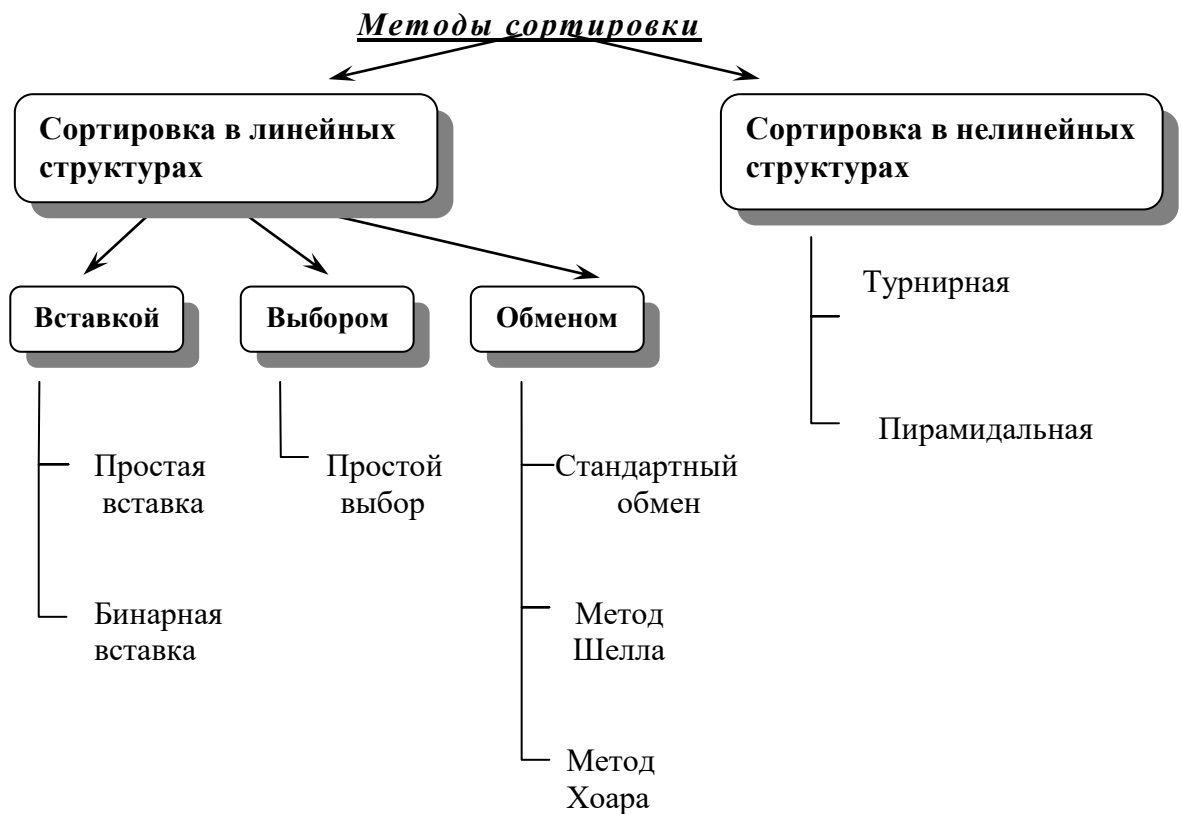
Продолжительность работы: - 2 часа.

Теоретические сведения

Упорядочение элементов множества в возрастающем или убывающем порядке называется сортировкой.

С упорядоченными элементами проще работать, чем с произвольно расположенными: легче найти необходимые элементы, исключить, вставить новые. Сортировка применяется при трансляции программ, при организации наборов данных на внешних носителях, при создании библиотек, каталогов, баз данных и т.д.

Алгоритмы сортировки можно разбить на следующие группы:



Обычно сортируемые элементы множества называют записями и обозначают через

$k_1, k_2, \dots, k_n$.

Сортировка выбором

Сортировка выбором состоит в том, что сначала в неупорядоченном списке выбирается и отделяется от остальных наименьший элемент. После этого исходный список оказывается изменённым. Изменённый список принимается за исходный и процесс

продолжается до тех пор, пока все элементы не будут выбраны. Очевидно, что выбранные элементы образуют упорядоченный список.

Например, требуется найти минимальный элемент списка:

{5, 11, 6, 4, 9, 2, 15, 7}

Процесс выбора показан на рис.1, где в каждой строчке выписаны сравниваемые пары. Выбираемые элементы с меньшим весом обведены кружком. Нетрудно видеть, что число сравнений соответствует на рисунке числу строк, а число перемещений – количеству изменений выбранного элемента.

{5, 11, 6, 4, 9, 2, 15, 7}

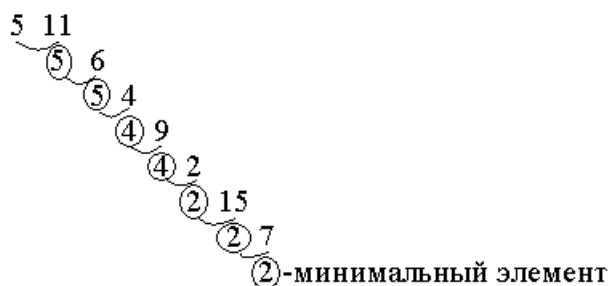


Рис.1. Сортировка выбором

Выбранный в исходном списке минимальный элемент размещается на предназначенном ему месте несколькими способами:

- Минимальный элемент после i -го просмотра перемещается на i -ое место нового списка ($i = 1, 2, \dots, n$), а в исходном списке на место выбранного элемента записывается какое-то очень большое число, превосходящее по величине любой элемент списка, при этом длина заданного списка остаётся постоянной. Изменённый таким образом список можно принимать за исходный.
- Минимальный элемент записывается на i -ое место исходного списка ($i = 1, 2, \dots, n$), а элемент с i -го места - на место выбранного. При этом очевидно, что уже упорядоченные элементы (а они будут расположены, начиная с первого места) исключаются из дальнейшей сортировки, поэтому длина каждого последующего списка (списка, участвующего в каждом последующем просмотре) должна быть на 1 элемент меньше предыдущего.
- Выбранный минимальный элемент, как и в предыдущем случае, перемещается на i -ое место заданного списка, а чтобы это i -ое место освободилось для записи очередного минимального элемента, левая от выбранного элемента часть списка перемещается вправо на одну позицию так, чтобы заполнилось место, занимаемое до этого выбранным элементом.

Сложность метода сортировки выбором порядка $O(n^2)$.

Сортировка вставкой

В этом методе из неупорядоченной последовательности элементов выбирается поочередно каждый элемент, сравнивается с предыдущим, уже упорядоченным, и помещается на соответствующее место.

Сортировку вставкой рассмотрим на примере заданной неупорядоченной последовательности элементов:

{40, 11, 83, 57, 32, 21, 75, 64}

Процедура сортировки отражена на рис.2, где кружком на каждом этапе обведён анализируемый элемент, стрелкой сверху отмечено место перемещения анализируемого элемента, в рамку заключены упорядоченные части последовательности.

Этапы:

- 1-й $\overline{40, 11}, 83, 57, 32, 21, 75, 64$
 $\boxed{11, 40}, 83, 57, 32, 21, 75, 64$
- 2-й $11, 40, \overline{83}, 57, 32, 21, 75, 64$
 $\boxed{11, 40, 83}, 57, 32, 21, 75, 64$
- 3-й $11, 40, \overline{83}, \overline{57}, 32, 21, 75, 64$
 $\boxed{11, 40, 57, 83}, 32, 21, 75, 64$
- 4-й $11, \overline{40, 57, 83}, \overline{32}, 21, 75, 64$
 $\boxed{11, 32, 40, 57, 83}, 21, 75, 64$
- 5-й $11, \overline{32, 40, 57, 83}, \overline{21}, 75, 64$
 $\boxed{11, 21, 32, 40, 57, 83}, 75, 64$
- 6-й $11, 21, 32, 40, 57, \overline{83}, \overline{75}, 64$
 $\boxed{11, 21, 32, 40, 57, 75, 83}, 64$
- 7-й $11, 21, 32, 40, 57, \overline{75}, \overline{83}, \overline{64}$
 $\boxed{11, 21, 32, 40, 57, 64, 75, 83}$

Рис. 2. Сортировка вставкой

На первом этапе сравниваются два начальных элемента. Поскольку второй элемент меньше первого, он перемещается на место первого элемента, который сдвигается вправо на одну позицию. Остальная часть последовательности остаётся без изменения. На втором этапе из неупорядоченной последовательности выбирается элемент и сравнивается с двумя упорядоченными ранее элементами. Так как он больше предыдущих, то остаётся на месте. Затем анализируются четвёртый, пятый и последующие элементы – до тех пор, пока весь список не будет упорядоченным, что имеет место на последнем (седьмом) этапе.

Разновидностью сортировки вставкой является метод **фон Неймана**. Пусть заданы два упорядоченных по возрастанию элементов одномерных массива: a размерности n и b размерности m . Требуется получить третий массив с размерности $n+m$, который содержал бы все элементы исходных массивов, упорядоченных по возрастанию.

Алгоритм решения этой задачи, известный как «сортировка фон Неймана» или сортировка слиянием, состоит в следующем: сначала анализируются первые элементы обоих массивов. Меньший элемент переписывается в новый массив. Оставшийся элемент последовательно сравнивается с элементами из другого массива. В новый массив после каждого сравнения попадает меньший элемент. Процесс продолжается до исчерпания элементов одного из массивов. Затем остаток другого массива дописывается в новый массив. Полученный новый массив упорядочен таким же образом, как исходные.

Сложность метода сортировки вставкой порядка $O(n^2)$.

Сортировка обменом

Сортировка обменом – метод, в котором элементы списка последовательно сравниваются между собой и меняются местами в том случае, если предшествующий элемент больше последующего.

Требуется, например, провести сортировку списка методом стандартного обмена или методом 'пузырька':

$\{40, 11, 83, 57, 32, 21, 75, 64\}$

Обозначим квадратными скобками со стрелками $\uparrow \quad \downarrow$ обмениваемые элементы, а $\lfloor \quad \rfloor$ - сравниваемые элементы. Первый этап сортировки показан на рис.3, а второй этап – на рис.4.

Нетрудно видеть, что после каждого просмотра списка все элементы, начиная с последнего, занимают свои окончательные позиции, поэтому их не следует проверять при следующих просмотрах. Каждый последующий просмотр исключает очередную позицию с найденным максимальным элементом, тем самым укорачивая список. После первого просмотра в последней позиции оказался больший элемент, равный 83 (исключаем его из дальнейшего рассмотрения).

Второй просмотр выявляет максимальный элемент, равный 75 (рис.4).

Процесс сортировки продолжается до тех пор, пока не будут сформированы все элементы конечного списка, либо не выполнится условие Айверсона.

Исходный список	40	11	83	57	32	21	75	64
Первый просмотр	$\uparrow$	$\uparrow$						
	11	40						
		40	$\uparrow$					
			83	$\uparrow$				
			57	83	$\uparrow$			
				32	83	$\uparrow$		
					21	83	$\uparrow$	
						75	83	$\uparrow$
							64	83
Полученный список	11	40	57	32	21	75	64	83

Рис. 3. Сортировка обменом (первый просмотр)

Исходный список	11	40	57	32	21	75	64
Второй просмотр	$\uparrow$	$\uparrow$					
	11	40					
		40	$\uparrow$				
			57	$\uparrow$			
			32	57	$\uparrow$		
				21	57	$\uparrow$	
					57	75	$\uparrow$
						64	75
Полученный список	11	40	32	21	57	64	75

Рис. 4. Сортировка обменом (второй просмотр)

Условие Айверсона: если в ходе сортировки при сравнении элементов не было сделано ни одной перестановки, то множество считается упорядоченным (условие Айверсона выполняется только при шаге $d=1$).

Модификацией сортировки стандартным обменом является **шейкерная** или челночная сортировка . Здесь, как и в методе пузырька проводится по парное сравнение элементов . При этом первый проход осуществляется слева направо, второй – справа налево и т.д. Иными словами меняется направление просмотра элементов списка.

Сложность метода стандартного обмена $O(n^2)$.

Сортировка Шелла

В методе Шелла сравниваются не соседние элементы, а элементы, расположенные на расстоянии d (где d – шаг между сравниваемыми элементами) $d = \lfloor n/2 \rfloor$. После каждого просмотра шаг d уменьшается вдвое. На последнем просмотре он сокращается до $d = 1$.

Например, пусть дан список, в котором число элементов чётно:

{40, 11, 83, 57, 32, 21, 75, 64}

Список длины n разбивается на $n/2$ частей, т.е. $d = \lfloor n/2 \rfloor = 4$.

Исходный массив	40	11	83	57	32	21	75	64
$d = 4$	↑				↑			
	32				40			
		└				└		
		11				21		
			↑				↑	
			75				83	
				└				└
				57				64
Полученный массив	32	11	75	57	40	21	83	64

Рис. 5. Метод Шелла ($d=4$)

Исходный массив	32	11	75	57	40	21	83	64
$d = 2$	↑		↑					
	32		75					
		└		└				
		11		57				
			↑		↑			
			40		75			
				└		└		
				21		57		
					└		└	
					75		83	
						└		└
						57		64
Полученный массив	32	11	40	21	75	57	83	64

Рис. 6. Метод Шелла ($d=2$)

При первом просмотре сравниваются элементы, отстоящие друг от друга на $d = 4$ (рис.5), т.е. k_1 и k_5 , k_2 и k_6 и т.д. Если $k_i > k_{i+d}$, то происходит обмен между позициями i и $(i+d)$. Перед вторым просмотром выбирается шаг $d = \lfloor d/2 \rfloor = 2$ (рис.6). Затем выбираем шаг $d = \lfloor d/2 \rfloor = 1$ (рис.7), т.е. имеем аналогию с методом стандартного обмена.

Сложность метода Шелла $O(0,3n(\log_2 n)^2)$.

Исходный список	32	11	40	21	75	57	83	64
$d = 1$	↑	↑						
	11	32						
		└	└					
		32	40					
			└	└				
			21	40				
				└	└			
				40	75			
					└	└		
					57	75		
						└	└	
						75	83	
							└	└
							64	83
Полученный список	11	32	21	40	57	75	64	83

Рис. 7. Метод Шелла ($d=1$)

Быстрая сортировка (сортировка Хоара)

В методе быстрой сортировки фиксируется какой-либо ключ (базовый), относительно которого все элементы с большим весом перебрасываются вправо, а с меньшим – влево. При этом весь список элементов делится относительно базового ключа на две части. Для каждой части процесс повторяется.

Поясним метод на примере.

На рис.8 представлен первый этап быстрой сортировки. В первой строке указана исходная последовательность.

Примем первый элемент последовательности за базовый ключ, выделим его квадратом и обозначим $k_0 = 40$. Установим два указателя: i и j , из которых i начинает отсчет слева ($i=1$), а j – справа ($j=n$).

Сравниваем базовый ключ k_0 и текущий ключ k_j . Если $k_0 \leq k_j$, то устанавливаем $j=j-1$ и проводим следующее сравнение k_0 и k_j . Продолжаем уменьшать j до тех пор, пока не достигнем условия $k_0 > k_j$. После этого меняем местами ключи k_0 и k_j (шаг 3 на рис.8).

Номер шага	$i \longrightarrow$ $\longleftarrow j$								Примечание
	40	11	83	57	32	21	75	64	Исходный список
1	40	---	---	---	---	---	---	64	$k_0 < k_j$
2	40	---	---	---	---	---	75	---	$k_0 < k_j$
3	40	---	---	---	---	21	---	---	Обмен; $k_0 > k_j$
	21	---	---	---	---	40	---	---	
4	---	11	---	---	---	---	40	---	$k_i < k_0$
5	---	---	83	---	---	---	40	---	Обмен; $k_i > k_0$
	---	---	40	---	---	---	83	---	
6	---	---	40	---	---	32	---	---	Обмен; $k_0 > k_j$
	---	---	32	---	---	40	---	---	
7	---	---	---	57	40	---	---	---	Обмен; $k_i > k_0$
	---	---	---	40	57	---	---	---	
	21	11	32	40	57	83	75	64	Полученный список

Рис. 8. Метод Хоара

Теперь начинаем изменять индекс $i = i + 1$ и сравнивать элементы k_i и k_0 . Продолжаем увеличение i до тех пор, пока не получим условие $k_i > k_0$, после чего следует обмен k_i и k_0 (см. шаг 5). Снова возвращаемся к индексу j , уменьшаем его. Чередуя уменьшение j и увеличение i , продолжаем этот процесс с обоих концов к середине до тех пор, пока не получим $i = j$ (см. шаг 7).

В отличие от предыдущих рассмотренных сортировок уже на первом этапе имеют место два факта: во-первых, базовый ключ $k_0 = 40$ занял своё постоянное место в сортируемой последовательности; во-вторых, все элементы слева от k_0 будут меньше него, а справа – больше него. Таким образом, по окончании первого этапа имеем:

21, 11, 32 40 57, 83, 75, 64
 левая часть правая часть

Указанная процедура сортировки применяется независимо к левой и правой частям. Сложность метода Хоара $O(n \log_2 n)$.

Сортировка в нелинейных структурах

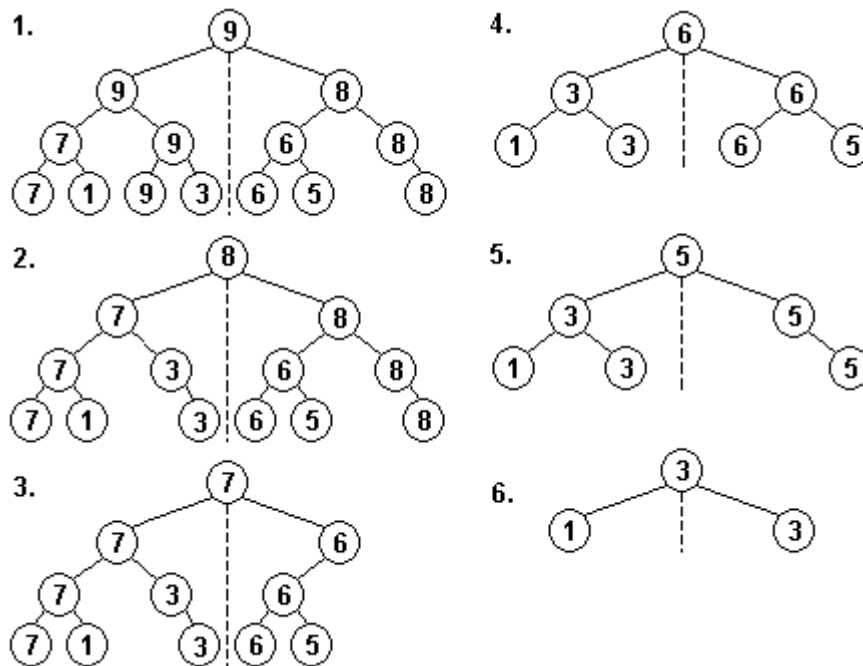
Сортировка в нелинейных структурах осуществляется только на бинарных деревьях (деревьях, из каждой вершины которого выходит по два ребра).

Турнирная сортировка

Свое название эта сортировка получила, потому что она используется при проведении соревнований, турниров и олимпиад. Элементы исходного множества представляются листьями дерева. Их попарное сравнение позволяет определить максимальный элемент.

Пример: Дано исходное множество { 7, 1, 9, 3, 6, 5, 8 }

Производится попарное сравнение элементов снизу-вверх. Найденный максимальный элемент помещается в результирующее множество.



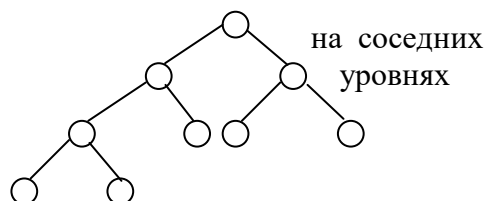
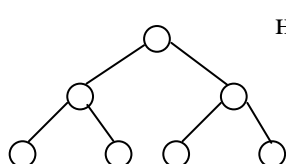
В результате будет получено упорядоченное множество { 9,8,7,6,5,3 }

Пирамидальная сортировка

Данный тип сортировки заключается в построение пирамидального дерева.

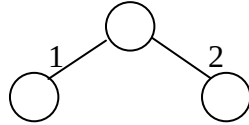
Пирамидальное дерево – это бинарное дерево обладающее тремя свойствами:

- В вершине каждой триады располагается элемент с большим весом.
- Листья бинарного дерева располагаются либо в одном уровне либо в двух соседних



- Листья нижнего уровня располагаются левее листьев более высокого уровня.

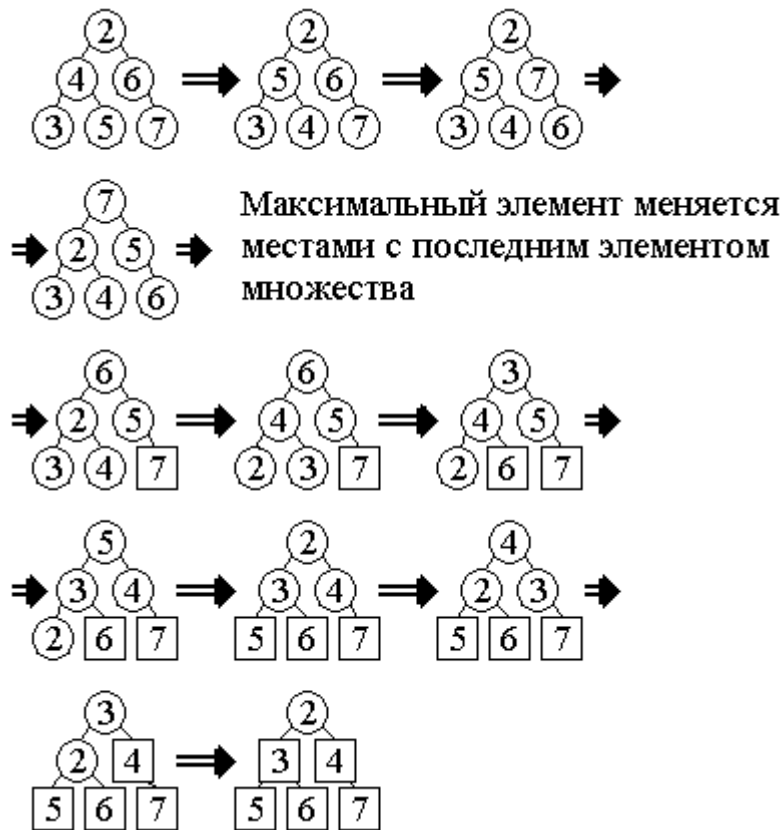
В ходе преобразования элементы триад сравниваются дважды, при этом элемент с большим весом перейдет вверх, а с меньшим - вниз.



1 - первое сравнение

2 - второе сравнение

Пример: Дано исходное множество { 2, 4, 6, 3, 5, 7 }

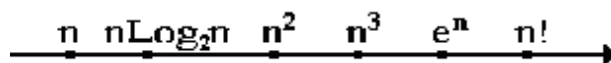


В результате будет получено упорядоченное множество { 2,3,4,5,6,7 }

Функция сложности алгоритма

Для оценки эффективности алгоритмов используется функция сложности алгоритма, которая обозначается заглавной буквой “O”, в круглых скобках записывается аргумент. Например, функция сложности $O(n^2)$ читается как функция сложности порядка n^2 . Функция сложности алгоритма – это функция, которая определяет количество сравнений, перестановок а так временные и ресурсные затраты на реализацию алгоритма.

Функция сложности принимает следующий ряд значений:



Функция сложности

Чем правее на оси расположена функция сложности, тем сложнее алгоритм.

Лабораторное задание

Для каждого из перечисленных методов сортировки провести анализ временных затрат для списков различной размерности.

Методика выполнения лабораторной работы

Для проведения лабораторной работы необходимо выполнить следующие действия.

1. Вызвать систему **Sort_new**, включающую в себя сортировку неупорядоченных списков в линейных и нелинейных структурах.

Путь к файлу: D:\ИПОВС\АиСД\SORT\Sort_new.exe

Система работает в диалоговом режиме с использованием “меню”. Вся необходимая информация во время работы системы отображается на экране дисплея и не требует специальных пояснений.

Основные функции системы

К - конец работы;

? - выдача краткого сообщения о командах;

U – задание условий для генерации неупорядоченного массива;

G - генерации неупорядоченного массива;

P – вывод массива;

W – вывод времени сортировки, количества элементов массива.

Сортировка методом:

1 – простого обмена;

2 – бинарной вставки;

3 – простой вставки;

4 – челночной;

5 – простого выбора;

6 – слиянием;

7 – Шелла;

8 – Хоара;

9 – пирамиды;

Esc – аварийное прерывание выполнения функции.

Краткое описание функции

U– задание условий для генерации неупорядоченного массива (задаются число элементов и границы элементов неупорядоченного массива). Заданные условия сохраняются до следующего вызова функции “U”;

G – генерация неупорядоченного массива, необходима перед каждой сортировкой;

P – вывод массива в текущем состоянии : неупорядоченный (до сортировки) или упорядоченный (после неё);

W - вывод времени сортировки ;

I – 9 – методы сортировки;

Для указанных в лабораторной работе методов сортировки провести следующие исследования:

а) выполнить сортировку массива из N чисел (варианты заданий даны в приложении ; номер варианта соответствует порядковому номеру фамилии студента в списке группы). Результаты занести в таблицу 1.

Таблица 1

Метод			
Количество элементов сортируемого массива N	N_1	N_2	N_3
Время сортировки t, с	t_1	t_2	t_3

2. Провести исследование методов сортировки упорядоченных списков с использованием программы **SORTALL**.

Путь к файлу: D:\ИПОВС\АиСД\SORT\Sortall.exe

Результаты занести в таблицу 1.

3. Провести исследование методов сортировки с использованием программ **WinSort** и **Sort**.

4. Оценить сложность рассмотренных методов сортировки;

а) провести анализ отклонения полученной в результате эксперимента сложности алгоритма от теоретической;

б) построить графические зависимости времени сортировки от количества элементов сортируемого массива.

Требования к отчёту

Отчёт должен содержать:

- 1) конспект лабораторной работы;
- 2) примеры сортировки ;
- 3) результаты выполнения работы;
- 4) выводы по работе;

Контрольные вопросы

1. Что понимается под сортировкой?
2. Каковы особенности сортировки: вставкой, выбором, обменом , Шелла, Хоара, турнирной, пирамидой?
3. Что включает в себя понятие сложности алгоритма?
4. В чём состоит методика анализа сложности алгоритмов сортировки?

Литература

1. Колдаев В.Д., Поддубная Л. М., Полосухин Б.М. Методы сортировки . М.:МИЭТ, 1985.
2. Колдаев В.Д., Поддубная Л. М., Полосухин Б.М. Лабораторный практикум по курсу « Теория алгоритмов и вычислительные методы » М.:МИЭТ, 1988
3. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М. : Мир, 2000.
4. Т. Кормен, Ч. Лейзерсон, Р. Ривест «Алгоритмы: построение и анализ». М.: МЦНМО, 2000.
5. Вирт Н. Алгоритмы и структуры данных.: Пер. С англ. - М.: Мир, 2001.
6. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.

Варианты заданий

Вариант	n1	n2	n3	Составить программу
1,15	1000	4000	6000	Простая вставка
2,16	800	3000	7000	Сортировка слиянием
3,17	2000	5000	6800	Метод Шелла
4,18	1500	3500	6000	Простой выбор
5,19	1000	2800	8500	Метод Хоара
6,20	500	2500	6500	Бинарная вставка
7,21	900	3000	8500	Шейкерная сортировка
8,22	1200	2000	7500	Простая вставка
9,23	2500	3500	6500	Шейкерная сортировка
10,24	1600	2800	8800	Метод Шелла
11,25	2200	3400	7200	Простой выбор
12,26	1900	2700	8000	Метод Хоара
13,27	1300	3500	6000	Бинарная вставка
14,28	1700	2800	7500	Сортировка слиянием

Предусмотреть в программе учет времени сортировки для указанных значений n_1 , n_2 , n_3 . Для сортировок Шелла, Хоара, пирамидальной значения n_1 , n_2 , n_3 удвоить. Составить программу, реализующую один из методов сортировки. Произвести расчет функции сложности разработанного алгоритма по полученным временным затратам.

Пояснения к выполнению работы.

Определить время выполнения какого-либо фрагмента программы можно с помощью библиотечной функции **gettime()**, прототип которой находится в заголовочном файле **dos.h**. Эта функция должна получать аргумент – адрес структуры типа **time**, в поля которой и заносится информация о текущем времени. В обобщенном виде программа, в которой выполняется контроль времени может выглядеть следующим образом:

```
#include <dos.h>
void main()
{
    time t1,t2;
    gettime(&t1);
    // Фрагмент, время работы которого измеряется.
    gettime(&t2);
    // ...
}
```

Для получения интервала времени следует вычесть поля структуры **t1** из полей структуры **t2**. Представить этот интервал в секундах можно следующим образом:

```
double delta_t=(t2.ti_hour*360000.+t2.ti_min*6000.+
t2.ti_sec*100.+t2.ti_hund-
(t1.ti_hour*360000.+t1.ti_min*6000.+
t1.ti_sec*100.+t1.ti_hund))/100.;
```

Современные процессоры выполняют сортировку небольших массивов за время, значительно меньшее, чем тысячная доля секунды (**t1.ti_hund**), поэтому для получения точного результата нужно выполнить сортировку достаточно большое число раз, например, миллион. При этом необходимо каждый раз сортировать один и тот же неупорядоченный массив. Лучше сортировать второй массив, каждый раз перед сортировкой копируя в него данные из исходного неупорядоченного массива. Копирование занимает некоторое время, которое необходимо учесть.

После завершения всего процесса следует проверить правильность сортировки, выведя содержимое второго массива на экран.

Итоговая программа в обобщенном виде может выглядеть следующим образом:

```
#include <dos.h>
#include <iostream.h>
const int N = 5; // Размер массива.
const unsigned long NN = 1000000; // Число сортировок.
void main()
{
    char arr1[N]={ /* Элементы неупорядоченного массива */ };
    char arr[N];
    time t1,t2;
    double t_copy,t_sort;

    gettimeofday(&t1);
    // Копирование из массива arr1 в массив arr.
    gettimeofday(&t2);
    t_copy = (t2.ti_hour*360000.+t2.ti_min*6000.+
t2.ti_sec*100.+t2.ti_hund-(t1.ti_hour*360000.+
t1.ti_min*6000.+t1.ti_sec*100.+t1.ti_hund))/100.;

    gettimeofday(&t1);
    // Цикл сортировки из большого числа проходов.
    gettimeofday(&t2);

    t_sort = (t2.ti_hour*360000.+t2.ti_min*6000.+
t2.ti_sec*100.+t2.ti_hund-(t1.ti_hour*360000.+
t1.ti_min*6000.+t1.ti_sec*100.+t1.ti_hund))/100.;
    t_sort-=t_copy;
    // Контроль содержимого массива arr после сортировки.
    // Вывод на экран времени сортировки t_sort.
}
```

Лабораторная работа №2 «Методы поиска».

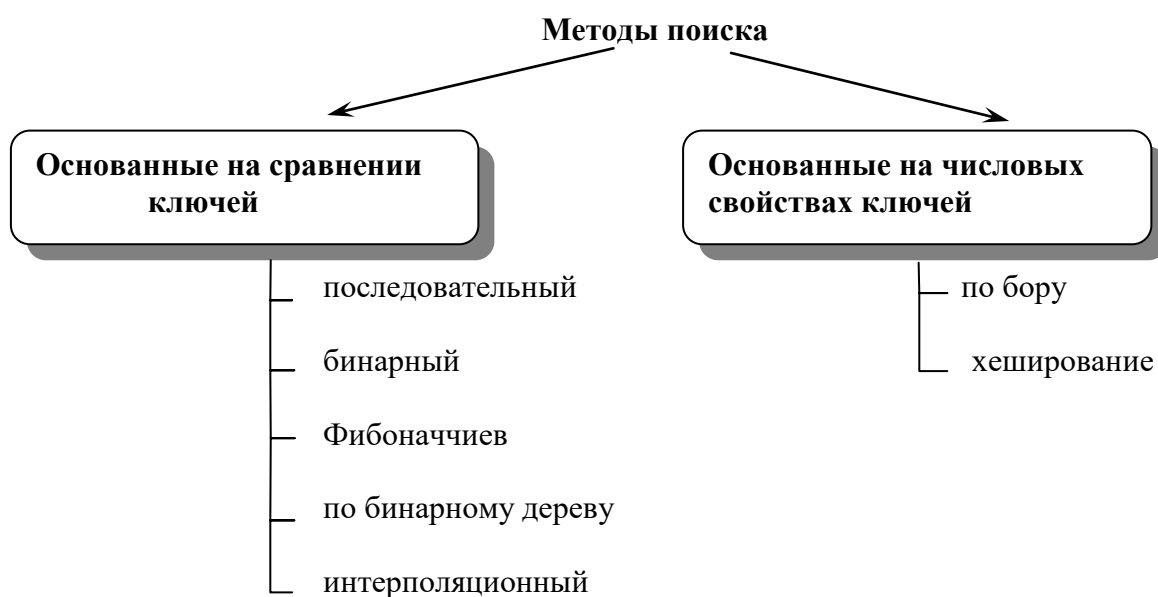
Цель работы: ознакомление с алгоритмами поиска в линейных и нелинейных структурах и оценкой эффективности алгоритмов.

Продолжительность работы: - 2 часа.

Теоретические сведения

Предметы (объекты), составляющие множество, называются его элементами. Элемент множества будет называться ключом, и обозначаться латинской буквой “k” с индексом, указывающим номер элемента.

Алгоритмы поиска можно разбить на следующие группы:



Задача поиска:

Пусть дано множество ключей $\{k_1, k_2, k_3 \dots k_n\}$.

Необходимо отыскать во множестве ключ k_i . Поиск может быть завершён в двух случаях:

- 1) Ключ во множестве отсутствует;
- 2) Ключ найден во множестве.

Последовательный поиск.

В последовательном поиске исходное множество не упорядоченно, т.е. имеется произвольный набор ключей $\{k_1, k_2, k_3 \dots k_n\}$. Метод заключается в том, что отыскиваемый ключ k_i последовательно сравнивается со всеми элементами множества. При этом поиск заканчивается досрочно, если ключ найден.

Бинарный поиск.

В бинарном поиске исходящее множество должно быть упорядоченно по возрастанию. Иными словами каждый последующий ключ больше предыдущего

$$\{k_1 \leq k_2 \leq k_3 \leq k_4 \dots k_{n-1} \leq k_n\}.$$

Отыскиваемый ключ сравнивается с центральным элементом множества, если он меньше центрального, то поиск продолжается в левом подмножестве, в противном случае в правом.

Центральный элемент находится по формуле $N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1$, где квадратные скобки обозначают, что от деления берётся только целая часть (всегда округляется в меньшую сторону). В методе бинарного поиска анализируются только центральные элементы.

Пример. Дано множество

{7,8,12,16,18,20,30,38,49,50,54,60,61,69,75,79,80,81,95,101,123,198}

Найти во множестве ключ $K=61$.

Шаг 1

$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 22/2 \rfloor + 1 = 12$

$K \sim k_{12}$

$61 > 60$ Дальнейший поиск в правом подмножестве {61,69,75,79,80,81,95,101,123,198}.

Значок " $\sim$ " обозначает сравнение элементов (чисел, значений).

Шаг 2

$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 12/2 \rfloor + 1 = 7$

$K \sim k_{19}$

$61 < 95$ Дальнейший поиск в левом подмножестве {61,69,75,79,80,81} (относительно предыдущего подмножества).

Шаг 3

$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 6/2 \rfloor + 1 = 4$

$K \sim k_{16}$

$61 < 79$ Дальнейший поиск в левом подмножестве {61,69,75,79}.

Шаг 4

$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 4/2 \rfloor + 1 = 3$

$K \sim k_{15}$

$61 < 75$ Дальнейший поиск в левом подмножестве {61,69}.

Шаг 5

$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 2/2 \rfloor + 1 = 2$

$K \sim k_{14}$

$61 < 69$ Дальнейший поиск в левом подмножестве.

Шаг 6

$K \sim k_{13}$

$61 = 61$.

Вывод: искомый ключ найден под номером 13.

Фибоначчиев поиск.

В этом поиске анализируются элементы, находящиеся в позициях, равных числам Фибоначчи. Числа Фибоначчи получаются по следующему правилу:

каждое последующее число равно сумме двух предыдущих чисел, например:

{1,2,3,5,8,13,21,34,55,...}.

Поиск продолжается до тех пор, пока не будет найден интервал между двумя ключами, где может располагаться отыскиваемый ключ.

Пример. Дано исходное множество ключей

{3,5,8,9,11,14,15,19,21,22,28,33,35,37,42,45,48,52}

Пусть отыскиваемый ключ равен 42. ($K=42$).

Последовательное сравнение отыскиваемого ключа будет проводиться в позициях, равных числам Фибоначчи: {1,2,3,5,8,13,21,...}

Шаг 1. $K \sim K_1$ $42 > 3 \Rightarrow$ отыскиваемый ключ сравнивается с ключом, стоящим в позиции равной числу Фибоначчи.

Шаг 2. $K \sim K_2$ $42 > 5 \Rightarrow$ сравнение продолжается с ключом, стоящим в позиции равной следующему числу Фибоначчи.

Шаг 3. $K \sim K_3$ $42 > 8 \Rightarrow$ сравнение продолжается

Шаг 4. $K \sim K_5$ $42 > 11 \Rightarrow$ сравнение продолжается

Шаг 5. $K \sim K_8$ $42 > 19 \Rightarrow$ сравнение продолжается

Шаг 6. $K \sim K_{13}$ $42 > 35 \Rightarrow$ сравнение продолжается

Шаг 7. $K \sim K_{18}$ $42 < 52 \Rightarrow$ найден интервал, в котором находится отыскиваемый ключ: от 13 до 18 позиции, т.е. $\{35, 37, 42, 45, 48, 52\}$

В найденном интервале поиск вновь ведётся в позициях равных числам Фибоначчи.

Интерполяционный поиск.

Исходное множество должно быть упорядочено по возрастанию весов.

Первоначальное сравнение осуществляется на расстоянии шага d , который определяется по формуле:

$$d = \left[\frac{(j-i)(K-K_i)}{K_j-K_i} \right]$$

Где:

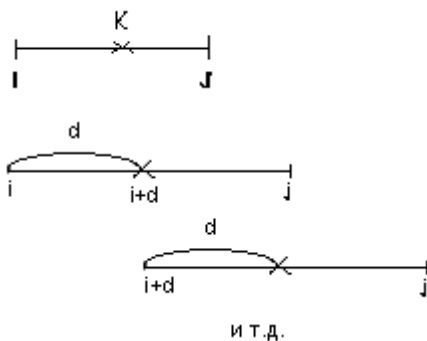
i – номер первого рассматриваемого элемента

j – номер последнего рассматриваемого элемента

K – отыскиваемый ключ

K_i, K_j – значения ключей в i и j позициях

$[]$ – целая часть от числа.



Идея метода заключается в следующем:

шаг d меняется после каждого этапа, по формуле приведённой выше.

Алгоритм заканчивает работу при $d=0$, при этом анализируются соседние элементы, после чего делается окончательное решение.

Этот метод прекрасно работает, если исходное множество представляет собой арифметическую прогрессию или множество, приближенное к ней.

Пример . Дано множество ключей:

$\{2, 9, 10, 12, 20, 24, 28, 30, 37, 40, 45, 50, 51, 60, 65, 70, 74, 76\}$

Пусть искомый ключ равен 70. ($K=70$).

Шаг 1. Определим шаг d для исходного множества ключей:

$$d = \left[\frac{(18-1)(70-2)}{(76-2)} \right] = 15$$

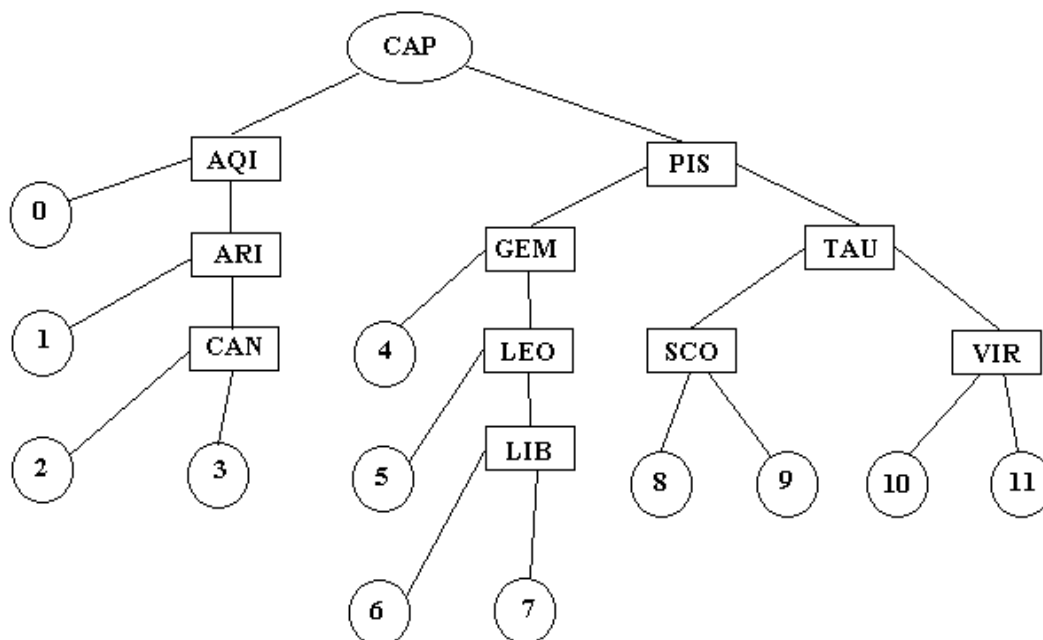
Сравниваем ключ, стоящий под шестнадцатым порядковым номером в данном множестве с искомым ключом:

$K_{16} \sim K$ $70=70$ ключ найден.

Поиск по бинарному дереву.

Использование структуры бинарного дерева позволяет быстро вставлять и удалять записи и производить эффективный поиск по таблице. Такая гибкость достигается добавлением в каждую запись двух полей для хранения ссылок.

Пусть дано бинарное дерево:



Требуется по бинарному дереву отыскать ключ SAG.

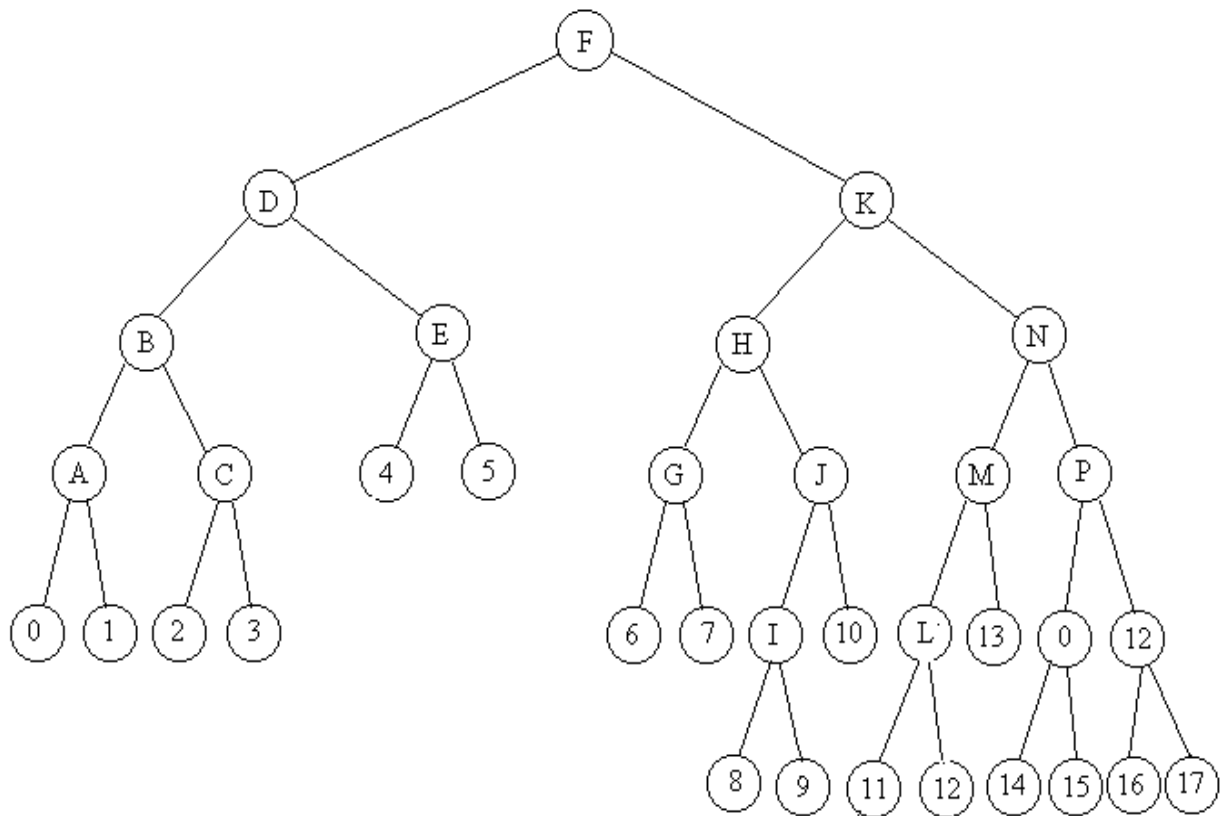
При просмотре от корня дерева видно, что по первой букве латинского алфавита, название SAG больше чем CAP. Следовательно, дальнейший поиск будем осуществлять в правой ветви. Это слово больше чем PIS - снова идем вправо; оно меньше чем TAU - идем влево; оно меньше чем SCO и попадаем в узел 8. Таким образом, название SAG должно находиться в узле 8.

При этом узлы дерева имеют следующую структуру:

Ключ	Информационная часть	Указатель на левое поддерево	Указатель на правое поддерево
	/может отсутствовать/		
KEY		LLINK	RLINK

Поиск по сбалансированному дереву. Бинарное дерево называется сбалансированным, если высота левого поддерева каждого узла отличается от высоты правого не более чем на ± 1 .

Например:



Сбалансированные бинарные деревья занимают промежуточное положение между оптимальными бинарными деревьями (все внешние узлы, которых расположены на двух смежных уровнях) и произвольными бинарными деревьями.

Бинарное дерево называется сбалансированным, если высота левого поддерева каждого узла отличается от высоты правого не более чем на ± 1 .

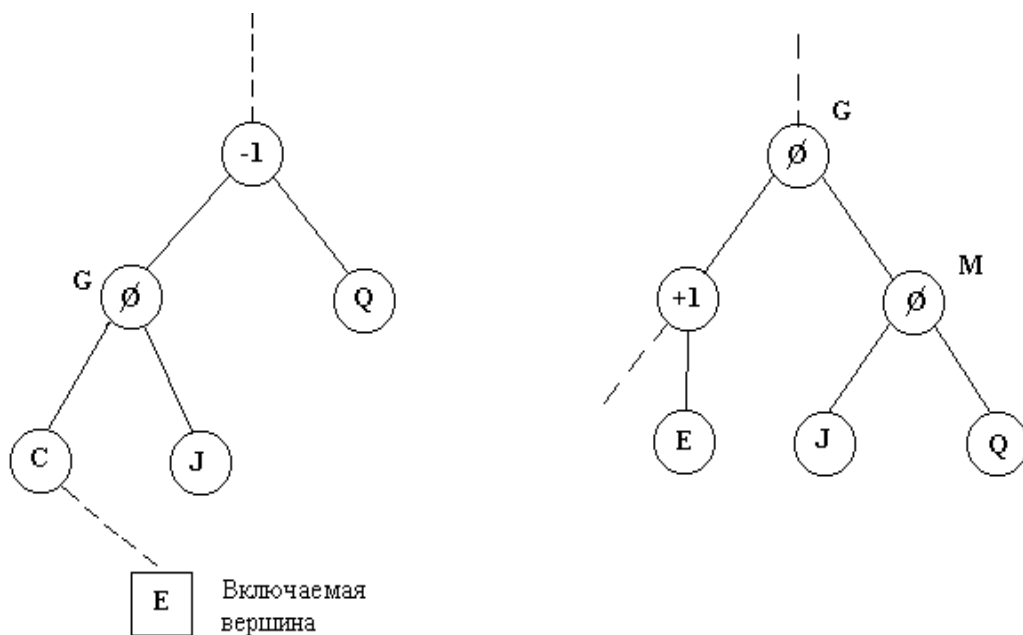
Рассмотрим следующую структуру узлов сбалансированного бинарного дерева:

Ключ	Указатель на левое поддерево	Указатель на правое поддерево	Показатель сбалансированности узла
KEY	LLINK	RLINK	B

где

B – показатель сбалансированности узла, то есть разность высот правого и левого поддерева ($B = +1; 0; -1$).

При восстановлении баланса дерева по высоте учитывается показатель B.



Символы +1, $\emptyset$, -1 указывают, что левое поддерево выше правого, поддеревья равны по высоте, правое поддерево выше левого.

Поиск по бору.

Особую группу методов поиска образует представление ключей в виде последовательности цифр и букв. Рассмотрим, например, имеющиеся во многих словарях буквенные высечки. Тогда по первой букве данного слова можно отыскивать страницы, содержащие все слова, начинающиеся с этой буквы. Развивая идею побуквенных высечек, получим схему поиска, основанную на индексации в структуре бора (термин использует часть слова выборка).

Бор представляет собой m -арное дерево. Каждый узел уровня h представляет множество всех ключей, начинающихся с определенной последовательности из h литер. Узел определяет m -путевое разветвление в зависимости от $(h + 1)$ -ой литеры.

Бор представляет собой таблицу, следующего вида:

Узлы	1	2	3	...	N
Символы					
Пробел()					

Пробел ()- обязательный символ таблицы.

В первом узле записывается первая буква или цифра ключа. Во втором узле к ней добавляется ещё один символ и т.д. Если слово начинающееся с определенной буквы (цифры) единственное, то оно сразу записывается в первом узле.

Пример. Дано множество

{A,AA,AB,ABC,ABCD,ABCA,ABCC,BOR,C,CC,CCC,CCCD,CCCB,CCCA}

От исходного множества перейдём к построению бора.

Исходный алфавит = {A,B,C,D}

BOR- единственное слово на букву B и оно побуквенно не разбивается.

Узлы Символы	1	2	3	4	5	6	7
–		A_	AB_	ABC_	C_	CC_	CCC_
A	2	AA		ABCA			CCCA
B	BOR	3					CCCB
C	5		4	ABCC	6	7	
D				ABCD			CCCD

Узлы бора представляют собой векторы, каждая компонента которых представляет собой либо ключ, либо ссылку (возможно пустую).

Узел 1 – корень, и первую букву следует искать здесь. Если первой оказалась, например, буква В, то из таблицы видно, что ему соответствует слово BOR. Если же первая буква А, то первый узел передает управление к узлу 2, где аналогичным образом отыскивается вторая буква. Узел 2 указывает, что вторыми буквами будут _, А, В и т. д.

Поиск хешированием.

В основе поиска лежит переход от исходного множества к множеству хеш-функций $h(k)$. Хеш-функция имеет следующий вид:

$$h(k) = k \bmod (m),$$

где k -ключ, m - целое число, $\bmod$ -целочисленный остаток от деления.

Например, пусть дано множество {9,1,4,10,8,5}.

Определим для него хеш-функцию $h(k) = k \bmod(m)$;

- Пусть $m=1$, тогда
 $h(k) = \{0, 0, 0, 0, 0, 0\}$;
- Пусть $m=20$, тогда
 $h(k) = \{9, 1, 4, 10, 8, 5\}$;
- Пусть m равно половине максимального ключа, тогда $m = \lfloor 10/2 \rfloor = 5$
 $h(k) = \{4, 1, 4, 0, 3, 0\}$;

Хеш-функция указывает адрес, по которому следует отыскивать ключ. Для разных ключей хеш-функция может принимать одинаковые значения, такая ситуация называется коллизией. Таким образом, поиск хешированием заключается в устранении коллизий.

Пример. Дано множество

{7, 13, 6, 3, 9, 4, 8, 5}

Найти ключ $K=27$.

Хеш-функция равна $h(k) = K \bmod(m)$;

$m = \lfloor 13/2 \rfloor = 6$ (т.к. 13- максимальный ключ)

$h(k) = \{1, 1, 0, 3, 3, 4, 2, 5\}$

Для устранения коллизий построим таблицу.

h(k)	Цепочки ключей
0	6
1	7,12
2	8
3	3, 9
4	4
5	5

По парным сравнением множества хеш-функций и множества исходных ключей, заполняем таблицу. Напоминаем, что хеш-функция указывает адрес, по которому следует отыскивать ключ.

Например, если отыскивается ключ $K=27$, тогда $h(k)=27 \bmod 6=3$ -это значит, что ключ $K=27$ может быть только в 3-й строке. Так как его там нет, то данный ключ отсутствует в исходном множестве.

Алгоритмы поиска словесной информации

В настоящее время наличие сверхпроизводительных микропроцессоров и дешевизна электронных компонентов позволяют делать значительные успехи в алгоритмическом моделировании. Рассмотрим несколько алгоритмов обработки слов.

Алгоритм Кнута - Морриса - Пратта

Алгоритм Кнута-Морриса-Пратта (КМП) получает на вход слово

$$X=x[1]x[2]...x[n]$$

и просматривает его слева направо буква за буквой, заполняя при этом массив натуральных чисел $l[1]...l[n]$, где

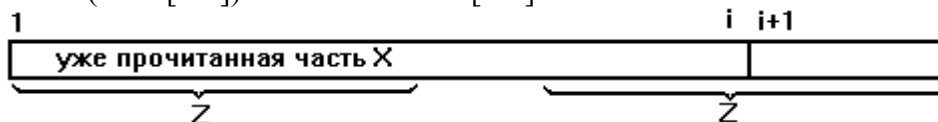
$$l[i]=\text{длина слова } l(x[1]...x[i])$$

Таким образом: $l[i]$ есть длина наибольшего начала слова $x[1]...x[i]$, одновременно являющегося его концом.

Используя алгоритм КМП определить, является ли слово A подсловом слова B ?

Решение. Применим алгоритм КМП к слову $A\#B$, где $\#$ - специальная буква, не встречающаяся ни в A , ни в B . Слово A является подсловом слова B тогда и только тогда, когда среди чисел в массиве l будет число, равное длине слова A .

Предположим, что первые i значений $l[1]...l[i]$ уже найдены. Читается очередная буква слова (т.е. $x[i+1]$) и вычисляется $l[i+1]$.



Другими словами, необходимо определить начала Z слова $x[1]...x[i+1]$,

одновременно являющиеся его концами -из них следует выбрать самое длинное.

Получаем такой рецепт отыскания слова Z . Рассмотрим все начала слова $x[1]...x[i]$, являющиеся одновременно его концами. Из них выберем подходящие - те, за которыми идет буква $x[i+1]$. Из подходящих выберем самое длинное. Приписав в его конец $x[i+1]$, получим искомое слово Z . Теперь пора воспользоваться сделанными приготовлениями и вспомнить, что все слова, являющиеся одновременно началами и концами данного слова, можно получить повторными применениями к нему функции l .

Вот что получается:

```

i:=1; l[1]:=0;
{таблица l[1]..l[i] заполнена правильно}
while i <> n do begin
  len:= l[i]
  {len - длина начала слова x[1]..x[i], которое является
  его концом; все более длинные начала оказались
  неподходящими}
  while (x[len+1]<>x[i+1]) and (len>0) do begin
    {начало не подходит, применяем к нему функцию l}

```

```

len:=l[len];
end;
    {нашли подходящее или убедились в отсутствии}
if x[len+1]=x[i+1] do begin
    {x[1]..x[len] - самое длинное подходящее начало}
    l[i+1]:=len+1;
end else begin
    {подходящих нет}
    l[i+1]:= 0;
end;
i:=i+1;
end;

```

Запишем алгоритм проверяющий, является ли слово $X=x[1]...x[n]$ подсловом слова $Y=y[1]...y[m]$

Решение. Вычисляем таблицу $l[1]...l[n]$ как раньше.

```

j:=0; len:=0;
    {len - длина максимального начала слова X, одновременно
    являющегося концом слова y[1]..j[j]}
while (len<>n) and (j<>m) do begin
    while (x[len+1]<>y[j+1]) and (len>0) do begin
        {начало не подходит, применяем к нему функцию l}
        len:= l[len];
    end;
    {нашли подходящее или убедились в отсутствии}
    if x[len+1]=y[j+1] do begin
        {x[1]..x[len] - самое длинное подходящее начало}
        len:=len+1;
    end else begin
        {подходящих нет}
        len:=0;
    end;
    j:=j+1;
end;
    {если len=n, слово X встретилось; иначе мы дошли до конца
    слова Y, так и не встретив X}

```

Алгоритм Бойера - Мура

Этот алгоритм делает то, что на первый взгляд кажется невозможным: в типичной ситуации он читает лишь небольшую часть всех букв слова, в котором ищется заданный образец. Пусть, например, отыскивается образец *abcd*. Посмотрим на четвертую букву слова: если, к примеру, это буква *e*, то нет никакой необходимости читать первые три буквы. (В самом деле, в образце буквы *e* нет, поэтому он может начаться не раньше пятой буквы.)

Приведем самый простой вариант этого алгоритма, который не гарантирует быстрой работы во всех случаях. Пусть $x[1]...x[n]$ - образец, который надо искать. Для каждого символа s найдем самое правое его вхождение в слово X , то есть наибольшее k , при котором $x[k]=s$. Эти сведения будем хранить в массиве $pos[s]$; если символ s вовсе не встречается, то будет удобно предположить $pos[s]=0$..

Решение.

положить все $pos[s]$ равными 0

```

for i:=1 to n do begin
    pos[x[i]]:=i;
end;

```

В процессе поиска будем хранить в переменной last номер буквы в слове, против которой стоит последняя буква образца. Вначале last=n (длина образца), затем last постепенно увеличивается.

```

last:=n;
{все предыдущие положения образца уже проверены}
while last<= m do begin {слово не кончилось}
    if x[m]<>y[last] then begin {последние буквы разные}
        last:=last+(n-pos[y[last]]);
        {n - pos[y[last]] - это минимальный сдвиг образца,
        при котором напротив y[last] встанет такая же
        буква в образце. Если такой буквы нет вообще,
        то сдвигаем на всю длину образца}
    end else begin
        если нынешнее положение подходит, т.е. если
        x[i]..x[n]=y[last-n+1]..y[last],
        то сообщить о совпадении;
        last:=last+1;
    end;
end;

```

Алгоритм Рабина

Этот алгоритм основан на простой идее. Представим себе, что в слове длины m ищется образец длины n. Вырежем окошко размера n и будем двигать его по входному слову. При этом проверяем, не совпадает ли слово в окошке с заданным образцом. Сравнивать по буквам долго. Вместо этого фиксируем некоторую функцию, определенную на словах длины n. Если значения этой функции на слове в окошке и на образце различны, то совпадения нет. Только если значения одинаковы, нужно проверять совпадение по буквам.

В чем выигрыш при таком подходе. Ведь чтобы вычислить значение функции на слове в окошке, все равно нужно прочесть все буквы этого слова. Так уж лучше их сразу сравнить с образцом. Тем не менее выигрыш возможен, так как при сдвиге окошка слово не меняется полностью, а лишь добавляется буква в конце и убирается в начале. Хорошо бы, чтобы по этим данным можно было рассчитать, как меняется функция.

Заменим все буквы в слове и образце их номерами, представляющими собой целые числа. Тогда удобной функцией является сумма цифр. (При сдвиге окошка нужно добавить новое число и вычесть пропавшее.)

Выберем некоторое число p (желательно простое) и некоторый вычет x по модулю p. Каждое слово длины n будем рассматривать как последовательность целых чисел (заменив буквы кодами). Эти числа будем рассматривать как коэффициенты многочлена степени n-1 и вычислим значение этого многочлена по модулю p в точке x. Это и будет одна из функций семейства (для каждой пары p и x получается, таким образом, своя функция). Сдвиг окошка на 1 соответствует вычитанию старшего члена (x^{n-1} следует вычислить заранее), умножению на x и добавлению свободного члена.

Следующее соображение говорит в пользу того, что совпадения не слишком вероятны. Пусть число p фиксировано и k тому же простое, а X и Y - два различных слова длины n. Тогда им соответствуют различные многочлены (предполагаем, что коды всех

букв различны - это возможно, если p больше числа букв алфавита). Совпадение значений функции означает, что в точке x эти два различных многочлена совпадают, то есть их разность обращается в 0. Разность есть многочлен степени $n-1$ и имеет не более $n-1$ корней. Таким образом, если n много меньше p , то случайному x мало шансов попасть в неудачную точку.

Лабораторное задание

Для каждого из перечисленных методов поиска провести анализ временных затрат для списков различной размерности.

Методика выполнения лабораторной работы

Для проведения лабораторной работы необходимо выполнить следующие действия.

1. Выполнить тестовый пример поиска с использованием программы **LAB_FIND**.

Путь к файлу : D:\ИПОВС\АиСД\POISK\ Lab_Find.exe

- а) выполнить поиск в массиве из N чисел (варианты заданий даны в приложении ; номер варианта соответствует порядковому номеру фамилии студента в списке группы). Результаты занести в таблицу 1.

Таблица 1

Метод			
Количество элементов массива N	N_1	N_2	N_3
Время поиска t , с	t_1	t_2	t_3

- б) оценить сложность рассмотренных методов поиска;
 - в) провести анализ отклонения полученной в результате эксперимента сложности алгоритма от теоретической;
 - г) построить графические зависимости времени поиска от количества элементов массива.
2. Исследовать методы поиска с использованием программ **PoiskNew** и **Poisk**.
Путь : D:\ИПОВС\АиСД\POISK\ PoiskNew (Poisk)
 3. Провести исследование метода хеширования
Путь : D:\ИПОВС\АиСД\POISK\ Хеширование

Требования к отчёту

Отчёт должен содержать:

- 1) конспект лабораторной работы;
- 2) примеры методов поиска ;
- 3) результаты выполнения работы;
- 4) выводы по работе;

Контрольные вопросы

1. Что понимается под поиском?
2. Каковы особенности поиска: последовательного, бинарного, интерполяционного, Фибоначчиевого, по бинарному дереву , по бору , хешированием ;

3. В чём состоит методика анализа сложности алгоритмов поиска?

Рекомендуемая литература:

1. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М. : Мир, 2000.
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест «Алгоритмы: построение и анализ». М.: МЦНМО, 2000.
3. Вирт Н. Алгоритмы и структуры данных.: Пер. С англ. - М.: Мир, 2001.
4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.

Варианты заданий

Провести анализ методов поиска для списков различной размерности и построить зависимости времени от числа элементов исходного множества.

Вариант	n1	n2	n3
1,15	1000	4000	6000
2,16	800	3000	7000
3,17	2000	5000	6800
4,18	1500	3500	6000
5,19	1000	2800	8500
6,20	500	2500	6500
7,21	900	3000	8500
8,22	1200	2000	7500
9,23	2500	3500	6500
10,24	1600	2800	8800
11,25	700	3400	7200
12,26	1900	2700	8000
13,27	1300	3500	6000
14,28	1700	2800	7500

Вариант	Составить программу
1,15	Бинарный поиск
2,16	Интерполяционный поиск
3,17	Поиск хешированием
4,18	Фибоначчиев поиск
5,19	Поиск по бинарному дереву
6,20	Поиск по бору
7,21	Фибоначчиев поиск
8,22	Поиск по бору
9,23	Интерполяционный поиск
10,24	Поиск по бору
11,25	Поиск по бинарному дереву
12,26	Поиск хешированием
13,27	Бинарный поиск
14,28	Фибоначчиев поиск

Использовать алгоритмы Кнута - Морриса – Пратта, Бойера – Мура, Рабина для поиска текстовой информации.

Лабораторная работа № 3 «ИТЕРАТИВНЫЕ И РЕКУРСИВНЫЕ АЛГОРИТМЫ»

Цель работы: изучить рекурсивные алгоритмы и рекурсивные структуры данных; научиться проводить анализ итеративных и рекурсивных процедур; исследовать эффективность итеративных и рекурсивных процедур при реализации на ПЭВМ.

Продолжительность работы: 2 часа.

Теоретические сведения

Эффективным средством программирования для некоторого класса задач является рекурсия. С ее помощью можно решать сложные задачи численного анализа, комбинаторики, алгоритмов трансляции, операций над списковыми структурами и т.д. Программы в этом случае имеют небольшие объемы по сравнению с итерацией и требуют меньше времени на отладку.

Под рекурсией понимают способ задания функции через саму себя, например способ задания факториала в виде

$$N! = (n-1)! * n$$

В программировании под рекурсивной процедурой (функцией) понимают способ обращения процедуры (функции) к самой себе.

Под итерацией понимают результат многократно повторяемой какой-либо операции, например представление факториала в виде:

$$n! = 1 * 2 * 3 \dots * n.$$

Среди широкого класса задач удобно представлять с использованием рекурсивных процедур (функций) те задачи, которые сводятся на подзадачи того же типа, но меньшей размерности.

Общая методика анализа рекурсии содержит три этапа:

1. Параметризация задачи, заключающаяся в выделении различных элементов, от которых зависит решение, в частности размерности решаемой задачи. После каждого рекурсивного вызова размерность должна убывать.
2. Поиск тривиального случая и его решение. Как правило, это ключевой этап в рекурсии, размерность задачи при этом часто равна 0 или 1.
3. Декомпозиция общего случая, имеющая целью привести задачу к одной или нескольким задачам того же типа, но меньшей размерности.

Рассмотрим понятие итеративного и рекурсивного алгоритмов на примере вычисления факториала.

Итеративный алгоритм. Наиболее простой и естественной формой представления итеративного алгоритма при реализации на ПЭВМ является описание его с использованием цикла. Программа итеративного алгоритма вычисления факториала представлена на рис.1. Программа состоит из процедуры-функции FACTORIAL и основной программы. В основной программе происходит ввод значения N, вызов процедуры-функции и печать результата.

```
{ИТЕРАТИВНОЕ ВЫЧИСЛЕНИЕ ФАКТОРИАЛА }  
PROGRAM FI;  
VAR  
FAC: LONGINT;  
N: INTEGER;  
{ ФУНКЦИЯ ВЫЧИСЛЕНИЯ ФАКТОРИАЛА }  
FUNCTION FACTORIAL (N: INTEGER): LONGINT;
```

```

    VAR  F: LONGINT;
    I: INTEGER;
BEGIN
    IF (N=0) OR (N=1) THEN FACTORIAL:=1 ELSE
    BEGIN
        F:= 1;
        FOR I:= 2 TO N DO
            F:= F * I;
        FACTORIAL:=F;
    END;
END;
{ ОСНОВНАЯ ПРОГРАММА }
BEGIN
Writeln ('ВВЕДИТЕ ЗНАЧЕНИЕ N')
Readln (N);
FAC:= FACTORIAL(N);    { ВЫЗОВ ФУНКЦИИ FACTORIAL }
Writeln(' ФАКТОРИАЛ =', FAC) ;
Readln;
END.

```

Рекурсивный алгоритм. Рекурсивное представление факториала имеет вид:

$$n! = (n-1)! * n$$

Проведем анализ этого выражения согласно методике.

1. Параметризация. В данном случае имеется всего один параметр N- целое число.
2. Поиск тривиального случая. При n=0 или n=1 значение факториала равно 1, что соответствует выходу из рекурсии.
3. Декомпозиция общего случая: n! вычисляется через меньшую размерность этой же задачи
(n-1)!

Программа рекурсивного алгоритма вычисления факториала представлена ниже.

```

{РЕКУРСИВНОЕ ВЫЧИСЛЕНИЕ ФАКТОРИАЛА}
PROGRAM FR ;
VAR
fac: longint;
n: integer;
{Функция вычисления факториала}
FUNCTION factorial (n: integer): longint ;
Begin
If (n=0) or (n=1) then factorial:= 1
Else factorial:= factorial (n-1)*n;
End;

{ ОСНОВНАЯ ПРОГРАММА }
BEGIN
Writeln (' введите значение n');
Readln (n);
fac:= factorial (n); { ВЫЗОВ ФУНКЦИИ FACTORIAL }
Writeln ('факториал =', fac);
Readln;
END.

```

Процедура-функция имеет следующие особенности:

- при вычислении факториала происходит обращение функции к самой себе (подчеркнуто в выражении), но с меньшим значением аргумента $N-1$ по сравнению с первым вызовом N .

$$\text{FACTORIAL } N: = \underline{\text{FACTORIAL } (N-1)} * N$$

- при вычислении факториала не используется цикл, что является существенной особенностью рекурсивного алгоритма.

Рассмотрим последовательность действий при рекурсивном вычислении факториала для $n = 3$:

- 1) внешний вызов из основной программы FACTORIAL (3);
- 2) первый рекурсивный вызов FACTORIAL (2);
в операторе $\text{FACTORIAL}: = \underline{\text{FACTORIAL } (N-1)} * n$,
где не происходят никакие вычисления - только вызов (подчеркнуто);
- 3) второй рекурсивный вызов FACTORIAL (1);
- 4) получение значения FACTORIAL (1): =1;
- 5) возврат из второго рекурсивного вызова и вычисление факториала $\text{FACTORIAL } (2): = 1 * 2 = 2$;
- 6) возврат из первого рекурсивного вызова и вычисление факториала $\text{FACTORIAL } (3): = 2 * 3 = 6$;
- 7) возврат в основную программу FAC: =6.

В программу рекурсивного вычисления факториала можно добавить стандартную функцию текущего времени GETTIME(Var Hour, Minute, Second, Sec100: WORD). Эта процедура может быть использована при подключении модуля DOS (USES DOS).

Рекурсивные структуры данных

Список (list) – набор элементов, расположенных в определенном порядке.

Список очереди (pushup list) – список, в котором последний поступающий элемент добавляется к нижней части списка.

Список с использованием указателей (linked list) – список, в котором каждый элемент содержит указатель на следующий элемент списка.

Однонаправленный и двунаправленный список – это линейный список, в котором все исключения и добавления происходят в любом месте списка.

Однонаправленный список отличается от двунаправленного списка только связью. То есть в однонаправленном списке можно перемещаться только в одном направлении (из начала в конец), а двунаправленном – в любом (рис.1.)

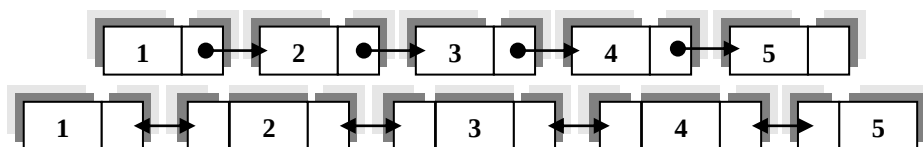
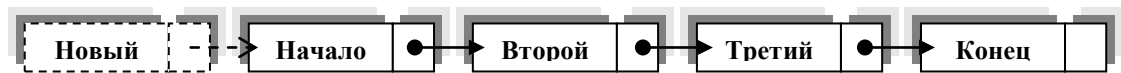


Рис.1 Однонаправленный и двунаправленный список

В однонаправленном списке структура добавления и удаления такая же только связь между элементами односторонняя.

Очередь — тип данных, при котором новые данные располагаются следом за существующими в порядке поступления; поступившие первыми данные при этом обрабатываются первыми.

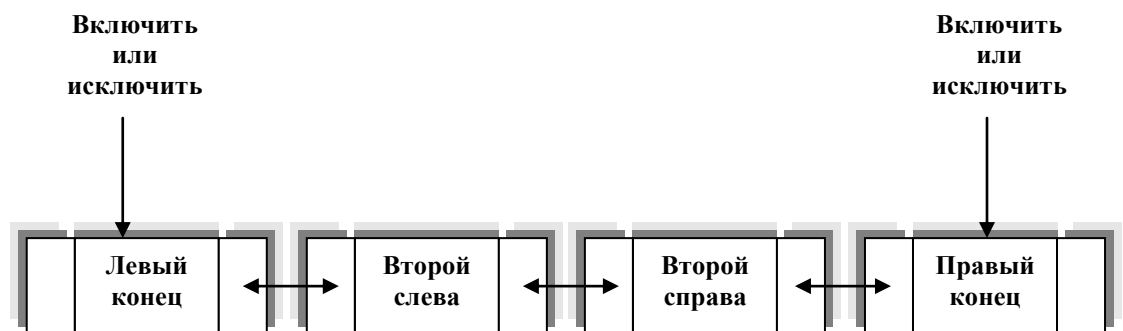
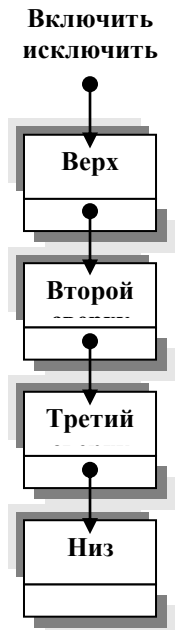


Очередь иногда называют — циклической памятью или списком типа FIFO ("first-in-first-out" — "первым включается — первым исключается"). Другими словами, у очереди есть голова (head) и хвост (tail).

В очереди новый элемент добавляется только с одного конца. Удаление элемента происходит на другом конце. В данном случае это может быть только 4 элемент. Очередь по сути однонаправленный список, только добавление и исключение элементов происходит на концах списка.

Стек (stack) — линейный список, в котором все включения и исключения делаются в одном конце списка. Стек называют пуш-даун (push-down) списком, реверсивной памятью, гнездовой памятью, магазином, списком типа LIFO ("last-in-first-out" — "последним включается — первым исключается"). Стек — часть памяти ОЗУ компьютера, которая предназначена для временного хранения байтов, используемых микропроцессором. Действия со стеком производится при помощи регистра указателя стека. Любое повреждение этой части памяти приводит к фатальному сбою.

Дек (deck) (стек с двумя концами) — линейный список, в котором все включения и исключения делаются на обоих концах списка. Еще один термин "архив" применялся к декам с ограниченным выходом, а деки с ограниченным входом называли "перечнями", или "реестрами".



Рекурсивно можно представить не только алгоритм решения задачи, но и обрабатываемую информацию. Рассмотрим применение рекурсивных процедур при обработке рекурсивных структур данных.

Предварительно обратимся к понятиям списка, указателя и динамической переменной языка Паскаль.

Информационную часть можно описать как INTEGER (целый тип), REAL (действительный), CHAR (символьный) и т.д. Для отображения указателя (горизонтальной стрелки) в языке Паскаль введен особый тип данных, который называется указателем. Для его описания нет ключевого слова, вместо него введен

символ ^.

Структуру данных, рассмотренную в виде списка, можно представить на языке Турбо Паскаль следующим образом:

```
TYPE LINK = ^ELEMENT;  
      ELEMENT = RECORD  
        INFORM: CHAR;  
        NEXT: LINK;  
      END;
```

Здесь LINK - указатель, указывает на ELEMENT. Структура элемента ELEMENT отражена в виде записи, состоящей из двух частей: INFORM и NEXT.

Обратите внимание на характер структуры данных. В начале описания тип данных LINK указывает на ELEMENT у которого, в свою очередь, одна из составляющих NEXT является типом указателя LINK, а LINK указывает на ELEMENT. Получается замкнутый круг. Такая структура данных называется рекурсивной.

В разделе определения типов допускается менять местами описание указателя и описание элемента, например

```
TYPE ELEMENT = RECORD  
  INFORM: CHAR;  
  NEXT: LINK;  
END;  
LINK = ^ELEMENT;
```

Чтобы иметь возможность использовать в программе переменные типа ELEMENT (то есть переменные, имеющие такую же структуру), необходимо описать их как переменные типа указателя, например

```
VAR A, B, C: LINK;
```

Где A, B, C называются переменными-указателями, которые обозначаются в программе со стрелкой ^: A ^, B ^, C ^.

Доступ к элементу записи осуществляется с указанием составного (уточненного) имени, содержащего внутри себя символ точки, например:

```
A^. INFORM: = 'Z';  
C^. INFORM: = B^. INFORM;
```

Каждый тип указателей среди своих возможных значений содержит значение NIL (зарезервированное слово языка Паскаль), которое не указывает ни на один элемент. Чаще всего NIL используется при формировании начала или конца списка, например

```
A^. LINK: = NIL;
```

Следует обратить внимание на важный факт: в определении типа данных переменные-указатели A, B, C описаны как указатели (VAR A, B, C : LINK) а не как переменные типа ELEMENT. В этом случае при трансляции память выделяется только для указателей, а для переменных, имеющих структуру записи ELEMENT не выделяется. В языке Паскаль существует понятие динамической переменной, которая начинает существовать в результате вызова стандартной процедуры NEW, например NEW (A). Это означает выделение области памяти и формирование её адреса для создания новой переменной, имеющей структуру записи.

Таким образом, если в программе была объявлена переменная типа указателя, то в результате вызова NEW (A) формируется переменная типа ELEMENT (состоящая из двух

частей INFORM и NEXT). Только после этого возможен доступ к элементу записи, например

$A^{\wedge}.INFORM := 'Z';$ или $A^{\wedge}.LINK := NIL;$

Если динамическая переменная становится ненужной, то выделенная область памяти освобождается с помощью стандартной процедуры DISPOSE, например DISPOSE (A).

Задача. Составить программу на языке Паскаль для формирования обхода узлов бинарного дерева.

Набор способа обхода дерева позволяет ввести отношение порядка для узлов дерева.

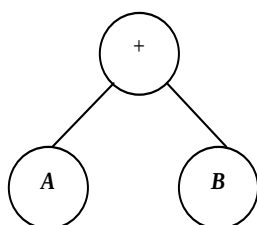
Решение. Для размещения узлов дерева в памяти ПЭВМ воспользуемся списковой структурой данных. Каждый элемент этой структуры соответствует некоторому узлу дерева и описывается записью следующего вида:

```
TREE=RECORD
    LEFT: ^TREE;
    RIGHT: ^TREE;
    IDENT: CHAR;
END;
```

где LEFT и RIGHT - указатели (адреса) элементов структуры данных, представляющие узлы, которые являются соответственно корнями левого и правого поддерева данного узла, в поле IDENT находится односимвольный идентификатор узла.

Наиболее распространены три способа обхода узлов дерева, которые получили следующие названия:

- обход в направлении слева направо (обратный порядок, инфиксная запись);
- сверху вниз (прямой порядок, префиксная запись);
- снизу вверх (концевой порядок, постфиксная запись).



<i>КЛП</i>	<i>Прямой</i>	<i>+AB</i>	<i>Префиксная</i>
<i>ЛКП</i>	<i>Обратный</i>	<i>A+B</i>	<i>Инфиксная</i>
<i>ЛПК</i>	<i>Концевой</i>	<i>AB+</i>	<i>Постфиксная</i>

Рис.2. Три различных обхода по бинарному дереву и соответствующие формы записи

В результате обхода дерева, приведенного на рис.3, каждым из трех способов порождаются следующие последовательности прохождения узлов:

слева направо: **A+B*C-D** (обратный порядок);

сверху вниз: ***+AB-CD** (прямой порядок);

снизу вверх: **AB+CD-*** (концевой порядок).

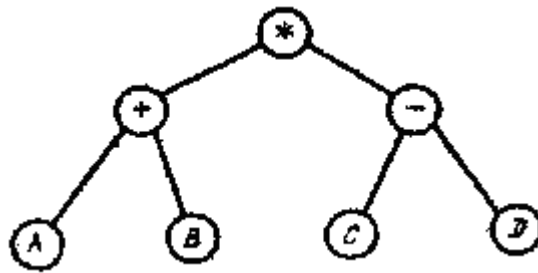


Рис.3. Бинарное дерево

Ниже приведены программы для создания в памяти ПЭВМ списковой структуры бинарного дерева (процедура CREATE) и обхода его узлов в порядке сверху вниз. (процедура PREORDER)

Листинг 1.

```
{ ФОРМИРОВАНИЕ БИНАРНОГО ДЕРЕВА }
TYPE                                { ОПИСАНИЕ ДЕРЕВА }
NODE = ^TREE;                      { ТИП УКАЗАТЕЛЯ УЗЛА ДЕРЕВА      }
TREE = RECORD                      { СТРУКТУРА УЗЛА БИНАРНОГО ДЕРЕВА }
    LEFT : ^TREE;                  { УКАЗАТЕЛЬ ЛЕВОГО ПОДДЕРЕВА  }
    RIGHT : ^TREE;                 { УКАЗАТЕЛЬ ПРАВОГО ПОДДЕРЕВА }
    IDENT : CHAR;                  { ИДЕНТИФИКАТОР УЗЛА ДЕРЕВА  }
END;
PROCEDURE CREATE(VAR A:NODE);

{ РЕКУРСИВНАЯ ПРОЦЕДУРА СОЗДАНИЯ В ОПЕРАТИВНОЙ ПАМЯТИ
СТРУКТУРЫ БИНАРНОГО ДЕРЕВА В ПЕРЕМЕННОЙ "А" ФОРМИРУЕТСЯ АДРЕС
КОРНЯ ПОЛУЧЕННОГО ДЕРЕВА ИЛИ ПОДДЕРЕВА }

VAR
    B : NODE;                      { АДРЕС ПОДДЕРЕВА ДАННОГО УЗЛА  }
    R : CHAR;                      { РАБОЧАЯ ПЕРЕМЕННАЯ          }
BEGIN
    NEW(A);                        { РЕЗЕРВИРОВАНИЕ ПАМЯТИ ДЛЯ НОВОГО УЗЛА
    ДЕРЕВА }
    WRITE(' ВВЕДИТЕ ИМЯ УЗЛА> ');
    READLN(A^.IDENT);
    WRITE(' ЕСТЬ ЛЕВОЕ ПОДДЕРЕВО У УЗЛА ',A^.IDENT,' ? (Y/N) ');
    READLN(R);
    IF (R='Y') THEN
        BEGIN
            CREATE(B);              { ФОРМИРОВАНИЕ ЛЕВОГО ПОДДЕРЕВА УЗЛА }
            A^.LEFT:=B;
        END
    ELSE
        A^.LEFT:=NIL;              { У ДАННОГО УЗЛА НЕТ ЛЕВОГО ПОДДЕРЕВА }

    WRITE(' ЕСТЬ ПРАВОЕ ПОДДЕРЕВО У УЗЛА ',A^.IDENT,' ? (Y/N) ');
    READLN(R);
    IF (R='Y') THEN
        BEGIN
            CREATE(B);              { ФОРМИРОВАНИЕ ПРАВОГО ПОДДЕРЕВА УЗЛА }
```

```

        A^.RIGHT:=B;
    END
    ELSE
        A^.RIGHT:=NIL;    { У ДАННОГО УЗЛА НЕТ ПРАВОГО ПОДДЕРЕВА }
    END;

```

Листинг 2.

```

{ РЕКУРСИВНАЯ ПРОЦЕДУРА ОБХОДА УЗЛОВ ДЕРЕВА СВЕРХУ-ВНИЗ }
PROCEDURE PREORDER(A:NODE);
BEGIN
    IF A<>NIL THEN
        BEGIN
            WRITE(A^.IDENT:2);           { ПЕЧАТЬ ИДЕНТИФИКАТОРА УЗЛА }
            PREORDER(A^.LEFT);           { ОБХОД ЛЕВОГО ПОДДЕРЕВА }
            PREORDER(A^.RIGHT);          { ОБХОД ПРАВОГО ПОДДЕРЕВА }
        END;
    END;
END;

```

В процедуре CREATE функция NEW(A) резервирует в памяти ПЭВМ область для размещения записи типа TREE. В операторе вводится идентификатор текущего узла дерева (один символ) и заносится в поле IDENT записи, адрес которой в данный момент хранятся в переменной A. Далее на экран выдается запрос, есть ли левое поддерево у данного узла. Если в ответ введен символ Y (да), то рекурсивно вызывается процедура CREATE для формирования левого поддерева. После ее завершения в переменной B возвращается адрес узла – корня левого поддерева и он запоминается в поле LEFT текущей записи.

Аналогично формируется правое поддерево. Выход из рекурсии происходит при обработке концевых вершин дерева. В записи, представляющей эти узлы, в поля LEFT и RIGHT заносится константа NIL - неопределенный адрес.

Если проанализировать последовательность прохождения узлов в порядке сверху вниз, то можно установить следующее. Для любого узла, например *, сначала фиксируется факт прохождения через данный узел, затем просматриваются все узлы, входящие в его левое поддерево, а в последнюю очередь просматриваются узлы, составляющие его правое поддерево. Тогда алгоритм обхода бинарного дерева в порядке сверху вниз имеет следующий вид.

Шаг 1. Посетить корень дерева (напечатать его идентификатор),

Шаг 2. Пройти сверху вниз левое поддерево корневого узла.

Шаг 3. Пройти сверху вниз правое поддерево.

Описанный алгоритм реализуется в приведенном примере рекурсивной процедурой. Условие $A \neq \text{NIL}$ позволяет обнаружить концевые вершины дерева и обеспечивает выход из рекурсии.

В основной программе выполняется последовательное обращение к описанным выше подпрограммам создания и обхода бинарного дерева. Заметим, что начальное значение переменной указателя ROOT определяется в процедуре CREATE. Это значение используется для указания адреса корневого узла сформированного бинарного дерева при обращении к процедуре обхода его узлов в порядке сверху вниз.

Лабораторное задание

Для проведения лабораторной работы необходимо выполнить следующие действия.

- 1). Исследовать работу стека, очереди и дека, определив алгоритмы работы указанных структур данных. Результаты записать в тетрадь.
Используя программу **DataStruct**, задать исходное множество элементов, содержащее 12- 15 элементов. Путь к программе:
Путь к файлу: D:\ИПОВС\АиСД\ DataStruct

Система работает в диалоговом режиме с использованием “меню”. Вся необходимая информация во время работы системы отображается на экране дисплея и не требует специальных пояснений.

- 2). Используя программы **Obhod1** либо **Obhod2**, сформировать бинарные деревья, содержащие 8, 10, 12 элементов и выполнить для каждого из них три вида обхода дерева.

Путь к программе: D:\ИПОВС\АиСД\ Obhod1 (Obhod2)

- 3). Составить и отладить программу итеративного и рекурсивного алгоритмов вычисления суммы.

$$\sum_{i=1}^n i$$

аналогично программам вычисления факториала.

Задание выполняется при домашней подготовке. Для определения времени выполнения программы (в секундах) необходимо воспользоваться стандартной встроенной функцией GETTIME .

Сравнить время выполнения этих алгоритмов.

4. Используя программу **TREE_WD**, сформировать бинарные деревья, содержащие 9, 10, 11 элементов и выполнить для каждого из них балансировку.

Требования к отчету

Отчет должен содержать:

- 1)конспект лабораторной работы;
- 2)три вида обхода дерева для своего варианта;
- 3)программу итеративного рекурсивного вычисления суммы;
- 4)результаты выполнения работы;
- 5)вывода по работе.

Контрольные вопросы

- 1.Что понимается под рекурсией и итерацией в математике?
- 2.Каковы особенности итеративного алгоритма?
- 3.Каковы особенности рекурсивного алгоритма?
- 4.В чем состоит методика анализа рекурсивного алгоритма?
- 5.В каких случаях целесообразно использовать рекурсивный или итеративный алгоритм
- 6.Приведите примеры итерации и рекурсии.
- 7.Все ли языки программирования дают возможность рекурсивного вызова процедур?
- 8.Приведите пример рекурсивной структуры данных.
- 9.Что такое указатели и динамические переменные в языке Паскаль?

ЛИТЕРАТУРА

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест «Алгоритмы: построение и анализ». М.: МЦНМО, 2000.
2. Вирт Н. Алгоритмы и структуры данных.: Пер. С англ. - М.: Мир, 2001.
3. Полосухин Б.М., Поддубная Л.М. Рекурсивные функции и алгоритмы. М.: МИЭТ, 1985.
4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.

Лабораторная работа № 4

«Алгоритмы построения остовного дерева сети».

Цель работы: ознакомление с алгоритмами построения остовного дерева графа (сети) и методикой оценки их эффективности.

Продолжительность работы: - 2 часа.

Теоретические сведения

Предположим, что необходимо принять решение, связанное с организацией сети компьютеров в различных территориальных пунктах. Это решение является довольно сложным и зависит от большого количества факторов, которые включают в себя вычислительные ресурсы, доступные в каждом пункте, соответствующие уровни потребностей, пиковые нагрузки на систему, возможное неэффективное использование основного ресурса в системе и, кроме того, стоимость предлагаемой сети. В эту стоимость входят приобретение оборудования, прокладка линий связи, обслуживание системы и т.д. Необходимо определить стоимость такой сети.

Нетрудно видеть, что сформулированная здесь задача имеет много других аналогов. Например, требуется соединить несколько населенных пунктов линиями телефонной связи таким образом, чтобы все эти пункты были связаны в сеть, и чтобы стоимость прокладки коммуникаций была минимальной. Вместо телефонных линий можно говорить о прокладке водопроводных коммуникаций, о строительстве дорог и т. д. Решение подобных задач возможно с использованием теории графов (сетей).

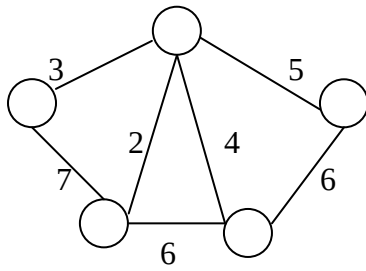
Пусть $G=(V,E)$ - связный неориентированный граф, содержащий циклы, т.е. замкнутые маршруты, где V – множество вершин, а E – множество ребер. Остовным (покрывающим) деревом называется подграф, не содержащий циклов, включающий все вершины исходного графа, для которого сумма весов ребер минимальна.

Введем понятие цикломатического числа (γ) показывающего, сколько ребер на графе нужно удалить, чтобы в нём не осталось ни одного цикла. Цикломатическое число равно, увеличенной на единицу разности между количеством ребер и количеством вершин графа.

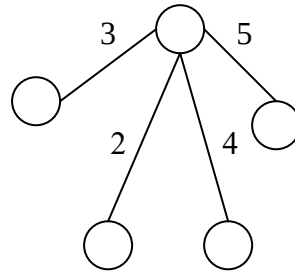
$$\gamma = n - m + 1, \text{ где } n - \text{ количество ребер, } m - \text{ количество вершин,}$$

Например, для графа, изображенного на рисунке, цикломатическое число равно

$$\gamma = n - m + 1 = 7 - 5 + 1 = 3$$



Это значит , что если на графе убрать три ребра , то в нем не останется ни одного цикла, а суммарный вес ребер равен 14.



Для построения остовного дерева графа используются алгоритмы Крускала и Прима. На рис.1 показана схема алгоритма Крускала.

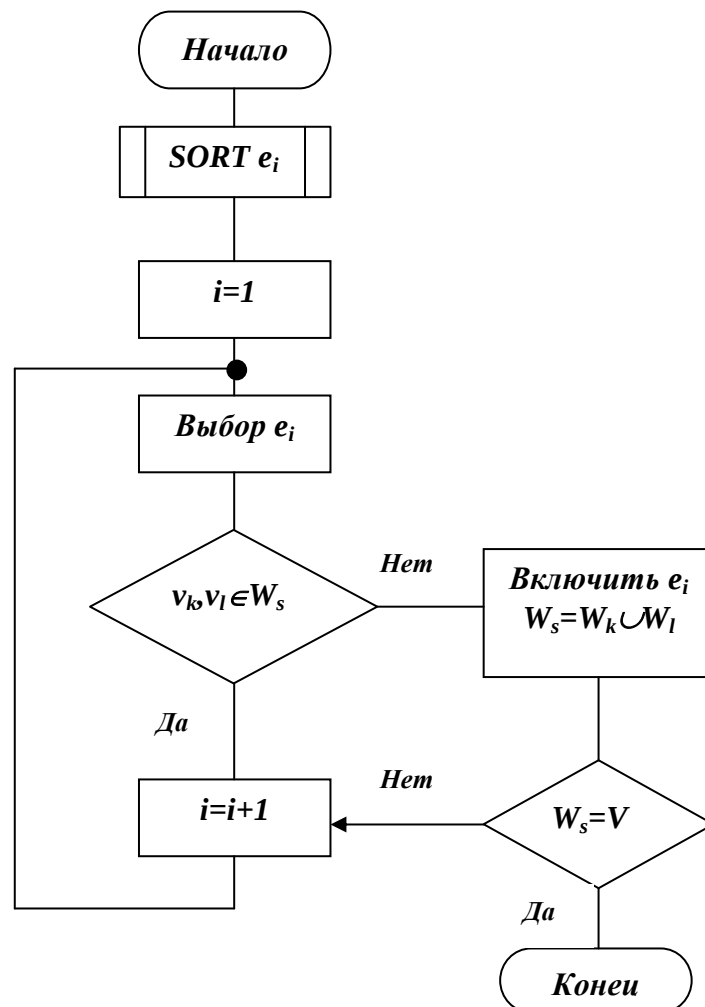


Рис.1. Схема алгоритма Крускала

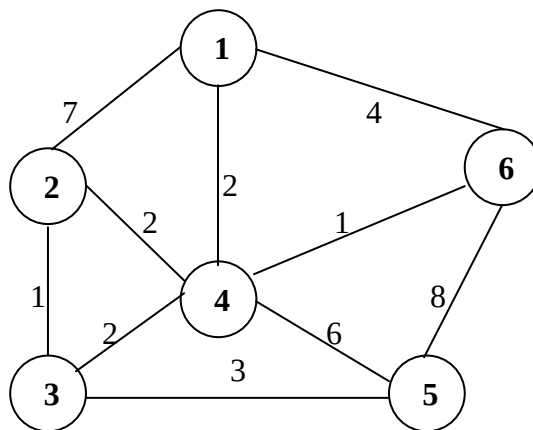
Блок **Sort e_i** предполагает предварительную сортировку весов ребер в порядке их возрастания. В начале работы алгоритма предполагается, что в искомом остове не проведено ни одно ребро (т.е. остов состоит из изолированного множества вершин $v_1, v_2, \dots, v_m$, где m - количество вершин графа). Поэтому считается, что множество W_s имеет вид: $W_s = \{ \{v_1\}, \{v_2\}, \dots, \{v_n\} \}$, где $\{v_i\}$ обозначает множество, состоящее из единственной изолированной вершины v_i . Проверка $v_k, v_l \in W_s$ предполагает установление факта: входят ли вершины v_k, v_l во множество W_s как изолированные, или они сами входят в подмножества постепенно увеличивающихся связанных вершин W_k, W_l , каждое из которых имеет вид: $W_k = \{ \dots, v_k, \dots \}$ и $W_l = \{ \dots, v_l, \dots \}$.

Если обе вершины v_k, v_l содержатся в одном из подмножеств W_k, W_l , то ребро (k, l) в остов не включается, в противном случае данное ребро включается в остов, а множества W_k, W_l объединяются. Работа алгоритма Крускала заканчивается, когда множество W_s совпадет по мощности с множеством V - множеством всех вершин графа. Нетрудно видеть, что это произойдет, когда все вершины графа окажутся связными.

Пример 1. Пусть дана схема микрорайона.

Необходимо соединить дома телефонным кабелем, таким образом, чтобы его длина была минимальной.

Схему микрорайона представим взвешенным графом.



Определим цикломатическое число графа $\gamma = n - m + 1 = 10 - 6 + 1 = 5$, т.е. на графе необходимо удалить 5 ребер.

Первоначально каждая вершина исходного графа помещается в одноэлементное подмножество, то есть считаем, что все вершины изолированы, т.е. не связаны. Ребро включается в остовное дерево, если оно связывает вершины, принадлежащие разным подмножествам, при этом вершины объединяются в новое подмножество.

Ребро	Вес	Множества вершин	Операция
{V1},{V2},{V3},{V4},{V5},{V6}			
{V2,V3}	1	{V2,V3},{V1},{V4},{V5},{V6}	Вкл.
{V4,V6}	1	{V2,V3},{V4,V6},{V1},{V5}	Вкл.
{V1,V4}	2	{V2,V3},{V1,V4,V6},{V5}	Вкл.
{V3,V4}	2	{V1,V2,V3,V4,V6},{V5}	Вкл.
{V2,V4}	2	—————	Искл.
{V3,V5}	3	{V1,V2,V3,V4,V5,V6}	Вкл.

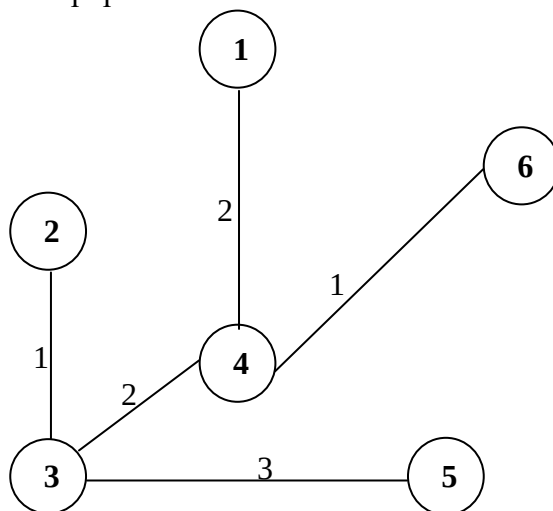
В таблицу последовательно включаются ребра в порядке возрастания их весов. Ребро {V2,V3} связывает две вершины, принадлежащие разным подмножествам {V2} и {V3}. Поэтому ребро включается в остовное дерево, а вершины объединяются в одно подмножество {V2,V3}.

Ребро {V4,V6} также связывает вершины из разных подмножеств, оно включается в остовное дерево, а вершины образуют подмножество {V4,V6}.

Вершины V2 и V4 находятся в одном подмножестве, поэтому ребро {V2,V4} исключается из рассмотрения.

Алгоритм заканчивает работу, когда все вершины объединяются в одно множество, при этом оставшиеся ребра не включаются в остовное дерево.

Последовательно просматривая таблицу, получим схему соединения телефонным кабелем домов в микрорайоне.

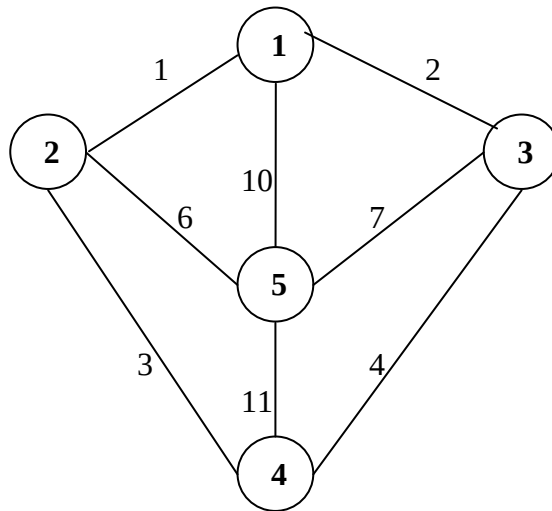


В методе Прима от исходного графа переходим к его представлению в виде матрицы смежности. На графе выбирается ребро минимального веса. Выбранное ребро вместе с вершинами образует первоначальный фрагмент остовного дерева. Затем анализируются длины ребер от каждой вершины фрагмента до оставшихся невыбранных вершин. Выбирается минимальное ребро и присоединяется к первоначальному фрагменту, и т. д.

Процесс продолжается до тех пор, пока в остовное дерево не будут включены все вершины исходного графа.

Пример 2. Пусть дана схема компьютерной сети.

Необходимо соединить компьютеры таким образом, чтобы длина проводки была минимальной.



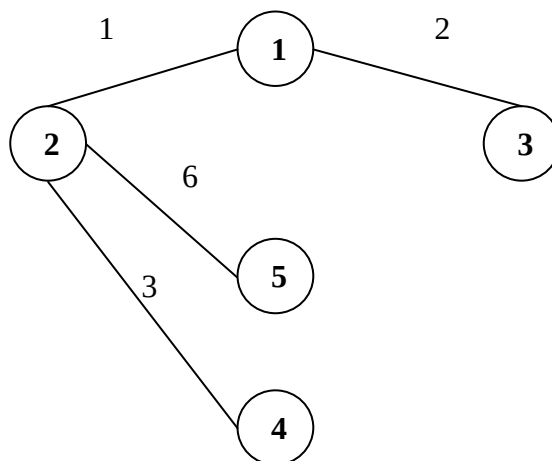
Определим цикломатическое число графа: $\gamma = n - m + 1 = 8 - 5 + 1 = 4$

Выбранные вершины	Невыбранные вершины			
	V1	V2	V3	V4
V5	10	<u>6</u>	7	11
V2	<u>1</u>	*	7	3
V1	*	*	<u>2</u>	3
V3	*	*	*	<u>3</u>

При просмотре строки таблицы находим минимальную величину веса ребра, выделяем её и больше этот столбец, в котором находится эта величина, в рассмотрении не участвует.

Для построения остовного дерева необходимо просмотреть столбцы таблицы снизу вверх и зафиксировать первое появление минимальной величины.

В итоге получим остовное дерево минимального веса:



Лабораторное задание

Для проведения лабораторной работы необходимо выполнить следующие действия.

1. Изучить основные понятия теории графов

Путь к файлу: D:\ИПОВС\АиСД\OSTOV\Теория графов

2. Вызвать систему **Ostov1**, включающую в себя изучение методов Крускала и Прима.

Путь к файлу: D:\ИПОВС\АиСД\OSTOV\Ostov1

Система работает в диалоговом режиме с использованием “меню”. Вся необходимая информация во время работы системы отображается на экране дисплея и не требует специальных пояснений.

Для графов, содержащих 10 и 12 вершин построить остовные деревья двумя методами.

По окончании работы выставляется оценка.

3. Вызвать системы **Ostov2** и **Ostov3** для исследования методов Крускала и Прима.

Путь к файлу: D:\ИПОВС\АиСД\OSTOV\Ostov2 (Ostov3)

Результаты записать в тетрадь.

Требования к отчёту

Отчёт должен содержать:

- 5) конспект лабораторной работы;
- 6) примеры работы методов Крускала и Прима;
- 7) результаты выполнения работы;
- 8) выводы по работе;

Контрольные вопросы

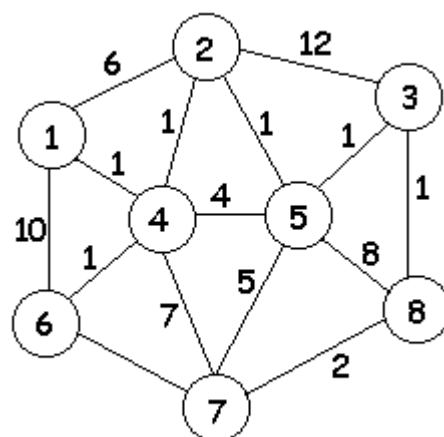
1. Что понимается под остовным деревом?
2. Каковы особенности методов Крускала и Прима?
3. В чём состоит методика анализа сложности алгоритмов построения остовного дерева графа?

Литература

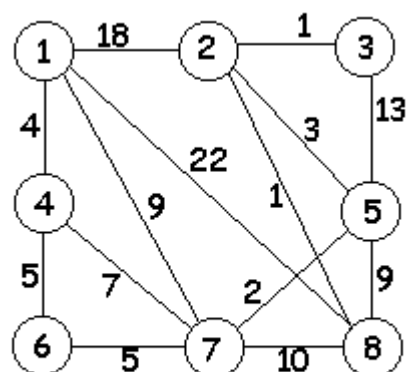
1. Т. Кормен, Ч. Лейзерсон, Р. Ривест. Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
2. Д.Кнут Искусство программирования , том 1. Основные алгоритмы. Уч . пос. М.:Изд. Дом " Вильямс ", 2000.
3. Вирт Н. Алгоритмы и структуры данных.: Пер. С англ. - М.: Мир, 2001.
4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.

Задание : Построить остовное дерево графа методами Крускала и Прима.

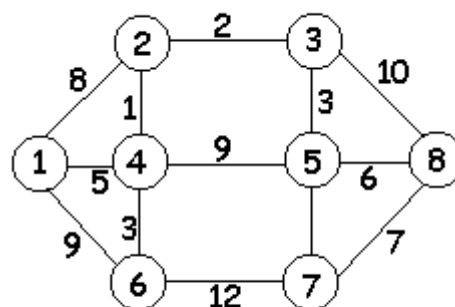
1; 24



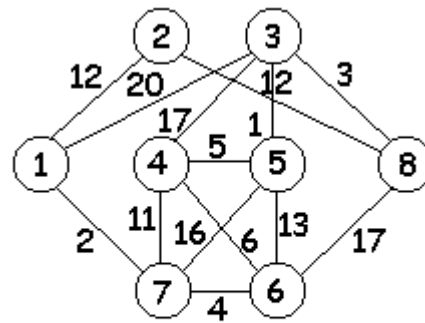
2; 23



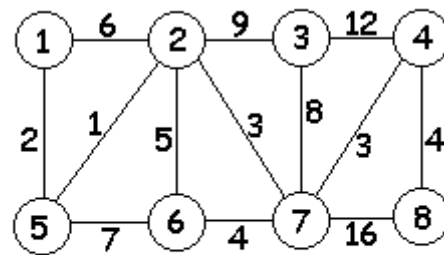
3; 22



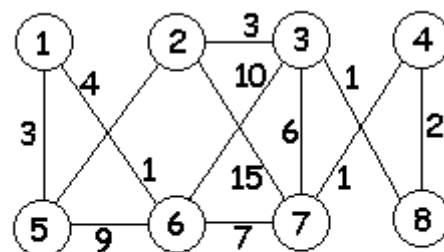
4; 21



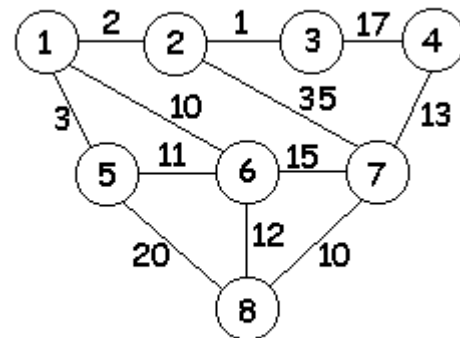
5; 20



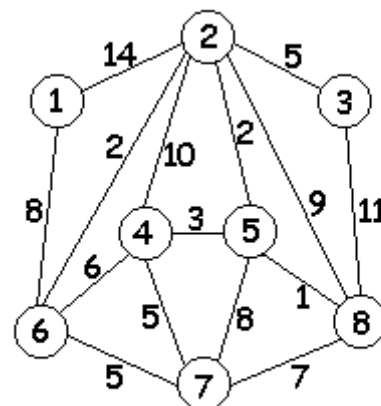
6; 19



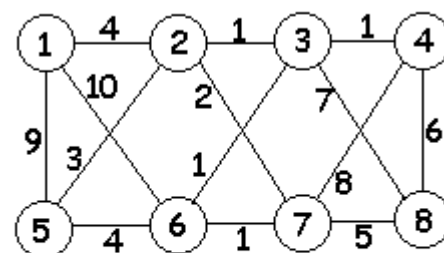
7; 18



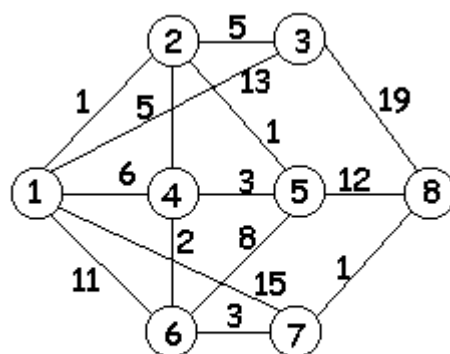
8; 17



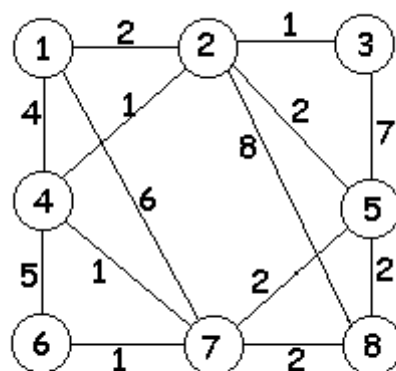
9; 16



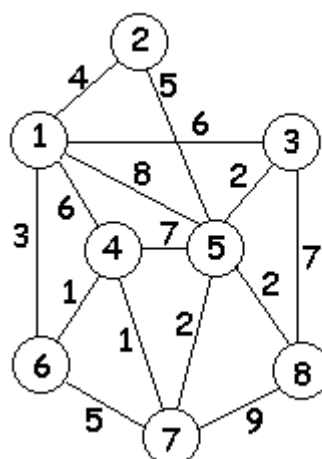
10; 15; 25



11; 14

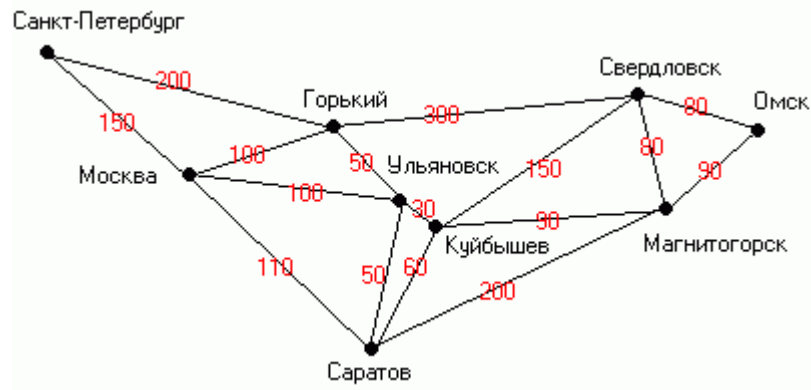


12; 13

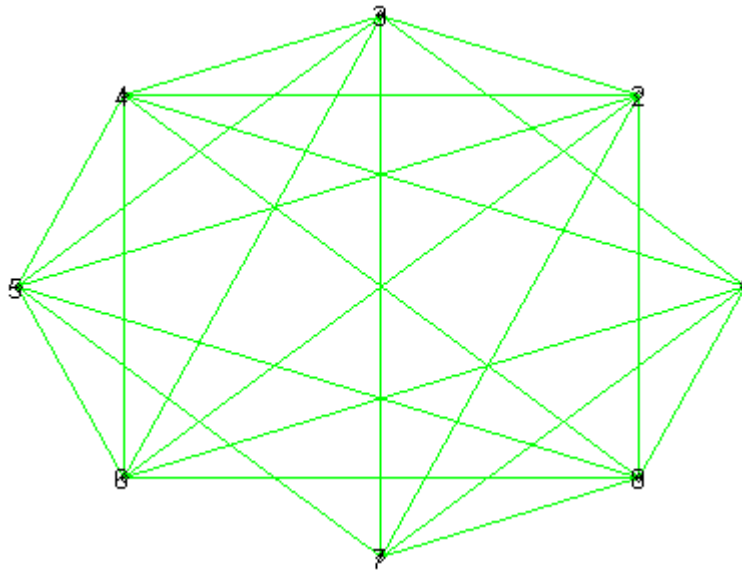


Решить задачи

Задача 1. Дана карта дорог между городами, где указана длина каждой дороги (данные не совпадают с настоящими). Соединить все города автострадой минимальной длины, т.е. построить остовное дерево методами Прима и Крускала.



Задача 2. Задать веса ребрам и построить остовное дерево графа



Задача 3. Между 9 планетами Солнечной системы введено космическое сообщение. Ракеты летают по следующим маршрутам: Земля-Меркурий, Плутон-Венера, Земля-Плутон, Плутон-Меркурий, Меркурий-Венера, Уран-Нептун, Нептун-Сатурн, Сатурн-Юпитер, Юпитер-Марс и Марс-Уран. Можно ли добраться с Земли до Марса?

Задача 4. Может ли в государстве, в котором из каждого города выходит 3 дороги, быть ровно 100 дорог?

Задача 5. Какое максимальное число висячих вершин может иметь дерево, обладающее 9 вершинами? Какое минимальное число висячих вершин оно может иметь? Сделайте рисунки таких деревьев.

Задача 6. В доме отдыха Вишкиль 57 корпусов. Электрик решил соединить телефонными проводами каждый корпус ровно с пятью другими. Сможет ли он это сделать?

Задача 7. У предприятия имеется 4 завода и 9 торговых точек. Каждый завод должен поставлять свою продукцию во все торговые точки. Сколько всего дорог необходимо проложить, чтобы осуществить задуманное? Определите вид полученного графа.

Задача 8. На участке 3 дома и 3 колодца. От каждого дома к каждому колодцу ведет тропинка. Когда владельцы домов поссорились, они задумали проложить дорогу от каждого дома к каждому колодцу так, чтобы не встречаться на пути к колодцам. Может ли осуществиться их намерение?

Задача 9. Муравей забрался в банку из-под сахара, имеющую форму куба. Сможет ли он последовательно обойти все рёбра куба, не проходя дважды по одному ребру?

Задача 10. На занятии 20 школьников решили каждый по 6 задач, причём каждая задача была решена ровно двумя школьниками. Можно ли организовать разбор всех задач таким образом, чтобы каждый школьник рассказал ровно по 3 задачи.

Лабораторная работа № 5.

АЛГОРИТМЫ НАХОЖДЕНИЯ НА ГРАФАХ КРАТЧАЙШИХ ПУТЕЙ.

Цель работы: изучение некоторых алгоритмов нахождения на графах кратчайших путей; исследование эффективности этих алгоритмов.

Продолжительность работы – 2 часа.

Теоретические сведения

Задача отыскания в графе (как ориентированном, так и неориентированном) кратчайшего пути имеет многочисленные практические приложения. С решением подобной задачи приходится встречаться в технике связи (например, при телефонизации населенных пунктов), на транспорте (при выборе оптимальных маршрутов доставки грузов), в микроэлектронике (при проектировании топологии микросхем) и т.д.

Путь между вершинами i и j графа считается кратчайшим, если вершины i и j соединены минимальным числом ребер (случай не взвешенного графа) или если сумма весов ребер, соединяющих вершины i и j , минимальна (для взвешенного графа).

В настоящее время известны десятки алгоритмов решения поставленной задачи.

Важным показателем алгоритма является его эффективность. Применительно к поставленной задаче эффективность алгоритма может зависеть в основном от двух параметров графа: число его вершин и число весов его ребер. В данной лабораторной работе для определения кратчайшего расстояния между вершинами графа исследуются два алгоритма: метод динамического программирования и метод Дейкстры.

Метод динамического программирования.

Метод рассматривает многостадийные процессы принятия решения. При постановке задачи динамического программирования формируется некоторый критерий. Процесс разбивается на стадии (шаги), в которых принимаются решения, приводящие к достижению общей поставленной цели. Таким образом, метод динамического программирования - метод пошаговой оптимизации.

Введем функцию f_i , определяющую минимальную длину пути из начальной вершины i . Обозначим через S_{ij} длину пути между вершинами i и j , а f_j - наименьшую длину пути между вершиной j и начальной вершиной.

Выбирая в качестве i такую вершину, которая минимизирует сумму $(S_{ij} + f_j)$, получаем уравнение

$$f_i = \min \{S_{ij} + f_j\}$$

Трудность решения данного уравнения заключается в том, что неизвестная функция входит в обе части равенства. В такой ситуации приходится прибегать к классическому методу последовательных приближений (итераций), используя рекуррентную формулу:

$$f_i^{(k+1)} = \min \{S_{ij} + f_j^{(k)}\},$$

где $f_j^{(k)}$ - k -е приближение функции.

Возможен другой подход к решению поставленной задачи с помощью метода стратегий. При движении из начальной точки i в конечную k получается приближение $f_i^{(0)} = S_{ik}$, где S_{ik} - длина пути между точками i и k . Следующее приближение – поиск решения в классе двухзвенных ломаных. Дальнейшие приближения ищутся в классе трехзвенных, четырехзвенных и других ломаных.

Пример 1. Определить кратчайший путь из вершины 1 в вершину 10 для графа, представленного на рис. 1.

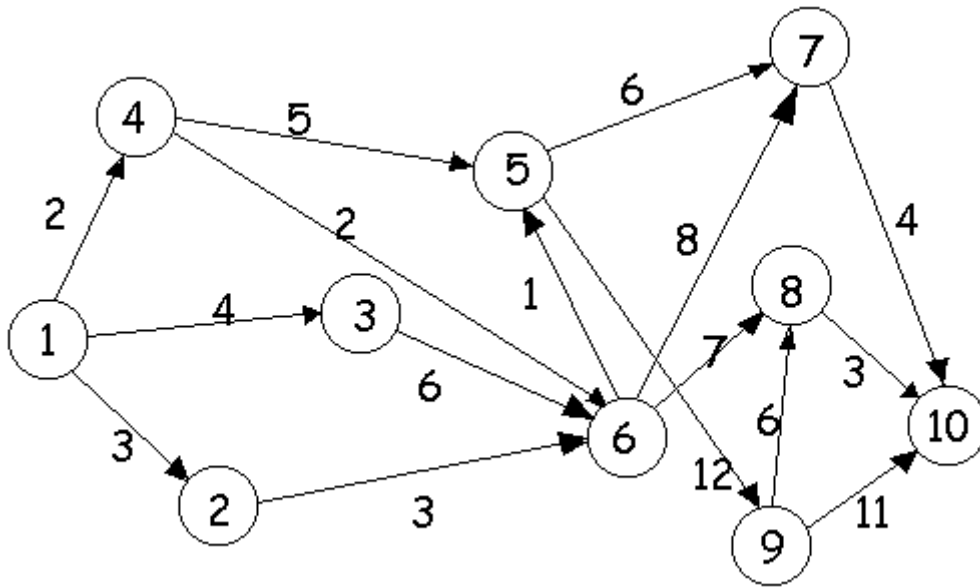


Рис.1. Взвешенный ориентированный граф

Начальные условия: $f_1 = 0$, $S_{11} = 0$.

Находим последовательно значения функции f_i (в условных единицах) для каждой вершины ориентированного графа:

$$f_2 = \min\{S_{21} + f_1\} = \{3 + f_1\} = \{3 + 0\} = 3;$$

$$f_3 = \min\{S_{31} + f_1\} = \{4 + f_1\} = \{4 + 0\} = 4;$$

$$f_4 = \min\{S_{41} + f_1\} = \{2 + f_1\} = \{2 + 0\} = 2;$$

$$f_6 = \min \left\{ \begin{array}{l} S_{64} + f_4 \\ S_{63} + f_3 \\ S_{62} + f_2 \end{array} \right\} = \min \left\{ \begin{array}{l} 2 + 2 \\ 6 + 4 \\ 3 + 3 \end{array} \right\} = 4;$$

$$f_5 = \min \left\{ \begin{array}{l} S_{54} + f_4 \\ S_{56} + f_6 \end{array} \right\} = \min \left\{ \begin{array}{l} 5 + 2 \\ 1 + 4 \end{array} \right\} = 5;$$

$$f_7 = \min \left\{ \begin{array}{l} S_{75} + f_5 \\ S_{76} + f_6 \end{array} \right\} = \min \left\{ \begin{array}{l} 6 + 5 \\ 8 + 4 \end{array} \right\} = 11;$$

$$f_9 = \min \{S_{95} + f_5\} = \min = \{12 + 5\} = 17;$$

$$f_8 = \min \left\{ \begin{array}{l} S_{86} + f_6 \\ S_{89} + f_9 \end{array} \right\} = \min \left\{ \begin{array}{l} 7 + 4 \\ 6 + 17 \end{array} \right\} = 11;$$

$$f_{10} = \min \left\{ \begin{array}{l} S_{10,7} + f_4 \\ S_{10,8} + f_8 \\ S_{10,9} + f_{10} \end{array} \right\} = \min \left\{ \begin{array}{l} 4 + 11 \\ 3 + 11 \\ 11 + 17 \end{array} \right\} = 14.$$

Длина кратчайшего пути составляет 14 условных единиц. Для выбора оптимальной траектории движения следует осуществить просмотр функций f_i в обратном порядке, то есть с f_{10} . Пусть $f_i = f_{10}$. В данном случае

$$f_{10} = \min \left\{ \begin{array}{l} 4 + f_1 \\ 3 + f_8 \\ 11 + f_9 \end{array} \right\} = \min \left\{ \begin{array}{l} 4 + 11 \\ 3 + 11 \\ 11 + 17 \end{array} \right\} = 14.$$

Получаем, что $(3 + f_8) = 14$, то есть $f_j = f_8$. Значит, из вершины 10 следует перейти к вершине 8. Имеем $f_i = f_8$. Рассмотрим функцию

$$f_8 = \min \left\{ \begin{array}{l} 7 + f_6 \\ 6 + f_9 \end{array} \right\} = \min \left\{ \begin{array}{l} 7 + 4 \\ 6 + 17 \end{array} \right\} = 11,$$

то есть $f_j = f_6$ и т.д.

Таким образом, получаем кратчайший путь от вершины 1 к вершине 10:
(1, 4, 6, 8, 10)

Рассмотренный метод определения кратчайшего пути может быть распространен и на неориентированные графы.

Пример 2. Для графа (рис. 1) найти кратчайший путь от вершины 1 до вершины 10, рассматривая граф как неориентированный. Матрица смежности весов графа в этом случае имеет вид:

	1	2	3	4	5	6	7	8	9	10
1	-	3	4	2	-	-	-	-	-	-
2	3	-	-	-	-	3	-	-	-	-
3	4	-	-	-	-	6	-	-	-	-
4	2	-	-	-	5	2	-	-	-	-
5	-	-	-	5	-	1	6	-	12	-
6	-	3	6	2	1	-	8	7	-	-
7	-	-	-	-	6	8	-	-	-	4
8	-	-	-	-	-	7	-	-	6	3
9	-	-	-	-	12	-	-	6	-	11
10	-	-	-	-	-	-	4	3	11	-

Записываем выражения для функции f_i

$$f_1 = 0; \quad f_2 = \min \begin{cases} f_1 + 3 \\ f_6 + 3 \end{cases} ; \quad f_3 = \min \begin{cases} f_1 + 4 \\ f_6 + 6 \end{cases} ; \quad f_4 = \min \begin{cases} f_1 + 2 \\ f_5 + 5 \\ f_6 + 2 \end{cases}$$

$$f_5 = \min \begin{cases} f_4 + 5 \\ f_6 + 1 \\ f_9 + 12 \\ f_7 + 6 \end{cases} ; \quad f_6 = \min \begin{cases} f_2 + 3 \\ f_3 + 6 \\ f_4 + 2 \\ f_5 + 1 \\ f_7 + 8 \\ f_8 + 7 \end{cases} ; \quad f_7 = \min \begin{cases} f_5 + 6 \\ f_6 + 8 \end{cases}$$

$$f_8 = \min \begin{cases} f_6 + 7 \\ f_9 + 6 \end{cases} ; \quad f_9 = \min \begin{cases} f_5 + 12 \\ f_8 + 6 \end{cases} ; \quad f_{10} = \min \begin{cases} f_7 + 4 \\ f_8 + 3 \\ f_9 + 11 \end{cases} .$$

Указанные целевые функции, представляют собой систему линейных алгебраических уравнений (в примере имеется 10 уравнений и 10 неизвестных). Рассмотрим один из вариантов ее решения, учитывая, что $f_1 = 0$. Тогда имеем:

$$f_2 = 3; \quad f_3 = 4; \quad f_4 = \min \begin{cases} 2 \\ f_5 + 5 \\ f_6 + 2 \end{cases} = 2;$$

$$f_5 = \min \begin{cases} 7 \\ f_6 + 1 \\ f_9 + 12 \\ f_7 + 6 \end{cases} = \min \begin{cases} 7 \\ f_6 + 1 \end{cases} ; \quad f_6 = \min \begin{cases} 6 \\ 10 \\ 4 \\ f_5 + 1 \\ f_7 + 1 \\ f_8 + 7 \end{cases} = \min \begin{cases} 4 \\ f_5 + 1 \end{cases} = 4$$

Подставив выражение для f_6 в f_5 , получим

$$f_5 = \min \begin{cases} 7 \\ 4 + 1 \\ f_5 + 1 + 1 \end{cases} = 5.$$

Тогда

$$f_7 = 11; \quad f_8 = \min \begin{cases} 11 \\ f_9 + 6 \end{cases}; \quad f_9 = \min \begin{cases} 17 \\ f_8 + 6 \end{cases}; \quad f_{10} = \min \begin{cases} 15 \\ f_8 + 3 \\ f_9 + 11 \end{cases}.$$

Подставляя f_9 в f_8 , получаем

$$f_8 = \min \begin{cases} 11 \\ 17 + 6 \\ f_8 + 6 + 6 \end{cases} = 11.$$

Окончательно имеем: $f_9 = 17$; $f_{10} = 14$.

Метод Дейкстры

Алгоритм Дейкстры предназначен для нахождения кратчайшего пути между вершинами в неориентированном графе.

Идея алгоритма следующая, сначала выберем путь до начальной вершины равным нулю, и заносим эту вершину во множество уже выбранных, расстояние от которых до оставшихся невыбранных вершин определено. На каждом следующем этапе находим следующую невыбранную вершину, расстояние до которой наименьшее и соединённую ребром с какой-нибудь вершиной из множества выбранных (это расстояние будет равно расстоянию до уже выбранной вершины плюс длина ребра).

Найти кратчайший путь на графе от вершины L до вершины D (рис.2).

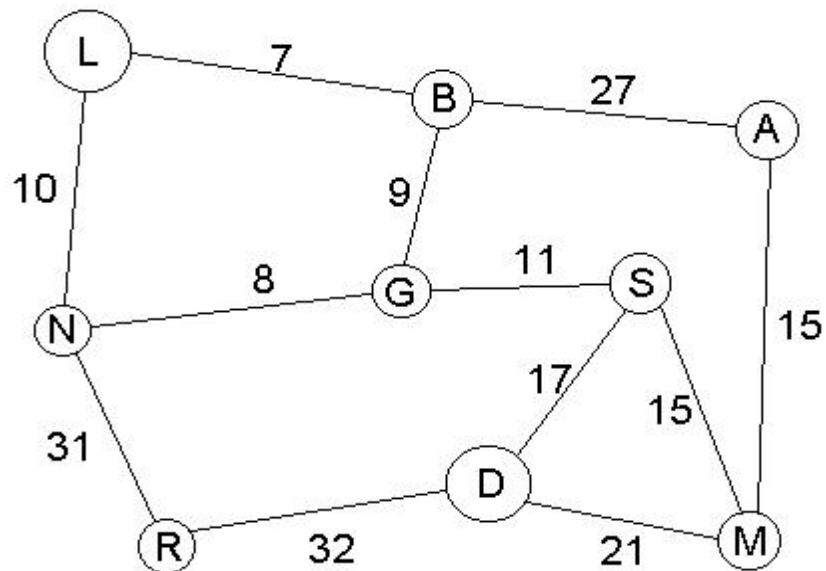


Рис.2. Взвешенный неориентированный граф

Запишем алгоритм в виде последовательности шагов.

Шаг 1. Определяются расстояния от начальной вершины L до всех остальных .

Длина пути до выбранной вершины	Выбранная вершина	Невыбранные вершины							
		B	G	N	R	D	S	M	A
0	L	7	∞	10	∞	∞	∞	∞	∞
7	B								

Шаг 2. Выбираем наименьшее расстояние от L до B, найденная вершина B принимается за вновь выбранную. Найденное наименьшее расстояние добавляется к длинам ребер от новой вершины B до всех остальных. Выбирается минимальное расстояние от B до N. Новая вершина N принимается за выбранную.

Длина пути до выбранной вершины	Выбранная вершина	Невыбранные вершины							
		B	G	N	R	D	S	M	A
0	L	7	∞	10	∞	∞	∞	∞	∞
7	B	∞	16	10	∞	∞	∞	∞	34
10	N								

Для наглядности в дальнейшем вместо знака ∞ будем ставить знак “-”.

Шаг 3. Определяются расстояния от вершины N до всех оставшихся (за исключением L и B) .

Длина пути до выбранной вершины	Выбранная вершина	Невыбранные вершины							
		B	G	N	R	D	S	M	A
0	L	7	∞	10	∞	∞	∞	∞	∞
7	B	∞	16	10	∞	∞	∞	∞	34
10	N	∞	16	∞	41	∞	∞	∞	34
16	G								

Расстояние от вершины L через вершину N до вершины G равно 18. Это расстояние больше, чем расстояние $LB+BG=16$, поэтому оно не учитывается в дальнейшем.

Продолжая аналогичные построения, построим таблицу. Таким образом, найдена длина кратчайшего пути между вершинами L и D (44 условных единицы).

Траектория пути определяется следующим образом.

Осуществляем обратный просмотр от конечной вершины к начальной.

Просматриваем столбец, соответствующий вершине, снизу вверх и фиксируем первое появление минимальной величины. В столбце, соответствующем вершине D, впервые минимальная длина 44 появилась снизу в четвертой строке. В этой строке указана вершина S, к которой следует перейти, то есть следующим нужно рассматривать столбец, соответствующий вершине S.

Длина пути до выбранной вершины	Выбранная вершина	Невыбранные вершины							
		B	G	N	R	D	S	M	A
0	L	7	-	10	-	-	-	-	-
7	B	-	16	10	-	-	-	-	34
10	N	-	16	-	41	-	-	-	34
16	G	-	-	-	41	-	$16+11=27$	-	34
27	S	-	-	-	41	$27+17=44$	-	$27+15=42$	34
34	A	-	-	-	41	44	-	42 [34+15]	-
41	R	-	-	-	-	44 [41+32]	-	42	-
42	M	-	-	-	-	44 [42+21]	-	-	-

Min

В этом столбце минимальная длина, равная 27, указывает следующую вершину G,

к которой следует перейти, и т.д. Таким образом, получаем траекторию пути:
(L, B, G, S, D).

Алгоритм Флойда

Пусть в матрице $A[i,j]$ записаны длины ребер графа (элемент $A[i,j]$ равен весу ребра, соединяющего вершины с номерами i и j , если же такого ребра нет, то в соответствующем элементе записано некоторое очень большое число). Построим новые матрицы $C_k[i,j]$ ($k=0,\dots,N$). Элемент матрицы $C_k[i,j]$ будет равен минимально возможной длине такого пути из i в j , что в качестве промежуточных вершин в этом пути используются вершины с номерами от 1 до k . То есть рассматриваются пути, которые могут проходить через вершины с номерами от 1 до k , но заведомо не проходят через вершины с номерами от $k+1$ до N . В матрицу записывается длина кратчайшего из таких путей. Если таких путей не существует, записывается то же большое число, которым обозначается отсутствие ребра.

Если вычислена матрица $C_{k-1}[i,j]$, то элементы матрицы $C_k[i,j]$ можно вычислить по следующей формуле: $C_k[i,j] := \min(C_{k-1}[i,j], C_{k-1}[i,k] + C_{k-1}[k,j])$. В самом деле, рассмотрим кратчайший путь из вершины i в вершину j , который в качестве промежуточных вершин использует только вершины с номерами от 1 до k . Тогда возможно два случая:

Этот путь не проходит через вершину с номером k . Тогда его промежуточные вершины — это вершины с номерами от 1 до $k-1$. Но тогда длина этого пути уже вычислена в элементе $C_{k-1}[i,j]$.

Этот путь проходит через вершину с номером k . Но тогда его можно разбить на две части: сначала из вершины i доходим оптимальным образом до вершины k , используя в качестве промежуточных вершины с номерами от 1 до $k-1$ (длина такого оптимального пути вычислена в $C_{k-1}[i,k]$), а потом от вершины k идем в вершину j опять же оптимальным способом, и опять же используя в качестве промежуточных вершин только вершины с номерами от 1 до k ($C_{k-1}[k,j]$).

Выбирая из этих двух вариантов минимальный, получаем $C_k[i,j]$.

Последовательно вычисляя матрицы C_0, C_1, C_2 и т.д. получим искомую матрицу C_N кратчайших расстояний между всеми парами вершин в графе.
Алгоритм Флойда можно свести к последовательности шагов

Присвоить $c_{ij}=0$ для всех $i=1,2,\dots,n$ и $c_{ij}=\infty$, если в графе отсутствует дуга (x_i, x_j) .

Присвоение начальных значений.

Шаг 1. присвоить $k=0$;

Шаг 2. $k=k+1$.

Шаг 3. Для всех $i < k$, таких, что $c_{ik} < \infty$, и для всех $j > k$, таких, что $c_{ik} < \infty$,
 $c_{ij} = [\min(c_{ij}, (c_{ik} + c_{kj}))]. \quad (9)$

Проверка на окончание.

Шаг 4. Если $k=n$, то матрица $[c_{ij}]$ дает длины всех кратчайших путей.

Остановка. Иначе перейти к шагу 2.

Алгоритм Йена

Пусть граф задан матрицей $A[i,j]$. Вершина s выбрана как начальная. Найдём длины k наименьших путей до каждой вершины от вершины s (рёбра в одном пути могут повторяться неоднократно), при условии, что эти пути существуют. Результат будет храниться в матрице B , размером $N \times k$, где N — количество вершин. Элемент массива $B[i,j]$ равен j -му по длине пути до вершины i .

Для каждой вершины в массиве C будем хранить количество уже найденных до неё наименьших путей. Изначально все элементы массива C равны нулю. Все элементы матрицы B делаем равными, какой-нибудь большой константе, заведомо большей всех возможных путей. Во время исполнения алгоритма в матрице B будем хранить лучшие k путей до каждой вершины, найденные во время исполнения, при этом первые $C[i]$ путей для вершины i найдены уже окончательно (элементы матрицы B , $B[i,1]$, $B[i,2]$, ..., $B[i,k]$ для всех i , упорядочены в возрастающем порядке).

Следовательно, можно действовать следующим образом: пусть уже найдены какие-то кратчайшие пути, тогда чтобы найти следующий по длине, попробуем удлинить каждый из уже полученных на одно ребро. Найдём наикратчайший из них, при чём оканчивающийся на вершину, до которой найдено менее k путей. Его можно вносить в таблицу результата.

Алгоритм Йена можно свести к последовательности шагов

Присвоение начальных значений.

Шаг 1. Найти P^1 . Присвоить $k=2$. Если существует только один кратчайший путь P^1 , включить его в список L_0 и перейти к шагу 2.

Если таких путей несколько, но меньше, чем K , включить один из них в список L_c , а остальные в список L_1 .

Перейти к шагу 2.

Если существует K или более кратчайших путей P^1 , то задача решена. Остановка.

Нахождение отклонений.

Шаг 2. Найти все отклонения $P_i^k(k-1)$ -го кратчайшего пути P^{k-1} для всех $i=1, 2, \dots, q_{k-1}$, выполняя для каждого i шаги с 3-го по 6-й.

Шаг 3. Проверить, совпадает ли подпуть, образованный первыми i вершинами любого из P^j путей ($j=1, 2, \dots, k-1$). Если да, то присвоить $c(x_i^{k-1}, x_{i+1}^j) = \infty$; иначе ничего не изменять. (При выполнении алгоритма вершина x_1 обозначается s).

Шаг 4. Найти кратчайшие пути S_i^k от x_i^{k-1} к t , исключая из рассмотрения вершины $s, x_1^{k-1}, x_3^{k-1}, \dots, x_i^{k-1}$. Если существует несколько кратчайших путей, то взять в качестве S_i^k один из них.

Шаг 5. Построить P_i^k , соединяя $R_i^k (=s, x_1^{-1}, x_3^{-1}, \dots, x_i^{-1})$ с S и поместить P в список L_1 .

Шаг 6. Заменить элементы матрицы весов, измененные на шаге их первоначальными значениями и возвратиться к шагу 3.

Выбор кратчайших отклонений.

Шаг 7. Найти кратчайший путь в списке L_1 . Обозначить этот путь P^k и переместить его из L_1 в L_0 .

Если $k=K$, то остановка. Список L_0 -список K -кратчайших путей.

Если $k < K$, то присвоить $k=k+1$, перейти к шагу 2.

Если в L_1 имеется более чем один кратчайший путь (h -путей), то поместить в L_0 любой из них и продолжать как выше до тех пор, пока увеличенное на h число путей, уже находящихся в L_1 , не совпадет с K или не превысит его. Тогда остановка.

Алгоритм Беллмана- Форда

Пусть в матрице $A[i,j]$ записаны длины ребер графа. Найдём наикратчайшие расстояния от заданной вершины s до всех остальных вершин графа. Алгоритм Беллмана-Форда решает эту задачу при наличии рёбер отрицательного веса.

Обозначим через $\text{МинСт}(s,v,k)$ наименьшую стоимость проезда из s в v менее чем с k пересадками.

$$\text{МинСт}(s,v,k+1) = \min(\text{МинСт}(s,v,k), \text{МинСт}(s,i,k) + A[i][v] \text{ (} i=1..n \text{)})$$

Искомым ответом является $\text{МинСт}(s,i,n)$ для всех $i=1..n$.

Чтобы найти не только длины наименьших путей до всех вершин, но и сами эти пути используем следующий приём. Параллельно с вычислением массива x будем вычислять матрицу $D[i,j]$. Если между вершинами s и j существует путь, то в элементе массива $D[j,0]$ будет храниться количество вершин в этом пути, а цепочка вершин, составленная из элементов с $D[j,1]$ по $D[j,D[j,0]]$, будет этим самым путём. Путь до вершины s содержит единственную вершину ($D[s,0]=1, D[s,1]=s$). Для вычисления матрицы D потребуется немного дополнить текст процедуры:

```

k:= 1;
for i := 1 to n do begin x[i] := a[s][i]; end;
{инвариант: x[i] := МинСт(s,i,k)}    Обозначим через МинСт(s,i,k) наименьшую
стоимость проезда из s в i менее чем с k пересадками. Тогда выполняется такое
соотношение:
for i := 1 to n do begin D[i,0]:=2; D[i,1]:=s; D[i,2]:=i; End;
D[s,0]:=1;

while k <> n do begin
  for i := 1 to n do begin
    for j := 1 to n do begin
      if x[i] > x[j]+a[j][i] then begin
        x[i] := x[j]+a[j][i];
        D[i]:=D[j];
        D[i,0]:=D[j,0]+1;
        D[i,D[i,0]]:=i;
      end;
    end
    {x[i] = МинСт(s,i,k+1)}
  end;
  k := k + 1;
end;

```

Литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест
Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
2. Д.Кнут Искусство программирования , том 1. Основные алгоритмы. Уч . пос. М.:Изд. Дом " Вильямс ", 2000.
3. Вирт Н. Алгоритмы и структуры данных.: Пер. С англ. - М.: Мир, 2001.
4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.
5. А. Ахо, Дж.Хопкрофт, Дж.Ульман, Структуры данных и алгоритмы М: СПб: Киев: Вильямс, 2001г.

Лабораторное задание.

Для выполнения лабораторной работы необходимо:

1) Найти кратчайший путь методом динамического программирования для трех графов. Зафиксировать параметры каждого графа (число вершин и ребер), значение пути и время решения задачи (ориентированные графы);

Путь к файлу: D:\ИПОВС\АиСД\KRATPUT\Kratput_new

2) Найти кратчайший путь методом Дейкстры для трех графов. Зафиксировать параметры каждого графа (число вершин и ребер), значение пути и время решения задачи.

Путь к файлу: D:\ИПОВС\АиСД\KRATPUT\Kratput1

Путь к файлу: D:\ИПОВС\АиСД\KRATPUT\Kratput2

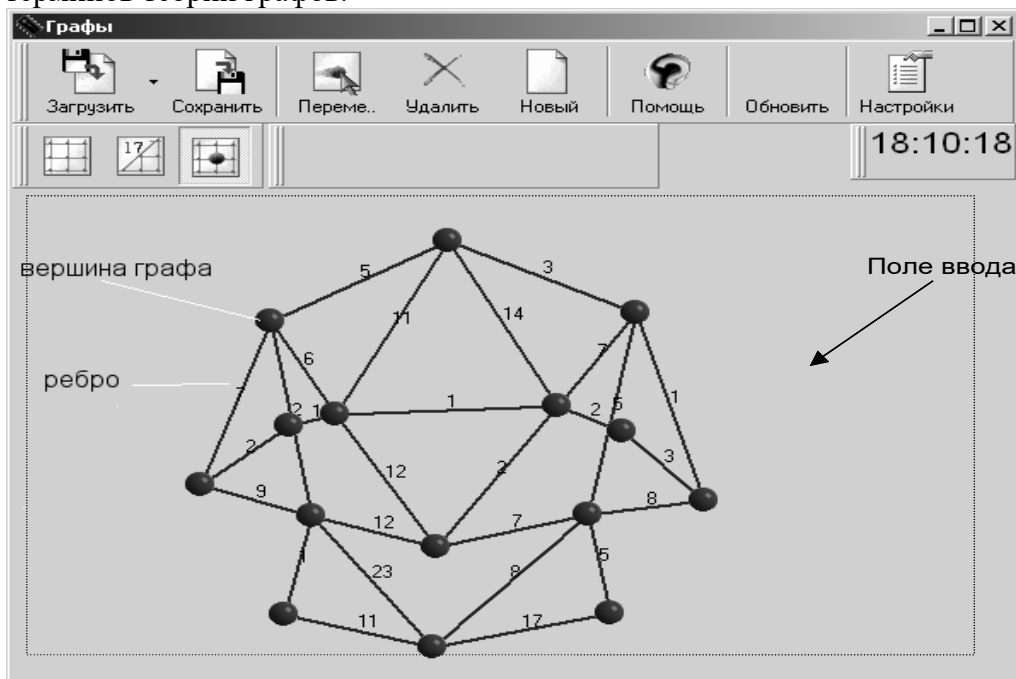
Путь к файлу: D:\ИПОВС\АиСД\KRATPUT\Kratput3

Система работает в диалоговом режиме с использованием «меню». Вся необходимая поясняющая информация отображается во время работы системы на экране монитора.

3) Изучить алгоритмы работы с графовыми структурами

Путь к файлу: D:\ИПОВС\АиСД\KRATPUT\Grafmod

Данная программа предназначена для демонстрации работы алгоритмов и облегчения понимания терминов теории графов.



Программа позволяет вводить, редактировать, сохранять графы в файл, загружать из файла. Также реализованы алгоритмы построения Эйлера цикла, нахождения кратчайшего пути, нахождения пути, содержащего наименьшее количество вершин, решение задачи о раскраске вершин, задачи Коммивояжера.

Для ввода вершины щелкните левой кнопкой мыши в свободное от кнопок поле. Для ввода ребра щелкните правой кнопкой мыши на одной вершине, затем на другой. Для ввода длины ребра щелкните левой кнопкой мыши на введенном ребре. При перемещении вершины нажмите кнопку «Переместить», щелкните левой кнопкой мыши на перемещаемую вершину, затем на свободное место, куда вы хотите переместить вершину, и нажмите кнопку «Переместить». Для удаления вершины нажмите на кнопку «Удалить», затем щелкните левой кнопкой мыши на вершинах, которые хотите удалить, затем еще раз на кнопку «Удалить».

При включенной кнопке «Выравнивать по сетке» новые вершины будут автоматически выравниваться по координатной сетке.

Третья панель служит для запуска алгоритмов программы. Первая кнопка запускает алгоритм решения задачи Коммивояжера. При нажатии на вторую кнопку программа раскрасит граф минимальным количеством цветов. Если выбрать две различные вершины (щелчком левой кнопки мыши) и нажать на третью кнопку, то программа найдет путь с наименьшим количеством вершин, а если четвертую, то найдет кратчайший путь между вершинами. При нажатии на пятую кнопку построится Эйлеров цикл.

4) Ознакомиться с алгоритмами Флойда, Йена, Беллмана- Форда

Требования к отчету

Отчет должен содержать:

- 1) Конспект лабораторной работы;
- 2) Схемы алгоритмов определения в графе кратчайшего расстояния между заданными вершинами для метода динамического программирования и метода Дейкстры;
- 3) Результаты выполнения работы;
- 4) Выводы по работе.

Контрольные вопросы

- 1) Какова теоретическая сложность алгоритмов, рассмотренных в данной работе?
- 2) В решении каких прикладных задач используется алгоритмы определение в графе кратчайших расстояний между заданными вершинами?
- 3) Может ли быть применен рассмотренный в работе алгоритм Дейкстры к определению кратчайшего расстояния в ориентированном графе?
- 4) Как работает алгоритм Дейкстры?
- 5) Как работает алгоритм динамического программирования применительно к задаче определения в графе кратчайших расстояний между вершинами?

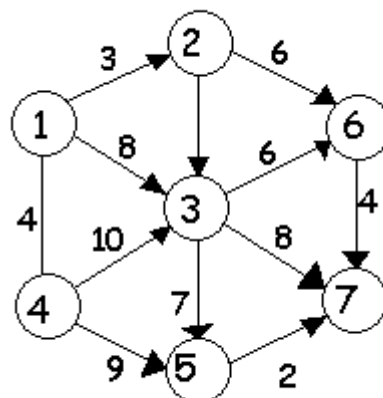
ВАРИАНТЫ ЗАДАНИЙ

Задание 1: Найти кратчайший путь на графе между тремя парами вершин методом динамического программирования

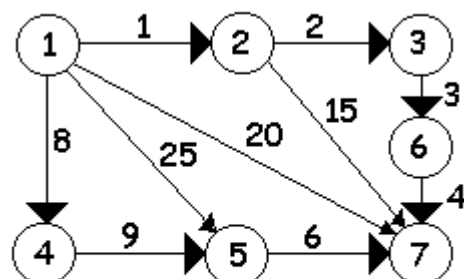
Варианты

Граф

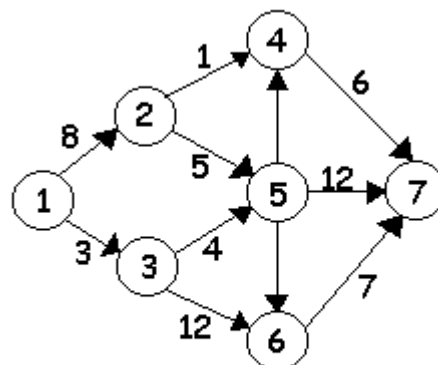
1, 13



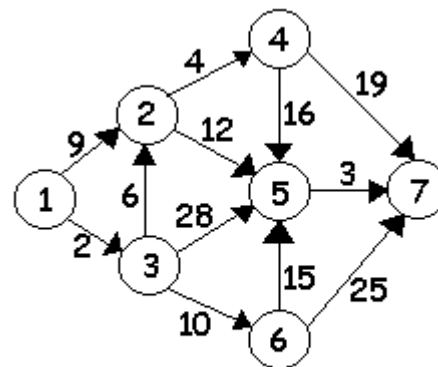
2, 14



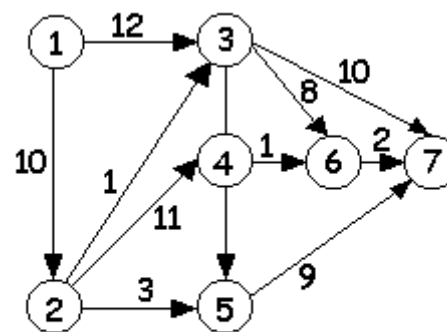
3, 15



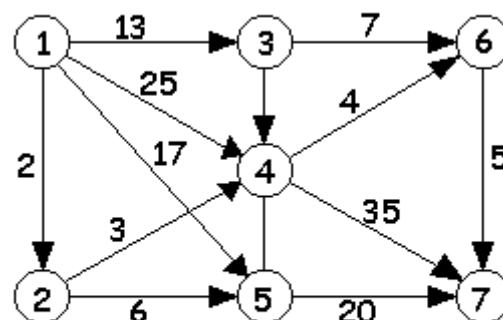
4, 16



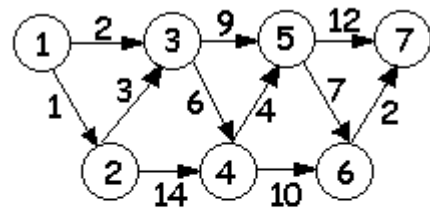
5, 17, 25



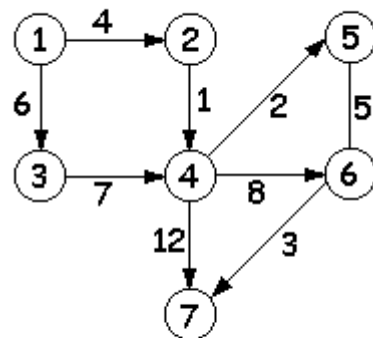
6, 18



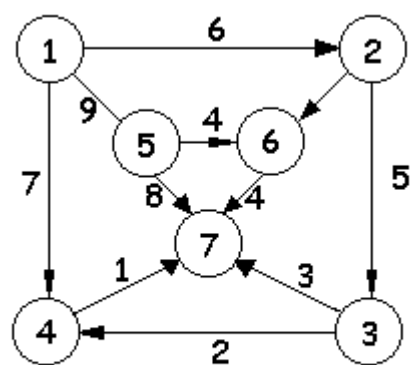
7, 19



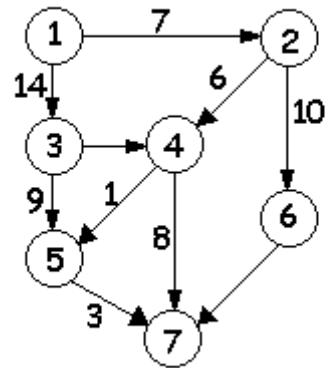
8, 20



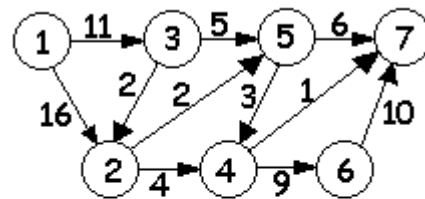
9; 21



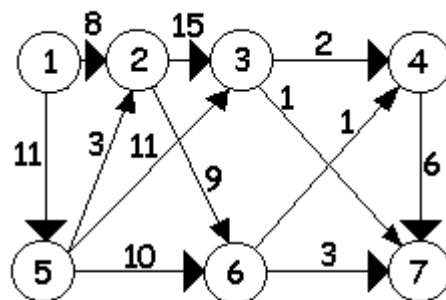
10; 22



11; 23

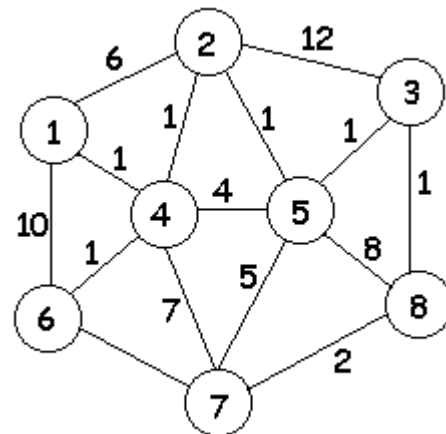


12; 24

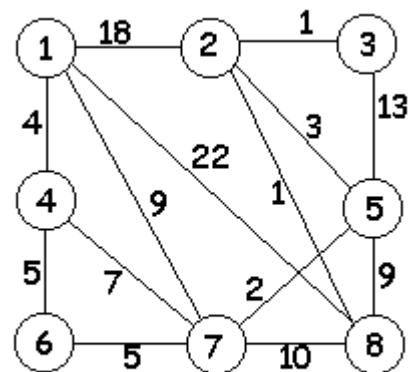


Задание 2: Найти кратчайший путь между тремя парами вершин методом Дейкстры

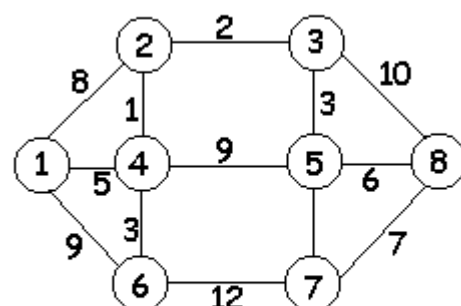
1; 24



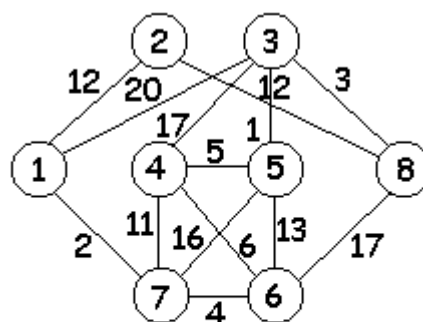
2; 23



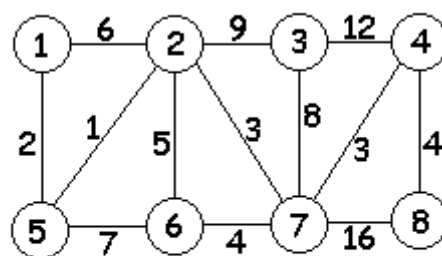
3; 22



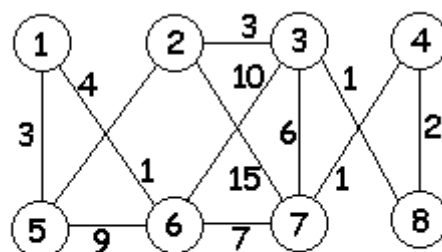
4; 21



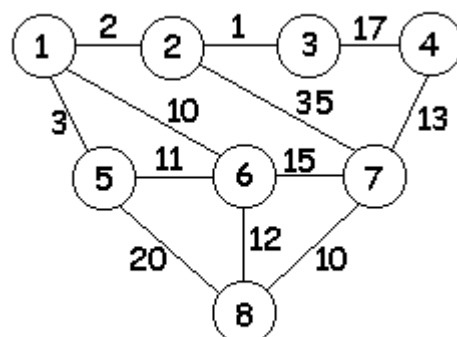
5; 20



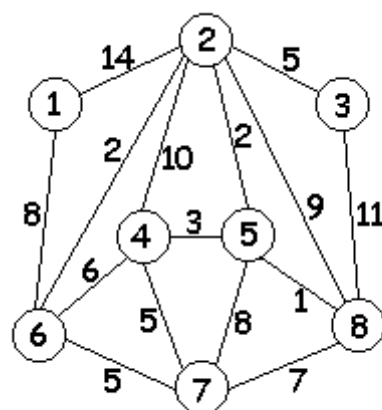
6; 19



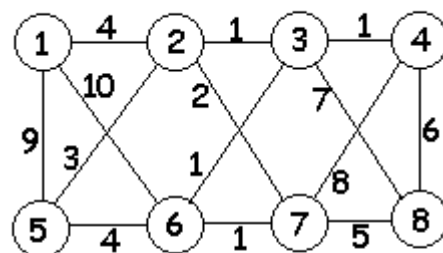
7; 18



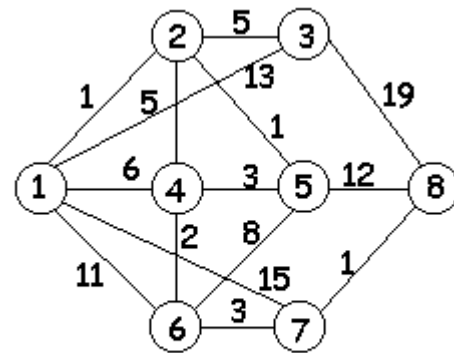
8; 17



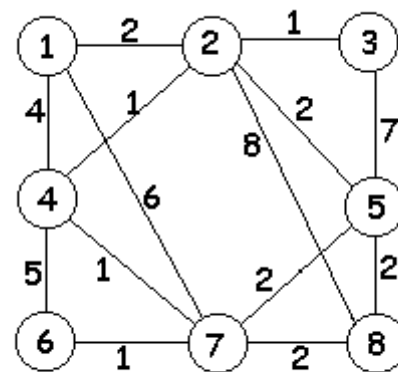
9; 16



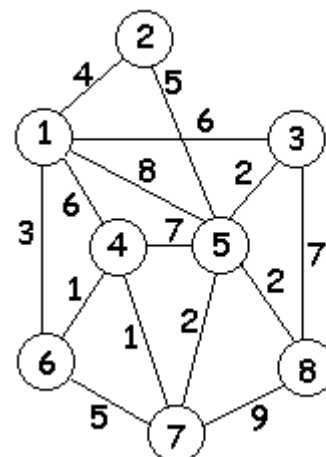
10; 15; 25



11; 14



12; 13



Решить задачи

Задание 1. Дана карта автомобильных дорог. Найти кратчайший путь, учитывая длины дорог, из Ярославля до Новосибирска. Использовать только те города, которые отмечены флажком. Совпадет ли кратчайший маршрут, если не брать во внимание длины дорог?



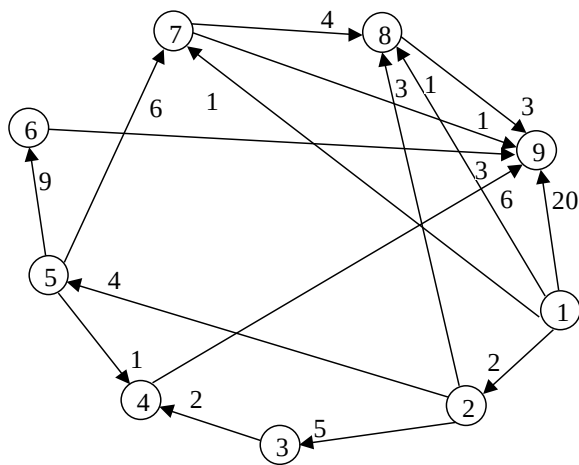
Задание 2. Дана карта Европы. Турагенству необходимо найти кратчайший туристический маршрут между произвольно выбранными странами Европы.



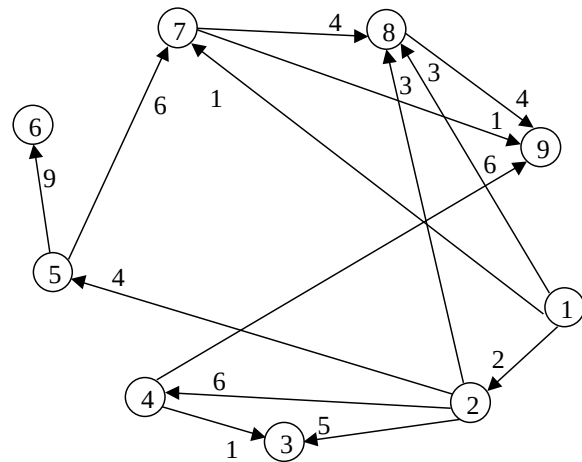
Задание 4. На участке 3 дома и 3 колодца. От каждого дома к каждому колодцу ведет тропинка. Когда владельцы домов поссорились, они задумали проложить дорогу от каждого дома к каждому колодцу так, чтобы не встречаться на пути к колодцам. Может ли осуществиться их намерение?

Составить программу для нахождения кратчайшего пути

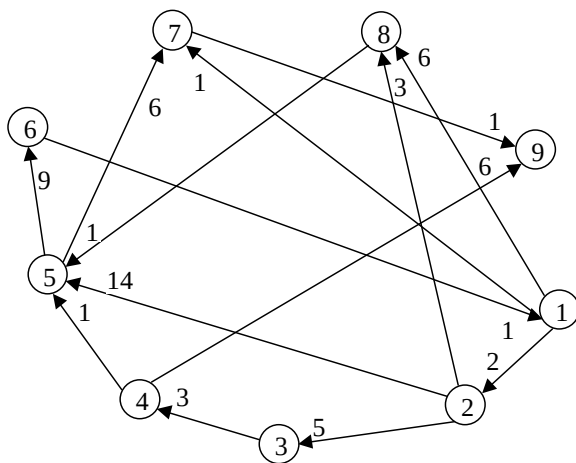
Вариант 1. Путь из 1 в 9



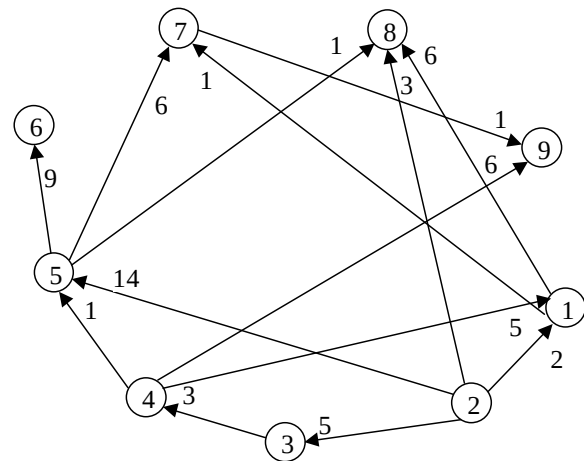
Вариант 3. Путь из 1 в 9



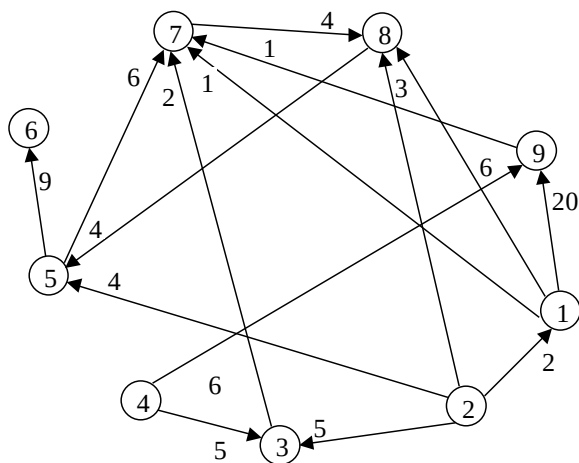
Вариант 2. Путь из 2 в 5



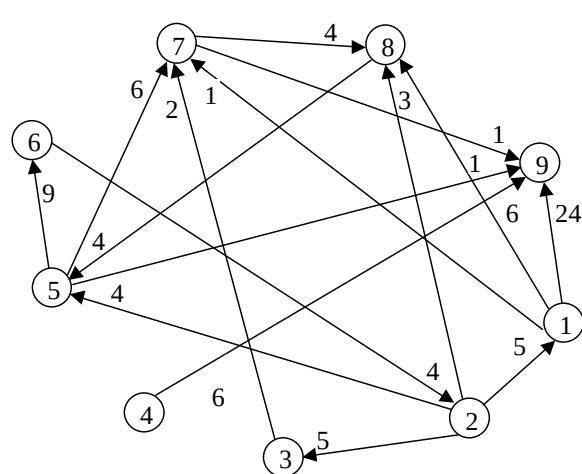
Вариант 4. Путь из 2 в 5



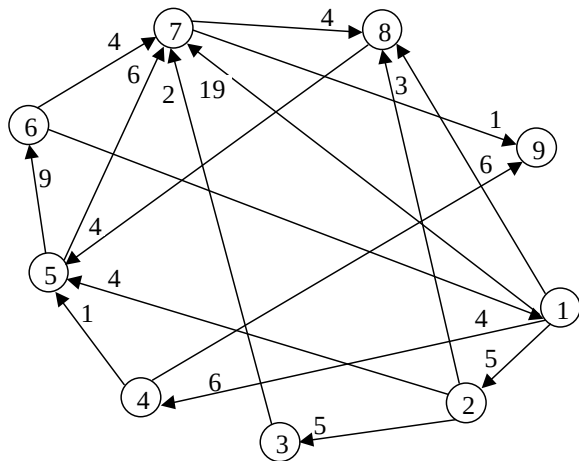
Вариант 5. Путь из 2 в 7



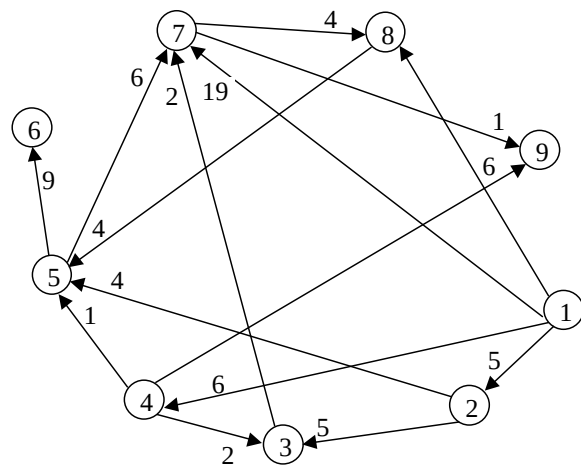
Вариант 8. Путь из 2 в 9



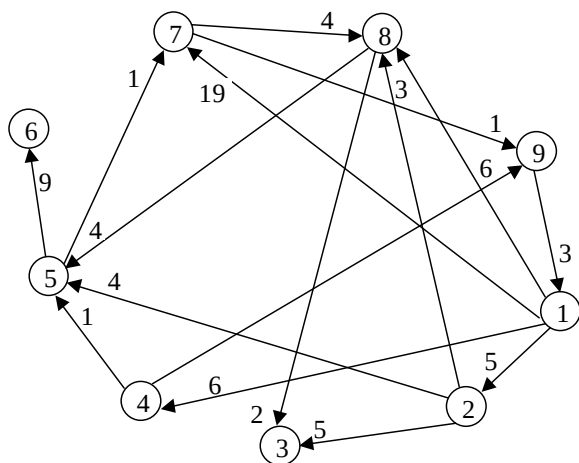
Вариант 6. Путь из 1 в 7



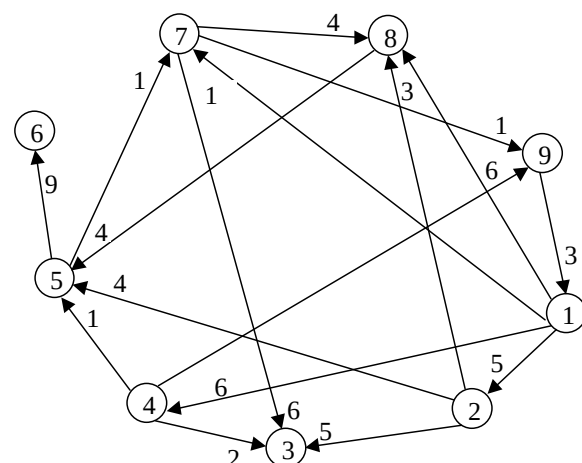
Вариант 9. Путь из 1 в 5



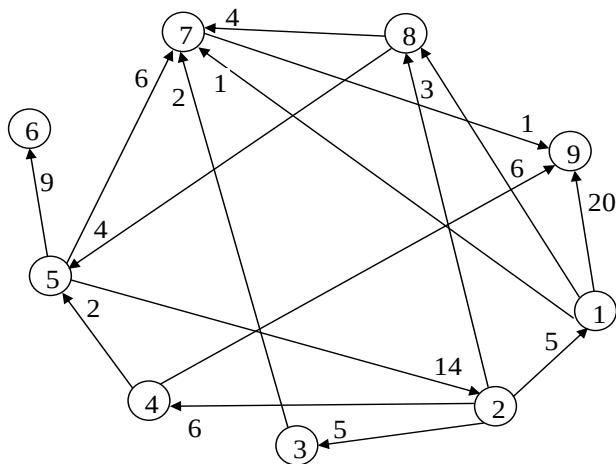
Вариант 7. Путь из 4 в 1



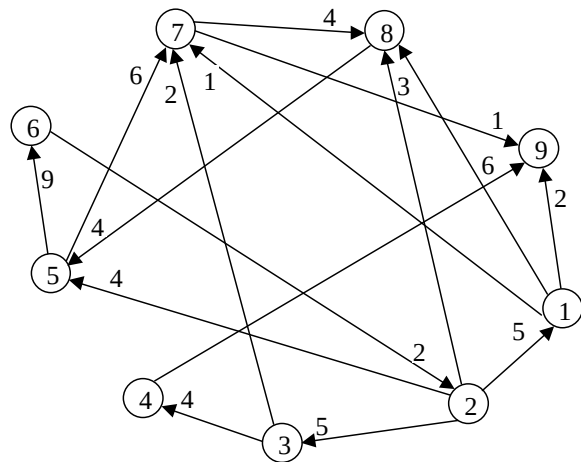
Вариант 10. Путь из 1 в 3



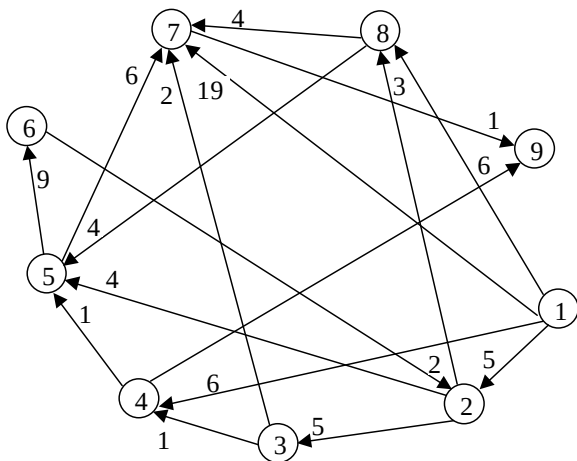
Вариант 11. Путь из 2 в 7



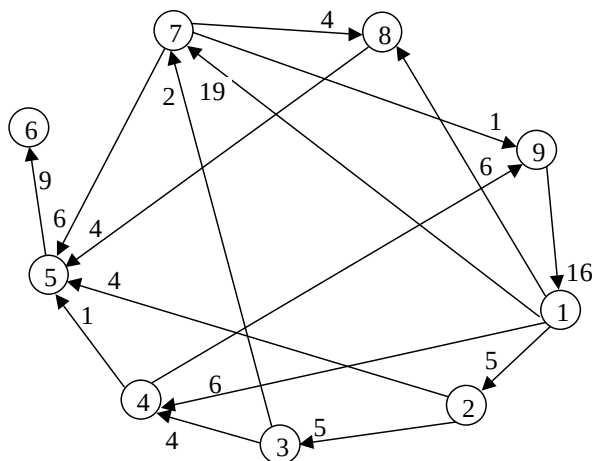
Вариант 14. Путь из 2 в 9



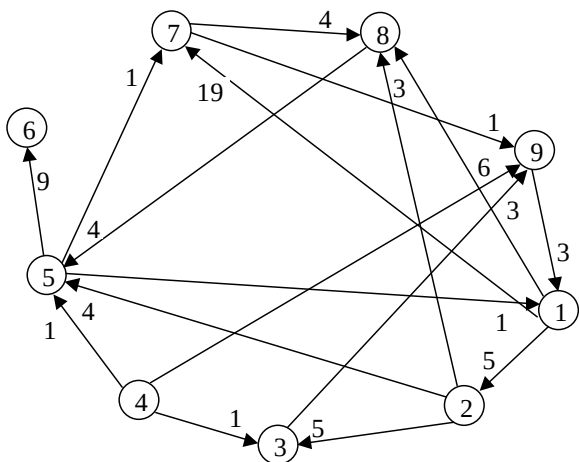
Вариант 12. Путь из 1 в 7



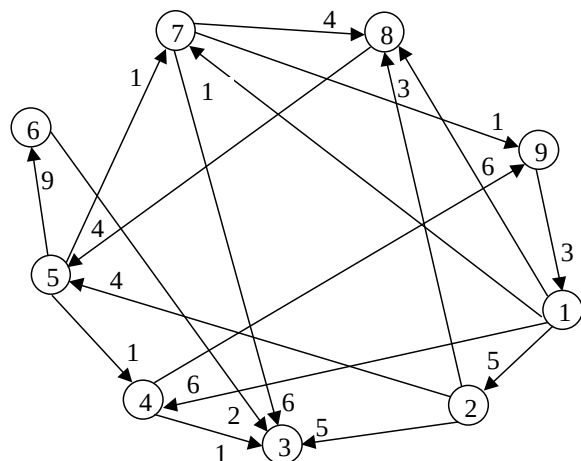
Вариант 15. Путь из 1 в 5



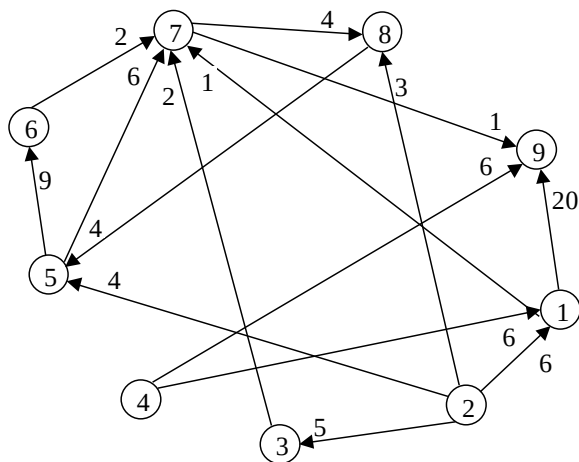
Вариант 13. Путь из 4 в 1



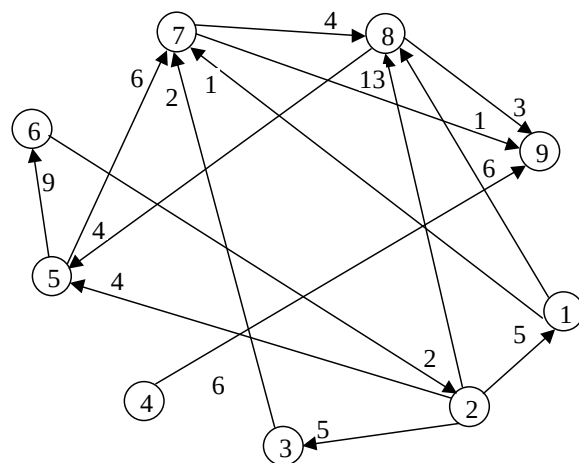
Вариант 16. Путь из 1 в 3



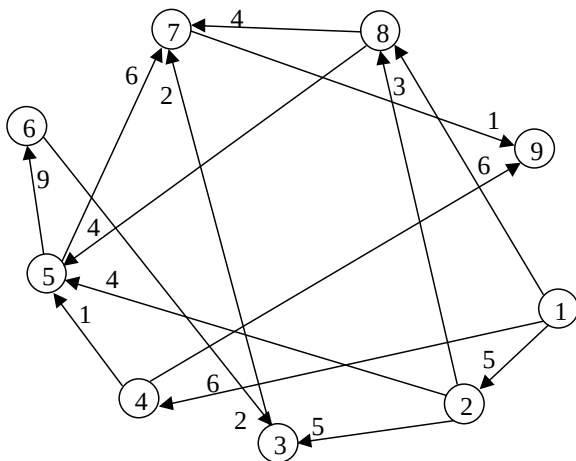
Вариант 17. Путь из 2 в 7



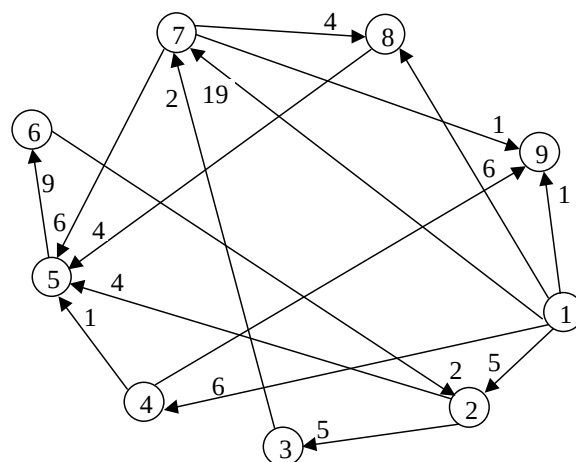
Вариант 20. Путь из 2 в 9



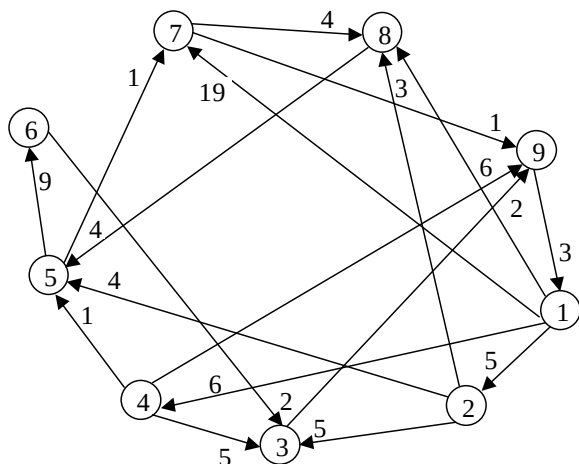
Вариант 18. Путь из 1 в 7



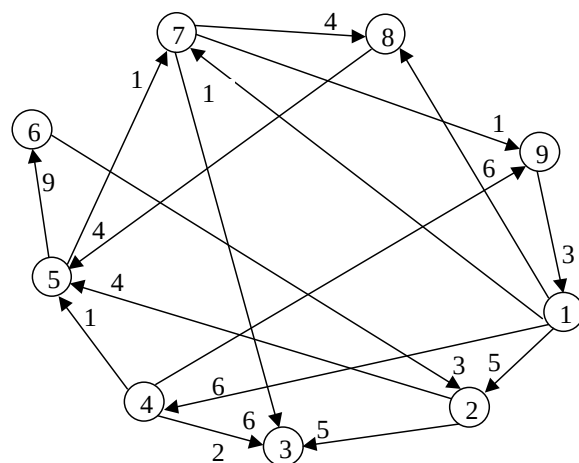
Вариант 21. Путь из 1 в 5



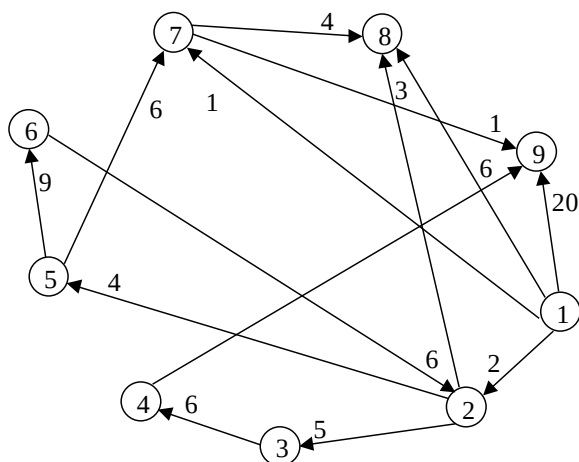
Вариант 19. Путь из 4 в 1



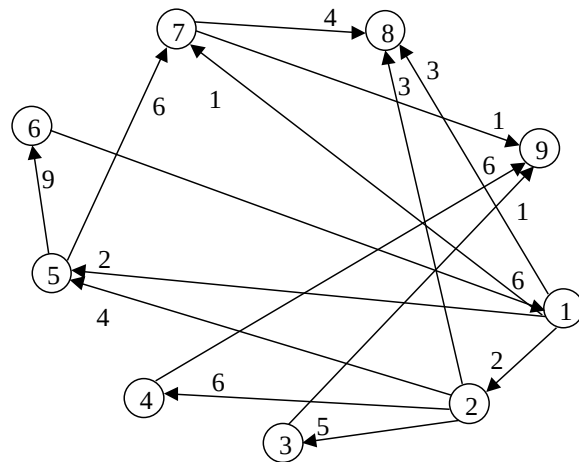
Вариант 22. Путь из 1 в 3



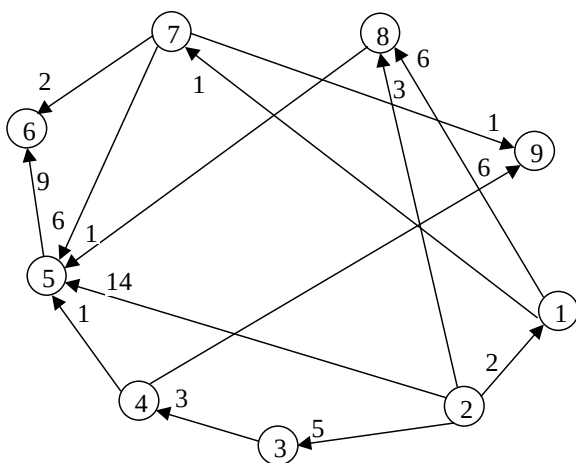
Вариант 23. Путь из 1 в 9



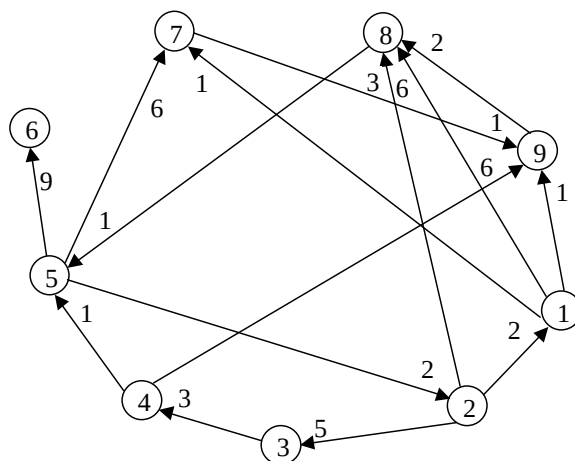
Вариант 25. Путь из 1 в 9



Вариант 24. Путь из 2 в 5



Вариант 26. Путь из 2 в 5



Лабораторная работа № 6

« Эвристические алгоритмы »

Цель работы: ознакомление с эвристическими алгоритмами и методикой оценки их эффективности .

Продолжительность работы: - 2 часа.

Теоретические сведения

Эвристический алгоритм – это алгоритм , в котором на определенном этапе используется интуиция разработчика . Если решение , принятое разработчиком , окажется неверным результат все равно будет получен , но за большее число шагов . Таким образом в эвристических алгоритмах можно увеличить скорость получения правильного результата

К эвристическим алгоритмам относятся : **волновой , двухлучевой , четырехлучевой , маршрутный, алгоритмы составления расписания.**

Волновой алгоритм.

Волновой алгоритм или алгоритм Ли первоначально использовался для поиска пути в лабиринте или в игровых задачах. В настоящее время алгоритм Ли (волновой) является основным в микроэлектронике для трассировки (соединения) элементов интегральных схем. Особенность алгоритма состоит в следующем:

- В лабиринте(на подложке) выбираются две точки А(начальная) и В(конечная). Из начальной точки в четырех направлениях выходит волна. Цифрами обозначается номер фронта волны или ее путевые координаты .

	1	
1	A	1
	1	

- Путевые координаты определяют шаг распространения волны. Каждый элемент первого фронта волны является источником вторичной волны.

		2		
	2	1	2	
2	1	A	1	2
	2	1	2	
		2		

Элементы второго фронта генерируют третий фронт и т.д. От запрещенных элементов волна не распространяется.

Процесс продолжается до тех пор, пока не будет достигнут конечный элемент. Траектория пути определяется обратным просмотром, от конечного к начальному. При этом разработчик задает приоритеты движения :

Вверх , Вниз , Влево , Вправо.

От того в каком порядке заданы приоритеты зависит скорость решения задачи.

При построении траектории используется два принципа:

- 1) Движение осуществляется строго по заданным приоритетам.
- 2) При построении трассы, т.е. траектории движения, значения путевых координат должны уменьшаться.

Пример 1. Пусть задан лабиринт, где запрещенные элементы заштрихованы.

Найти путь между элементами А и В.

На первом этапе от элемента А распространяется волна до тех пор пока она не достигнет

конечного элемента В. На втором этапе выбираются приоритеты движения от конечной точки В к начальной А. Приоритеты выбираются исходя из взаимного расположения начального и конечного элемента. Предположим, что выбраны парадоксальные (логически неверные) приоритеты : вверх, вправо, вниз, влево. В этом случае трасса все равно будет построена, но за большее число шагов (сравнений).

6		10	9	8		10			
5				7		9	10		
4				5	6		8	9	В
3	2			4	5		7	8	
	1	2	3	4		6			9
1	А	1	2	3	4	5	6	7	8
2	1		3				7		
3	2		4	5	6		8	9	10

Двухлучевой алгоритм.

В двухлучевом алгоритме из начального и конечного элементов одновременно выходят по два луча, трасса считается проведенной, если пересекаются два разноименных луча (от разных источников). Если на пути луча встречается запрещенный элемент, его обход осуществляется по второму приоритетному направлению, характерному для лучей выходящих из одной точки.

Если же оба направления оказываются заблокированными запрещёнными элементами, либо достигнут край координатной сетки, то движение луча прекращается.

Выбор направления движения лучей происходит исходя из интуиции разработчика. Существуют варианты распространения лучей, которые выбираются из таблицы, после вычисления значений α и β .

$$\alpha = \begin{cases} 1, & \text{если } X_A - X_B \geq 0 \\ 0, & \text{если } X_A - X_B < 0 \end{cases} \quad \beta = \begin{cases} 1, & \text{если } Y_A - Y_B \geq 0 \\ 0, & \text{если } Y_A - Y_B < 0 \end{cases}$$

где (X_A, Y_A) – координаты начального элемента,
 (X_B, Y_B) – координаты конечного элемента.

Исходя из значений α и β выбирается вид распространения лучей:

$\alpha = 0, \beta = 0$	$\alpha = 1, \beta = 0$	$\alpha = 0, \beta = 1$	$\alpha = 1, \beta = 1$

Пример 2. Осуществить трассировку элементов А и В .

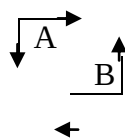
10										
9			<u>A</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>
8			1							<u>6</u>
7			2							<u>5</u>
6			3				7	6		<u>4</u>
5			4					5		<u>3</u>
4			5	6	7			4		<u>2</u>
3								3		<u>1</u>
2								2	1	<u>B</u>
1										
	1	2	3	4	5	6	7	8	9	10

$$\alpha = 3 - 10 = -7$$

$$\beta = 9 - 2 = 7$$

$$\alpha = 0$$

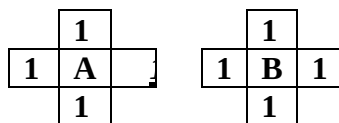
$$\beta = 1$$



- Вариант распространения лучей

Четырехлучевой алгоритм.

Из начальной и конечной точек выходят одновременно по четыре луча. Лучи движутся строго по заданным направлениям и “затухают”, если достигли края координатной сетки, либо встретили запрещенный элемент.



Пример 3.

		2					
		1					
1	A						
	<u>1</u>				1		
	2	2	<u>1</u>	B	1	2	

Маршрутный алгоритм.

Маршрутный алгоритм получил свое название, потому что осуществляет одновременно и формирование фронта и прокладывание трассы. Источником волны на каждом шаге является конечный элемент участка трассы проложенной на предыдущих шагах.

В маршрутном алгоритме рассматриваются восьмиэлементная окрестность исходного элемента.

i-1,j-1	i,j-1	i+1,j-1
i-1,j	A	i+1,j
i-1,j+1	i,j+1	i+1,j+1

От каждого элемента окружения оценивается расстояние d до конечного элемента В.

$$d = \sqrt{(x_i - x_b)^2 + (y_i - y_b)^2} \quad \text{или} \quad d = |x_i - x_b| + |y_i - y_b|$$

Таким образом определяются восемь значений расстояний, из которых выбирается минимальное. Элемент для которого d оказалось минимальным считаем элементом трассы. Процесс повторяется до тех пор пока расстояние не будет равным нулю ($d=0$) т.е. пока не будет достигнут конечный элемент. Обход запрещенных элементов осуществляется исходя из интуиции разработчика.

Пример:

1	2			9	B	
3	A			7		
3	A	4		8	10	
5			6			

$$dl = \sqrt{(1-5)^2 + (4-5)^2} = \sqrt{17}$$

$$d2 = \sqrt{(2-5)^2 + (4-5)^2} = \sqrt{10}$$

$$d3 = \sqrt{(1-5)^2 + (3-5)^2} = \sqrt{20}$$

$$d4 = \sqrt{(3-5)^2 + (3-5)^2} = \sqrt{8}$$

$$d5 = \sqrt{(1-5)^2 + (2-5)^2} = \sqrt{25} = 5$$

$$d6 = \sqrt{(3-5)^2 + (2-5)^2} = \sqrt{13}$$

$$\min = d4;$$

$$d7 = \sqrt{2}; d8 = \sqrt{5};$$

$$d9 = 1 ; d10 = 2 ;$$

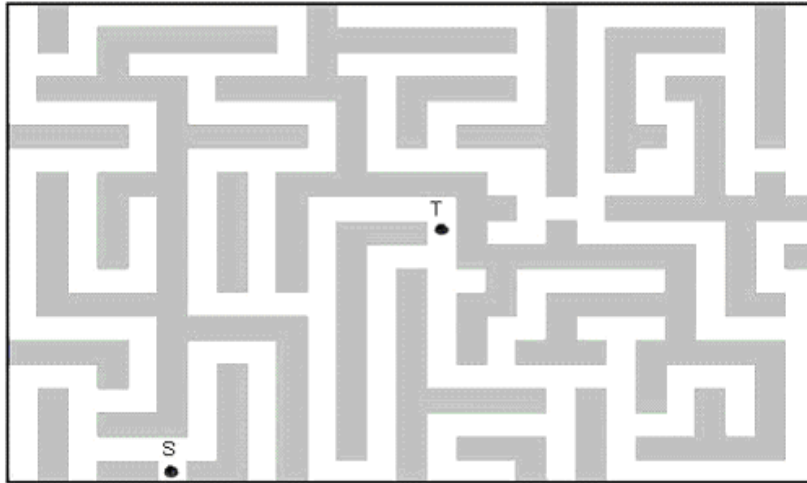
```
min = d7;
```

$$d\epsilon = 0;$$

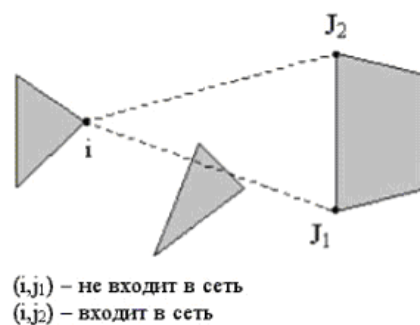
Геометрическая модель задачи о лабиринте

В лабиринте с произвольными препятствиями найти кратчайший путь между заданными точками.

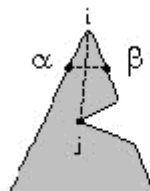
Решение: Так как препятствия на местности образуют многоугольники, или какие либо другие геометрические фигуры (которые с некоторыми погрешностями тоже можно изобразить в виде многоугольников), то кратчайшая трасса будет являться ломанной с узлами в вершинах этих многоугольников. Звено ломаной – это либо сторона многоугольника, либо прямолинейный отрезок, проходящий вне многоугольников и соединяющий две вершины одного и того же или разных многоугольников. Для решения этой задачи нужно построить сеть (ломаную), а так же соединить точки s и t с простреливаемыми из них вершинами, если эти точки не являются вершинами многоугольников.



Формирование сети, т. е. матрицы расстояний C размером $n \times n$ (n – общее число вершин всех многоугольников плюс два для учета старта и финиша) представляет собой тройной цикл. Внешний – по i – перебор вершин, откуда стреляют; средний – по j (j от $i+1$ до n , чтобы не повторяться) – это перебор вершин, куда стреляют; и внутренний – по k – это проверка, не пересекает ли k -я сторона какого-либо многоугольника отрезок соединения.



Последнее условие, в случае, как на рис. 8, проверяется по стандартным формулам аналитической геометрии: выписывается уравнение прямой, проходящей через i , j , выписывается уравнение прямой проходящей через концы отрезка k , решением системы из этих двух уравнений находится точка пересечения и устанавливается, лежит ли точка пересечения внутри рассматриваемых отрезков. Если да, то $d_{ij} = \infty$, конец цикла по k , если нет пересечения по окончанию цикла по k , то вычисляется Евклидово расстояние d_{ij} .



В случае на рис. 9, ситуация сложнее: между вершинами i и j не проходит ни какой стены, а j из i не простреливается. Чтобы преодолеть эту трудность, нужно ввести

характеристику i угла препятствия: g_i присвоив $g_i = 0$, если $\alpha_i < \pi$ (“вогнутый” угол), или $g_i = 1$, если $\alpha_i > \pi$ (“выпуклый” угол). Так, для угла с вершиной i на рис. 9 $g_i = 1$, а для угла с вершиной j $g_j = 0$.

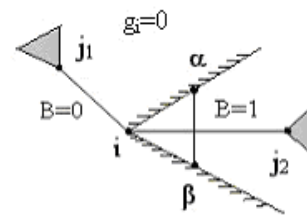
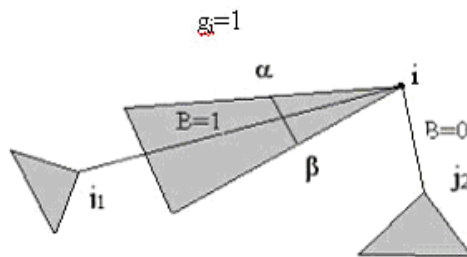
Если крайние вершины x_i и x_{i+3} ($x_i, x_{i+1}, x_{i+2}, x_{i+3}$ – последовательные вершины многоугольника) лежат по одну сторону от прямой, проходящей через соседние вершины x_{i+1}, x_{i+2} , то $g_{i+1} = g_{i+2}$, иначе $g_{i+1} \neq g_{i+2}$.

$$(x - x_{i+1})(y_{i+2} - y_{i+1}) - (x_{i+2} - x_{i+1})(y - y_{i+1}) = 0$$

Если при подстановке в это уравнение точек (x_i, y_i) и (x_{i+3}, y_{i+3}) в левой части получаются числа с одинаковым знаком, то $g_{i+1} = g_{i+2}$, иначе $g_{i+1} \neq g_{i+2}$. После этого цикла будут известны все g_i точно или с точностью до наоборот. Остается абсолютно установить g_i хотя бы для одной вершины. Это легко сделать, потому что экстремальная вершина имеет $g_0 = 1$.

Теперь можно справиться с трудностью, показанной на рис. 9. Из вершины i не простреливается никакая вершина j , защищенная углом с вершиной i . Чтобы исключить из рассмотрения загороженные вершины, нужно отступить от вершины i по сторонам угла на величину ϵ заведомо меньшую, чем длина стороны, построив таким образом точки α и β .

После чего нужно ввести бинарную величину B , $B = 1$, если отрезки $\alpha\beta$ и ij пересекаются; и $B = 0$, если отрезки $\alpha\beta$ и ij не пересекаются. Как например на рис. 10



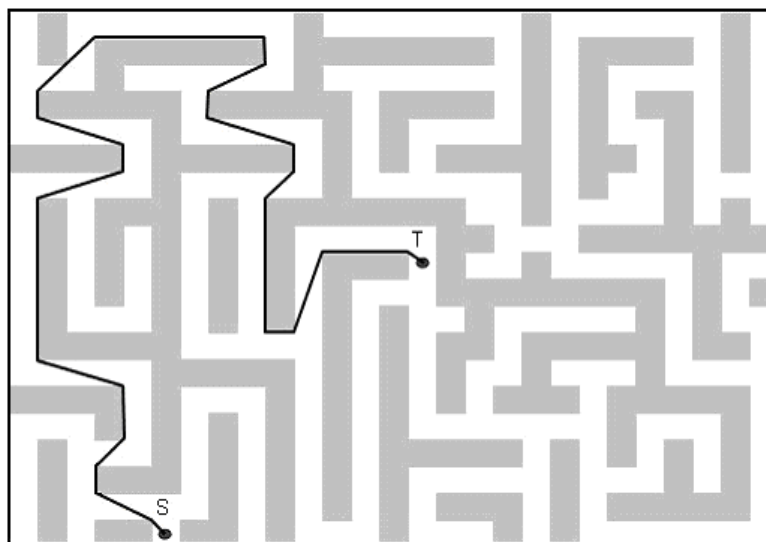
Всего имеется четыре возможности:

- 1) $B = 1$ и $g_i = 0$
- 2) $B = 0$ и $g_i = 1$
- 3) $B = 1$ и $g_i = 0$
- 4) $B = 1$ и $g_i = 1$

Ясно что вершина j не простреливается – в случаях 2 и 3 (при нечетном $B + g_i$).

Теперь можно построить сеть.

После того как сеть построена, можно приступать к нахождению кратчайших путей, воспользовавшись любым из выше рассмотренных алгоритмов (в зависимости от поставленной задачи).



Алгоритмы составления расписания.

Предположим, что имеется множество n одинаковых процессоров, обозначенных $P_1, P_2, \dots, P_n$, и m независимых заданий $J_1, J_2, \dots, J_m$, которые нужно выполнить. Процессоры могут работать одновременно, и любое задание можно выполнять на любом процессоре. Если задание загружено в процессор, оно остается там до конца обработки. Время обработки задания J_i известно и равно $t_i, i = 1, 2, \dots, m$. Организовать обработку заданий таким образом, чтобы выполнение всего набора заданий было завершено как можно быстрее.

Система работает следующим образом: первый освободившийся процессор берет из списка следующее задание. Если одновременно освобождаются два или более процессоров, то выполнять очередное задание из списка будет процессор с наименьшим номером.

Пример. Пусть имеется три процессора и шесть заданий, время выполнения каждого из которых равно: $t_1 = 2, t_2 = 5, t_3 = 8, t_4 = 1, t_5 = 5, t_6 = 1$.

Рассмотрим расписание $L = (J_2, J_5, J_1, J_4, J_6, J_3)$. В начальный момент времени $T=0$, процессор P_1 начинает обработку задания J_2 , процессор P_2 - задания J_5 , а процессор P_3 - задания J_1 . Процессор P_3 заканчивает выполнение задания J_1 в момент времени $T = 2$ и начинает обрабатывать задание J_4 , пока процессоры P_1 и P_2 все еще работают над своими первоначальными заданиями. При $T=3$ процессор P_3 опять заканчивает задание J_4 и начинает обрабатывать задание J_6 , которое завершается в момент $T=4$. Тогда он начинает выполнять последнее задание J_3 . Процессоры P_1 и P_2 заканчивают задания при $T=5$, но, так как список L пуст, они останавливаются. Процессор P_3 завершает выполнение задания J_3 при $T=12$.

Рассмотренное расписание проиллюстрировано на рис.1. временной диаграммой, известной как *схема Ганта*. Очевидно, что расписание не оптимально. Можно «подобрать», например, расписание $L^* = (J_3, J_2, J_5, J_1, J_4, J_6)$, которое позволяет завершить все задания за $T^* = 8$ единиц времени (рис.2.).

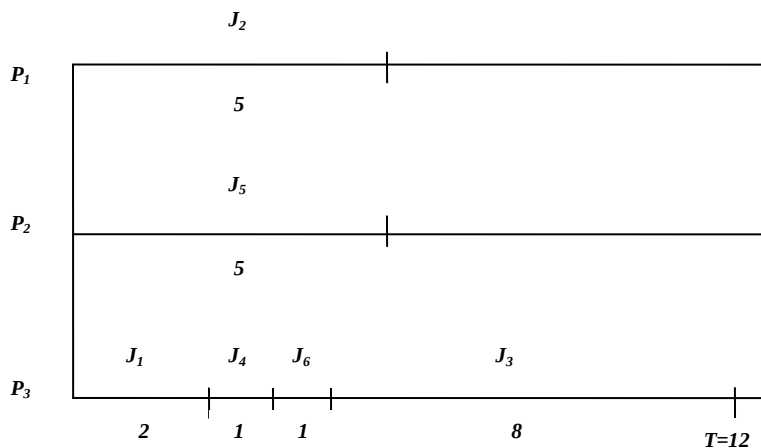


Рис.1. Схема Ганта для расписания L

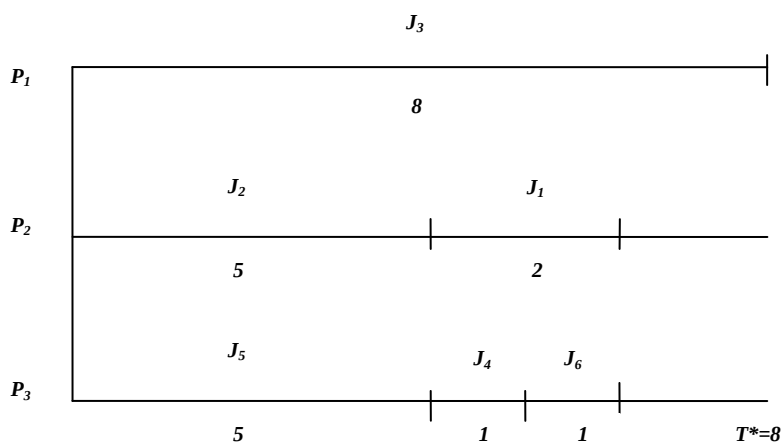


Рис.2. Схема Ганта для оптимального расписания

Теперь рассмотрим другой тип задач по составлению расписания для многопроцессорных систем. Вместо вопроса о быстрейшем завершении набора заданий фиксированным числом процессоров теперь поставим вопрос о минимальном числе процессоров, необходимых для завершения данного набора заданий за фиксированное время T_0 . Конечно, время T_0 будет не меньше времени выполнения самого трудоемкого задания.

В такой постановке задача составления расписания эквивалентна следующей задаче упаковки. Пусть каждому процессору P_j соответствует ящик B_j размера T_0 . Пусть каждому заданию J_i соответствует предмет размера t_i , равного времени выполнения задания J_i , где $i = 1, 2, \dots, n$. Теперь для решения задачи по составлению расписания нужно построить алгоритм, позволяющий разместить все предметы в минимальном количестве ящиков. Конечно, нельзя заполнять ящики сверх их объема T_0 , и предметы нельзя дробить на части.

Литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест
Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
2. Д.Кнут Искусство программирования, том 1. Основные алгоритмы. Уч. пос. М.:Изд. Дом " Вильямс ", 2000.
3. Вирт Н. Алгоритмы и структуры данных.: Пер. с англ. - М.: Мир, 2001.
4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учеб. пособие. М : Финансы и статистика, 2004.
5. А. Ахо, Дж.Хопкрофт, Дж.Ульман, Структуры данных и алгоритмы М: СПб: Киев: Вильямс, 2001г.

Лабораторное задание.

Для выполнения лабораторной работы необходимо:

- 1) Ознакомиться с эвристическими алгоритмами.
- 2) Осуществить трассировку элементов интегральных схем, размером 10x10, 20x20, 30x30. Зафиксировать параметры трассировок всеми рассмотренными методами.

Путь к файлу: D:\ИПОВС\АиСД\HEVRIST\Hevrlist1

Путь к файлу: D:\ИПОВС\АиСД\ HEVRIST\Hevrlist2

Путь к файлу: D:\ИПОВС\АиСД\ HEVRIST\Hevrlist3

Путь к файлу: D:\ИПОВС\АиСД\ HEVRIST\Hevrlist4

Система работает в диалоговом режиме с использованием «меню». Вся необходимая поясняющая информация отображается во время работы системы на экране монитора.

- 3) Составить оптимальное расписание работы четырех процессоров, для которых известно $t_1, \dots, t_{11}$.
- 4) Составить алгоритм оптимальной упаковки 12 предметов, размером от 1 до 4 в ящики размером 6.
- 5) Составить программу эвристического алгоритма (по заданию преподавателя)

Требования к отчету

Отчет должен содержать:

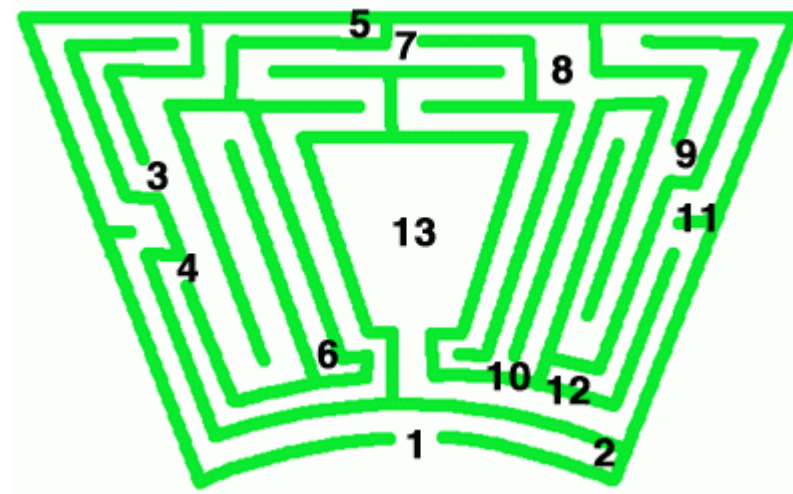
- 5) Конспект лабораторной работы;
- 6) Схемы волнового и лучевых алгоритмов;
- 7) Результаты выполнения работы;
- 8) Выводы по работе.

Контрольные вопросы

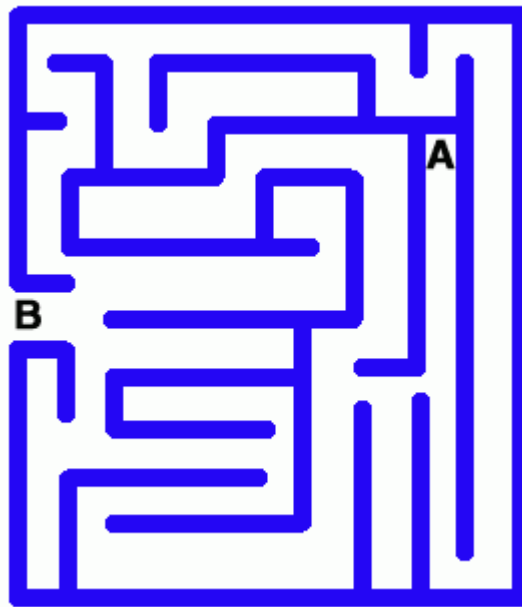
- 6) Какова теоретическая сложность алгоритмов, рассмотренных в данной работе?
- 7) Особенности работы волнового, лучевых и маршрутного алгоритмов?
- 8) Принципы составления оптимального расписания работы параллельных процессоров?
- 9) В чем особенности задачи упаковки?
- 10) Принципы решения задачи о джипе?
- 6) Как построить дерево решений в задаче о кодовом замке?

Решить задачи

Пример 1. Найти выход из произвольной точки лабиринта в саду Хемптон Корт. Отождествив коридоры лабиринта с ребрами, а перекрестки, тупики, входы и выходы - с вершинами, перейти к связному графу, представляющему схему лабиринта.



Пример 2. Нарисуйте граф, соответствующий лабиринту. Найдите путь, по которому можно пройти от пункта А до В лабиринта, используя предложенные выше алгоритмы.



Пример 3. (Задача о джипе). Пусть необходимо пересечь на джипе **1000** -километровую пустыню, израсходовав при этом минимум горючего. Объем топливного бака джипа **500** литров, горючее расходуется равномерно, по одному литру на километр. При этом в точке старта имеется неограниченный резервуар с топливом. Так как в пустыне нет складов с горючим, необходимо установить собственные хранилища и наполнять их топливом из бака машины. Конечно, проще было бы ехать на грузовике, загруженным бочками с бензином, но тогда не было бы задачи о джипе.

Итак, идея задачи ясна: нужно из точки старта отъезжать с полным баком на некоторое расстояние, устраивать там первый склад, оставлять там какое-то количество горючего из бака, но такое, чтобы хватило вернуться назад. В точке старта вновь производится полная заправка и делается попытка второй склад продвинуть в пустыню дальше. Но где обустраивать эти склады и сколько горючего оставлять в каждом из них?

Пример 4. (Задача о кодовом замке). Пусть кодовый замок состоит из набора N переключателей, каждый из которых может быть в положении “вкл” или “выкл”. Замок открывается только при одном наборе положений переключателей, из которых не менее $[N/2]$ (целая часть от $N/2$) , находятся в положении “вкл”. Построить алгоритм перебора комбинаций, чтобы не пропустить нужную и не набирать ту, которая заведомо к успеху не приведет.

Промоделируем каждую возможную комбинацию вектором из N нулей и единиц. На i -м месте будет **1**, если i -й переключатель находится в положении “вкл” и **0**, если i -й переключатель - в положении “выкл”. Множество всех возможных N -векторов моделируется с помощью бинарного (или двоичного) дерева. Если количество

переключателей в замке равно N , то в дереве просмотра будет N уровней. Решить задачу для $N=4$.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине
«Структуры и алгоритмы компьютерной обработки данных»
для студентов направления подготовки
02.03.01 «Математика и компьютерные науки»

Самостоятельная работа как важнейшая форма учебного процесса

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения. Государственным стандартом предусматривается, как правило, 50% часов из общей трудоемкости дисциплины на самостоятельную работу студентов (далее СРС). В связи с этим, обучение в ВУЗе включает в себя две, практически одинаковые по объему и взаимовлиянию части – процесса обучения и процесса самообучения. Поэтому СРС должна стать эффективной и целенаправленной работой студента.

Концепцией модернизации российского образования определены основные задачи профессионального образования - "подготовка квалифицированного работника соответствующего уровня и профиля, конкурентоспособного на рынке труда, компетентного, ответственного, свободно владеющего своей профессией и ориентированного в смежных областях деятельности, способного к эффективной работе по специальности на уровне мировых стандартов, готового к постоянному профессиональному росту, социальной и профессиональной мобильности".

Решение этих задач невозможно без повышения роли самостоятельной работы студентов над учебным материалом, усиления ответственности преподавателей за развитие навыков самостоятельной работы, за стимулирование профессионального роста студентов, воспитание творческой активности и инициативы.

К современному специалисту общество предъявляет достаточно широкий перечень требований, среди которых немаловажное значение имеет наличие у выпускников определенных способностей и умения самостоятельно добывать знания из различных источников, систематизировать полученную информацию, давать оценку конкретной финансовой ситуации. Формирование такого умения происходит в течение всего периода обучения через участие студентов в практических занятиях, выполнение контрольных заданий и тестов, написание курсовых и выпускных квалификационных работ. При этом самостоятельная работа студентов играет решающую роль в ходе всего учебного процесса.

Формы самостоятельной работы студентов разнообразны. Они включают в себя:

- изучение и систематизацию официальных государственных документов - законов, постановлений, указов, нормативно-инструкционных и справочных материалов с использованием информационно-поисковых систем "Консультант-плюс", "Гарант", глобальной сети "Интернет";
- изучение учебной, научной и методической литературы, материалов периодических изданий с привлечением электронных средств официальной, статистической, периодической и научной информации;
- подготовку докладов и рефератов, написание курсовых и выпускных квалификационных работ;
- участие в работе студенческих конференций, комплексных научных исследованиях.

Самостоятельная работа приобщает студентов к научному творчеству, поиску и решению актуальных современных проблем.

Цели и основные задачи СРС

Ведущая цель организации и осуществления СРС должна совпадать с целью обучения студента – подготовкой специалиста и бакалавра с высшим образованием. При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности.

Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

Виды самостоятельной работы

В образовательном процессе высшего профессионального образовательного учреждения выделяется два вида самостоятельной работы – аудиторная, под руководством преподавателя, и внеаудиторная. Тесная взаимосвязь этих видов работ предусматривает дифференциацию и эффективность результатов ее выполнения и зависит от организации, содержания, логики учебного процесса (межпредметных связей, перспективных знаний и др.):

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Основными видами самостоятельной работы студентов без участия преподавателей являются:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- написание рефератов;
- подготовка к семинарам и лабораторным работам, их оформление;
- составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

- подготовка рецензий на статью, пособие;
- выполнение микроисследований;
- подготовка практических разработок;
- выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;
- компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов.

(В зависимости от особенностей факультета перечисленные виды работ могут быть расширены, заменены на специфические).

Основными видами самостоятельной работы студентов с участием преподавателей являются:

- текущие консультации;
- коллоквиум как форма контроля освоения теоретического содержания дисциплин: (в часы консультаций, предусмотренных учебным планом);
- прием и разбор домашних заданий (в часы практических занятий);
- прием и защита лабораторных работ (во время проведения л/р);
- выполнение курсовых работ (проектов) в рамках дисциплин (руководство, консультирование и защита курсовых работ (в часы, предусмотренные учебным планом);
- выполнение учебно-исследовательской работы (руководство, консультирование и защита УИРС);
- прохождение и оформление результатов практик (руководство и оценка уровня сформированности профессиональных умений и навыков);
- выполнение выпускной квалификационной работы (руководство, консультирование и защита выпускных квалификационных работ) и др.

Организация СРС

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Организацию самостоятельной работы студентов обеспечивают: факультет, кафедра, учебный и методический отделы, преподаватель, библиотека, ТСО, ИВТ, издательство и др.

Деятельность студентов по формированию и развитию навыков учебной самостоятельной работы.

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, саморефлексии и становится активным самостоятельным субъектом учебной деятельности.

Выполняя самостоятельную работу под контролем преподавателя студент должен:

- освоить минимум содержания, выносимый на самостоятельную работу студентов и предложенный преподавателем в соответствии с ФГОС ВО по данной дисциплине.
- планировать самостоятельную работу в соответствии с графиком самостоятельной работы, предложенным преподавателем.
- самостоятельную работу студент должен осуществлять в организационных формах, предусмотренных учебным планом и рабочей программой преподавателя.
- выполнять самостоятельную работу и отчитываться по ее результатам в соответствии с графиком представления результатов, видами и сроками отчетности по самостоятельной работе студентов.

студент может:

сверх предложенного преподавателем (при обосновании и согласовании с ним) и минимума обязательного содержания, определяемого ФГОС ВО по данной дисциплине:

- самостоятельно определять уровень (глубину) проработки содержания материала;
- предлагать дополнительные темы и вопросы для самостоятельной проработки;
- в рамках общего графика выполнения самостоятельной работы предлагать обоснованный индивидуальный график выполнения и отчетности по результатам самостоятельной работы;
- предлагать свои варианты организационных форм самостоятельной работы;
- использовать для самостоятельной работы методические пособия, учебные пособия, разработки сверх предложенного преподавателем перечня;
- использовать не только контроль, но и самоконтроль результатов самостоятельной работы в соответствии с методами самоконтроля, предложенными преподавателем или выбранными самостоятельно.

Самостоятельная работа студентов должна оказывать важное влияние на формирование личности будущего специалиста, она планируется студентом самостоятельно. Каждый студент самостоятельно определяет режим своей работы и меру труда, затрачиваемого на овладение учебным содержанием по каждой дисциплине. Он выполняет внеаудиторную работу по личному индивидуальному плану, в зависимости от его подготовки, времени и других условий.

Общие рекомендации по организации самостоятельной работы

Основной формой самостоятельной работы студента является изучение конспекта лекций, их дополнение, рекомендованной литературы, активное участие на практических и семинарских занятиях. Но для успешной учебной деятельности, ее интенсификации, необходимо учитывать следующие субъективные факторы:

1. Знание школьного программного материала, наличие прочной системы знаний, необходимой для усвоения основных вузовских курсов. Это особенно важно для математических дисциплин. Необходимо отличать пробелы в знаниях, затрудняющие усвоение нового материала, от малых способностей. Затратив силы на преодоление этих пробелов, студент обеспечит себе нормальную успеваемость и поверит в свои способности.

2. Наличие умений, навыков умственного труда:

а) умение конспектировать на лекции и при работе с книгой;
б) владение логическими операциями: сравнение, анализ, синтез, обобщение, определение понятий, правила систематизации и классификации.

3. Специфика познавательных психических процессов: внимание, память, речь, наблюдательность, интеллект и мышление. Слабое развитие каждого из них становится серьезным препятствием в учебе.

4. Хорошая работоспособность, которая обеспечивается нормальным физическим состоянием. Ведь серьезное учение - это большой многосторонний и разнообразный труд. Результат обучения оценивается не количеством сообщаемой информации, а качеством ее усвоения, умением ее использовать и развитием у себя способности к дальнейшему самостоятельному образованию.

5. Соответствие избранной деятельности, профессии индивидуальным способностям. Необходимо выработать у себя умение саморегулировать свое эмоциональное состояние и устранять обстоятельства, нарушающие деловой настрой, мешающие намеченной работе.

6. Овладение оптимальным стилем работы, обеспечивающим успех в деятельности. Чередование труда и пауз в работе, периоды отдыха, индивидуально обоснованная норма продолжительности сна, предпочтение вечерних или утренних занятий, стрессоустойчивость на экзаменах и особенности подготовки к ним,

7. Уровень требований к себе, определяемый сложившейся самооценкой.

Адекватная оценка знаний, достоинств, недостатков - важная составляющая самоорганизации человека, без нее невозможна успешная работа по управлению своим поведением, деятельностью.

Одна из основных особенностей обучения в высшей школе заключается в том, что постоянный внешний контроль заменяется самоконтролем, активная роль в обучении принадлежит уже не столько преподавателю, сколько студенту.

Зная основные методы научной организации умственного труда, можно при наименьших затратах времени, средств и трудовых усилий достичь наилучших результатов.

Эффективность усвоения поступающей информации зависит от работоспособности человека в тот или иной момент его деятельности.

Работоспособность - способность человека к труду с высокой степенью напряженности в течение определенного времени. Различают внутренние и внешние факторы работоспособности.

К внутренним факторам работоспособности относятся интеллектуальные особенности, воля, состояние здоровья.

К внешним:

- организация рабочего места, режим труда и отдыха;
- уровень организации труда - умение получить справку и пользоваться информацией;
- величина умственной нагрузки.

Выдающийся русский физиолог Н. Е. Введенский выделил следующие условия продуктивности умственной деятельности:

- во всякий труд нужно входить постепенно;

- мерность и ритм работы. Разным людям присущ более или менее разный темп работы;
- привычная последовательность и систематичность деятельности;
- правильное чередование труда и отдыха.

Отдых не предполагает обязательного полного бездействия со стороны человека, он может быть достигнут простой переменой дела. В течение дня работоспособность изменяется. Наиболее плодотворным является *утреннее время (с 8 до 14 часов)*, причем максимальная работоспособность приходится на период с 10 до 13 часов, затем *послеобеденное* - (с 16 до 19 часов) и *вечернее* (с 20 до 24 часов). Очень трудный для понимания материал лучше изучать в начале каждого отрезка времени (лучше всего утреннего) после хорошего отдыха. Через 1-1,5 часа нужны перерывы по 10 - 15 мин, через 3 - 4 часа работы отдых должен быть продолжительным - около часа.

Составной частью научной организации умственного труда является овладение техникой умственного труда.

Физически здоровый молодой человек, обладающий хорошей подготовкой и нормальными способностями, должен, будучи студентом, отдавать *учению 9-10 часов в день* (из них 6 часов в вузе и 3 - 4 часа дома). Любой предмет нельзя изучить за несколько дней перед экзаменом. Если студент в году работает систематически, то он быстро все вспомнит, восстановит забытое. Если же подготовка шла аврально, то у студента не будет даже общего представления о предмете, он забудет все сданное.

Следует взять за правило: *учиться ежедневно, начиная с первого дня семестра.*

Время, которым располагает студент для выполнения учебного плана, складывается из двух составляющих: одна из них - это аудиторная работа в вузе по расписанию занятий, другая - внеаудиторная самостоятельная работа. Задания и материалы для самостоятельной работы выдаются во время учебных занятий по расписанию, на этих же занятиях преподаватель осуществляет контроль за самостоятельной работой, а также оказывает помощь студентам по правильной организации работы.

Чтобы выполнить весь объем самостоятельной работы, необходимо заниматься по 3 - 5 часов ежедневно. Начинать самостоятельные внеаудиторные занятия следует с первых же дней семестра, пропущенные дни будут потеряны безвозвратно, компенсировать их позднее усиленными занятиями без снижения качества работы и ее производительности невозможно. Первые дни семестра очень важны для того, чтобы включиться в работу, установить определенный порядок, равномерный ритм на весь семестр. Ритм в работе - это ежедневные самостоятельные занятия, желательно в одни и те же часы, при целесообразном чередовании занятий с перерывами для отдыха. Вначале для того, чтобы организовать ритмичную работу, требуется сознательное напряжение воли. Как только человек втянулся в работу, принуждение снижается, возникает привычка, работа становится потребностью.

Если порядок в работе и ее ритм установлены правильно, то студент изо дня в день может работать, не снижая своей производительности и не перегружая себя. Правильная смена одного вида работы другим позволяет отдыхать, не прекращая работы.

Таким образом, первая задача организации внеаудиторной самостоятельной работы - это составление расписания, которое должно отражать время занятий, их характер (теоретический курс, практические занятия, графические работы, чтение), перерывы на обед, ужин, отдых, сон, проезд и т.д. Расписание не предопределяет содержания работы, ее содержание неизбежно будет изменяться в течение семестра. Порядок же следует закрепить на весь семестр и приложить все усилия, чтобы поддерживать его неизменным (кроме исправления ошибок в планировании, которые могут возникнуть из-за недооценки объема работы или переоценки своих сил).

При однообразной работе человек утомляется больше, чем при работе разного характера. Однако не всегда целесообразно заниматься многими учебными дисциплинами в один и тот же день, так как при каждом переходе нужно вновь сосредоточить внимание, что может привести к потере времени. Наиболее целесообразно ежедневно работать не более чем над двумя-тремя дисциплинами.

Начиная работу, не нужно стремиться делать вначале самую тяжелую ее часть, надо выбрать что-нибудь среднее по трудности, затем перейти к более трудной работе. И напоследок оставить легкую часть, требующую не столько больших интеллектуальных усилий, сколько определенных моторных действий (черчение, построение графиков и т.п.).

Самостоятельные занятия требуют интенсивного умственного труда, который необходимо не только правильно организовать, но и стимулировать. При этом очень важно уметь поддерживать устойчивое внимание к изучаемому материалу. Выработка внимания требует значительных волевых усилий. Именно поэтому, если студент замечает, что он часто отвлекается во время самостоятельных занятий, ему надо заставить себя сосредоточиться. Подобную процедуру необходимо проделывать постоянно, так как это является тренировкой внимания. Устойчивое внимание появляется тогда, когда человек относится к делу с интересом.

Следует правильно организовать свои занятия по времени: 50 минут - работа, 5-10 минут - перерыв; после 3 часов работы перерыв - 20-25 минут. Иначе нарастающее утомление повлечет неустойчивость внимания. Очень существенным фактором, влияющим на повышение умственной работоспособности, являются систематические занятия физической культурой. Организация активного отдыха предусматривает чередование умственной и физической деятельности, что полностью восстанавливает работоспособность человека.

Самостоятельная работа студента - необходимое звено становления исследователя и специалиста

Прогресс науки и техники, информационных технологий приводит к значительному увеличению научной информации, что предъявляет более высокие требования не только к моральным, нравственным свойствам человека, но и в особенности, постоянно возрастающие требования в области образования – обновление, модернизация общих и профессиональных знаний, умений специалиста.

Всякое образование должно выступать как динамический процесс, присущий человеку и продолжающийся всю его жизнь. Овладение научной мыслью и языком науки является необходимой составляющей в самоорганизации будущего специалиста исследователя. Под этим понимается не столько накопление знаний, сколько овладение научно обоснованными способами их приобретения. В этом, вообще говоря, состоит основная задача вуза.

Специфика вузовского учебного процесса, в организации которого самостоятельной работе студента отводятся все больше места, состоит в том, что он является как будто бы последним и самым адекватным звеном для реализации этой задачи. Ибо во время учебы в вузе происходит выработка стиля, навыков учебной (познавательной) деятельности, рациональный характер которых будет способствовать постоянному обновлению знаний высококвалифицированного выпускника вуза.

Однако до этого пути существуют определенные трудности, в частности, переход студента от синтетического процесса обучения в средней школе, к аналитическому в высшей. Это связано как с новым содержанием обучения (расширение общего образования и углубление профессиональной подготовки), так и с новыми, неизвестными до сих пор формами: обучения (лекции, семинары, лабораторные занятия и т.д.). Студент получает не только знания, предусмотренные программой и учебными пособиями, но он

также должен познакомиться со способами приобретения знаний так, чтобы суметь оценить, что мы знаем, откуда мы это знаем и как этого знания мы достигли. Ко всему этому приходят через собственную самостоятельную работу.

Это и потому, что самостоятельно приобретенные знания являются более оперативными, они становятся личной собственностью, а также мотивом поведения, развивают интеллектуальные черты, внимание, наблюдательность, критичность, умение оценивать. Роль преподавателя в основном заключается в руководстве накопления знаний (по отношению к первокурсникам), а в последующие годы учебы, на старших курсах, в совместном установлении проблем и заботе о самостоятельных поисках студента, а также контролирования за их деятельностью. Отметим, что нельзя ограничиваться только приобретением знаний предусмотренных программой изучаемой дисциплины, надо постоянно углублять полученные знания, сосредотачивая их на какой-нибудь узкой определенной области, соответствующей интересам студента. Углубленное изучение всех предметов, предусмотренных программой, на практике является возможным, и хорошая организация работы позволяет экономить время, что создает условия для глубокого, систематического, заинтересованного изучения самостоятельно выбранной студентом темы.

Конечно, все советы, примеры, рекомендации в этой области, даваемые преподавателем, или определенными публикациями, или другими источниками, не гарантируют никакого успеха без проявления собственной активности в этом деле, т.е. они не дают готовых рецептов, а должны способствовать анализу собственной работы, ее целей, организации в соответствии с индивидуальными особенностями. Учитывая личные возможности, существующие условия жизни и работы, навыки, на основе этих рекомендаций, возможно, выработать индивидуально обоснованную совокупность методов, способов, найти свой стиль или усовершенствовать его, чтобы изучив определенный материал, иметь время оценить его значимость, пригодность и возможности его применения, чтобы, в конечном счете, обеспечить успешность своей учебы с будущей профессиональной деятельности

Методические рекомендации для студентов по отдельным формам самостоятельной работы

С первых же сентябрьских дней на студента обрушивается громадный объем информации, которую необходимо усвоить. Нужный материал содержится не только в лекциях (запомнить его – это только малая часть задачи), но и в учебниках, книгах, статьях. Порой возникает необходимость привлекать информационные ресурсы Интернет. Система вузовского обучения подразумевает значительно большую самостоятельность студентов в планировании и организации своей деятельности. Вчерашнему школьнику сделать это бывает весьма непросто: если в школе ежедневный контроль со стороны учителя заставлял постоянно и систематически готовиться к занятиям, то в вузе вопрос об уровне знаний вплотную встает перед студентом только в период сессии. Такая ситуация оборачивается для некоторых соблазном весь семестр посвятить свободному времяпрепровождению («когда будет нужно – выучу!»), а когда приходит пора экзаменов, материала, подлежащего усвоению, оказывается так много, что никакая память не способна с ним справиться в оставшийся промежуток времени.

Работа с книгой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Различают два вида чтения; первичное и вторичное. *Первичное* - это внимательное, неторопливое чтение, при котором можно остановиться на трудных местах. После него не должно остаться ни одного непонятого слова. Содержание не всегда может быть понятно после первичного чтения.

Задача *вторичного* чтения - полное усвоение смысла целого (по счету это чтение может быть и не вторым, а третьим или четвертым).

Правила самостоятельной работы с литературой

Как уже отмечалось, самостоятельная работа с учебниками и книгами (а также самостоятельное теоретическое исследование проблем, обозначенных преподавателем на лекциях) – это важнейшее условие формирования у себя научного способа познания. Основные советы здесь можно свести к следующим:

- Составить перечень книг, с которыми Вам следует познакомиться; «не старайтесь запомнить все, что вам в ближайшее время не понадобится, – советует студенту и молодому ученому Г. Селье, – запомните только, где это можно отыскать» (Селье, 1987. С. 325).

- Сам такой перечень должен быть систематизированным (что необходимо для семинаров, что для экзаменов, что пригодится для написания курсовых и дипломных работ, а что Вас интересует за рамками официальной учебной деятельности, то есть что может расширить Вашу общую культуру...).

- Обязательно выписывать все выходные данные по каждой книге (при написании курсовых и дипломных работ это позволит очень сэкономить время).

- Разобраться для себя, какие книги (или какие главы книг) следует прочитать более внимательно, а какие – просто просмотреть.

- При составлении перечней литературы следует посоветоваться с преподавателями и научными руководителями (или даже с более подготовленными и эрудированными сокурсниками), которые помогут Вам лучше сориентироваться, на что стоит обратить большее внимание, а на что вообще не стоит тратить время...

• Естественно, все прочитанные книги, учебники и статьи следует конспектировать, но это не означает, что надо конспектировать «все подряд»: можно выписывать кратко основные идеи автора и иногда приводить наиболее яркие и показательные цитаты (с указанием страниц).

• Если книга – Ваша собственная, то допускается делать на полях книги краткие пометки или же в конце книги, на пустых страницах просто сделать свой «предметный указатель», где отмечаются наиболее интересные для Вас мысли и обязательно указываются страницы в тексте автора (это очень хороший совет, позволяющий экономить время и быстро находить «избранные» места в самых разных книгах).

• Если Вы раньше мало работали с научной литературой, то следует выработать в себе способность «воспринимать» сложные тексты; для этого лучший прием – научиться «читать медленно», когда Вам понятно каждое прочитанное слово (а если слово незнакомое, то либо с помощью словаря, либо с помощью преподавателя обязательно его узнать), и это может занять немалое время (у кого-то – до нескольких недель и даже месяцев); опыт показывает, что после этого студент каким-то «чудом» начинает буквально заглатывать книги и чуть ли не видеть «сквозь обложку», стоящая это работа или нет...

• «Либо читайте, либо перелистывайте материал, но не пытайтесь читать быстро... Если текст меня интересует, то чтение, размышление и даже фантазирование по этому поводу сливаются в единый процесс, в то время как вынужденное скорочтение не только не способствует качеству чтения, но и не приносит чувства удовлетворения, которое мы получаем, размышляя о прочитанном», – советует Г. Селье (Селье, 1987. – С. 325-326).

• Есть еще один эффективный способ оптимизировать знакомство с научной литературой – следует увлечься какой-то идеей и все книги просматривать с точки зрения данной идеи. В этом случае студент (или молодой ученый) будет как бы искать аргументы «за» или «против» интересующей его идеи, и одновременно он будет как бы общаться с авторами этих книг по поводу своих идей и размышлений... Проблема лишь в том, как найти «свою» идею...

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют **четыре основные установки в чтении научного текста**:

1. информационно-поисковый (задача – найти, выделить искомую информацию)
2. усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)
3. аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)
4. творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

С наличием различных установок обращения к научному тексту связано существование и нескольких **видов чтения**:

1. библиографическое – просматривание карточек каталога, рекомендательных списков, сводных списков журналов и статей за год и т.п.;
2. просмотрное – используется для поиска материалов, содержащих нужную информацию, обычно к нему прибегают сразу после работы со списками литературы и

каталогами, в результате такого просмотра читатель устанавливает, какие из источников будут использованы в дальнейшей работе;

3. ознакомительное – подразумевает сплошное, достаточно подробное прочтение отобранных статей, глав, отдельных страниц, цель – познакомиться с характером информации, узнать, какие вопросы вынесены автором на рассмотрение, провести сортировку материала;

4. изучающее – предполагает доскональное освоение материала; в ходе такого чтения проявляется доверие читателя к автору, готовность принять изложенную информацию, реализуется установка на предельно полное понимание материала;

5. аналитико-критическое и творческое чтение – два вида чтения близкие между собой тем, что участвуют в решении исследовательских задач. Первый из них предполагает направленный критический анализ, как самой информации, так и способов ее получения и подачи автором; второе – поиск тех суждений, фактов, по которым или в связи с которыми, читатель считает нужным высказать собственные мысли.

Из всех рассмотренных видов чтения основным для студентов является изучающее – именно оно позволяет в работе с учебной литературой накапливать знания в различных областях. Вот почему именно этот вид чтения в рамках учебной деятельности должен быть освоен в первую очередь. Кроме того, при овладении данным видом чтения формируются основные приемы, повышающие эффективность работы с научным текстом.

Основные виды систематизированной записи прочитанного:

1. Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

2. Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

3. Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

4. Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

5. Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;

2. Выделите главное, составьте план;

3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;

4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной

последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Практические занятия

Для того чтобы практические занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

Самопроверка

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, используя лист опорных сигналов, воспроизвести по памяти определения, выводы формул, формулировки основных положений и доказательств.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

Консультации

Если в процессе самостоятельной работы над изучением теоретического материала или при решении задач у студента возникают вопросы, разрешить которые самостоятельно не удастся, необходимо обратиться к преподавателю для получения у него разъяснений или указаний. В своих вопросах студент должен четко выразить, в чем он

испытывает затруднения, характер этого затруднения. За консультацией следует обращаться и в случае, если возникнут сомнения в правильности ответов на вопросы самопроверки.

Подготовка к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий, особенно по математике - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неустойчивые занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Правила подготовки к зачетам и экзаменам:

- Лучше сразу сориентироваться во всем материале и обязательно расположить весь материал согласно экзаменационным вопросам (или вопросам, обсуждаемым на семинарах), эта работа может занять много времени, но все остальное – это уже технические детали (главное – это ориентировка в материале!).

- Сама подготовка связана не только с «запоминанием». Подготовка также предполагает и переосмысление материала, и даже рассмотрение альтернативных идей.

- Готовить «шпаргалки» полезно, но пользоваться ими рискованно. Главный смысл подготовки «шпаргалок» – это систематизация и оптимизация знаний по данному предмету, что само по себе прекрасно – это очень сложная и важная для студента работа, более сложная и важная, чем простое поглощение массы учебной информации. Если студент самостоятельно подготовил такие «шпаргалки», то, скорее всего, он и экзамены сдавать будет более уверенно, так как у него уже сформирована общая ориентировка в сложном материале.

- Как это ни парадоксально, но использование «шпаргалок» часто позволяет отвечающему студенту лучше демонстрировать свои познания (точнее – ориентировку в знаниях, что намного важнее знания «запомненного» и «тут же забытого» после сдачи экзамена).

- Сначала студент должен продемонстрировать, что он «усвоил» все, что требуется по программе обучения (или по программе данного преподавателя), и лишь после этого он вправе высказать иные, желательно аргументированные точки зрения.