МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ по дисциплине

«Теория алгоритмов»

для студентов направления подготовки

02.03.01 «Математика и компьютерные науки»

направленность (профиль)

«Анализ данных и искусственный интеллект»

Ставрополь 2025 г.

Введение

Практические занятия — это целенаправленная форма организации педагогического процесса, направленная на углубление научно-теоретических знаний и овладение определенными методами работы, в процессе которых вырабатываются умения и навыки выполнения тех или иных учебных действий в данной сфере науки.

Практические занятия предназначены для углубленного изучения учебных дисциплин и играют важную роль в выработке у студентов умений и навыков применения полученных знаний для решения практических задач совместно с педагогом. Кроме того, они развивают научное мышление и речь, позволяют проверить знания студентов и выступают как средства оперативной обратной связи.

Цель практических занятий - углублять, расширять, детализировать знания, полученные на лекции, в обобщенной форме и содействовать выработке навыков профессиональной деятельности.

Данные методические указания к практическим занятиям разработаны в соответствии с рабочей программой по дисциплине «Теория алгоритмов» для студентов направления подготовки 02.03.01 «Математика и компьютерные науки».

Тема 2. Машины Тьюринга и вычислимые по Тьюрингу функции

Учебные цели: приобрести первичные умения и навыки по использованию основных понятий машины Тьюринга и сочетаний машин Тьюринга к решению простейших арифметических задач, сводящихся к вычислению значений эффективно вычислимых функций.

Учебные вопросы занятия:

1. Машины Тьюринга и их функционирование
2. Композиция машин Тьюринга
3. Композиция машин Тьюринга
4. Итерация машин Тьюринга
5. Разветвление машин Тьюринга
6. Построение машин Тьюринга
7. Самостоятельная работа

Вопросы для проверки готовности студентов к занятию

1. Какие алфавиты включает в себя машина Тьюринга?
2. Какие состояния выделяются особо во внутренней памяти?
3. Что еще входит в конструкцию машины Тьюринга?
4. Что понимается под конфигурацией машины Тьюринга?
5. Что представляет собой программа машины Тьюринга? Какими способами ее можно задать?
6. Что понимают под данными и результатами в машине Тьюринга?
7. Какие шаги машины являются элементарными?
8. В каких случаях считается, что машина Тьюринга применима к начальной информации, а в каких случаях – не применима?
9. Охарактеризовать функционирование машины Тьюринга.
10. Сформулировать тезис Тьюринга.
11. Дать определение композиции машин Тьюринга.
12. Дать определение итерации машины Тьюринга.
13. Дать определение разветвления машин Тьюринга.

Предварительные сведения

Машина Тьюринга дает нам еще один путь уточнения интуитивного понятия алгоритма. Этот путь связан с машинной математикой, а сущность понятия алгоритма раскрывается посредством рассмотрения процессов, происходящих в простой гипотетической модели вычислительного устройства. Вычисления машиной Тьюринга описываются через последовательность конфигураций с помощью программы, определяющей переходы машины от одной конфигурации к другой. Программа машины Тьюринга может быть задана либо системой команд, либо таблицей, либо диаграммой переходов.

Такие сочетания и операции над машинами Тьюринга как композиция, разветвление и итерация позволяют при конструировании машин использовать стандартные приемы, аналогами которых в реальном программировании являются подпрограммы, ветвления, циклы, модульное программирование.

Справедлив следующий тезис Тьюринга: *всякий алгоритм может быть задан посредством функциональной схемы Тьюринга и реализован в соответствующей машине Тьюринга*. Тезисы Черча и Тьюринга эквивалентны.

На занятии следует рассмотреть задачи двух главных типов: по заданной машине Тьюринга Т найти результат ее применения к заданному слову u, то есть Т(u); построить машину Тьюринга, решающую данный класс задач.

1. Машины Тьюринга и их функционирование

Машина Тьюринга имеет три алфавита:

1. *Внешний алфавит*, включающий пустой символ.

2. *Внутренний алфавит* (внутренняя память), состоящий из конечного множества состояний Q=

соответствующий сдвиг. Затем машина переходит к выполнению следующей команды и т.д.

Задачи

1.1. Выяснить, применима ли машина Тьюринга Т, задаваемая программой *P*, к слову *u*. Если применима, то найти результат применения машины Т к слову *u*. Предполагается, что во внешнем алфавите *A*={∅,1}символ ∅ - пустой, а в алфавите состояний *Q*={q_0, q_1, q_2, q_3}q_1 - начальное состояние, q_0 – заключительное и в начальной конфигурации управляющая головка обозревает самую левую единицу на ленте, если не оговорено противное. Программу *P* задать системой команд, таблицей и диаграммой переходов.

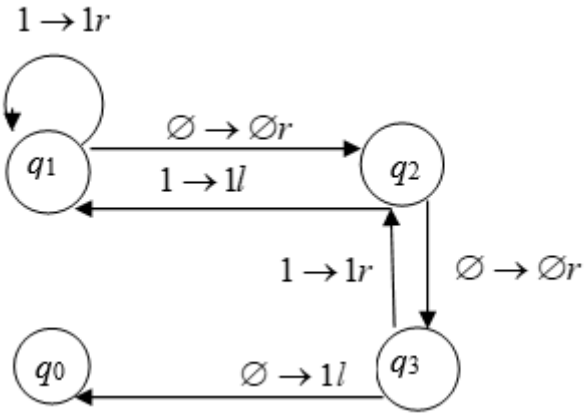
1.1.1.

q_1	$q_2 \varnothing r$	$q_1 1 r$
q_2	$q_3 \varnothing r$	$q_1 1 l$
q_3	$q_0 \varnothing s$	$q_2 1 r$

Запишем последовательно все переходы от одной конфигурации к другой согласно программе, начиная с начальной конфигурации:

1:	…	∅	1	1	1	∅	∅	1	1	∅	…
			q_1								
2:	…	∅	1	1	1	∅	∅	1	1	∅	…
				q_1							
3:	…	∅	1	1	1	∅	∅	1	1	∅	…
					q_1						
4:	…	∅	1	1	1	∅	∅	1	1	∅	…
						q_1					
5:	…	∅	1	1	1	∅	∅	1	1	∅	…
							q_2				
6:	…	∅	1	1	1	∅	∅	1	1	∅	…
								q_3			
7:	…	∅	1	1	1	∅	∅	1	1	∅	…
									q_2		
8:	…	∅	1	1	1	∅	∅	1	1	∅	…
								q_1			
9:	…	∅	1	1	1	∅	∅	1	1	∅	…
									q_1		
10:	…	∅	1	1	1	∅	∅	1	1	∅	…
										q_1	
11:	…	∅	1	1	1	∅	∅	1	1	∅	…
										q_2	
12:	…	∅	1	1	1	∅	∅	1	1	∅	…
										q_3	
13:	…	∅	1	1	1	∅	∅	1	1	∅	…
										q_0	
										stop	

Таким образом, $T(u) = 111\varnothing\varnothing 11 = 1^3\varnothing^2 1^2 = u$. Изобразим программу машины Т диаграммой переходов:

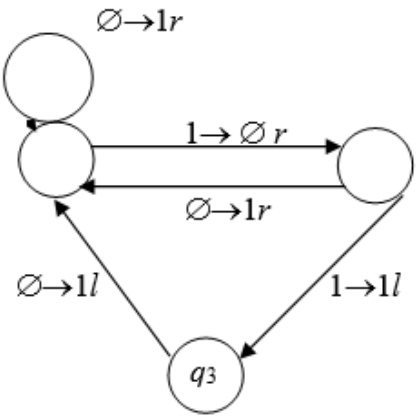


Легко проследить по этой диаграмме переходов процесс преобразования вектора *u* в вектор T(*u*).

Конфигурация 6 совпадает с конфигурацией 4, следовательно, машина Тьюринга не применима к входному слову $u=1\,^{3}\varnothing\,1^{3}$.

Ответкзадаче 1.1.1 (в) - 1∅

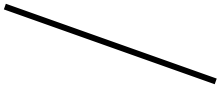
A $Q\backslash\{q_0\}$	∅	1
q_1	$q_1 1r$	$q_2 \varnothing r$
q_2	$q_1 1r$	$q_3 1l$
q_3	$q_1 1l$	-



Видим, что машина последовательно заполняет ленту единицами, но никогда не остановится.

2. Композиция машин Тьюринга

Определение. Пусть T_1 и T_2 – две машины Тьюринга соответственно с внешними и внутренними алфавитами A_1, A_2, Q_1 и Q_2, причем $Q_1 \cap Q_2 = Ø$. Композицией машин T_1 и T_2 называется машина $T_2 \circ T_1$ с внешним алфавитом



$P_{T2 \circ T1}$:

Применяя эту программу к слову $u=1^2\varnothing\ 1^3\varnothing 1^2$, получим заключительную конфигурацию K_0=…∅11∅111∅11∅…, то есть всего 18 конфигураций.

Ответ:$T_{2^\circ}T_1$ (u)=u.

3. Итерация машины Тьюринга

Определение. Пусть q_0 – заключительное состояние машины Тьюринга Т, q_i – какое-либо рабочее состояние. Заменим всюду в программе P машины Т состояние q_0 на q_i. Получим программу

а) $u=1^{2x}$,
б) $u=1^{2x+1}$,
i=1;
где$x\geq 1$.

до пустого символа, машина переходит в состояние q_1. На этом завершается первая итерация. В результате этой итерации слева слова будут две 1 заменены пустыми символами, а справа слова будет добавлена 1. Затем этот процесс повторяется до тех пор, пока половина единиц не будет заменена пустыми символами. Результатом работы машины

P_1:

q_{11}	$q_{12}\emptyset r$	$q_{11}\ 1\ r$
q_{12}		$q_{13}\ 1\ r$
q_{13}		$q_{13}\ 1\ r$

P_2:

q_{21}	$q_{22}\emptyset l$	$q_{21}\ 1l$
q_{22}	$q_{20}\emptyset r$	$q_{22}\ 1l$

P_3:

A / q_{3j}	$\emptyset$	1
q_{31}	$q_{32}\emptyset r$	$q_{31}\ 1\ r$
q_{32}	$q_{30}\ 1s$	$q_{31}\ 1\ r$

а)

б)

Не трудно увидеть, что в случае

Решение. Будем натуральные числа изображать в так называемом унарном коде с помощью символа $|^n$, $n \in N$, что не нарушает общности рассуждений.

Например, число 7_{10} будет изображаться в виде $|||||||$, при этом в каждой ячейке будет находиться только один символ $|$, причем данное натуральное число будет изображаться символами $|$, записанными подряд без пропусков ячеек.

Введем алфавит $A=\{a_0, |, +\}$, где a_0 обозначает пустой символ, а символ + - знак сложения. Например, сумма чисел 4 и 3 на ленте будет выглядеть так:

…	a_0							+						a_0	…

q_1

Идея решения задачи заключается в том, что управляющая головка, обозревая самый левый символ $|$, заменяет его пустым символом a_0, перемещается вправо до тех пор, пока не достигнет символа +. Затем она заменяет символ + символом $|$ и возвращается к первому символу $|$ слева и переходит в заключительное состояние q_0. Поэтому внутренний алфавит Q состоит из 4 состояний q_0, q_1, q_2, и q_3. Тогда один из вариантов программы P искомой машины Тьюринга может быть таким:

P_3:

A / $Q\backslash\{q_0\}$	a_0	+	1
q_1	-	$q_0 a_0 r$	$q_2 a_0 r$
q_2	-	$q_3 1\ l$	$q_2\ 1\ r$
q_3	$q_0 a_0 r$	-	$q_3 1\ l$

6. Самостоятельная работа

Построить композицию $T_2 \circ T_1$ машин Тьюринга T_1 и T_2 по паре состояний (q_{10}, q_{21}) (q_{20} – заключительное состояние машины T_2). Программу машины $T_2 \circ T_1$ задать таблицей, системой команд и диаграммой переходов. Применить композицию машин T_1 и T_2 к заданному слову u. Выписать последовательность конфигураций от начальной конфигурации до результирующей.

P_1:

A / q_{1j}	0	1
q_{11}	$q_{10} 0\ l$	$q_{12}\ 1\ r$
q_{12}	$q_{13}\ 0\ r$	$q_{13}\ 1\ r$
q_{13}	$q_{11}\ 0\ r$	$q_{11}\ 0\ r$

P_2:

A / q_{2j}	0	1
q_{21}	$q_{22} 1\ l$	$q_{22} 1\ l$
q_{22}	$q_{20}\ 0\ r$	$q_{21} 0\ l$

Слово u каждому студенту дается в соответствии с его вариантом.

1. 14.
2. 15.
3. 16.

4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.

Тема 3. Рекурсивные функции

***Учебные цели*:** Основной учебной целью является отработка первичных умений и навыков использования базовых вычислимых функций, операторов суперпозиции функций, примитивной рекурсии и минимизации к решению простейших задач формальной арифметики целых неотрицательных чисел. Главным является формирование методологического понимания факта замкнутости класса примитивно рекурсивных функций относительно применения конечного числа базовых вычислимых функций и примитивной рекурсии, а также того, что класс примитивно рекурсивных функций принципиально не может исчерпать весь класс эффективно вычислимых функций. Студенты должны научиться распознавать классы вычислимых, примитивно рекурсивных, частично рекурсивных и общерекурсивных функций, *понимать сущность тезиса Черча и закрепить определение алгоритма посредством частично рекурсивных функций*.

***Учебные вопросы занятия*:**

1. Замкнутость класса примитивно рекурсивных функций
2. Оператор минимизации
3. Частично рекурсивные и общерекурсивные функции и их использование для решения задач

Вопросы для проверки готовности студентов к занятию

1. Привести интуитивное описание алгоритма и перечислить основные свойства алгоритма.
2. Какие функции называются эффективно вычислимыми?
3. Почему возникла необходимость уточнения интуитивного понятия алгоритма?
4. Перечислить простейшие вычислимые функции и указать их область определения.
5. Дать определение оператора суперпозиции функций в аналитической форме.
6. Является ли суперпозиция функций всюду определенной?
7. Как может быть вычислена суперпозиция функций?
8. Дать аналитическое определение схемы примитивной рекурсии.
9. Дать определение примитивно рекурсивной функции и в индуктивной форме.
10. Как следует понимать высказывание о том, что класс примитивно рекурсивных функций замкнут?
11. Чем вызвано расширение средств построения вычислимых функций?
12. Дать определение оператора минимизации.
13. Дать определение частично рекурсивной функции.
14. Сформулировать тезис Черча.
15. Дать определение общерекурсивной функции.
16. Какое существует соотношение между классами вычислимых, примитивно рекурсивных, частично рекурсивных и общерекурсивных функций?

Разумеется, эти вопросы следует обсуждать не все сразу, а по мере необходимости в ходе проведения занятия.

Предварительные сведения

Данное занятие служит основой для математического определения понятия алгоритма и формулировке тезиса Черча. Умения, приобретенные в ходе этого занятия, потребуются для успешного использования оператора и схемы примитивной рекурсии при изучении технологий программирования. С другой стороны, студенты должны овладеть конкретным выбором средств, с помощью которых строятся вычислимые функции.

Из простейших функций с помощью применения конечного числа операторов суперпозиции и примитивной рекурсии можно получить большое число целочисленных функций, которые являются эффективно вычислимыми. Тем самым было установлено, что эти функции имеют примитивно рекурсивное описание, которое однозначно определяет процедуру вычисления их значений. Следовательно, их естественно отнести к классу эффективно вычислимых функций. Однако не все вычислимые функции являются примитивно рекурсивными, например, функция Аккермана является вычислимой, но не примитивно рекурсивной. Поэтому возникла проблема поиска средств для расширения класса примитивно рекурсивных функций, которые позволяли бы строить новые вычислимые функции. С этой целью был введен оператор минимизации, использование которого привело к понятию частично рекурсивной функции. Как оказалось, класс частично рекурсивных функций замкнут и совпадает с классом всех вычислимых функций, что и сформулировано в тезисе Черча: *любая эффективно вычислимая функция является частично рекурсивной*. Частным случаем частично рекурсивных функций являются общерекурсивные функции.

Данное практическое занятие посвящено закреплению определений операторов суперпозиции, примитивной рекурсии и минимизации, уточнению интуитивного понятия алгоритма посредством частично рекурсивных функций, а также решению простейших задач арифметики целых неотрицательных чисел с использованием перечисленных операторов.

1. Замкнутость класса примитивно рекурсивных функций

Простейшие вычислимые функции

К простейшим (или базовым) функциям относятся:

1. оператор сдвига (или следования)

1.2.1. $g_1(x_1,x_2,\ldots,x_n) = f(x_2,x_1,x_3,\ldots,x_n)$ (перестановка аргументов);
1.2.2. $g_2(x_1,x_2,\ldots,x_n) = f(x_2,x_3,\ldots,x_n,x_1)$ (циклическая перестановка аргументов);
1.2.3. $g_3(x_1,x_2,\ldots,x_{n-1}) = f(x_1,x_1,x_2,\ldots,x_{n-1})$ (отождествление аргументов);
1.2.4. $g_4(x_1,x_2,\ldots,x_n,x_{n+1}) = f(x_1,x_2,\ldots,x_n)$ (введение фиктивного аргумента).

Докажем, например, примитивную рекурсивность функции 1.2.2.

Функции g_1, g_2, g_3 и g_4 получаются из функций $f(x_1,x_2,\ldots,x_n)$ и

2. Оператор минимизации

Переход от функции $f(x_1, x_2,\ldots,x_n, y)$ к функции

$\varphi(x_1, x_2,\ldots,x_n) = \mu y[f(x_1, x_2,\ldots,x_n, y) = 0]$.

называется применение оператора минимизации (μ-оператора).

С помощью μ-оператора находится наименьшее значение y при фиксированных значениях $x_1, x_2,\ldots,x_n$, при котором $f(x_1, x_2,\ldots,x_n, y)=0$.

Для 2-местной функции $f(x, y)$ оператор минимизации записывается в виде

$\varphi(x)=\mu y[f(x, y) = 0]$.

Для вычисления значений функции φ можно предложить следующий алгоритм:

1. Вычисляем $f(x_1, x_2,\ldots,x_n, 0)$. Если значение равно 0, то полагаем $\varphi(x_1, x_2,\ldots,x_n)=0$. Если $f(x_1, x_2,\ldots,x_n, 0)\neq 0$, то переходим к следующему шагу.
2. Вычисляем $f(x_1, x_2,\ldots,x_n, 1)$. Если $f(x_1, x_2,\ldots,x_n, 1)=0$, то полагаем $\varphi(x_1, x_2,\ldots,x_n)=1$. Если $f(x_1, x_2,\ldots,x_n, 1)\neq 0$, то переходим к следующему аналогичному шагу, полагая $y=2,3,\ldots$

Если окажется, что для $\forall y\in N_0 f(x_1, x_2,\ldots,x_n, y)\neq 0$, то функцию $\varphi(x_1, \ldots, x_n)$ в этом случае считают *неопределенной*. Но возможно, что существует такое y_0, что $f(x_1, x_2,\ldots,x_n, y_0)=0$ и, значит, есть наименьшее y, при котором $f(x_1, x_2,\ldots,x_n, y)=0$. И в то же время может случиться, что при некоторых z, $0 < z < y_0$, значение функции $f(x_1, x_2,\ldots,x_n, z)$ не определено. Очевидно, что в этом случае процесс вычисления не дойдет до y_0. В этом случае функцию φ также считают неопределенной.

Иногда μ-оператор определяют с ограничением, полагая

$$\varphi(x_1, x_2,\ldots,x_n, y) = \mu y[f(x_1, x_2,\ldots,x_n, y) = z] \text{ для } y\leq z,$$

которое дает гарантию окончанию вычислений.

Задачи

2.1. Рассмотрим функцию $\varphi(x, y) = x - y$, определяющую разность двух целых неотрицательных чисел x и y. Получить эту функцию, применив оператор минимизации.

Решение.По определению, вычесть из числа x число y, значит, найти такое число z, что $x= z+ y$. Введем функцию $f(x, y, z) = y+ z- x$. Требуется найти такое наименьшее неотрицательное число z, чтобы $f(x, y, z)=0$, т.е. чтобы $y+ z= x$. Применяя к этой функции μ-оператор, получим:

$\varphi(x, y) = x-y= \mu z[y+z=x] = \mu z[$

2.2.1. $f(x_1)=3$, $i=1$.

2.2.2. $f(x_1)=[x_1/2]$, $i=1$, где символом $[x_1/2]$ обозначена целая часть числа $x_1/2$ и $[x_1/2]=m$, если $x_1=2m$ или $x_1=2m+1$, $m\geq 0$.

2.2.3. $f(x_1, x_2)=$

3.1.4.

Тема 4. Нормальные алгоритмы Маркова

Учебные цели: изучить нормальные алгоритмы Маркова, рассмотреть композицию, итерацию и разветвление нормальных алгоритмов

Учебные вопросы занятия:

1. Нормальные алгоритмы: основные понятия.
2. Композиция, итерация и разветвление нормальных алгоритмов.
3. Принцип нормализации Маркова. Эквивалентность различных теорий алгоритмов.

Основные теоретические сведения

Наряду с рекурсивными функциями и машинами Тьюринга еще одной формализацией интуитивного понятия алгоритма является теория нормальных алгоритмов. Эта теория была разработана советским математиком А.А.Марковым (1903-1979) в конце 40-х – начале 50-х годов XX века. Эти алгоритмы представляют собой некоторые правила по переработке слов в каком-либо алфавите, так что исходные данные и искомые результаты для алгоритмов являются словами в некотором алфавите. В качестве элементарной операции, на базе которой строятся нормальные алгоритмы, А.А. Марков выделил подстановку в слово одного подслова вместо другого.

Нормальные алгоритмы: основные понятия

О.1.1.**Алфавит А** – всякое непустое конечное множество символов. Символы алфавита называются **буквами.**

О.1.2. Любая конечная последовательность букв алфавита А называется **словом** в данном алфавите.

Пустое слово не имеет в своем составе ни одной буквы и обозначается Λ.

Множество всех слов в алфавите А обозначается A^*.

О.1.3. Алфавит В называется **расширением** алфавита А, если $A \subset B$.

В этом случае всякое слово в алфавите А является словом и в алфавите В.

Слова будем обозначать заглавными буквами латинского алфавита P, Q, R, S, T, … (с индексами или без). Алгоритмы будем обозначать M, M_1, M_2, …

О.1.4. Говорят, что слово Т **входит в слово** Q (является **подсловом** слова Q), если существуют такие (возможно пустые) слова U, W, что Q = UTW.

Пример 1.

Пусть А – алфавит русских букв. Рассмотрим слова:

$$P_1 = \text{ПАРАГРАФ},\ P_2 = \text{ГРАФ},\ P_3 = \text{РА}.$$

Слово P_2 – подслово слова P_1; слово P_3 – подслово слов P_2 и P_1, причем в P_1 оно входит дважды.

О.1.5.**Алгоритмом в алфавите** А называется всякая эффективно вычислимая функция, областью определения и множеством значений которой служат какие-нибудь подмножества множества A^* (возможно совпадающие).

О.1.6. Пусть Р – слово в алфавите А, т.е. $P \in A^*$. Говорят, что алгоритм М **применим к слову** Р, если Р содержится в области определения алгоритма М.

О.1.7. Если алфавит В является расширением алфавита А, то всякий алгоритм в алфавите В называется **алгоритмом над алфавитом** А.

Большинство известных алгоритмов можно разбить на некоторые простейшие шаги. Согласно А.А. Маркову, в качестве элементарной операции, на базе которой строятся алгоритмы, выделим **подстановку** одного слова вместо другого.

Пусть P и Q – слова в алфавите A, т.е. $P,Q \in A^*$.

О.1.8.**Марковской подстановкой** называется операция над упорядоченной парой слов (P,Q), состоящая в следующем. В заданном слове $R \in A^*$ находят первое вхождение слова P (если оно есть) и, не изменяя остальных частей слова R, заменяют в нем это вхождение словом Q. Полученное слово называется **результатом применения марковской подстановки** (P,Q) к слову R. Если вхождения слова P в слово R нет, то считается, что марковская подстановка (P,Q) **не применима** к слову R.

Частными случаями марковских подстановок являются подстановки с пустыми словами: (Λ,Q), (P,Λ), (Λ,Λ).

Пример 2.

К слову R = ШРАМ применим подстановку (РА,АР). В нашем случае P = РА, Q = АР. Результатом будет слово ШАРМ.

Пример 3.

В таблице рассматриваются различные **примеры марковских подстановок.**

преобразуемое слово	**марковская подстановка**	**результат**
13**85789**26	(85789, 00)	13**00**26
Т**АРА**РАМ	(АРА,Λ)	ТРАМ
ФУНКЦИЯ	(Λ,β-)	β-ФУНКЦИЯ
ЛОГ**ИКА**	(ИКА,Λ)	ЛОГ
КНИГА	(Λ,Λ)	КНИГА
ПОЛЯНА	(ПОР,Т)	не применима

Для обозначения марковской подстановки (P,Q) используется запись $\mathbf{P \to Q}$. Она называется **формулой простой подстановки** (P,Q).

Некоторые подстановки (P,Q) являются заключительными. Для обозначения таких подстановок используется запись $\mathbf{P \to\cdot\ Q}$. Она называется **формулой заключительной подстановки.**

Слово P называется **левой частью**, а слово Q – **правой частью** в формуле подстановки. Каждое из слов P и Q может быть пустым словом.

Предполагается, что стрелка → и точка · не являются буквами алфавита A, в котором построены слова P и Q.

О.1.9. Пусть $P \to (\cdot) Q$ обозначает одну из формул подстановки $P \to Q$ или $P \to \cdot Q$. Упорядоченный конечный список формул подстановок в алфавите A

$$\begin{cases} P_1 \to (\cdot) Q_1, \\ P_2 \to (\cdot) Q_2, \\ \ldots\ldots\ldots\ldots \\ P_r \to (\cdot) Q_r \end{cases} \quad \textbf{(1)}$$

называется **схемой (записью) нормального алгоритма** в A.

Данная схема определяет (детерминирует) алгоритм преобразования слов, называемый **нормальным алгоритмом Маркова.**

О.1.10. **Нормальным алгоритмом Маркова в алфавите A** называется правило M построения последовательности $\{R_i\}$ слов в алфавите A, исходя из данного слова R в этом алфавите. В качестве начального слова R_0 последовательности берется слово R. Пусть для некоторого $i \geq 0$ слово R_i построено и процесс построения рассматриваемой последовательности еще не завершился. Если при этом в схеме нормального алгоритма нет формул подстановки, левые части которых входили бы в R_i, то R_{i+1} полагают равным R_i, и процесс построения последовательности считается завершившимся. Если же в схеме имеются формулы с левыми частями, входящими в R_i, то в качестве R_{i+1} берется результат марковской подстановки первой из таких формул к слову R_i. Процесс построения

последовательности считается **завершившимся**, если на данном шаге была применена формула заключительной подстановки, и **продолжающимся** в противном случае.

Если процесс построения упомянутой последовательности обрывается, то говорят, что рассматриваемый **нормальный алгоритм M применим к слову** R.

Последний член S последовательности называется **результатом применения нормального алгоритма** M к слову R. При этом говорят, что нормальный алгоритм M перерабатывает R в S и записывают M(R) = S.

Последовательность $\{R_i\}$ записывается следующим образом:

$$R_0 \to R_1 \to R_2 \to .... \to R_{m-1} \to R_m,$$

где $R_0 = R$, $R_m = S$.

Работа нормального алгоритма M, задаваемого приведенной выше схемой, может быть описана следующим образом.

Пусть дано слово R в алфавите A. Ищем первую в схеме алгоритма M формулу подстановки $P_k \to (\cdot)\, Q_k$ такую, что слово P_k входит в слово R. Если ни одно из слов P_1, P_2, …, P_r не входит в R, то алгоритм M не применим к слову R.

Если находим первую в схеме алгоритма M формулу подстановки $P_k \to (\cdot)\, Q_k$ такую, что слово P_k входит в слово R, то совершаем подстановку слова Q_k вместо самого левого (первого) вхождения слова P_k в слово R.

Пусть R_1 – результат такой подстановки. Если $P_k \to (\cdot)\, Q_k$ – формула заключительной подстановки, то работа алгоритма заканчивается, и его результатом является слово R_1: $M(R) = R_1$.

Если подстановка $P_k \to (\cdot)\, Q_k$ не является заключительной, то применяем к слову R_1 тот же поиск, который был только что применим к слову R, и т.д.

Если мы в конце концов получили такое слово R_m ($m \geq 1$), что алгоритм M не применим к R_m (т.е. ни одно из слов P_1, P_2, …, P_r не входит в R_m), то работа алгоритма M заканчивается и R_m будет его результатом: $M(R) = R_m$. При этом, возможно, что описанный процесс никогда не закончится (будет продолжаться бесконечно). В таком случае так же считаем, что алгоритм M не применим к слову R.

Замечание

Мы определили понятие нормального алгоритма в алфавите A. Если же алгоритм задан в некотором расширении алфавита A, то говорят, что он есть **нормальный алгоритм над алфавитом A**. Различие между такими алгоритмами существенно. Всякий нормальный алгоритм в алфавите A использует только буквы из A, тогда как нормальный алгоритм над алфавитом A может использовать и некоторые дополнительные буквы, не входящие в A, хотя применяться он будет к словам в алфавите A и выдавать будет в результате слово в алфавите A.

Ясно, что всякий нормальный алгоритм в алфавите A будет так же и нормальным алгоритмом над алфавитом A. Обратное утверждение неверно: существуют алгоритмы в A, которые определяются нормальными алгоритмами над A, но сами не являются нормальными алгоритмами в A.

Примеры нормальных алгоритмов

Пример 4.

Пусть A = {a,b} – алфавит. Рассмотрим схему нормального алгоритма в A:

$$\begin{cases} a \to \cdot\Lambda, \\ b \to b. \end{cases}$$

Данный алгоритм всякое слово R в алфавите A, содержащее хотя бы одно вхождение переменной a, перерабатывает в слово, которое получается из R вычеркиванием в нем самого левого (первого) вхождения буквы a. Пустое слово оно перерабатывает в пустое.

Например:

- a a b a b $\Rightarrow$ a b a b

- a b $\Rightarrow$b
- b a $\Rightarrow$ b
- a a$\Rightarrow$a
- b a b $\Rightarrow$bb

Алгоритм не применим к таким словам, которые содержат только букву b (в этом случае процесс продолжается бесконечно).

Пример 5.

Пусть A = $\{a_0,a_1,\ldots,a_n\}$–алфавит. Рассмотрим схему: $\begin{cases} a_0 \to \Lambda, \\ a_1 \to \Lambda, \\ \ldots\ldots\ldots\ldots \\ a_n \to \Lambda, \\ \Lambda \to \cdot\Lambda. \end{cases}$

Она определяет нормальный алгоритм, перерабатывающий всякое слово в алфавите A в пустое слово.

Например: $a_1a_2a_1a_3\mathbf{a_0}\Rightarrow\mathbf{a_1}a_2a_1a_3\Rightarrow a_2\mathbf{a_1}a_3\Rightarrow\mathbf{a_2}a_3\Rightarrow\mathbf{a_3}\Rightarrow\Lambda\Rightarrow\Lambda$.

Пример 6.

В алфавите A = {1} схема $\Lambda\to\cdot1$ определяет нормальный алгоритм, который к каждому слову в алфавите A (все такие слова суть: Λ, 1, 11, 111, …) приписывает слева 1. Следовательно, данный алгоритм вычисляет функцию $\lambda(n) = n + 1$.

Композиция, итерация и разветвление нормальных алгоритмов

Пусть M_1 и M_2 – нормальные алгоритмы, а P – слово.

Запись $\mathbf{M_1(P) \cong M_2(P)}$ означает, что либо алгоритмы M_1 и M_2 оба не применимы к слову P, либо оба к нему применимы и $M_1(P) = M_2(P)$.

О.2.1. Два алгоритма M_1 и M_2 над алфавитом A называются **вполне эквивалентными** относительно алфавита A, если для любого слова P в алфавите A выполняется $M_1(P) \cong M_2(P)$.

О.2.2. Два алгоритма M_1 и M_2 над алфавитом A называются **эквивалентными** относительно алфавита A, если $M_1(P) \cong M_2(P)$ всякий раз, когда P есть слово в A, и хотя бы одно из слов $M_1(P)$ или $M_2(P)$ определено и является словом в A.

Уточним понятие вычислимости натуральных функций от натуральных аргументов с помощью нормальных алгоритмов.

Зафиксируем алфавит A = {1}и его расширение K = {*, 1}.

Представление натуральных чисел в нормальных алгоритмах

Для всякого натурального числа $n\in N$ определим по индукции следующие слова в алфавите A: $\overline{0}=1$, $\overline{n+1}=\overline{n}1$, т.е. $\overline{1}=11$, $\overline{2}=111$ и т.д. Слова $\overline{n}$ называются **цифрами.** Таким образом, натуральное число n в алфавите A = {1} представляется словом, состоящим из n + 1 единиц.

Представление упорядоченных n-ок натуральных чисел в нормальных алгоритмах

Рассмотрим расширенный алфавит К = {*, 1}. Поставим в соответствие каждой упорядоченной n-ке натуральных чисел $(k_1,k_2,\ldots,k_n)$ слово $\overline{k_1} * \overline{k_2} * \ldots * \overline{k_n}$ в алфавите К, которое обозначим $\overline{(k_1,k_2,\ldots,k_n)}$: $\overline{(k_1,k_2,\ldots,k_n)} = \overline{k_1} * \overline{k_2} * \ldots * \overline{k_n}$.

Пример 7.

Запись $\overline{(1,4,0,2)}$ обозначает слово 11 * 11111 * 1 * 111.

Пусть $\varphi(x_1,x_2,\ldots,x_n)$ – частичная эффективно вычислимая функция n аргументов, заданная на множестве N^n и принимающая значения во множестве N, т.е. f: $N^n \to N$.

О.2.3. Функция $\varphi(x_1,x_2,\ldots,x_n)$ называется **частично вычислимой по Маркову** (или **частично нормально вычислимой**), если существует нормальный алгоритм, вычисляющий эту функцию, т.е. такой алгоритм M_φ над алфавитом К, что

$$M_\varphi\left(\overline{(k_1,k_2,\ldots,k_n)}\right) = \varphi(k_1,k_2,\ldots,k_n)$$

всякий раз, когда хотя бы одна из частей этого равенства определена (тогда определена и другая часть этого равенства и эти части равны).

При этом предполагается, что нормальный алгоритм M_φ не применим к словам, отличным от слов вида $\overline{(k_1,k_2,\ldots,k_n)}$.

О.2.4. Если функция $\varphi(x_1,x_2,\ldots,x_n)$ всюду определена, т.е. для любой упорядоченной n-ки натуральных чисел, и является частично вычислимой по Маркову, то такая функция называется **вычислимой по Маркову** (или **нормально вычислимой**).

О.2.5. Нормальный алгоритм называется **замкнутым**, если его схема содержит формулу подстановки вида $\Lambda \to \cdot$ Q, где Q – произвольное слово.

Композиция нормальных алгоритмов

Два нормальных алгоритма можно сочетать следующим образом. Предписано, исходя из произвольных начальных данных, сначала применить первый алгоритм, а затем к результату его работы – второй алгоритм. Это предписание составляет новый алгоритм – композицию двух данных алгоритмов.

Пусть M_1 и M_2 – нормальные алгоритмы в алфавите А.Обозначим через $M_1^\bullet$ и $M_2^\bullet$ нормальные алгоритмы, которые получаются соответственно из схем алгоритмов M_1 и M_2 добавлением в конце новой формулы подстановки $\Lambda \to \cdot \Lambda$.

Сопоставим каждой букве $b \in A$ новую букву $\bar{b}$, которую назовем **двойником** буквы b.

Пусть $\bar{A}$ – алфавит, состоящий из всех двойников букв алфавита А. Выберем еще две буквы α и β, не принадлежащие $A \cup \bar{A}$.

Обозначим через Z_{M_1} схему, полученную из схемы нормального алгоритма $M_1^\bullet$ заменой в ней точки в каждой заключительной формуле подстановки буквой α, и обозначим через Z_{M_2} схему, которая получается путем замены в схеме нормального алгоритма $M_2^\bullet$ всех букв алфавита А их двойниками, всех точек – буквами β с последующей заменой всех формул подстановки вида $\Lambda \to$ Q и $\Lambda \to \cdot$ Q соответственно формулами подстановки $\alpha \to \alpha Q$ и $\alpha \to \alpha\beta Q$.

Пусть $b,c \in A$. Зададим алгоритм M_3 в алфавите $A \cup \bar{A} \cup \{\alpha,\beta\}$ схемой:

$$\begin{cases} b\alpha \to \alpha b, \\ \alpha b \to \alpha \bar{b}, \\ \bar{b}c \to \bar{\bar{b}}c, \\ \bar{b}\beta \to \beta \bar{b}, \\ \beta \bar{b} \to \beta b, \\ b\bar{c} \to bc, \\ \alpha\beta \to \cdot\Lambda, \\ Z_{M_1}, \\ Z_{M_2}. \end{cases} \qquad \textbf{(2)}$$

О.2.6.Алгоритм M_3 над алфавитом А, определяемый схемой (2), такой, что для любого слова Р из А

$$\mathbf{M_3(P) \cong M_2(M_1(P))},$$

называется **композицией нормальных алгоритмов** M_1 и M_2. Обозначение: $M_3 = M_1 \circ M_2$.

Из построения схемы (2) и определения 2.6 следует, что композиция двух нормальных алгоритмов является так же нормальным алгоритмом.

Итерация нормальных алгоритмов

Пусть даны алгоритмы M_1 и M_3 в алфавите А и произвольное слово $P_0 \in A$. Применим алгоритм M_1 к словуP_0. Если в результате получится некоторое слово P_1, то применим M_3 к P_1. Если окажется, что $M_3(P_1) = \Lambda$, то процесс заканчивается, если же $M_3(P_1) \neq \Lambda$, то применим M_1 к P_1.

Если в результате получится некоторое слово P_2, то снова применим M_3 к P_2 и опять, если $M_3(P_2) = \Lambda$, то процесс останавливается, если же $M_3(P_2) \neq \Lambda$, то применим M_1 к P_2 и т.д.

О.2.7. Построенный выше алгоритм M_2 называется **итерацией (повторением)** алгоритма M_1, управляемой алгоритмом M_3.

Очевидно, что $M_2(P_0) = Q$ тогда и только тогда, когда существует последовательность слов $P_0, P_1, \ldots, P_n$ $(n > 0)$ такая, что $P_n = Q$, $M_3(P_n) = \Lambda$, $P_i = M_1(P_{i-1})$ при $0 < i \leq n$ и $M_3(P_i) \neq \Lambda$ при $0 < i < n$.

В определении 2.7 не сказано, что рассматриваемые алгоритмы являются нормальными. При выполнении некоторых условий алгоритм M_2 оказывается нормальным. Чтобы определить эти условия, рассмотрим ряд понятий.

Пусть М – некоторый нормальный алгоритм в алфавите А и В – некоторое расширение алфавита А. В начало схемы алгоритма М добавим всевозможные формулы подстановки вида $b \to b$, где b – буква из В\А. Полученная таким образом схема определяет некоторый нормальный алгоритм M_B в алфавите В, который неприменим ни к какому слову, содержащему буквы из множества В\А, и такой, что

$$M_B(P) \cong M(P)$$

для любого слова Р в А. Алгоритм M_B вполне эквивалентен алгоритму М относительно алфавита А и называется **формальным распространением** алгоритма М на алфавит В.

Т.2.1.Пусть M_1 и M_3 – нормальные алгоритмы, А – объединение их алфавитов, M_{11} и M_{33} – формальные распространения соответственно M_1 и M_3 на алфавит А. Тогда существует нормальный алгоритмM_2 над А, являющийся итерацией (повторением) алгоритма M_{11}, управляемой алгоритмом M_{33}.

Разветвление нормальных алгоритмов

К исходным данным могут быть применены разные алгоритмы в зависимости от того, обладают эти данные каким-то свойством или нет.

Т.2.2.Пусть M_1, M_2, M_3 – нормальные алгоритмы и А – объединение их алфавитов. Тогда существует нормальный алгоритм М над алфавитом А, называемый **разветвлением** алгоритмов M_1 и M_2, управляемым алгоритмом M_3, такой, что

$$M(P) \cong \begin{cases} M_2(P), & \text{если } P - \text{слово в алфавите А и } M_3(P) = \Lambda, \\ M_1(P), & \text{если } P - \text{слово в алфавите А и } M_3(P) \neq \Lambda, \end{cases}$$

причем алгоритм М применим только к тем словам в алфавите А, к которым применим алгоритм M_3.

Принцип нормализации Маркова. Эквивалентность различных теорий алгоритмов

Создатель теории нормальных алгоритмов советский математик А.А. Марков выдвинул гипотезу, получившую название **принцип нормализации Маркова**: всякий алгоритм может быть реализован нормальным алгоритмом Маркова.

Другая формулировка: всякий алгоритм F в алфавите А эквивалентен относительно А некоторому нормальному алгоритму М над алфавитом А.

Так же как и тезисы Тьюринга и Черча, данный принцип не является теоремой и не может быть строго доказан. Он выдвинут на основании математического и практического опыта.

Установим эквивалентность теории нормальных алгоритмов теориям машин Тьюринга и рекурсивных функций.

Т.3.1. Класс всех частично рекурсивных функций совпадает с классом всех частично нормально вычислимых функций (частично вычислимых по Маркову).

Т.3.2. Класс нормально вычислимых функций совпадает с классом функций, вычислимых по Тьюрингу.

Т.3.3. (итоговая)

Следующие классы функций (заданных на множестве натуральных чисел и принимающих значения в этом же множестве) совпадают:

1) класс всех функций, вычислимых по Тьюрингу;
2) класс всех частично рекурсивных функций;
3) класс всех нормально вычислимых функций.

Практические задания

1. Пусть для слов в алфавите А = {a,b,c} заданы следующие марковские подстановки:

А) $ab \to dc$	Г) $cb \to d$	Ж) $dac \to acd$	К) $b \to a$
Б) $bc \to a$	Д) $abc \to \Lambda$	З) $da \to \Lambda$	Л) $a \to bd$
В) $dd \to bb$	Е) $cba \to \Lambda$	И) $ac \to dc$	

Примените каждую из них максимально возможное число раз к словам: ddacbabc, cbabcdac.

2. Примените к словам в алфавите А= {a,b}:

А) bbaab	Г) aaaa	Ж) abbbba	К) abbabba
Б) aabbbaa	Д) bbbb	З) baab	Л) abbbaaab
В) bababab	Е) aabaabb	И) bbbaaa	

следующие нормальные алгоритмы:

$$S_1:\begin{cases} bb \to ba \\ ba \to a \\ a \to \Lambda \\ b \to \cdot\Lambda \end{cases} \quad \text{и} \quad S_2:\begin{cases} ba \to a \\ bb \to b \\ ab \to \Lambda \\ \Lambda \to \cdot b \end{cases}$$

3. Написать формулу для функции f, вычисляемой нормальным алгоритмом. Проверить работу алгоритма над некоторым набором значений аргументов.

а) $f(x,y):\begin{cases} \alpha 1 \to 1\beta 1\alpha \\ \alpha * 1 \to \Lambda \\ \beta \to \gamma \\ \gamma \to \cdot\Lambda \\ \Lambda \to \alpha \end{cases}$ б) $f(x,y,z):\begin{cases} \alpha 1 \to \alpha \\ \alpha * 1 \to \beta \\ \beta 1 \to 11\beta \\ \beta * \to \cdot\Lambda \\ \Lambda \to \alpha \end{cases}$

Тема 5. Сложность вычислений и массовых проблем

***Учебные цели*:** расширить и углубить знания студентов путем анализа сложности алгоритмов сортировки массивов, умножения матриц и некоторых задач на графах.

***Учебные вопросы занятия*:**

1. Анализ сложности алгоритмов сортировки
2. Нижние оценки числа арифметических операций в матричной алгебре
3. Анализ сложности алгоритмов на графах

Вопросы для проверки готовности студентов к занятию

1. Дать определение временной и емкостной сложности алгоритма.

2. Охарактеризовать нотацию сложности с помощью символа «**О**большое». Какие еще нотации используются при исследовании асимптотической сложности алгоритмов? Когда и кем был введен символ **O**?

3. Доказать, что следующие утверждения истинны:

а) 17 имеет порядок $\mathbf{O}(1)$;

б) $n(n-1)/2$ имеет порядок $\mathbf{O}(n^2)$;

в) $\max(n^3, 10\,n^2)$ имеет порядок $\mathbf{O}(n^3)$;

г)

Основная операция – сравнение двух элементов. Мы также всегда будем предполагать, что во всех рассматриваемых методах сортировки не используются промежуточные массивы.

Второй тип задач посвящен исследованию асимптотической вычислительной сложности умножения квадратных матриц, элементы которых берутся из произвольного кольца. Мы увидим, что традиционный алгоритм умножения матриц имеет сложность $O(n^3)$. Но этот алгоритм можно асимптотически улучшить до порядка сложности $O(n^{2.81})$.

Многие задачи теоретического и прикладного характера можно сформулировать в терминах неориентированных или ориентированных графов. В третьем типе задач мы обсудим некоторые из основных задач на графах, решения которых полиномиально зависят (в смысле времени выполнения) от числа вершин и, следовательно, ребер (дуг) графа.

1. Анализ сложности алгоритмов сортировки

Проведем краткий обзор основных алгоритмоввнутренней сортировки.

1. Метод простых вставок. Данный алгоритм наиболее прост для программирования, не требует дополнительного пространства памяти и вполне эффективен при малых значениях *n*. При больших значениях *n* он становится медленным, если только исходный массив не окажется сразу почти упорядоченным.

2. Метод прямого выбора. Этот алгоритм довольно прост и особенно подходит в случае, когда имеется специальное оборудование для быстрого поиска наименьшего элемента в массиве. Этот метод требует наличия всех исходных элементов массива до начала сортировки, а элементы вывода он порождает последовательно один за другим. Картина, по существу, противоположна методу прямых вставок, в котором исходные элементы должны поступать последовательно, но вплоть до завершения сортировки ничего не известно об окончательном выводе.

3. Метод прямого обмена или, по-другому, «пузырьковая сортировка», является наихудшим методом из всех сравниваемых методов, хотя ее алгоритм наиболее прост для понимания. Этот метод представляет скорее исторический и методологический интерес, чем практический. Интересно сравнить метод прямого обмена с методом прямого выбора. Метод прямого обмена можно рассматривать как алгоритм прямого выбора, в котором иногда выбирается более одного элемента за один раз. По этой причине при сортировке методом пузырька производится меньше сравнений, чем при прямом выборе и она, как может показаться, предпочтительнее. Но в действительности соответствующая методу пузырька программа работает более чем вдвое медленнее программы, соответствующей методу прямого выбора. Сортировка методом прямого обмена проигрывает из-за того, что в ней выполняется слишком много обменов, в то время как при сортировке методом прямого выбора производится очень мало пересылок данных.

4. Шейкерная сортировка - одна из модификаций метода прямого обмена - продолжает оставаться плохой по сравнению с методом прямых вставок и методом прямого выбора, за исключением случая уже упорядоченного массива, хотя в шейкерной сортировке удается несколько сократить среднее число сравнений.

5. Метод Шелла. Алгоритм Шелла (1959 г.) или сортировка с убывающим шагом является одной из наиболее удачных модификаций метода прямых вставок, использует минимальный объем памяти и довольно эффективен при умеренно больших значениях *n*. В каждом из промежуточных процессов сортировки участвуют либо сравнительно короткие части массива, либо уже относительно хорошо упорядоченные части, поэтому на каждом этапе сортировки можно пользоваться методом прямых вставок; элементы массива довольно быстро достигают своего конечного положения. Для достижения большей эффективности при сортировке этим методом желательно пользоваться взаимно

простыми значениями шагов. При произвольной последовательности шагов сортировки не удается отыскать наихудший случай для алгоритма Шелла, поэтому попытки анализа этого алгоритма в общем случае приводят лишь к приближенному асимптотическому поведению максимального времени работы алгоритма.

6. Сортировка с помощью бинарного дерева. Выберем для изучения один из методов выбора из дерева - пирамидальную сортировку. Соответствующая программа более эффективна при больших значениях *n*, а именно, с ростом *n* пирамидальная сортировка превосходит по скорости метод Шелла. Пирамидальная сортировка эффективна лишь в среднем случае. Наихудший случай, как оказалось, ненамного хуже среднего.

7. Метод Хоара. На практике алгоритм Хоара является одним из самых полезных и универсальных алгоритмов внутренней сортировки, поскольку он требует мало памяти и опережает своих конкурентов по быстрой сортировке по среднему времени работы в 2-3 раза. Этот алгоритм сортирует массив, элементы которого расположены в обратном порядке, практически с той же скоростью, что и массив, порожденный случайным образом, однако, он работает очень медленно в редком на практике случае, когда исходный массив первоначально уже упорядочен.

8. Метод слияния. Различные методы слияния для сортировки массивов используются редко. Они предназначены для сортировки последовательных файлов.

Задачи

Прямые методы сортировки

1.1. Провести анализ сложности сортировки массивовс помощью алгоритма прямого выбора.

Решение. Сортировка массива размером n в неубывающем (невозрастающем) порядке методом прямого выбора выполняется по следующему алгоритму:

Выбирается элемент данного массива с минимальным (максимальным) значением (или ключом).

Выбранный элемент и первый элемент меняются местами.

Этот процесс повторяется с оставшимися n-1 элементами, затем с n-2 элементами и т.д. до тех пор, пока не останется один, самый больший (меньший) элемент.

Всего потребуется n-1 раз выполнить указанную последовательность действий. В процессе сортировки будет увеличиваться отсортированная часть массива, а неотсортированная часть, соответственно, уменьшаться.

В отличие от сортировки вставками, когда на каждом шаге рассматривается только один очередной элемент исходного массива и все элементы отсортированной части, среди которых ищется место для вставки, при прямом выборе для поиска одного элемента с наименьшим (наибольшим) ключом просматриваются все элементы исходного массива и найденный элемент помещается как очередной в отсортированную часть массива.

Напомним, что во всех рассматриваемых методах сортировки мы не используем промежуточных массивов.

В рассматриваемом методе сортировки число C сравнений ключей не зависит от начального порядка их расположения и $C=(n^2-n)/2$. Число перестановок минимально в случае изначально упорядоченных ключей: $M_{min}=0$, и максимально: $M_{max}=n^2/4 + 3(n-1)$, если первоначально ключи располагались в обратном порядке. Используя аппроксимацию интегралом сумму дискретных слагаемых, можно доказать, что среднее число перестановок элементов определяется приближенным равенством $M_{ср.}\approx n(\ln(n)+g)$, где g=0,57721... - константа Эйлера, связанная с n-ой частичной суммой H_n гармонического ряда $H_n=1+1/2+1/3+...+1/n$, а именно, $H_n - \ln n \to g$ при увеличении n.

Схему сортировки методом прямого выбора проиллюстрируем на примере.

Исходный массив:	22	-14	13	28	6	41	-5	18	0	35	6	**-15**
1-й шаг:	**-15**	-14	13	28	6	41	-5	18	0	35	6	**22**
2-й шаг:	-15	**-14**	13	28	6	41	-5	18	0	35	6	22
3-й шаг:	-15	-14	**-5**	28	6	41	**13**	18	0	35	6	22
4-й шаг:	-15	-14	-5	**0**	6	41	13	18	**28**	35	6	22
5-й шаг:	-15	-14	-5	0	**6**	41	13	18	28	35	6	22
6-й шаг:	-15	-14	-5	0	6	**6**	13	18	28	35	**41**	22
7-й шаг:	-15	-14	-5	0	6	6	**13**	18	28	35	41	22
8-й шаг:	-15	-14	-5	0	6	6	13	**18**	28	35	41	22
9-й шаг:	-15	-14	-5	0	6	6	13	18	**22**	35	41	**28**
10-й шаг:	-15	-14	-5	0	6	6	13	18	22	**28**	41	**35**
11-й шаг:	-15	-14	-5	0	6	6	13	18	22	28	**35**	**41**

1.2.Провести анализ сложности сортировки массивовс помощью алгоритма прямого обмена.

Решение. Алгоритм данного метода сортировки заключается в последовательных просмотрах массива от начала к концу и обмене местами *соседних* элементов, если они образуют *инверсию*. Опишем этот алгоритм подробнее.

Пусть задан массив *A* размером *n*. Начинаем просмотр массива с первой пары элементов *A*[1] и *A*[2]. Если первый элемент этой пары больше второго, то меняем их местами, иначе оставляем их без изменения и сравниваем вторую пару элементов *A*[2] и *A*[3]. Если A[2]>A[3], то меняем их местами и т.д. На первом шаге будут просмотрены все пары элементов массива *A*[*i*] и *A*[*i*+1] для *i* от 1 до *n*-1. В результате такого просмотра и необходимых обменов *максимальный* элемент переместится в конец массива. На втором шаге аналогичная процедура проводится с первого до (*n*-1)-го элемента. Тем самым второй по величине элемент массива переместится на предпоследнее место. Эти действия продолжаются до тех пор, пока количество элементов в неотсортированной части массива не уменьшится до двух. На последнем шаге выполняется упорядочение оставшихся двух элементов. После (*n*-1)-го шагов массив окажется отсортированным по неубыванию.

Сортировку методом прямого обмена называют еще методом «пузырька». Это название происходит от образной интерпретации, при которой элементы данного массива расположены вертикально и в процессе выполнения сортировки на каждом шаге более легкие элементы, как пузырьки в ванной с водой, поднимаются до уровня, соответствующего их весу.

Для иллюстрации отсортируем по неубыванию методом простого обмена массив, рассмотренный в предыдущей задаче (в таблице на каждом шаге отсортированная и неотсортированная части массива отделены вертикальной чертой):

Исходный массив:	22	-14	13	28	6	41	-5	18	0	35	6	-15
1-й шаг:	-14	13	22	6	28	-5	18	0	35	6	-15	41
2-й шаг:	-14	13	6	22	-5	18	0	28	6	-15	35	41
3-й шаг:	-14	6	13	-5	18	0	22	6	-15	28	35	41
4-й шаг:	-14	6	-5	13	0	18	6	-15	22	28	35	41
5-й шаг:	-14	-5	6	0	13	6	-15	18	22	28	35	41
6-й шаг:	-14	-5	0	6	6	-15	13	18	22	28	35	41
7-й шаг:	-14	-5	0	6	-15	6	13	18	22	28	35	41
8-й шаг:	-14	-5	0	-15	6	6	13	18	22	28	35	41
9-й шаг:	-14	-5	-15	0	6	6	13	18	22	28	35	41
10-й шаг:	-14	-15	-5	0	6	6	13	18	22	28	35	41
11-й шаг:	-15	-14	-5	0	6	6	13	18	22	28	35	41

Иногда используют просмотр массива в порядке, обратном описанному выше, как более естественном. В этом случае параметр *i*меняется от 2 до *n*, а параметр *j* - от *n* до *i*-1.

Время работы обменной сортировки определяется тремя величинами: числом *A* просмотров массива, числом *C* сравнений элементов и числом *M* перемещений элементов (присваиваний). Если элементы исходного массива различны и расположены в случайном порядке, то можно предположить, что их ключи образуют случайную перестановку множества {1,2,..., *n*}. Очевидно, что в наилучшем случае, когда исходный массив уже упорядочен, $A_{min}=1$, $C_{min}= n-1$, $M_{min}=0$. Максимум этих величин достигается в наихудшем случае, когда элементы исходного массива расположены в обратном порядке. Тогда число просмотров массива $A_{max}= n-1$, число сравнений $C_{max}=(n-1)+(n-2)+...+1= n(n-1)/2$, а число B_{max} обменов определяется числом инверсий в перестановке из элементов массива, т.е. $B_{max}= (n-1)+(n-2)+...+2+1= n(n-1)/2$. Так как для каждого обмена местами двух элементов массива требуется по три присваивания, то $M_{max}=3B_{max}=3/2n(n-1)$.

Наиболее сложным является определение асимптотического значения средних величин $A_{ср.}$, $C_{ср.}$, и $M_{ср.}$ Д. Кнут доказал, что

$A_{ср.}= n -$

размеров. Таким образом, метод обменной сортировки не обладает никакими достоинствами, за которые его можно было бы рекомендовать для практического использования, если не считать интересных теоретических задач, к которым он приводит при анализе сложности соответствующих алгоритмов.

Быстрые методы сортировки

Можно доказать, что среднее расстояние, на которое должен продвигаться каждый из n элементов массива размером nво время сортировки, равно $n/3$ позиций. Это число является целью в поиске улучшений, т.е. в поиске более эффективных методов сортировки. Все прямые методы сортировки фактически передвигают каждый элемент массива на каждом элементарном шаге на 1 позицию. Поэтому они требуют $O(n^2)$ таких шагов. Следовательно, в основу любых улучшений должен быть положен принцип перемещения элементов на каждом шаге сортировки на большие расстояния. Такой подход позволяет существенно уменьшить время сортировки, но соответствующие алгоритмы являются более сложными для понимания по сравнению с алгоритмами прямых методов. Для быстрых методов нет простых и точных формул для подсчета эффективности того или иного алгоритма. Существующие формулы дают лишь грубую меру для производительности алгоритма как функции от размера сортируемого массива, поэтому для практических целей полезно иметь некоторые экспериментальные данные для конкретных алгоритмов и конкретных ЭВМ.

1.3. *Провести анализ сложности сортировки массивов методом Шелла*.

Решение. В 1959 г. Д. Шеллом было предложено усовершенствование сортировки с помощью прямого включения. Метод Шелла, называемый также слиянием с обменом или сортировкой с помощью включений с уменьшающимися расстояниями, состоит в том, что выбирается некоторый интервал h между сравниваемыми элементами, который уменьшается после каждого просмотра массива. При этом для последовательности интервалов должны выполняться условия:

$h_1=1, \quad h_i<h_{i+1}, \quad i=1, 2,..., p-1,$

где p – количество интервалов (шагов).

Таким образом, вначале сравниваются (и, если необходимо, меняются местами) далеко стоящие элементы массива, а при последнем просмотре – соседние элементы. Выигрыш получается за счет того, что на каждом шаге либо участвует сравнительно небольшое число элементов, либо эти элементы уже довольно хорошо упорядочены, и требуется небольшое количество обменов. Последний просмотр по методу Шелла выполняется по тому же алгоритму, что и при сортировке с помощью прямого включения. Предыдущие просмотры подобны просмотрам по методу прямого включения, но в них сравниваются не соседние элементы, а элементы, отстоящие друг от друга на заданное расстояние h_i. Большой шаг на начальных этапах сортировки позволяет уменьшить число вторичных сравнений на последующих этапах.

Для иллюстрации отсортируем методом Шелла ранее рассмотренный массив с последовательностью интервалов $h_4=7$, $h_3=3$, $h_2=2$, $h_1=1$ (вертикальной чертой отделены соответствующие шагам интервалы).

Исходный массив: 22 -14 13 28 6 41 -5 18 0 35 6 -15

1-ый шаг ($h_4=7$): | 18 -14 13 6 -15 41 -5 | 22 0 35 28 6 |

2-ой шаг ($h_3=3$): | -5 -15 0 | 6 -14 6 | 18 22 13 | 35 28 41 |

3-ий шаг ($h_2=2$): | -14 -15 | -5 6 | 0 6 | 13 22 | 18 35 | 28 41 |

4-ый шаг($h_1=1$): -15 -14 -5 0 6 6 13 18 22 28 35 41

На 1-ом шаге в исходном массиве выделяется и сортируется 5 частей по 2 элемента каждая: 22 18, -14 0, 13 35, 28 6 и 6 –15. Элементы 41 и -5 остаются на своих местах. На

втором шаге в массиве, отсортированном на первом шаге, выделяется три части по 4 элемента каждая: 18 6 -5 35, -14 –15 22 28, 13 41 0 6. После сортировки каждой из этих трех частей мы выделяем в полученном массиве две части по 6 элементов: -14 –5 0 13 18 28 и -15 6 6 22 35 41. После сортировки этих частей переходим к последнему шагу, на котором остается поменять местами пары соседних элементов: -14 и –15, 6 и 0, 22 и 18, 35 и 28.

Последовательность интервалов можно уменьшать по-разному, лишь бы последний был единичным, ведь в самом плохом случае последний шаг доведет сортировку до конца. Чтобы обеспечить выполнение условия окончания поиска места для включения элемента, необходимо на каждом шаге сортировки установить свой барьер, поэтому приходится расширять исходный массив на h_p компонентов влево.

Для того чтобы выбрать хорошую последовательность интервалов сортировки методом Шелла, необходимо проанализировать время работы алгоритма сортировки как функцию от этих интервалов. Это приводит к ряду трудных, но красивых и интересных математических проблем, многие из которых не решены до настоящего времени. В частности, до сих пор не удалось найти возможную наилучшую последовательность расстояний для больших размеров массивов. Тем не менее, известно довольно много интересных фактов поведения сортировки методом Шелла. Приведем некоторые из них, в частности, проанализируем *двухпроходную*сортировку (т.е. когда p=2 и шаги сортировки $h_2 = h$ и h_1=1).

На время выполнения алгоритма сортировки влияют пять факторов: размер сортируемого массива n, число интервалов (шагов) p, сумма интервалов $s = h_1+h_2+\ldots+h_p$, число сравнений элементов массива C и число обменов B. При этом число B является основным фактором, от которого зависит время работы алгоритма сортировки, и равно числу инверсий в сортируемых частях массива на каждом шаге.

Назовем процесс сортировки массива на k-ом шаге h_k*–сортировкой*. Тогда сортировка методом Шелла состоит из h_p-сортировки, за которой следует h_{p-1}-сортировка,…, за которой следует h_1-сортировка. Массив, в котором $a[i] \le a[i+h_k]$ при $1 \le I \le n- h_k$, будем называть h_k*-упорядоченным*. В простейшем случае, когда имеется всего два шага h_2=2 и h_1=1, среднее число инверсий в случайной 2-упорядоченной перестановке равно $\lfloor n/2 \rfloor \cdot 2^{n-2}/C_n^{\lfloor n/2 \rfloor}$, где символом $\lfloor x \rfloor$ обозначено наибольшее целое число, не превосходящее x («пол» от x), в отличие от символа $\lceil x \rceil$, обозначающего наименьшее целое число, большее или равное x («потолок» от x). Применяя к выражению для вычисления среднего числа инверсий формулу Стирлинга

В 1967 г. Д.Хант, рассматривая общий случай двухпроходной сортировки методом Шелла, когда шаги сортировки равны h и 1, доказал, что среднее число инверсий в h-упорядоченной перестановке множества $\{1, 2,\ldots, n\}$ можно вычислить по формуле

$$I(n,h) = \frac{2^{2q+1} q!q!}{(2q+1)!}(C_h^2 q(q+1) + C_r^2 (q+1) - \frac{1}{2} C_{h-r}^2 q),$$

где $q = \lfloor n/h \rfloor$, $r \equiv n \bmod h$.

Идея этого доказательства состоит в том, что в h-упорядоченной перестановке содержится r упорядоченных частей массива длиной q+1 и h-r частей длины q. Каждая инверсия образуется из элементов двух различных частей, а каждая пара различных упорядоченных частей в случайной h-упорядоченной перестановке определяет случайную 2-упорядоченную перестановку. Поэтому среднее число инверсий равно сумме средних значений числа инверсий во всех парах различных частей массива. Проведя тождественные преобразования, мы и получим вышеприведенную формулу.

В первом приближении функция

числа Фибоначчи: 1, 2, 3, 5, 8, 13, …

В случае 2) время работы алгоритма имеет порядок $O(n^{3/2})$. Это является *верхнейоценкой* максимального времени работы алгоритма. На основании многочисленных эмпирических тестов были выявлены формулы для вычисления среднего числа обменов при $100 \le n \le 60000$ для различных последовательностей шагов.

1.4.*Провести анализ сложности сортировки массивовс помощью дерева*.

Решение. Метод сортировки с помощью прямого выбора основан, как мы видели, на повторяющихся поисках наименьшего элемента среди n элементов массива; далее – наименьшего среди оставшихся n-1 элементов и т.д. Всего, как было показано в задаче 1.1, требуется $C=1/2(n^2-n)$ сравнений. Усовершенствовать рассматриваемый метод сортировки можно, оставляя после каждого прохода больше информации, чем просто фиксирование единственного минимального элемента. Одним из таких усовершенствований является использование двоичных (бинарных) деревьев, которые обычно определяются рекурсивно, а именно: *это либо пустое множество, либо конфигурация, в которой вверху находится один узел, называемый* ***корнем****, из которого выходят дуги в два других узла*; *из каждого из этих узлов выходят дуги в два других и т.д*.Каждый из узлов, кроме корня, называется *вершиной* дерева. Говорят, что дуги ведут «*от отцов к сыновьям*»» или «*от предков кпотомкам*». У каждой вершины, кроме корня, ровно два потомка и один предок.

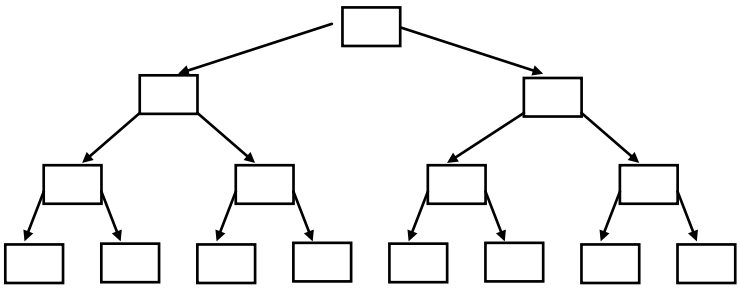


Рисунок 11.1 - Полное двоичное дерево

Мы привели определение *полного* бинарного дерева, хотя можно рассматривать и неполные бинарные деревья, когда не у каждого узла обязательно два потомка.На рисунке 1 приведена схема полного двоичного дерева, состоящего из 15 узлов.

Предположим для простоты, что количество подлежащих сортировке элементов есть степень двойки (хотя это и не обязательно). Запишем все элементы сортируемого по возрастанию массива в самый нижний ряд дерева, а остальную часть дерева заполним по правилу: элемент в звене высшего уровня – это меньший из элементов в вершинах-потомках. Тем самым в корень дерева после n–1 сравнений будет записан наименьший элемент массива. Пусть, например, задан целочисленный массив, состоящий из 16 элементов: 43, 12, 24, 2, 48,16, 8, 36, 6,19, 40, 28, 30, 7, 26, 5. Тогда соответствующее бинарное дерево схематично можно представить в виде следующего рисунка 2.

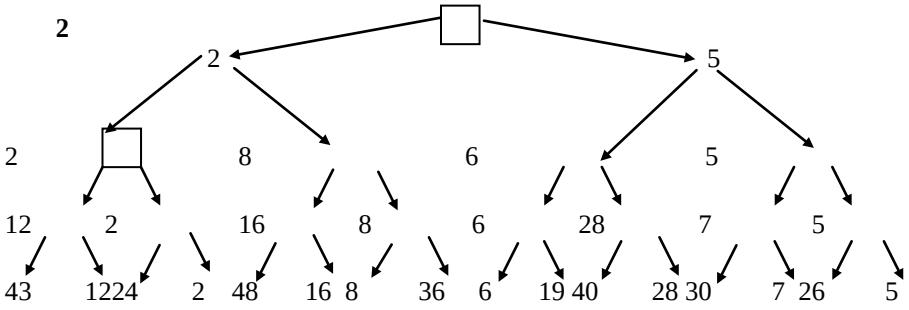


Рисунок 11.2 - Выбор наименьшего элемента

Изымем из сортируемого массива минимальный элемент (в нашем случае – число 2). Это можно сделать, идя от корня: от предка переходим к тому потомку, где записан тот же элемент. Изъяв минимальный элемент, заменим его символом «$*$» (иногда используют символ «∞»). В нашем примере после изъятия числа 2 получим дерево, изображенное на рисунке 3.

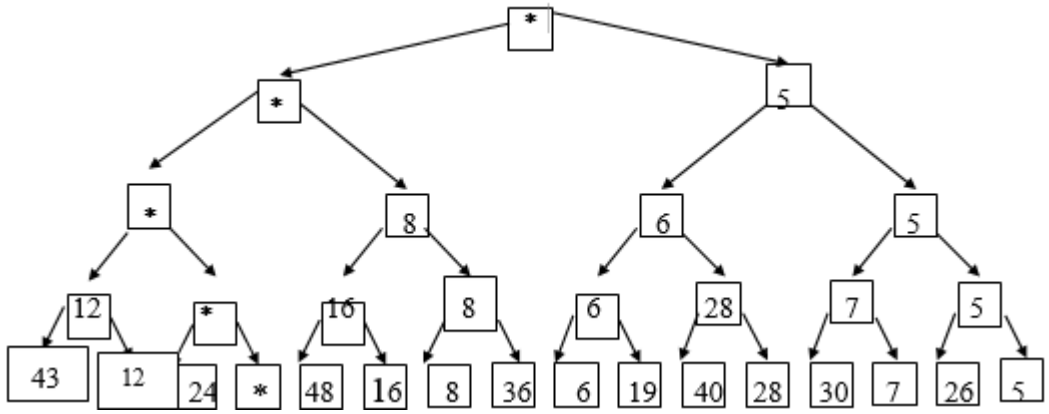


Рисунок 11.3 - Исключение наименьшего элемента

После этого скорректируем более низкие уровни. Для этого снова надо пройти путь к корню, полагая $\min(t, *) = t$. Тогда в корне дерева появится второй по величине элемент (в нашем примере – число 5). На втором этапе получим следующий рисунок:

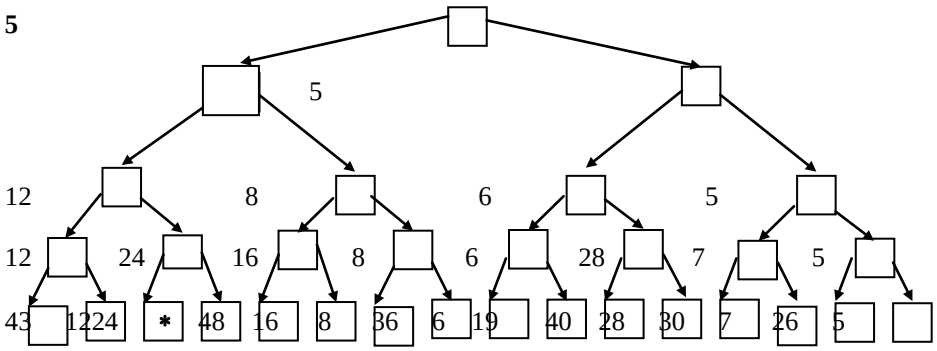


Рисунок 11.4 - Заполнение «дырок»

Затем изымаем число 5 и корректируем более низкие уровни. Получим последовательно числа 5, 6, 7, 8, …, 48 и после 16 таких шагов дерево окажется пустым.

В общем же случае для каждого из n элементов массива требуется произвести $\lceil \log_2 n \rceil$ сравнений, поэтому на весь процесс понадобится порядка $n\log_2 n$ элементарных операций плюс еще n шагов на построение дерева. Это весьма существенное улучшение не только метода прямого выбора, требующего порядка n^2 шагов, но и даже метода Шелла, где нужно$O(n^{1.667})$ шагов. Естественно, что из-за сохранения избыточной информации при сортировке массива посредством выбора из древообразной структуры каждый отдельный шаг усложняется. Поэтому наша следующая задача – найти более эффективные приемы организации этой информации. Прежде всего, необходимо избавиться от «дырок», которыми в конечном итоге будет заполнено все дерево, и которые порождают много ненужных сравнений. Кроме того, надо найти такое представление дерева из n элементов, которое требует лишь n ячеек памяти, а не $2n - 1$, как предлагалось выше. Заметим, что для того, чтобы знать, куда вставлять следующий символ «$*$», необходимо запомнить, откуда пришел элемент, находящийся в корне дерева. Поэтому вершины разветвления дерева в действительности содержат индексы-указатели, описывающие позицию элемента,

а не сам элемент. Отсюда следует, что необходима память для n исходных элементов массива, $n-1$ индексов-указателей и n выводимых элементов.

До сих пор в наших примерах выбора из дерева предполагалось, что размер массива n есть степень двойки. В действительности можно работать с произвольным значением n, так как полное бинарное дерево с n концевыми узлами не трудно построить для любого n, добавляя при необходимости пустые вершины.

Заканчивая наш обзор основных методов внутренней сортировки, мы попытаемся сравнить их эффективность. Как и ранее, n будет означать размер сортируемого массива, C и M соответственно число необходимых сравнений элементов (ключей) массива и число обменов. Для всех прямых методов сортировки можно дать, как мы видели, точные аналитические формулы. Они приводятся в таблице 1, в которой столбцы **min**, **avg** и **max**определяют соответственно минимальное, усредненное и максимальное по всем $n!$ перестановкам из n элементов значения.

Для быстрых методов сортировки нет простых и точных формул для вычисления числа сравнений, обменов и времени выполнения алгоритма. Рассуждения и выкладки, проведенные в разделе 4, вводят лишь грубую меру для производительности алгоритмов как функции от размера сортируемого массива. Для практических целей полезно иметь некоторые эмпирические данные, которые несут информацию о продолжительности того или иного метода сортировки, тем более, что формулы не учитывают других затрат на вычисления, кроме сравнений ключей и обменов элементов массива.

Таблица 11.1 - Сравнение прямых методов сортировки

Метод		min	avg	max
Прямые вставки	C	$n-1$	$(n^2+n-2)/4$	$(n^2-n)/2$
	M	$3(n-1)$	$(n^2+9n-10)/4$	$(n^2+3n-4)/2$
Прямой выбор	C	$(n^2-n)/2$	$(n^2-n)/2$	$(n^2-n)/2$
	M	0	$n(\ln n+0.57)$	$n^2/4+3(n-1)$
Прямой обмен	C	$n-1$	$3(n^2-n)/4$	$(n^2-n)/2$
	M	0	$0.75(n^2-n)$	$1.5(n^2-n)$

Ясно, что эти факторы во многом зависят от типа компьютера, тем не менее, результаты экспериментов достаточно информативны. Ниже в таблице 2 собраны экспериментальные данные о времени работы в секундах рассмотренных нами прямых и быстрых методов внутренней сортировки. Соответствующие алгоритмы реализованы в среде программирования Турбо Паскаль 7.0 на домашнем персональном компьютере типа Pentium с тактовой частотой 100 Мгц. Элементы всех целочисленных сортируемых массивов сформированы датчиком псевдослучайных чисел в диапазоне от –5000 до 5000. Для каждого размера массива проводилось 3-4 испытания и вычислялось среднее время сортировки, наихудшие и наилучшие случаи не рассматривались. Прочерки в некоторых клетках таблицы 2 указывают на нехватку оперативной памяти компьютера. Данные приведены с точностью до сотых долей секунды.

Таблица 11.2 - Время работы программ различных методов сортировки

Размер сортируемого массива / Методы сортировки	1000	5000	10000	15000	20000	25000	30000
Метод прямых вставок	0.5	2.80	8.67	21.20	39.16	62.39	91.13
Метод прямого обмена	0.16	4.33	18.23	43.88	81.61	130.27	193.26
Метод Шелла	0.00	0.05	0.10	0.24	0.40	0.60	0.87
Метод бинарного дерева	0.00	0.05	0.16	0.27	0.38	-	-
Метод Хоара	0.00	0.00	0.05	0.05	0.10	0.10	0.16
Метод слияния (все стадии)	3.46	22.35	48.16	-	-	-	-

1.5. Найти количество различных чисел среди элементов данного массива. Число действий должно иметь порядок nlogn.

Указание. Отсортировать числа, а затем посчитать количество различных чисел, просматривая элементы массива по порядку.

1.6. Дано n отрезков [a[i], b[i]] на прямой (i = 1..n). Найти максимальное число k, для которого существует точка прямой, покрытая k отрезками («максимальное число слоев»). Число действий – nlogn.

Указание. Упорядочим все левые и правые концы отрезков вместе (при этом левый конец считается меньше правого конца, расположенного в той же точке прямой). Далее сдвигаемся слева направо, считая число слоев. Встреченный левый конец увеличивает число слоев на 1, правый – уменьшает. Отметим, что примыкающие друг к другу отрезки обрабатываются правильно: сначала идет левый конец (правого отрезка), а затем – правый (левого отрезка).

1.7. Дано n точек на плоскости. Построить их выпуклую оболочку, то есть минимальную выпуклую фигуру, их содержащую (образно говоря, построить колечко, натянутую на вбитые в доску гвозди). Число операций должно иметь порядок O(nlogn).

Указание. Упорядочим точки по координате x, а при равных координатах x – по координате y. Затем, рассматривая точки по очереди, будем строить выпуклую оболочку.

1.8. Пусть имеется n различных по весу камней и весы, которые позволяют за одно взвешивание определить, какой из двух выбранных нами камней тяжелее. Надо упорядочить камни по весу, сделав как можно меньше взвешиваний. Разумеется, число взвешиваний зависит не только от выбранного нами алгоритма, но и от того, как оказались расположены камни. Сложностью алгоритма назовем число взвешиваний при наихудшем расположении камней. Доказать, что сложность любого алгоритма сортировки n камней не меньше logn!.

Решение. Пусть имеется алгоритм сложности не более d. Для каждого из n! возможных расположений камней запротоколируем результаты взвешиваний; их можно записать в виде последовательности из не более чем d нулей и единиц. Для единообразия дополним последовательность нулями, чтобы ее длина стала равной d. Тем самым у нас имеется n! последовательностей из d нулей и единиц. Все эти последовательности разные, иначе наш алгоритм дал бы одинаковые ответы для разных порядков (и один из ответов был бы неправильным). Получаем

Это рассуждение показывает, что любой алгоритм сортировки, использующий только сравнения элементов массива и их перестановки, требует не менее Cnlogn действий, так что наши алгоритмы близки к оптимальным. Однако алгоритм сортировки, использующий другие операции, может действовать быстрее.

2. Нижние оценки числа арифметических операций в матричной алгебре

2.1. Показать, что обычный алгоритм умножения матриц требует $O(n^3)$ шагов.

Решение. Если почти все элементы матрицы-произведения равны 1, то можно перемножить две булевы матрицы в среднем за время $O(n^2)$, вычисляя каждую сумму

Число операций подсчитывается тривиально. Доказательство того, что в результате получаются требуемые величины c_{ij}, представляет собой простое алгебраическое упражнение на использование аксиом кольца.

2.4. Рассмотрим еще один алгоритм Штрассена умножения двух квадратных матриц второго порядка:

3.3. *Найти наименьшую стоимость проезда из города* i *в город* j *для всех* i *за время* $O(n^3)$.

Решение. Для

Учебно-методическое дисциплины «Теория алгоритмов»

Перечень основной литературы:

1. Брыкалова А.А. Теория алгоритмов [Электронный ресурс]: лабораторный практикум / А.А. Брыкалова. - Электрон. текстовые данные. - Ставрополь: Северо-Кавказский федеральный университет, 2016. - 134 с. - 2227-8397. - Режим доступа: http://www.iprbookshop.ru/69439.html

2. Макоха А.Н. Математическая логика и теория алгоритмов [Электронный ресурс]: учебное пособие / А.Н. Макоха, А.В. Шапошников, В.В. Бережной. - Электрон. текстовые данные. - Ставрополь: Северо-Кавказский федеральный университет, 2017. - 418 с. - 2227-8397. - Режим доступа: http://www.iprbookshop.ru/69397.html

Перечень дополнительной литературы:

1. Теория алгоритмов: учебное пособие / Министерство образования и науки РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»; сост. А.А. Брыкалова. - Ставрополь: СКФУ, 2016. - 129 с.: ил. - Библиогр. в кн.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=467402

2. Перемитина Т.О. Математическая логика и теория алгоритмов: учебное пособие / Т.О. Перемитина; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск: ТУСУР, 2016. - 132 с.: ил. - Библиогр.: с.130.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=480886

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

http://www.allmath.ru/ Вся математика – высшая математика, прикладная математика, математические методы в экономике, финансовая математика.

http://www.exponenta.ru/ Образовательный математический сайт.

http://www.math.ru/ Ресурс содержит книги, видео-лекции, занимательные математические факты, различные по уровню и тематике задачи, отдельные истории из жизни ученых и др.

http://www.mathedu.ru/ Электронная библиотека по математике и вопросам ее преподавания. Включает популярные книги и пособия, методические руководства, учебники, журналы, исторические работы, авторефераты, диафильмы.

http://www.Math-Net.ru/ Общероссийский математический портал представляет собой современную информационную систему, представляющую российским и зарубежным математикам различные возможности в поиске информации о математической жизни в России.

http://www.mathtest.ru/ Математика в помощь школьнику и студенту (тесты по математике on-line).

http://www.mccme.ru/ Московский центр непрерывного математического образования.

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

Методические указания
по организации и проведению самостоятельной работы
по дисциплине
«ТЕОРИЯ АЛГОРИТМОВ»
для студентов направления подготовки
02.03.01 Математика и компьютерные науки
направленность (профиль)
«Анализ данных и искусственный интеллект»

Общие положения

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 (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

– формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);

– написание докладов;

– подготовка к семинарам, практическим и лабораторным работам, их оформление;

– составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

– выполнение учебно-исследовательских работ, проектная деятельность;

– подготовка практических разработок и рекомендаций по решению проблемной ситуации;

– 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;

– выполнение курсовых работ (проектов) в рамках дисциплин;

– выполнение выпускной квалификационной работы и др.

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).

Самостоятельная работа по дисциплине **«Теория алгоритмов»** направлена на формирование следующих **компетенций**:

ПК-2. Способен преподавать математику и информатику в образовательных организациях общего образования и среднего профессионального образования на основе полученного фундаментального образования и научного мировоззрения

1. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора профессиональных компетенций будущего бакалавра по направлению подготовки «Математика и компьютерные науки».

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.

2. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
4 семестр					
ПК-2	Самостоятельное изучение литературы	Собеседование Защита реферата	34	2	36
ПК-2	Решение практических задач	Конспект	34	2	36
		Итого за 4 семестр	**68**	**4**	**72**
		Итого	**68**	**4**	**72**

3. Порядок выполнения самостоятельной работы студентом

3.1. Методические рекомендации по работе с учебной литературой

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.

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***:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

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).

Основные виды систематизированной записи прочитанного:

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

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

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

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

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.

3.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные 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.

3.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

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.

3.4. Методические рекомендации по написанию научных текстов (докладов, рефератов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т, п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Реферат (доклад) - это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Реферат не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в реферате должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки реферата студентом.

Выполнение реферата начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания реферата. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста реферата предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки реферат сдается на кафедру для его оценивания руководителем.

Требования к написанию реферата

Написание 1 реферата является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема реферата может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Реферат должен быть написан научным языком.

Объем реферата должен составлять 20-25 стр.

Структура реферата:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.
- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к

раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.

- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса
- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.
- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;
- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- реферат должен быть представлен в сброшюрованном виде.

Порядок защиты реферата:

Защита реферата проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту реферата отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите реферата приветствуется использование мультимедиа-презентации.

Оценка реферата

Реферат оценивается по следующим критериям:

• соблюдение требований к его оформлению;

• необходимость и достаточность для раскрытия темы приведенной в тексте реферата информации;

• умение студента свободно излагать основные идеи, отраженные в реферате;

• способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

Описание шкалы оценивания

Максимально возможный балл за весь текущий контроль устанавливается равным 55. Текущее контрольное мероприятие считается сданным, если студент получил за него не менее 60% от установленного для этого контроля максимального балла. Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

Темы рефератов

1. Способы описания алгоритмов. Примеры алгоритмов. Алгоритм Евклида.
2. Неформальное понятие алгоритма.
3. Характеристики алгоритмов. Задачи построения “хороших” алгоритмов. Алгоритмы вычисления значений полиномов. Схема Горнера.
4. Алгоритмы поиска максимального и минимального элементов в массиве.
5. Сортировка массивов. Анализ и сравнение алгоритмов.
6. Машины Тьюринга. Устройство машины Тьюринга. Работа машины Тьюринга. Команды машины Тьюринга. Программа машины Тьюринга. Конфигурации. Пример машины Тьюринга.
7. Вычислимые по Тьюрингу функции. Примеры: $f(x) = x + 1$ в кодах и десятичной системе исчисления, $f(x) = 2$ в кодах. Основная гипотеза теории алгоритмов.
8. Рекурсивные функции. Простейшие функции. Суперпозиция функций. Схема примитивной рекурсии. Примеры частично рекурсивных функций: $f(x) = x + 2$, $f(x) = 2$, $f(x, y) = x + y$.
9. Операция минимизации. Частично рекурсивные функции. Примеры частично рекурсивных функций: $f(x) = x - 1$, $f(x, y) = xy$, $f(x, y) = \max(x - y, 0)$, $f(x, y) = x - y$. Соотношение классов интуитивно вычислимых и частично рекурсивных функций.
10. Алгоритмы преобразования слов. Нормальные алгоритмы Маркова. Подстановки. Схема алгоритма. Выполнение нормального алгоритма. Примеры нормальных алгоритмов.
11. Нормальный алгоритм, приписывающий символ к заданному слову справа. Эквивалентность алгоритмов преобразования слов. Принцип нормализации алгоритмов.
1. Нормально вычислимые функции. Примеры нормально вычислимых функций в кодах: $f(x) = x + 1$, $f(x, y) = x - y$, $f(x) = 2x$, остаток и частное при делении на 3. Теорема (о совпадении трёх классов функций).
12. Неразрешимые алгоритмические проблемы. Коды символов. Шифры команд. Шифры программ. Шифры конфигураций.
13. Теорема о существовании функций невычислимых по Тьюрингу.
14. Примеры неразрешимых алгоритмических проблем. Теоремы о распознавании применимости и самоприменимости машин Тьюринга.
15. Определение и свойства разрешимых и перечислимых множеств. Теорема Поста.

Список литературы для выполнения СРС

Перечень основной литературы:

1. Брыкалова А.А. Теория алгоритмов [Электронный ресурс]: лабораторный практикум / А.А. Брыкалова. - Электрон. текстовые данные. - Ставрополь: Северо-Кавказский федеральный университет, 2016. - 134 с. - 2227-8397. - Режим доступа: http://www.iprbookshop.ru/69439.html

2. Макоха А.Н. Математическая логика и теория алгоритмов [Электронный ресурс]: учебное пособие / А.Н. Макоха, А.В. Шапошников, В.В. Бережной. - Электрон. текстовые данные. - Ставрополь: Северо-Кавказский федеральный университет, 2017. - 418 с. - 2227-8397. - Режим доступа: http://www.iprbookshop.ru/69397.html

Перечень дополнительной литературы:

1. Теория алгоритмов: учебное пособие / Министерство образования и науки РФ, Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет»; сост. А.А. Брыкалова. - Ставрополь: СКФУ, 2016. - 129 с.: ил. - Библиогр. в кн.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=467402

2. Перемитина Т.О. Математическая логика и теория алгоритмов: учебное пособие / Т.О. Перемитина; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск: ТУСУР, 2016. - 132 с.: ил. - Библиогр.: с.130.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=480886

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

http://www.allmath.ru/ Вся математика – высшая математика, прикладная математика, математические методы в экономике, финансовая математика.

http://www.exponenta.ru/ Образовательный математический сайт.

http://www.math.ru/ Ресурс содержит книги, видео-лекции, занимательные математические факты, различные по уровню и тематике задачи, отдельные истории из жизни ученых и др.

http://www.mathedu.ru/ Электронная библиотека по математике и вопросам ее преподавания. Включает популярные книги и пособия, методические руководства, учебники, журналы, исторические работы, авторефераты, диафильмы.

http://www.Math-Net.ru/ Общероссийский математический портал представляет собой современную информационную систему, представляющую российским и зарубежным математикам различные возможности в поиске информации о математической жизни в России.

http://www.mathtest.ru/ Математика в помощь школьнику и студенту (тесты по математике on-line).

http://www.mccme.ru/ Московский центр непрерывного математического образования.