Автономная некоммерческая организация среднего профессионального образования

«УРАЛЬСКИЙ ПРОМЫШЛЕННО-ЭКОНОМИЧЕСКИЙ ТЕХНИКУМ»

\

**Методические указания**

**к практическим занятиям**

по дисциплине

**Теория алгоритмов**

**Укрупненная группа: 09.00.00 И**нформатика и вычислительная техника

**Специальность:** 09.02.03 Программирование в компьютерных системах

2016

|  |  |
| --- | --- |
| Одобрена цикловой комиссией  информатики и вычислительной техники  Председатель комиссии  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Г. Максимова  Протокол №  от « » \_\_\_\_\_\_\_\_\_\_\_\_ 201\_\_г. | Методические указания разработаны на основе Федерального государственного образовательного стандарта по специальности среднего профессионального образования, входящей в состав укрупненной группы специальностей 09.00.00 «Информатика и вычислительная техника» 09.02.03 «Программирование в компьютерных системах»  *УТВЕРЖДАЮ*  Заместитель директора по  учебной работе АН ПОО «Уральский промышленно-экономический техникум»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Б. Чмель  « » \_\_\_\_\_\_\_\_\_\_\_\_ 201 \_\_ г. |

Разработчик: **Максимова О.Г.** преподаватель дисциплины

*«Теория алгоритмов»* АН ПОО «Уральский промышленно-экономический техникум»

Содержание

[Пояснительная записка 4](#_Toc480026116)

[Практическое занятие № 1 Реализация алгоритмов в машине Тьюринга 6](#_Toc480026117)

[Практическое занятие № 2 Рекурсивные функции 22](#_Toc480026118)

[Практическое занятие № 3 Нормальные алгоритмы 30](#_Toc480026119)

[Практическое занятие № 4 Применение метода последовательной детализации 31](#_Toc480026120)

[Практическое занятие № 5 Применение рекурсивных методов 34](#_Toc480026121)

[Практическое занятие № 6 Реализация методов перебора 39](#_Toc480026122)

[Практическое занятие № 7-8 Реализация методов сортировки данных 42](#_Toc480026123)

[Практическое занятие № 9-10 Определение сложности алгоритмов 67](#_Toc480026124)

[Практическое занятие № 11 Определение сложности рекурсивных алгоритмов 72](#_Toc480026125)

[Список рекомендуемой литературы 76](#_Toc480026126)

# Пояснительная записка

Рабочая программа учебной дисциплины «Теория алгоритмов» является частью основной профессиональной образовательной программы в соответствии с ФГОС по специальности среднего профессионального образования 09.02.03 «Программирование в компьютерных системах», входящей в состав укрупненной группы специальностей «Информатика и вычислительная техника»

Дисциплина «Теория алгоритмов» в составе общепрофессиональных дисциплин входит в профессиональный цикл.

В результате освоения обязательной части дисциплины обучающийся должен уметь:

* разрабатывать алгоритмы для конкретных задач;
* определять сложность работы алгоритмов.

В результате освоения обязательной части дисциплины обучающийся должен знать:

* основные модели алгоритмов;
* методы построения алгоритмов;
* методы вычисления сложности работы алгоритмов.

Содержание дисциплины должно быть ориентировано на подготовку обучающихся по базовой и углубленной подготовке к освоению профессиональных модулей ОПОП по специальности Программирование в компьютерных системах и овладению профессиональными компетенциями (ПК):

ПК 1.1. Выполнять разработку спецификаций отдельных компонент.

ПК 1.2. Осуществлять разработку кода программного продукта на основе готовых спецификаций на уровне модуля.

В результате освоения дисциплины у обучающихся по базовой подготовке формируются общие компетенции (ОК):

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.

ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения заданий.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

ОК 10. Исполнять воинскую обязанность, в том числе с применением полученных профессиональных знаний (для юношей).

Рабочей программой учебной дисциплины предусмотрено проведение 22 часов практических занятий.

# Практическое занятие № 1 Реализация алгоритмов в машине Тьюринга

**Цель занятия**: Научиться реализовывать алгоритмы с помощью машины Тьюринга

**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

На протяжении многих десятилетий ученые давали различные определения понятия «алгоритм». Существуют как точные (формальные) определения, так и интуитивные (неформальные). Но, изучая любое из этих определений, можно сделать вывод, что для выполнения алгоритма требуются некие исходные данные, набор элементарных команд, который будет оперировать с этими данными, и исполнитель алгоритма.

Чтобы дать точное определение алгоритма, необходимо определить, как задаются исходные данные и как задаются элементарные шаги. С точки зрения математики любой объект может быть описан некоторым набором фраз на определенном языке. Иначе говоря, он может быть закодирован цепочкой символов. Если объектом является число, то его можно представить в виде цепочки символов алфавита {0,1}. Если объектом является текст, то его можно представить цепочкой символов алфавита, содержащего буквы, цифры, знаки препинания. Если это графическое изображение, то его можно представить массивом пикселей, каждый их которых представляется степенями интенсивности трех основных цветов.

Таким образом, можно сказать, что алгоритм – это преобразование слов из заданного алфавита в другие слова. С этой точки зрения алгоритмы рассматривались такими учеными как Тьюринг, Пост, Марков.

Рассмотрим способ задания алгоритмов, предложенный американским ученым Аланом Тьюрингом в 1937 году. Тьюринг предложил для описания алгоритмов использовать некоторый математический аппарат, названный машиной, который содержит две основных части:

неограниченную в обе стороны ленту, разделенную на ячейки, в каждой их которых может быть записан один символ алфавита или ничего не быть записано.

автомат (головку для считывания/записи информации с ленты).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | S1 | S2 | S3 | S4 | S5 | … | … |

M

Автомат перемещается вдоль ленты и в каждый момент времени может обозревать и читать и изменять содержимое только одной ячейки. Автомат перемещается вдоль ленты в соответствии с некоторым алгоритмом.

С каждой машиной Тьюринга связаны два конечных алфавита: алфавит входных символов **A={a1,a2,a3,…}** и алфавит состояний **Q={q1,q2,q3,…}**. Таким образом, в каждый момент времени машина Тьюринга должна находиться в одном из возможных состояний Q. С разными машинами Тьюринга могут быть связаны разные алфавиты. Какая именно команда программы будет выполняться в данный момент времени, зависит от читаемого головкой символа и состоянием машины. Результатом выполнения команды являются: новый символ, записанный в текущую ячейку, перемещение головки на одну позицию влево или вправо и новое состояние. В некоторых случаях новый символ может быть равен старому, а перемещение может отсутствовать.

Таким образом, формат команды имеет вид: **a q b r D**. Здесь a – читаемый символ, q – текущее состояние, b – новый символ, r – новое состояние, D – направление движения. Будем обозначать буквой *e* пустую ячейку. Значения D выбираются из множества **{L,R,S}**, где L – движение влево, R – движение вправо и S – отсутствие движения.

Таким образом, команда **1 q3 0 q6 L** означает: «если находясь в состоянии q3, машина Тьюринга обозревает ячейку, в которой записано число 1, то необходимо записать в эту ячейку число 0, произвести сдвиг головки влево и перейти в состояние q6.

Стартовая конфигурация: на ленте находятся исходные данные – строка символов в алфавите A. Машина находится в начальном состоянии q1. Головка машины обозревает некоторую ячейку ленты с записанным там символом a. В момент старта выполняется первая команда. Затем машина переходит в новое состояние, читает новый символ и т.д. Машина прекращает функционирование, как только для некоторой пары (a,q) не окажется команды в программе. Такая ситуация считаются завершающей. Оставшаяся запись на ленте считается записью результата. Иногда для обозначения завершения программы используется переход к завершающему состоянию q0.

Таким образом, машина Тьюринга реализует вычисление некоторой функции – отображения исходной строки символов в результирующую строку.

Существует несколько способов представления программы машины Тьюринга. Наиболее употребительны два из них:

1. двумерная таблица,
2. диаграмма.

В двумерной таблице строки помечаются различными символами алфавита, а столбцы – именами различных состояний машины. Каждой команде соответствует единственная клетка в таблице. Для команды a q b r D она определяется следующим образом: в клетку, находящуюся на пересечении строки с символом a и столбца с состоянием q записывается тройка b r D.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q1 | … | q | … | qk |
| a1 |  |  |  |  |  |
| … |  |  |  |  |  |
| a |  |  | b r D |  |  |
| … |  |  |  |  |  |
| am |  |  |  |  |  |

Для некоторых пар (a,q) в программе может не быть команд, то есть соответствующие им клетки остаются пустыми. При достижении в работе пустой клетки машина Тьюринга останавливается.

**Пример 1**. Построить машину Тьюринга, которая увеличивает заданное двоичное число на 1, то есть вычисляет функцию S(x)=x+1. В начальный момент времени головка располагается напротив левого крайнего символа числа.

Алфавит A= {e,0,1}. Разберем алгоритм работы.

1. Перевести головку машины к последнему символу числа.
2. Если текущая цифра числа 0, то заменить его на 1 и закончить выполнение алгоритма
3. Если текущая цифра числа 1, то заменить его на 0, перейти в соседнюю слева ячейку и вернуться к шагу 2.
4. Если при перемещении влево достигнута пустая ячейка, то занести в ней 1 и закончить выполнение алгоритма.

|  |  |  |  |
| --- | --- | --- | --- |
|  | q1 | q2 | q3 |
| 0 | 0q1R | 1q3S |  |
| 1 | 1q1R | 0q2L |  |
| e | eq2L | 1q3S |  |

Проверить работу алгоритма на числах 100, 101, 111.

**Пример 2**. Построить машину Тьюринга для вычисления функции Z(x)=0, то есть функции, которая превращает запись любого аргумента в запись нуля. В начальный момент времени головка располагается напротив левого крайнего символа числа.

|  |  |  |
| --- | --- | --- |
|  | q1 | q2 |
| 0 | eq1R |  |
| 1 | eq1R |  |
| e | 0q2S |  |

Преобразовать программу таким образом, чтобы 0 заносился на место последнего символа исходного числа.

**Пример 3**. На ленте машины Тьюринга содержится последовательность символов “+”. Написать программу, которая каждый второй символ “+” заменит на “-“. Замена начинается с правого конца последовательности. Автомат в состоянии q1 обозревает произвольный символ последовательности.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 |
| E | eq2L | eq4S | eq4S |  |
| + | +q1R | +q3L | -q2L |  |
| - |  |  |  |  |

Проверить на последовательностях ++++, +++

**Пример 4**. На ленте машины Тьюринга находится десятичное число. Определить, делится ли это число на 5 без остатка. Если делится, то записать справа от числа слово «да», иначе – «нет». Автомат обозревает произвольный символ последовательности.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 |
| E | eq2L |  | нq4R | еq5R | тq8S | дq7R | аq8S |  |
| 1 | 1q1R | 1q3R |  |  |  |  |  |  |
| 2 | 2q1R | 2q3R |  |  |  |  |  |  |
| 3 | 3q1R | 3q3R |  |  |  |  |  |  |
| 4 | 4q1R | 4q4R |  |  |  |  |  |  |
| 5 | 5q1R | 5q6R |  |  |  |  |  |  |
| 6 | 6q1R | 6q3R |  |  |  |  |  |  |
| 7 | 7q1R | 7q3R |  |  |  |  |  |  |
| 8 | 8q1R | 8q3R |  |  |  |  |  |  |
| 9 | 9q1R | 9q3R |  |  |  |  |  |  |
| 0 | 0q1R | 0q6R |  |  |  |  |  |  |
| н |  |  |  |  |  |  |  |  |
| е |  |  |  |  |  |  |  |  |
| т |  |  |  |  |  |  |  |  |
| д |  |  |  |  |  |  |  |  |
| а |  |  |  |  |  |  |  |  |

**Пример 5**. Даны два натуральных числа m и n в унарной системе счисления, использующей цифру “|”. Числа разделены одной пустой клеткой. Автомат в начальном состоянии обозревает самый правый символ последовательности. Разработать машину Тьюринга, которая на ленте оставит сумму чисел m и n. Самостоятельно!

|  |  |  |  |
| --- | --- | --- | --- |
|  | q1 | q2 | q3 |
| e |  | |q3S |  |
| | | eq2L | |q2L |  |

Другой способ представления программы машины Тьюринга – диаграмма.

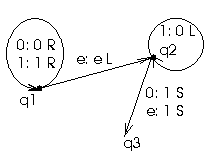
Диаграмма представляет собой геометрический объект, состоящий из вершин (обозначаемых точками или окружностями), и дуг (рисуемых в виде направленных отрезков прямой со стрелкой на одном из концов или в виде отрезков несамопересекающихся кривых). Каждой вершине приписывается состояние машины Тьюринга. Таким образом, вершин в диаграмме ровно столько, сколько имеется состояний. Дуге, соединяющей две вершины q1 и q2, приписывается некоторый символ *a* алфавита A и двойка *b D* так, что запись **a qi b qj D** образует команду программы машины Тьюринга.

**qi a: b D qj**

.

Дуга (стрелка) символизирует переход из состояния qi в состояние qj при условии, что головка читает символ *a*. Одновременно с этим символ *a* заменяется на символ *b* и совершается движение D. По этой причине диаграмму называют диаграммой (графом) переходов.

**Пример**. Изобразить диаграмму переходов для задачи 1.



**Композиции машин Тьюринга**

Пусть перед нами поставлена задача и мы сумели разбить её на части. При этом для решения каждой из частей задачи была построена машина Тьюринга. Встает вопрос: как организовать совместную работу этих машин для решения полной задачи? Для этих целей используются композиции машин Тьюринга.

Первая композиция: последовательное соединение машин.

Пусть даны две программы машины Тьюринга. Первая получает на ленте входные данные и начинает работу. После конечного числа шагов попытка выбрать из таблицы очередную команду заканчивается безрезультатно, так как соответствующая клетка таблицы пуста. Первая программа останавливается и на ленте остается какой-то результат. В этот момент начинает работать вторая программа, заданная другой таблицей и использующая результат работы первой программы в качестве исходных данных. Для организации такой работы достаточно построить объединенную таблицу машины Тьюринга, приписав в первой таблице вторую (справа) и заполнив пустые клетки первой таблицы командой i p1 S - командой перехода к первому (начальному) состоянию p1 второй программы. Здесь i – буква алфавита, соответствующая строке, в которой находится данная клетка.

Вторая композиция – итерация. В этом случае мы повторно выполняем одну и ту же программу конечно число раз. По окончании первого выполнения на ленте остается промежуточный результат, который является исходными данными для второго выполнения и т.д. Для обеспечения второго и последующего выполнений необходимо в некоторые пустые клетки вписать команду перехода на начало i q1 S. Во все пустые клетки её вписывать нельзя, так как измененная таким образом программа не сможет заканчиваться. Итерация машины Тьюринг соответствует циклам в языках программирования.

**Пример**. Построить композицию машин Тьюринга для вычисления функции *f(x)=2x+1*. В начальный момент времени головка располагается напротив левого крайнего символа числа.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 |  |  | p1 | p2 | p3 |
| E | eq2L | eq4S | 1q4L | ep1S |  | e | ep2L | 1p3S |  |
| 0 | 0q1R | 0q2L | 1q2L |  |  | 0 | 0p1R | 1p3S |  |
| 1 | 1q1R | 2q2L | 3q2L |  |  | 1 | 1p1R | 2p3S |  |
| 2 | 2q1R | 4q2L | 5q2L |  |  | 2 | 2p1R | 3p3S |  |
| 3 | 3q1R | 6q2L | 7q2L |  |  | 3 | 3p1R | 4p3S |  |
| 4 | 4q1R | 8q2L | 9q2L |  |  | 4 | 4p1R | 5p3S |  |
| 5 | 5q1R | 0q3L | 1q3L |  |  | 5 | 5p1R | 6p3S |  |
| 6 | 6q1R | 2q3L | 3q3L |  |  | 6 | 6p1R | 7p3S |  |
| 7 | 7q1R | 4q3L | 5q3L |  |  | 7 | 7p1R | 8p3S |  |
| 8 | 8q1R | 6q3L | 7q3L |  |  | 8 | 8p1R | 9p3S |  |
| 9 | 9q1R | 8q3L | 9q3L |  |  | 9 | 9p1R | 0p2L |  |

**Задача 1**. Построить машину Тьюринга (в виде таблицы), которая увеличивает двоичное число в два раза. В начальный момент времени головка располагается напротив левого крайнего символа числа. В конце работы головка так же должна располагаться напротив крайнего левого символа числа. Выполнить трассировку алгоритма для исходного значения 1101.

***Решение***

Алфавит A= { e , 0, 1 }

Поскольку умножение двоичного числа на два эквивалентно приписыванию справа от числа одного нуля, алгоритм работы будет следующий:

Сдвинуться вправо и встать на первый пустой символ справа от числа.

Заменить текущий пустой символ на 0.

Сдвинуться влево и встать на крайний левый символ числа.

|  |  |  |  |
| --- | --- | --- | --- |
|  | q1 | q2 | q3 |
| 0 | 0 q1R | 0 q2 L |  |
| 1 | 1 q1R | 1 q2 L |  |
| e | 0 q2 L | e q3 R |  |

Проведем трассировку (пошаговое выполнение) данного алгоритма для числа 1101. Название состояния, в котором мы находимся, будем писать сверху над тем символом, возле которого расположена читающая головка. В левом столбике будем указывать, какая команда была выполнена, а справа отражать состояние ленты машины Тьюринга после выполнения этой команды.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | в начале работы |  |  | q1 |  |  |  |  |  |  |
|  | … | e | 1 | 1 | 0 | 1 | e | e | … |
|  |  |  |  |  |  |  |  |  |  |  |
|  | 1 q1 R |  |  |  | q1 |  |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | e | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 1 q1 R |  |  |  |  | q1 |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | e | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 0 q1 R |  |  |  |  |  | q1 |  |  |  |
| … | e | 1 | 1 | 0 | 1 | e | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 1 q1 R |  |  |  |  |  |  | q1 |  |  |
| … | e | 1 | 1 | 0 | 1 | e | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 0 q2 L |  |  |  |  |  | q2 |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 1 q2 L |  |  |  |  | q2 |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 0 q2 L |  |  |  | q2 |  |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 1 q2 L |  |  | q2 |  |  |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |
| 1 q2 L |  | q2 |  |  |  |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |
| е q3 R |  |  | q3 |  |  |  |  |  |  |
| … | e | 1 | 1 | 0 | 1 | 0 | e | … |
|  |  |  |  |  |  |  |  |  |  |  |

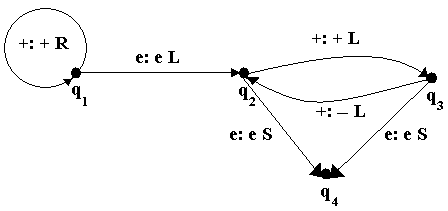
**Задача 2**. На ленте машины Тьюринга содержится последовательность символов “+”. Построить машину Тьюринга (в виде таблицы и в виде диаграммы), которая каждый второй символ “+” заменит на “–“. Замена начинается с правого конца последовательности. Автомат в состоянии q1 обозревает произвольный символ последовательности.

***Решение***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 |
| + | + q1 R | + q3 L | – q2 L |  |
| – |  |  |  |  |
| e | e q2 L | e q4 S | e q4 S |  |

В состоянии q1 автомат перемещается в правый край последовательности. Cостояние q2 служит для того, чтобы пропустить символ +, стоящий в нечетной позиции (если считать, что нумерация символов в строке производится справа налево), после замены одного «плюса» происходит переход в состояние q3. В состоянии q3 символ +, стоящий в четной позиции, заменяется на –, после чего происходит возврат в состояние q2. Выход из состояний q2 и q3 происходит в случае если достигнут пустой символ, т.е. если последовательность символов «+» обработана полностью. Данный пример иллюстрирует программирование простейшего цикла с помощью машины Тьюринга.

Запишем этот алгоритм в виде диаграммы.



**Задача 3**. Даны два натуральных числа m и n (m>n) в унарной системе счисления, использующей цифру “|”. Числа разделены одной пустой клеткой. Разработать машину Тьюринга, которая на ленте оставит разность чисел m и n (m − n). Число m записано левее числа n. Автомат в начальном состоянии обозревает самый правый символ числа n. Реализуйте данный алгоритм в программе Algo2000.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 | q6 |
| | | e q2 L | | q2 L | | q3 L | e q5 R | | q5 R | | q6 R |
| e |  | e q3 L | e q4 R |  | e q6 R | e q1 L |

Здесь так же имеет место циклический алгоритм. Распишем данный алгоритм по шагам.

Удаляем последнюю цифру в числе n (заменяем на пустой символ).

Сдвигаемся влево, последовательно пропуская число n, пустой символ-разделитель между числами и число m, останавливаемся, достигнув крайнюю левую цифру числа m.

Удаляем крайний левый символ в числе m (заменяем на пустой символ).

Возвращаемся на крайний правый символ числа n, после чего переходим к шагу 1.

Шагу 1 соответствует состояние q1, шагу 2 – состояния q2, q3, шагу 3 – состояние q4, шагу 4 – состояния q5,q6. Завершение работы машины Тьюринга происходит в состоянии q1, если закончились цифры в числе n.

***Справочная информация по работе с программой Algo2000:***

Прежде, чем вводить алгоритм, нужно ввести внешний алфавит.

В ячейках верхней строки записываются символы внутреннего алфавита: Q1,Q2...Qn.

В ячейках первого столбца записываются символы внешнего алфавита.

В ячейках других столбцов и строчек помещаются команды машины Тьюринга.

Формат команды машины Тьюринга: aKq, где:

a - новое содержание текущей ячейки (новый символ внешнего алфавита, который заносится в текущую ячейку),

K - команда лентопротяжного механизма машины Тьюринга (влево, вправо, стоп)

q - новое внутренне состояние машины Тьюринга

Чтобы ввести команду в ячейку нужно:

1) Ввести символ внешнего алфавита

2) Ввести одну из команд лентопротяжного механизма:

• "Влево" - ввести левую угловую скобку "<"

• "Вправо" - ввести правую угловую скобку ">"

• "Cтоп" - ввести восклицательный знак "!"

3) Ввести номер нового внутреннего состояния. Нужно ввести только цифру, букву Q вводить не надо.

Пример: чтобы ввести команду

# **->** Q1

нужно последовательно набрать на клавиатуре символы: # > 1 , где:

# - указание машине Тьюринга записать символ "#" в текущую ячейку ленту

> - указание машине Тьюринга сдвинуть каретку вправо

1 - указание машине Тьюринга перейти в новое внутреннее состояние Q1

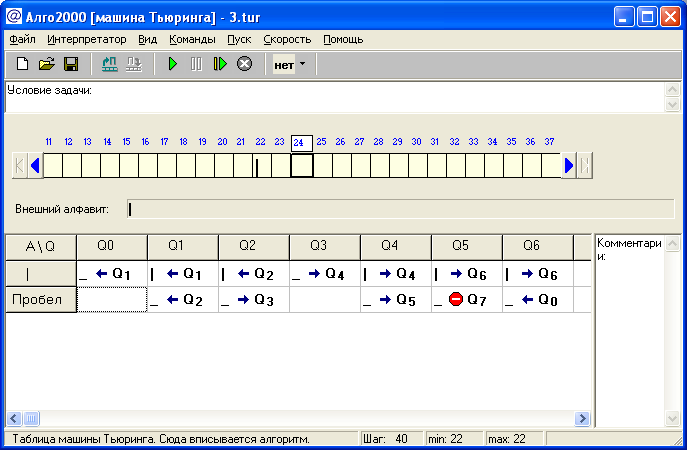
Пробелы в команде игнорируются.

Чтобы указать в команде в качестве символа внешнего алфавита "пробел", нужно ввести в ячейку символ подчеркивания "\_", пробел или ввести сразу команду лентопротяжного механизма.

Если команда не была введена правильно, то в ячейку таблицы ничего не запишется.

Чтобы удалить, вставить, добавить столбцы, очистить строки или столбцы воспользуйтесь соответствующими пунктами главного или всплывающего меню или панелью инструментов.

***Реализация алгоритма в программе Algo2000.***



**Задача 4**. Построить композицию машин Тьюринга для вычисления функции *f*(*x*)*=*2*x+*1, где x – десятичное число. В начальный момент времени головка располагается напротив левого крайнего символа числа.

Построим машину Тьюринга M1, вычисляющую *g*(*x*)=2*x*, и машину Тьюринга M2, вычисляющую *h*(*x*)=*x*+1. **В обеих машинах начнем работу из одного и того же положения головки относительно числа, закончим работу в том же положении с которого начали, а именно начальное и конечное положение головки установим напротив крайнего левого символа числа.**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M1: | q1 | q2 | q3 | q4 | q5 |  | M2: | p1 | p2 | p3 | p4 |
| e | e q2 L | e q4 S | 1 q4 L | e q5 R |  |  | e | e p2 L | 1 p3 L | e p4 R |  |
| 0 | 0 q1 R | 0 q2 L | 1 q2 L | 0 q4 L |  |  | 0 | 0 p1 R | 1 p3 L | 0 p3 L |  |
| 1 | 1 q1 R | 2 q2 L | 3 q2 L | 1 q4 L |  |  | 1 | 1 p1 R | 2 p3 L | 1 p3 L |  |
| 2 | 2 q1 R | 4 q2 L | 5 q2 L | 2 q4 L |  |  | 2 | 2 p1 R | 3 p3 L | 2 p3 L |  |
| 3 | 3 q1 R | 6 q2 L | 7 q2 L | 3 q4 L |  |  | 3 | 3 p1 R | 4 p3 L | 3 p3 L |  |
| 4 | 4 q1 R | 8 q2 L | 9 q2 L | 4 q4 L |  |  | 4 | 4 p1 R | 5 p3 L | 4 p3 L |  |
| 5 | 5 q1 R | 0 q3 L | 1 q3 L | 5 q4 L |  |  | 5 | 5 p1 R | 6 p3 L | 5 p3 L |  |
| 6 | 6 q1 R | 2 q3 L | 3 q3 L | 6 q4 L |  |  | 6 | 6 p1 R | 7 p3 L | 6 p3 L |  |
| 7 | 7 q1 R | 4 q3 L | 5 q3 L | 7 q4 L |  |  | 7 | 7 p1 R | 8 p3 L | 7 p3 L |  |
| 8 | 8 q1 R | 6 q3 L | 7 q3 L | 8 q4 L |  |  | 8 | 8 p1 R | 9 p3 L | 8 p3 L |  |
| 9 | 9 q1 R | 8 q3 L | 9 q3 L | 9 q4 L |  |  | 9 | 9 p1 R | 0 p2 L | 9 p3 L |  |

Для организации композиции данных машин в одну, в машине M1 в состоянии q5 поставим команды перехода к машине M2.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M1: | q1 | q2 | q3 | q4 | q5 |  | M2: | p1 | p2 | p3 | p4 |
| e | e q2 L | e q4 S | 1 q4 L | e q5 R |  |  | e | e p2 L | 1 p3 L | e p4 R |  |
| 0 | 0 q1 R | 0 q2 L | 1 q2 L | 0 q4 L | 0 p1 S |  | 0 | 0 p1 R | 1 p3 L | 0 p3 L |  |
| 1 | 1 q1 R | 2 q2 L | 3 q2 L | 1 q4 L | 1 p1 S |  | 1 | 1 p1 R | 2 p3 L | 1 p3 L |  |
| 2 | 2 q1 R | 4 q2 L | 5 q2 L | 2 q4 L | 2 p1 S |  | 2 | 2 p1 R | 3 p3 L | 2 p3 L |  |
| 3 | 3 q1 R | 6 q2 L | 7 q2 L | 3 q4 L | 3 p1 S |  | 3 | 3 p1 R | 4 p3 L | 3 p3 L |  |
| 4 | 4 q1 R | 8 q2 L | 9 q2 L | 4 q4 L | 4 p1 S |  | 4 | 4 p1 R | 5 p3 L | 4 p3 L |  |
| 5 | 5 q1 R | 0 q3 L | 1 q3 L | 5 q4 L | 5 p1 S |  | 5 | 5 p1 R | 6 p3 L | 5 p3 L |  |
| 6 | 6 q1 R | 2 q3 L | 3 q3 L | 6 q4 L | 6 p1 S |  | 6 | 6 p1 R | 7 p3 L | 6 p3 L |  |
| 7 | 7 q1 R | 4 q3 L | 5 q3 L | 7 q4 L | 7 p1 S |  | 7 | 7 p1 R | 8 p3 L | 7 p3 L |  |
| 8 | 8 q1 R | 6 q3 L | 7 q3 L | 8 q4 L | 8 p1 S |  | 8 | 8 p1 R | 9 p3 L | 8 p3 L |  |
| 9 | 9 q1 R | 8 q3 L | 9 q3 L | 9 q4 L | 9 p1 S |  | 9 | 9 p1 R | 0 p2 L | 9 p3 L |  |

Теперь можно записать полученный результат в виде одной машины Тьюринга – M3.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M3: | q1 | q2 | q3 | q4 | q5 | p1 | p2 | p3 | p4 |
| e | e q2 L | e q4 S | 1 q4 L | e q5 R |  | e p2 L | 1 p3 L | e p4 R |  |
| 0 | 0 q1 R | 0 q2 L | 1 q2 L | 0 q4 L | 0 p1 S | 0 p1 R | 1 p3 L | 0 p3 L |  |
| 1 | 1 q1 R | 2 q2 L | 3 q2 L | 1 q4 L | 1 p1 S | 1 p1 R | 2 p3 L | 1 p3 L |  |
| 2 | 2 q1 R | 4 q2 L | 5 q2 L | 2 q4 L | 2 p1 S | 2 p1 R | 3 p3 L | 2 p3 L |  |
| 3 | 3 q1 R | 6 q2 L | 7 q2 L | 3 q4 L | 3 p1 S | 3 p1 R | 4 p3 L | 3 p3 L |  |
| 4 | 4 q1 R | 8 q2 L | 9 q2 L | 4 q4 L | 4 p1 S | 4 p1 R | 5 p3 L | 4 p3 L |  |
| 5 | 5 q1 R | 0 q3 L | 1 q3 L | 5 q4 L | 5 p1 S | 5 p1 R | 6 p3 L | 5 p3 L |  |
| 6 | 6 q1 R | 2 q3 L | 3 q3 L | 6 q4 L | 6 p1 S | 6 p1 R | 7 p3 L | 6 p3 L |  |
| 7 | 7 q1 R | 4 q3 L | 5 q3 L | 7 q4 L | 7 p1 S | 7 p1 R | 8 p3 L | 7 p3 L |  |
| 8 | 8 q1 R | 6 q3 L | 7 q3 L | 8 q4 L | 8 p1 S | 8 p1 R | 9 p3 L | 8 p3 L |  |
| 9 | 9 q1 R | 8 q3 L | 9 q3 L | 9 q4 L | 9 p1 S | 9 p1 R | 0 p2 L | 9 p3 L |  |

Можно осуществить перенумерацию состояний, тогда получим.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M3: | q1 | q2 | q3 | q4 | q5 | q6 | q7 | q8 | q9 |
| e | e q2 L | e q4 S | 1 q4 L | e q5 R |  | e q7 L | 1 q8 L | e q9 R |  |
| 0 | 0 q1 R | 0 q2 L | 1 q2 L | 0 q4 L | 0 q6 S | 0 q6 R | 1 q8 L | 0 q8 L |  |
| 1 | 1 q1 R | 2 q2 L | 3 q2 L | 1 q4 L | 1 q6 S | 1 q6 R | 2 q8 L | 1 q8 L |  |
| 2 | 2 q1 R | 4 q2 L | 5 q2 L | 2 q4 L | 2 q6 S | 2 q6 R | 3 q8 L | 2 q8 L |  |
| 3 | 3 q1 R | 6 q2 L | 7 q2 L | 3 q4 L | 3 q6 S | 3 q6 R | 4 q8 L | 3 q8 L |  |
| 4 | 4 q1 R | 8 q2 L | 9 q2 L | 4 q4 L | 4 q6 S | 4 q6 R | 5 q8 L | 4 q8 L |  |
| 5 | 5 q1 R | 0 q3 L | 1 q3 L | 5 q4 L | 5 q6 S | 5 q6 R | 6 q8 L | 5 q8 L |  |
| 6 | 6 q1 R | 2 q3 L | 3 q3 L | 6 q4 L | 6 q6 S | 6 q6 R | 7 q8 L | 6 q8 L |  |
| 7 | 7 q1 R | 4 q3 L | 5 q3 L | 7 q4 L | 7 q6 S | 7 q6 R | 8 q8 L | 7 q8 L |  |
| 8 | 8 q1 R | 6 q3 L | 7 q3 L | 8 q4 L | 8 q6 S | 8 q6 R | 9 q8 L | 8 q8 L |  |
| 9 | 9 q1 R | 8 q3 L | 9 q3 L | 9 q4 L | 9 q6 S | 9 q6 R | 0 q7 L | 9 q8 L |  |

Таким образом, мы получили машину Тьюринга для вычисления функции *f*(*x*)*=*2*x+*1 и использовали процедурный подход, разбили сложную задачу на подзадачи, реализовали каждую подзадачу, а далее, как из кирпичиков сложили основную задачу.

**Задача 5**. На информационной ленте машины Тьюринга записано число N (N>2) в унарной системе счисления. Читающая головка находится напротив крайнего левого символа в записи числа. Выполните трассировку программы для некоторого числа. Определите, какую задачу она решает. Придумайте, как можно решить эту же задачу, используя меньшее число состояний. Запишите свое решение в виде программы-таблицы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 |
| | | | q1 R | e q3 L | | q3 L |  |
| e | e q2 L |  | e q4 R |  |

Исходные данные: **| | | |**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | в начале работы |  |  | q1 |  |  |  |  |  |  |
|  | … | e | | | | | | | | |  |  | … |
|  |  |  |  |  |  |  |  |  |  |  |
|  | | q1 R |  |  |  | q1 |  |  |  |  |  |
| … | e | | | | | | | | | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q1 R |  |  |  |  | q1 |  |  |  |  |
| … | e | | | | | | | | | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q1 R |  |  |  |  |  | q1 |  |  |  |
| … | e | | | | | | | | | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q1 R |  |  |  |  |  |  | q1 |  |  |
| … | e | | | | | | | | | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| e q2 L |  |  |  |  |  | q2 |  |  |  |
| … | e | | | | | | | | | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| e q3 L |  |  |  |  | q3 |  |  |  |  |
| … | e | | | | | | | e | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q3 L |  |  |  | q3 |  |  |  |  |  |
| … | e | | | | | | | e | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q3 L |  |  | q3 |  |  |  |  |  |  |
| … | e | | | | | | | e | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| | q3 L |  | q3 |  |  |  |  |  |  |  |
| … | e | | | | | | | e | e |  | … |
|  |  |  |  |  |  |  |  |  |  |
| е q4 R |  |  | q4 |  |  |  |  |  |  |
| … | e | | | | | | | e | e |  | … |
|  |  |  |  |  |  |  |  |  |  |  |

Алгоритм реализует уменьшение значения унарного числа на 1.

Другой вариант алгоритма:

|  |  |
| --- | --- |
|  | q1 |
| | | e q1 R |
| e |  |

**Пример 1**

Дано число n в десятичной системе счисления. Разработайте машину Тьюринга, которая увеличивала бы заданное число на 2. В начальный момент времени автомат обозревает крайнюю левую цифру. В конечном состоянии он так же должен обозревать крайнюю левую цифру числа. Составить программу-таблицу и нарисовать диаграмму переходов. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии. Отметьте запрещенные клетки таблицы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 |
| e | eq2L | Х | 1q5S | eq5R | Х |
| 0 | 0q1R | 2q4L | 1q4L | 0q4L |  |
| 1 | 1q1R | 3q4L | 2q4L | 1q4L |  |
| 2 | 2q1R | 4q4L | 3q4L | 2q4L |  |
| 3 | 3q1R | 5q4L | 4q4L | 3q4L |  |
| 4 | 4q1R | 6q4L | 5q4L | 4q4L |  |
| 5 | 5q1R | 7q4L | 6q4L | 5q4L |  |
| 6 | 6q1R | 8q4L | 7q4L | 6q4L |  |
| 7 | 7q1R | 9q4L | 8q4L | 7q4L |  |
| 8 | 8q1R | 0q3L | 9q4L | 8q4L |  |
| 9 | 9q1R | 1q3L | 0q3L | 9q4L |  |

q1 – перемещение автомата к последней цифре числа

q2 – добавление к последней цифре числа значения 2

q3 – увеличение разрядов числа на 1, если последняя цифра числа 8 или 9

q4 – перевод автомата в крайнюю левую позицию

q5 – завершение работы

Протестировать решение на числах 72, 39, 98, 999.



На информационной ленте машины Тьюринга содержится непрерывная последовательность символов «%». Сконструируйте машину Тьюринга, которая заменит каждый третий символ на символ «$» (отсчет символов начинается с правого края строки). В начальный момент времени автомат обозревает произвольный символ заданной строки символов. В конечный момент времени он должен обозревать крайний левый символ. Составить программу-таблицу и нарисовать диаграмму переходов. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии. Отметьте запрещенные клетки таблицы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 |
| e | eq2L | eq5R | eq5R | eq5R | X |
| % | %q1R | %q3L | %q4L | $q2L |  |
| $ | X | X | X | X |  |

Протестировать на значениях $, $$$, $$$$$.

На информационной ленте машины Тьюринга записано число N (N>2) в унарной системе счисления. Читающая головка находится напротив крайнего левого символа в записи числа. Выполните трассировку программы для некоторого числа. Определите, какую задачу она решает. Придумайте, как можно решить эту же задачу, используя меньшее число состояний. Запишите свое решение в виде программы-таблицы. Отметьте запрещенные клетки таблицы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 |
| | | | q1 R | e q3 L | e q4 L | | q4 L |  |
| e | e q2 L |  |  | e q5 R |  |

Вычитает из числа 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | q1 | q2 | q3 |
| | | eq2R | eq3R |  |
| e | X | X | X |

Дано число X в двоичной системе счисления. Постройте две машины Тьюринга: первая должна решать задачу умножения двоичного числа на 8, вторая – задачу вычитания 1 из заданного двоичного числа. Если при вычитании в старшем разряде получается ноль, заменять его пустым символом. Используя композицию построенных машин Тьюринга решить две задачи: вычисления Y1 = 8X – 1 и Y2 = (X – 1)\*8. В начальный момент времени автомат обозревает крайний левый непустой символ. Во всех машинах Тьюринга отметьте запрещенные клетки таблицы.

Чтобы умножить двоичное число на 8, достаточно приписать справа от него три нуля.

Y=X\*8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 |
| e | 0q2R | 0q3R | 0q4L | eq5R | X |
| 0 | 0q1R | X | X | 0q4L |  |
| 1 | 1q1R | X | X | 1q4L |  |

Y=X-1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | p1 | p2 | p3 | p4 | p5 |
| e | ep2L | X | ep4R | X | X |
| 0 | 0p1R | 1p2L | 0p3L | ep5R |  |
| 1 | 1p1R | 0p3L | 1p3L | 1p5S |  |

Объединяем обе части для вычисления выражения Y1=8X-1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | q1 | q2 | q3 | q4 | q5 | p1 | p2 | p3 | p4 | p5 |
| e | 0q2R | 0q3R | 0q4L | eq5R | X | ep2L | X | ep4R | X | X |
| 0 | 0q1R | X | X | 0q4L | 0p1S | 0p1R | 1p2L | 0p3L | ep5R |  |
| 1 | 1q1R | X | X | 1q4L | 1p1S | 1p1R | 0p3L | 1p3L | 1p5S |  |

Объединяем обе части для вычисления выражения Y1=(X-1)\*8

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | p1 | p2 | p3 | p4 | p5 | q1 | q2 | q3 | q4 | q5 |
| e | ep2L | X | ep4R | X | X | 0q2R | 0q3R | 0q4L | eq5R | X |
| 0 | 0p1R | 1p2L | 0p3L | ep5R | 0q1S | 0q1R | X | X | 0q4L | 0p1S |
| 1 | 1p1R | 0p3L | 1p3L | 1p5S | 1q1S | 1q1R | X | X | 1q4L | 1p1S |

Протестировать на значениях 11, 100

# Практическое занятие № 2 Рекурсивные функции

**Цель занятия**: Научиться работать с рекурсивными функциями

**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

Рекурсивные функции очень хорошо иллюстрируют понятие алгоритма. Если рассуждать упрощенно, то для рекурсивной функции должен существовать алгоритм, вычисляющей ее значения. Вообще говоря, большая часть известных числовых функций являются рекурсивными.

Полезно вспомнить, как определяются Элементарные функции. Вначале рассматривается несколько классов функций: алгебраические, тригонометрические, показательные, логарифмические. Элементарная функция определяется как Суперпозиция (или сложная функция) этих функций.

Рекурсивные функции строятся аналогичным образом.

Обратите внимание, что все функции в данном параграфе определены на множестве http://matica.org.ua/images/stories/MLITA/image679.png. Если это необходимо, в обозначении функции верхний индекс указывает число переменных. Так, функция http://matica.org.ua/images/stories/MLITA/image680.png зависит от http://matica.org.ua/images/stories/MLITA/image346.png переменных. Таким образом, http://matica.org.ua/images/stories/MLITA/image681.png.

Рассмотрим вначале примитивно-рекурсивные функции.

Простейшие примитивно-рекурсивные функции Задаются следующим образом.

· Функция следования задается формулой: http://matica.org.ua/images/stories/MLITA/image682.png (или http://matica.org.ua/images/stories/MLITA/image683.png).

· Функция аннулирования задается формулой: http://matica.org.ua/images/stories/MLITA/image684.png.

· Функция тождества определяется следующим образом: http://matica.org.ua/images/stories/MLITA/image685.png, то есть эта функция произвольному http://matica.org.ua/images/stories/MLITA/image346.png-мерному вектору сопоставляет его http://matica.org.ua/images/stories/MLITA/image665.png-ю координату.

Из простейших примитивно-рекурсивных функций можно получить примитивно-рекурсивные функции с помощью следующих двух операторов.

· Оператор суперпозиции. Пусть http://matica.org.ua/images/stories/MLITA/image686.png, http://matica.org.ua/images/stories/MLITA/image687.png, http://matica.org.ua/images/stories/MLITA/image688.png, …, http://matica.org.ua/images/stories/MLITA/image689.png – примитивно-рекурсивные функции. Тогда функция

http://matica.org.ua/images/stories/MLITA/image690.png

Получена с помощью оператора суперпозиции.

Оператор суперпозиции – это оператор построения сложной функции. Если мы умеем вычислять функции http://matica.org.ua/images/stories/MLITA/image691.png, http://matica.org.ua/images/stories/MLITA/image692.png, …, http://matica.org.ua/images/stories/MLITA/image693.png и http://matica.org.ua/images/stories/MLITA/image694.png, то значения функции http://matica.org.ua/images/stories/MLITA/image695.png могут быть получены последовательным вычислением значений функций http://matica.org.ua/images/stories/MLITA/image691.png, http://matica.org.ua/images/stories/MLITA/image692.png, …, http://matica.org.ua/images/stories/MLITA/image693.png на некотором наборе значений http://matica.org.ua/images/stories/MLITA/image696.png переменных http://matica.org.ua/images/stories/MLITA/image697.png, и вычислением значения функции http://matica.org.ua/images/stories/MLITA/image694.png на наборе значений http://matica.org.ua/images/stories/MLITA/image698.png, http://matica.org.ua/images/stories/MLITA/image699.png, …, http://matica.org.ua/images/stories/MLITA/image700.png

Пример. Функция http://matica.org.ua/images/stories/MLITA/image701.png получается суперпозицией функций 0(X) и S(X): http://matica.org.ua/images/stories/MLITA/image702.png. Аналогичным образом можно получить функции вида http://matica.org.ua/images/stories/MLITA/image703.png для всех значений N.

· Оператор примитивной рекурсии из известных примитивно-рекурсивных функций http://matica.org.ua/images/stories/MLITA/image704.png и http://matica.org.ua/images/stories/MLITA/image705.png позволяет строить новую функцию http://matica.org.ua/images/stories/MLITA/image706.png. Так,

http://matica.org.ua/images/stories/MLITA/image707.pnghttp://matica.org.ua/images/stories/MLITA/image708.png,

http://matica.org.ua/images/stories/MLITA/image709.png

http://matica.org.ua/images/stories/MLITA/image710.png.

Тогда функция http://matica.org.ua/images/stories/MLITA/image711.png получена с помощью оператора примитивной рекурсии, что выражается обозначением http://matica.org.ua/images/stories/MLITA/image712.png.

Таким образом, сначала (при фиксированных значениях http://matica.org.ua/images/stories/MLITA/image713.png) определяется значение функции http://matica.org.ua/images/stories/MLITA/image714.png при http://matica.org.ua/images/stories/MLITA/image715.png, а затем каждое следующее значение функции (зависящее от http://matica.org.ua/images/stories/MLITA/image716.png) выражается через предыдущее значение (зависящее от http://matica.org.ua/images/stories/MLITA/image414.png).

Пусть http://matica.org.ua/images/stories/MLITA/image717.png. Тогда функция http://matica.org.ua/images/stories/MLITA/image718.png есть постоянная. Обозначим ее следующим образом: http://matica.org.ua/images/stories/MLITA/image719.png. Функция http://matica.org.ua/images/stories/MLITA/image720.png зависит от двух переменных. Обозначим ее так: http://matica.org.ua/images/stories/MLITA/image721.png. Тогда

http://matica.org.ua/images/stories/MLITA/image722.png,

http://matica.org.ua/images/stories/MLITA/image723.pnghttp://matica.org.ua/images/stories/MLITA/image724.png.

Для произвольного http://matica.org.ua/images/stories/MLITA/image346.png получаем (обозначения http://matica.org.ua/images/stories/MLITA/image725.png, http://matica.org.ua/images/stories/MLITA/image553.png, http://matica.org.ua/images/stories/MLITA/image554.png, …, http://matica.org.ua/images/stories/MLITA/image726.png вводятся в предположении, что набор http://matica.org.ua/images/stories/MLITA/image713.png фиксирован):

http://matica.org.ua/images/stories/MLITA/image707.pnghttp://matica.org.ua/images/stories/MLITA/image727.png,

http://matica.org.ua/images/stories/MLITA/image728.pnghttp://matica.org.ua/images/stories/MLITA/image729.png,

http://matica.org.ua/images/stories/MLITA/image730.pnghttp://matica.org.ua/images/stories/MLITA/image731.png,

. . . . . . . . . . . . . . . . . .

http://matica.org.ua/images/stories/MLITA/image709.pnghttp://matica.org.ua/images/stories/MLITA/image732.png.

. . . . . . . . . . . . . . . . . .

Пример. Даны функции http://matica.org.ua/images/stories/MLITA/image733.png и http://matica.org.ua/images/stories/MLITA/image734.png. Определим функцию http://matica.org.ua/images/stories/MLITA/image735.png, полученную из данных функций по схеме примитивной рекурсии.

Решение. Найдем значения функции http://matica.org.ua/images/stories/MLITA/image735.png.

http://matica.org.ua/images/stories/MLITA/image736.pnghttp://matica.org.ua/images/stories/MLITA/image733.png,

http://matica.org.ua/images/stories/MLITA/image737.pnghttp://matica.org.ua/images/stories/MLITA/image738.pnghttp://matica.org.ua/images/stories/MLITA/image739.png;

http://matica.org.ua/images/stories/MLITA/image740.pnghttp://matica.org.ua/images/stories/MLITA/image741.pnghttp://matica.org.ua/images/stories/MLITA/image742.png;

http://matica.org.ua/images/stories/MLITA/image743.pnghttp://matica.org.ua/images/stories/MLITA/image744.pnghttp://matica.org.ua/images/stories/MLITA/image745.png.

Можно предположить, что http://matica.org.ua/images/stories/MLITA/image746.pnghttp://matica.org.ua/images/stories/MLITA/image747.png.

Докажем последнюю формулу методом математической индукции по переменной http://matica.org.ua/images/stories/MLITA/image414.png.

1. Проверим формулу при http://matica.org.ua/images/stories/MLITA/image748.png.

http://matica.org.ua/images/stories/MLITA/image736.pnghttp://matica.org.ua/images/stories/MLITA/image749.png, то есть при http://matica.org.ua/images/stories/MLITA/image750.png формула верна.

2. Допустим, что предположение индукции верно при http://matica.org.ua/images/stories/MLITA/image751.png, то есть, верна формула http://matica.org.ua/images/stories/MLITA/image752.pnghttp://matica.org.ua/images/stories/MLITA/image753.png.

Докажем, что предположение индукции верно при http://matica.org.ua/images/stories/MLITA/image754.png, то есть, верна формула http://matica.org.ua/images/stories/MLITA/image755.pnghttp://matica.org.ua/images/stories/MLITA/image756.png. Выразим http://matica.org.ua/images/stories/MLITA/image757.png с помощью схемы примитивной рекурсии.

http://matica.org.ua/images/stories/MLITA/image758.pnghttp://matica.org.ua/images/stories/MLITA/image759.pnghttp://matica.org.ua/images/stories/MLITA/image760.png.

Таким образом, на основании метода математической индукции формула http://matica.org.ua/images/stories/MLITA/image746.pnghttp://matica.org.ua/images/stories/MLITA/image747.png доказана для всех http://matica.org.ua/images/stories/MLITA/image761.png.

Теперь строго определим примитивно-рекурсивные функции.

Определение. 1) Простейшие примитивно-рекурсивные функции примитивно-рекурсивны.

2) Примитивно-рекурсивными являются функции, полученные из примитивно-рекурсивных функций с помощью операторов суперпозиции и (или) примитивной рекурсии.

3) Функция является примитивно-рекурсивной тогда и только тогда, когда это следует из 1) и 2).

Пример. Покажем, что функция http://matica.org.ua/images/stories/MLITA/image762.png примитивно-рекурсивна.

Доказательство. http://matica.org.ua/images/stories/MLITA/image763.png, http://matica.org.ua/images/stories/MLITA/image764.png, следовательно, функция http://matica.org.ua/images/stories/MLITA/image765.png должна зависеть от одной переменной, а функция http://matica.org.ua/images/stories/MLITA/image766.png – от трех. Пользуясь заданием функции, найдем ее значения:

http://matica.org.ua/images/stories/MLITA/image767.png.

http://matica.org.ua/images/stories/MLITA/image768.png

http://matica.org.ua/images/stories/MLITA/image769.png

То есть http://matica.org.ua/images/stories/MLITA/image770.png.

Таким образом, функция http://matica.org.ua/images/stories/MLITA/image762.png получена по схеме примитивной рекурсии (http://matica.org.ua/images/stories/MLITA/image771.png) из примитивно-рекурсивных функций, следовательно, сама является примитивно-рекурсивной.

Примитивно-рекурсивными, в частности, являются следующие функции: http://matica.org.ua/images/stories/MLITA/image772.png, http://matica.org.ua/images/stories/MLITA/image773.png, http://matica.org.ua/images/stories/MLITA/image774.png, http://matica.org.ua/images/stories/MLITA/image775.png, http://matica.org.ua/images/stories/MLITA/image776.png, http://matica.org.ua/images/stories/MLITA/image777.png, http://matica.org.ua/images/stories/MLITA/image778.png http://matica.org.ua/images/stories/MLITA/image779.png

Операция минимизации по http://matica.org.ua/images/stories/MLITA/image665.png-ой переменной функции http://matica.org.ua/images/stories/MLITA/image680.png обозначается следующим образом: http://matica.org.ua/images/stories/MLITA/image780.png, и определяется так.

Рассмотрим уравнение относительно http://matica.org.ua/images/stories/MLITA/image414.png:

http://matica.org.ua/images/stories/MLITA/image781.png. (1)

Это уравнение решается подбором, вместо переменной http://matica.org.ua/images/stories/MLITA/image414.png последовательно подставляются 0,1,2,… При этом возможны случаи.

· На некотором шаге левая часть соотношения (1) не определена. Следовательно, на наборе http://matica.org.ua/images/stories/MLITA/image782.png операция минимизации не определена.

· На каждом шаге левая часть соотношения (1) определена, но равенство не выполняется ни при каких значениях http://matica.org.ua/images/stories/MLITA/image414.png. Следовательно, на наборе http://matica.org.ua/images/stories/MLITA/image782.png операция минимизации не определена.

· Левая часть соотношения (1) определена при http://matica.org.ua/images/stories/MLITA/image783.png, но при http://matica.org.ua/images/stories/MLITA/image784.png равенство не выполняется, а при http://matica.org.ua/images/stories/MLITA/image785.png выполняется. В этом случае число http://matica.org.ua/images/stories/MLITA/image786.png считается значением операции минимизации на наборе http://matica.org.ua/images/stories/MLITA/image787.png.

Пример. [13]. Найти функции, получаемые из данной числовой функции http://matica.org.ua/images/stories/MLITA/image788.png с помощью оператора минимизации по каждой ее переменной.

Решение. Минимизируем функцию по переменной http://matica.org.ua/images/stories/MLITA/image555.png. Рассмотрим уравнение

http://matica.org.ua/images/stories/MLITA/image789.png. (2)

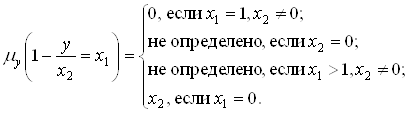
1. Если http://matica.org.ua/images/stories/MLITA/image790.png, http://matica.org.ua/images/stories/MLITA/image791.png, то при подстановке http://matica.org.ua/images/stories/MLITA/image750.png получаем верное равенство.

2. Если http://matica.org.ua/images/stories/MLITA/image792.png, то левая часть равенства (2) не определена.

3. Если http://matica.org.ua/images/stories/MLITA/image793.png, http://matica.org.ua/images/stories/MLITA/image791.png, то при подстановке http://matica.org.ua/images/stories/MLITA/image794.png в левой части равенства (2) появляется выражение http://matica.org.ua/images/stories/MLITA/image795.png, не имеющее смысла, и в этом случае операция минимизации не определена.

4. Если http://matica.org.ua/images/stories/MLITA/image796.png, то получаем равенство http://matica.org.ua/images/stories/MLITA/image797.png. Оно имеет смысл при http://matica.org.ua/images/stories/MLITA/image790.png, то есть http://matica.org.ua/images/stories/MLITA/image750.png, что рассмотрено в первом пункте, и при http://matica.org.ua/images/stories/MLITA/image798.png, то есть http://matica.org.ua/images/stories/MLITA/image794.png. При http://matica.org.ua/images/stories/MLITA/image799.png равенство не имеет смысла.

Таким образом,



Минимизируем функцию по переменной http://matica.org.ua/images/stories/MLITA/image556.png. Рассмотрим уравнение

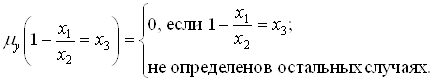
http://matica.org.ua/images/stories/MLITA/image801.png.

Это уравнение на самом первом шаге, при подстановке вместо http://matica.org.ua/images/stories/MLITA/image414.png нуля теряет смысл, значит, операция минимизации по второй переменной http://matica.org.ua/images/stories/MLITA/image802.png нигде не определена.

Минимизируем функцию по переменной http://matica.org.ua/images/stories/MLITA/image803.png. Рассмотрим уравнение

http://matica.org.ua/images/stories/MLITA/image804.png. (3)

Если левая часть соотношения (3) имеет смысл и равенство (3) выполнено, то оно выполнено и при подстановке в это соотношение переменной http://matica.org.ua/images/stories/MLITA/image414.png на первом шаге, то есть при http://matica.org.ua/images/stories/MLITA/image750.png. В остальных случаях значение операции минимизации не определено.



Определение. Частично-рекурсивной функцией называется числовая функция, получаемая за конечное число шагов из простейших примитивно-рекурсивных функций с помощью операторов суперпозиции, примитивной рекурсии и минимизации.

Определение. Числовая функция называется Общерекурсивной, если она частично-рекурсивна и всюду определена.

Определение. Функция http://matica.org.ua/images/stories/MLITA/image806.png называется Эффективно вычислимой, если существует алгоритм, позволяющий вычислить ее значения.

В данном определении алгоритм понимается в интуитивном значении, следовательно, интуитивным является и понятие эффективно вычислимой функции.

Имеет место следующий тезис.

Тезис Черча. Каждая интуитивно вычислимая функция является частично-рекурсивной.

Тезис является недоказуемым, так как он связывает нестрогое понятие интуитивно вычислимой функции и строгое математическое понятие частично-рекурсивной функции.

Тезис может быть опровергнут построением примера интуитивно вычислимой, но не частично-рекурсивной функции.

1. Определить функцию http://matica.org.ua/images/stories/MLITA/image735.png, полученную из функций http://matica.org.ua/images/stories/MLITA/image807.png и http://matica.org.ua/images/stories/MLITA/image808.png по схеме примитивной рекурсии.

1) http://matica.org.ua/images/stories/MLITA/image733.png, http://matica.org.ua/images/stories/MLITA/image809.png.

2) http://matica.org.ua/images/stories/MLITA/image733.png, http://matica.org.ua/images/stories/MLITA/image810.png.

3) http://matica.org.ua/images/stories/MLITA/image733.png, http://matica.org.ua/images/stories/MLITA/image811.png.

4) http://matica.org.ua/images/stories/MLITA/image733.png, http://matica.org.ua/images/stories/MLITA/image812.png.

5) http://matica.org.ua/images/stories/MLITA/image813.png, http://matica.org.ua/images/stories/MLITA/image734.png.

6) http://matica.org.ua/images/stories/MLITA/image814.png, http://matica.org.ua/images/stories/MLITA/image815.png.

7) http://matica.org.ua/images/stories/MLITA/image816.png, http://matica.org.ua/images/stories/MLITA/image817.png.

8) http://matica.org.ua/images/stories/MLITA/image814.png, http://matica.org.ua/images/stories/MLITA/image818.png.

9) http://matica.org.ua/images/stories/MLITA/image816.png, http://matica.org.ua/images/stories/MLITA/image819.png.

10) http://matica.org.ua/images/stories/MLITA/image816.png, http://matica.org.ua/images/stories/MLITA/image820.png.

2. Доказать, что следующие функции примитивно-рекурсивны.

1) http://matica.org.ua/images/stories/MLITA/image772.png.

2) http://matica.org.ua/images/stories/MLITA/image773.png.

3) http://matica.org.ua/images/stories/MLITA/image821.png

4) http://matica.org.ua/images/stories/MLITA/image775.png.

5) http://matica.org.ua/images/stories/MLITA/image777.png.

6) http://matica.org.ua/images/stories/MLITA/image779.png

7) http://matica.org.ua/images/stories/MLITA/image776.png.

3. Записать схему примитивной рекурсии для произвольных примитивно-рекурсивных функций при

1) http://matica.org.ua/images/stories/MLITA/image764.png;

2) http://matica.org.ua/images/stories/MLITA/image822.png;

3) http://matica.org.ua/images/stories/MLITA/image823.png.

4. Найти функции, получаемые из данной числовой функции http://matica.org.ua/images/stories/MLITA/image824.png с помощью оператора минимизации по каждой ее переменной.

1) http://matica.org.ua/images/stories/MLITA/image825.png.

2) http://matica.org.ua/images/stories/MLITA/image826.png.

3) http://matica.org.ua/images/stories/MLITA/image827.png.

4) http://matica.org.ua/images/stories/MLITA/image828.png.

5) http://matica.org.ua/images/stories/MLITA/image829.png.

6) http://matica.org.ua/images/stories/MLITA/image830.png.

7) http://matica.org.ua/images/stories/MLITA/image831.png.

8) http://matica.org.ua/images/stories/MLITA/image832.png.

9) http://matica.org.ua/images/stories/MLITA/image833.png.

10) http://matica.org.ua/images/stories/MLITA/image834.png.

# Практическое занятие № 3 Нормальные алгоритмы

**Цель занятия**: Научиться работать с нормальными алгорифмами Маркова

**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

Программа (ориг. схема) алгоритма Маркова представляет собой конечный линейно упорядоченный набор редукций (правил) двух видов:

u → v (обыкновенное правило) и u → · v (заключительное правило), где u, v — слова в алфавите ∑. Вычисление начинается с входного слова P ϵ ∆∙ . Применяется первая редукция, редекс которой u входит в P. Применение редукции состоит в замене самого левого вхождения редекса на контрактум v. Затем те же действия производят с полученным словом, и т.д. Процесс прерывается в двух случаях: когда применилась заключительная редукция или когда ни одна из редукций не может быть применена. В обоих случаях полученное слово считается результатом вычисления.

В следующих задачах требуется построить нормальные алгорифмы, реализующие указанные преобразования слов. Ниже \_, \_ — метапеременные в схемах редукций, обозначающие произвольные буквы указанного алфавита (одна схема представляет блок редукций, порядок которых несущественен), \_, \_, — фиксированные различные буквы из \_ \ \_.1. Приписать заданное слово Q к входному слову а) слева, б) справа.

а) ! · Q b) \_\_ ! \_\_ (\_ 2 \_)

\_ ! · Q

! \_

2. Отсортировать входное слово P 2 {0, 1}\_ по возрастанию.

10 ! 01

3. Извлечь из входного слова P 2 {0, 1,#}\_ двоичное слово, располо-

женное между первым и вторым вхождением разделителя #.

\_ ! (\_ 2 {0, 1,#})

! ·

\_\_ ! \_\_ (\_ 2 {0, 1})

\_ !

\_\_ ! \_ (\_ 2 {0, 1})

\_ ! \_

# ! \_

# Практическое занятие № 4 Применение метода последовательной детализации

**Цель занятия**: Научиться использовать метод последовательной детализации при разработке алгоритмов

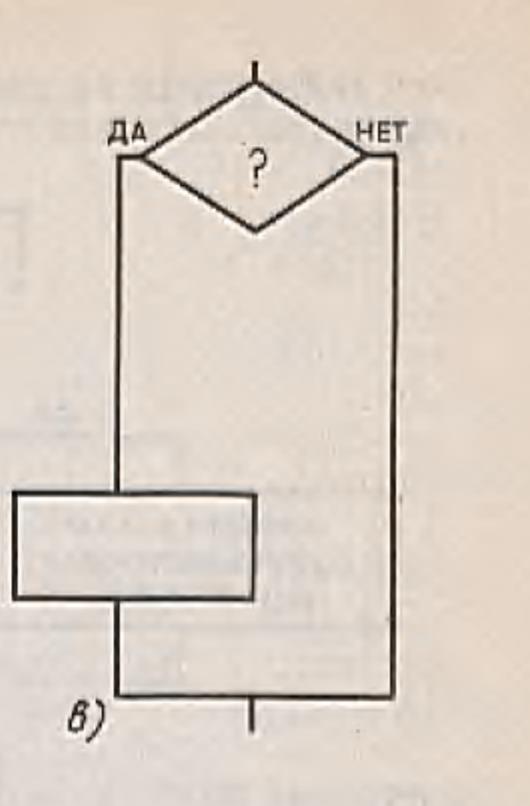
**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

**Задание №1. Ветвление и последовательная детализация алгоритма**

***Ветвление*** – это такая форма организации действий, при которой в зависимости от выполнения или невыполнения некоторого условия совершается либо одна, либо другая последовательность действий.

**Неполная форма ветвления**

****

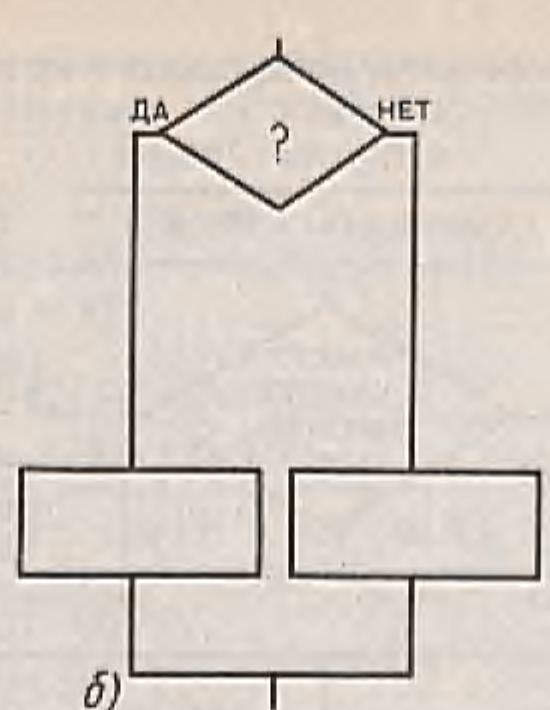
Неполная команда ветвления имеет следующий формат:

**если <условие>**

**то <серия>**

**кв** (конец ветвления)

**Полная форма ветвления**

****

Полная команда ветвления имеет следующий формат:

**если <условие>**

**то <серия 1>**

**иначе <серия 2>**

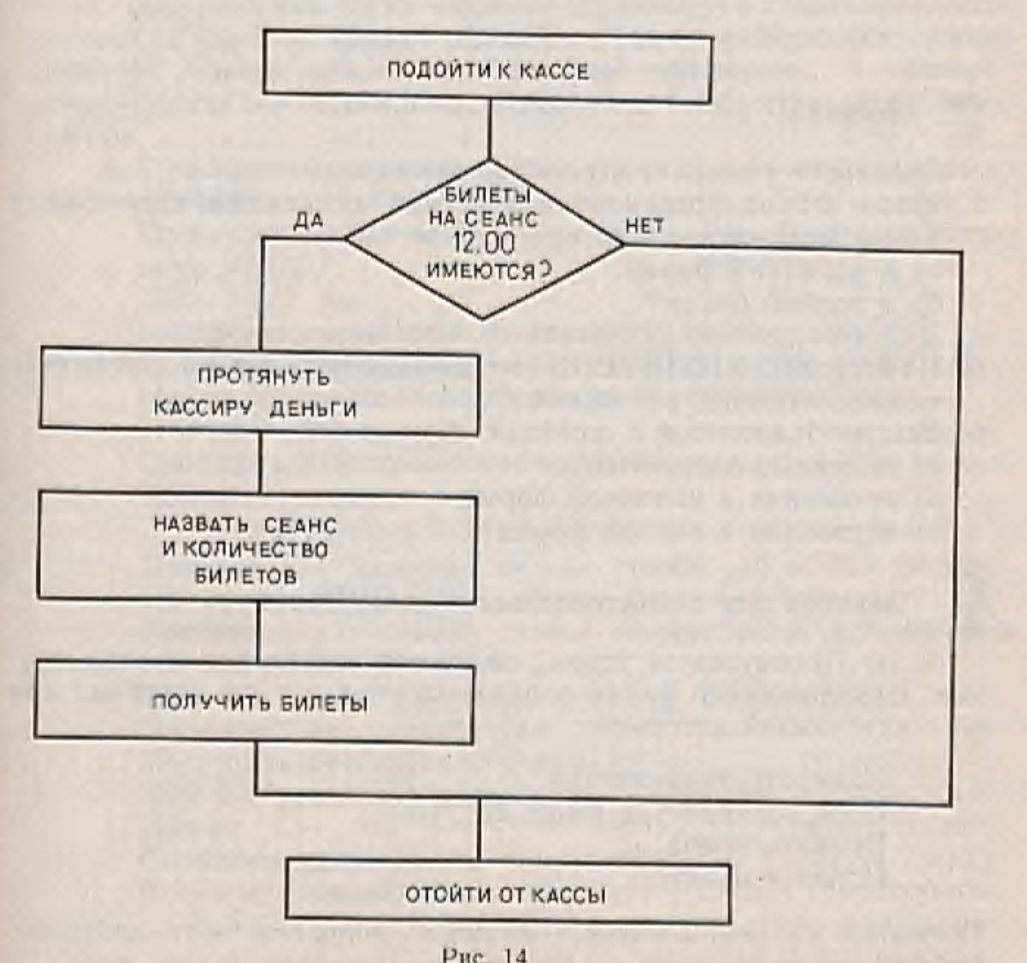
**кв** (конец ветвления)

**Пример ветвления.**

Пример 1. Допустим, вы собрались пойти в кинотеатр на сеанс 12.00. Алгоритм покупки может выглядеть так:

* Подойти к кассе.
* Если билеты на сеанс 12.00 имеются, то купить билеты.
* Отойти от кассы.

Представим это в виде блок схемы.



**Задание № 2**

Проснувшись утром, школьник почувствовал недомогание. Находившийся рядом злоумышленник тут же составил для него следующий алгоритм:

* + Измерить температуру.
  + Если температура выше 37 градусов, то:
  + Вызвать врача.
  + Пойти в школу.

Несмотря на недомогание, школьник исправил этот алгоритм, добавив всего две строки. Какие строки добавил школьник? Напишите его алгоритм. Представьте его в виде блок-схемы.

**Задание № 3**

Однажды школьник решил из своего дома позвонить приятелю. Злоумышленник, который и на этот раз оказался рядом, предложил ему следующий алгоритм:

* Подойти к телефону.
* Снять трубку .
* Набрать номер.
* Подождать 6 секунд.
* Если знакомый ответит, то:
* Сказать: «Здравствуй!»
* Сообщить последние новости.
* Узнать, что нового и как жизнь.
* Сказать:»До свидания!»
* Положить трубку.
* Конец ветвления.
* Отойти от телефона.

Школьник решил воспользоваться этим алгоритмом, и через некоторое время у него отключили телефон. Объясните почему.

# Практическое занятие № 5 Применение рекурсивных методов

**Цель занятия**: знакомство обучающихся с рекурсивными подпрограммами, механизмами их реализации, формирование и развитие навыков составления рекурсивных подпрограмм

**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

Примерный ход урока:

Объяснение нового материала можно начать с традиционного примера в теме «Рекурсивные подпрограммы» - показать, что вычисление значения факториала натурального числа N (N!) возможно двумя способами – итерационным и рекурсивным:

|  |  |
| --- | --- |
| Итерационный способ вычисления | Рекурсивный |
| N⋅⋅⋅⋅3 ⋅2 ⋅N!=1 | http://textarchive.ru/images/1102/2203816/1a9d0631.gif |
| 5⋅ 4 ⋅ 3 ⋅ 2 ⋅Например, 5!= 1 | 5!=4!\*5 |

Затем ввести определение частично-рекурсивной функции, привести описание вычисления N!, объяснить механизм реализации рекурсивного алгоритма (прямой и обратный ход рекурсии), обсудить достоинства (более изящный и компактный текст программы) и недостатки рекурсивного алгоритма (сравнительно медленное исполнение алгоритма). Необходимо подробно остановиться на процессе сохранения информации в стеке (это область памяти, организованная по принципу «первым зашёл, последним вышел»). Важно, чтобы учащиеся понимали, что в стек заносятся адреса команд (далее можно говорить – команды), которые не могут быть сразу выполнены в момент вызова подпрограммы, а также все локальные переменные. Занесённые в стек команды исполняются на обратном ходе рекурсии.

Затем предложите учащимся на основе составленной рекурсивной подпрограммы-функции для вычисления N! составить процедуру, обсудите перечень и типы параметров процедуры. Например:

Procedure Fact(N:integer; var F: longint);

Begin

If N=1 Then F:=1

Else

begin

Fact(N-1, F);

F:=F\*N

End

End;

Обратите внимание учащихся, что вычисление результата при каждом вызове будет осуществляться в два шага: рекурсивный вызов процедуры для аргумента (N-1) и умножение полученного результата на N.

В качестве примера рекурсивной процедуры без параметра, не вычисляющей результатов, можно решить следующую задачу:

Вводить последовательность символов, до тех пор, пока не будет введён символ ‘\*’. Вывести введённые символы в обратном порядке. Если попытаться реализовать данную задачу обычным методом, с помощью цикла, то потребуется немало усилий (запомнить ведённую последовательность в массиве, затем массив вывести в обратном порядке). Рекурсивный алгоритм выглядит очень эффектно:

programposl;

P

Результат работы программы:

Вводится последовательность: ABCDEF\*

Результат: FEDCBA

На данном примере можно показать механизм выполнения рекурсивной процедуры:

Команда write (c) запоминается в стеке при каждом рекурсивном вызове до тех пор, пока не будет введён символ '\*'.

Затем команды извлекаются из стека и выполняются. Таким образом, реализуется вывод в обратном порядке.

rocedurePrint\_Symb;

var c:char;

begin

read(c);

if c<>'\*' then

begin

Print\_Symb;

write (c)

end

end;

begin

Print\_Symb

end.

Задание для самостоятельного решения:

Составить рекурсивную подпрограмму вычисления N-го элемента последовательности Фибоначчи.

При выполнении задания следует обратить внимание учащихся на порядок выполнения задания: сначала записать частично-рекурсивную функцию вычисления N-го элемента последовательности, только потом составлять подпрограмму.

Данное задание рекомендуем выполнять в парах (один ученик составляет подпрограмму функцию, другой – процедуру) с перекрёстной проверкой результатов.

Домашнее задание. § 2.3.1, разобрать все примеры, выполнить упр. №2, 3.

Урок 2. Тема урока: Рекурсивные подпрограммы. Семинар-практикум.(1 час).

Цель урока: развитие навыков составления рекурсивных подпрограмм.

Примерный ход урока.

Проверка и обсуждение домашнего задания. Учащиеся демонстрируют и объясняют решение задания. Если кто-либо из учащихся испытывал затруднение при составлении рекурсивного алгоритма в задании №2, обратите внимание, что надо сначала составить описание частично-рекурсивной функции, затем только оформлять подпрограмму. К каждой задаче пусть учащиеся составят рекурсивную процедуру и проверят её работу. Такое решение задач (с функцией и процедурой) способствует, во-первых, пониманию реализации механизма выполнения рекурсии, во-вторых, закреплению навыков записи и вызова процедур, их отличий от функций.

Далее можно организовать работу в парах по выполнению заданий, направленных на закрепление понимания механизма рекурсивного процесса и составление рекурсивного алгоритма. Учитель выступает в роли консультанта, следит за процессом работы, в случае необходимости помогает. Важно, чтобы каждая группа справилась со своим заданием полностью.

Задания могут быть следующими:

Группа 1:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание |  | Ответ: |
|  | Определите, какую задачу решает данная подпрограмма.  Ответ запишите в столбце справа. | Function N1( n: longint):integer;  Begin  If N=0 then N1:=0  Else N1:=N1( N div 10) + N mod 10  End; |  |
|  | Вычислите значение N1(12345), N1(-1005) | Ответ запишите в столбце справа. |  |
|  | Перепишите подпрограмму-функцию в виде процедуры, дайте ей имя, соответствующее выполняемой задаче | | |
|  | Составьте рекурсивную подпрограмму вычисления суммы N элементов арифметической прогрессии, используя рекурсивную подпрограмму вычисления k-го элемента прогрессии. Проверьте работу программы на компьютере. | | |

Задачи для решения в группах:

К заданию 1:

|  |  |
| --- | --- |
| Function N2(a,b:integer):integer;  Begin  If b=0 then N2:=a  Else if b>0 then N2:=N2(a+1, b-1)  Else N2:=N2(a-1, b+1)  End; | Function N3( N :integer):integer;  Begin  If n=1 then N3:=n  Else N3:=N3(N div 2);  Write (N mod 2)  End; |
| Function N4( a,b:integer):integer;  Begin  If (a=0) or (b=0) then N4:=a +b  Else if a>b  then N4:=N4(a mod b, b)  else N4:=N4 (a, b mod a)  End; | Function N5( a,b:integer):integer;  Begin  If (a=0) or (b=0) then N5:=a +b  Else if a>b then N5:=N5(a - b, b)  Else N5:=N5 (a, b - a)  End; |
| Function N6(N : integer):integer;  Begin  If n=1 then N6:=1  Else if N>0  then N6:=N6(N -1)+2\*n-1  else N6:=N6(N +1)-2\*n-1  End; | procedure reverse (N: integer);  Begin  Write (N mod 10);  If (N Div 10)<>0 then  reverse (N Div 10)  end; |

К заданию 4:

1. Составьте рекурсивную подпрограмму вычисления суммы N элементов геометрической прогрессии, используя рекурсивную подпрограмму вычисления k-го элемента прогрессии. Проверьте работу программы на компьютере.
2. Составьте рекурсивную подпрограмму вычисления суммы N элементов последовательности Фибоначчи, используя рекурсивную подпрограмму вычисления k-го элемента прогрессии. Проверьте работу программы на компьютере.
3. Составьте рекурсивную подпрограмму вычисления суммы N слагаемых: 1!, 2!, 3!...N! , используя рекурсивную подпрограмму вычисления k!. Проверьте работу программы на компьютере.
4. Составьте рекурсивную подпрограмму вычисления суммы N слагаемых: a, a2, a3, …, aN используя рекурсивную подпрограмму вычисления ak (где а- действительное число, . Проверьте работу программы на компьютере.
5. Составьте рекурсивную подпрограмму программу вычисления S=1!+ 3!+5!+…(2N+1)!, используя рекурсивную подпрограмму вычисления факториала числа.

# Практическое занятие № 6 Реализация методов перебора

Задачи поиска предназначены для определения нахождения элемента, обладающего заданным свойством, в определенной совокупности данных, в частности, в массиве.

*Линейный поиск.*

*Поиск наибольшего и наименьшего элемента в массиве.*

Дан ряд чисел http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image435.gif , http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image437.gif , …, http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image439.gif , …, http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image441.gif . Разработать алгоритм поиска наибольшего и наименьшего числа в этом ряду с указанием номера (индекса) его расположения.

Очевидный способ поиска наибольшего (наименьшего) числа в заданном ряду *n* чисел включает следующие действия. Взять первое число ряда и сказать, что оно наибольшее (наименьшее), а индекс его равен 1. Затем взять второе число ряда и сравнить с предполагаемым максимальным (минимальным) первым числом. Если второе число больше предполагаемого (максимального) первого числа, взять третье число ряда и сравнить с первым. Так следует действовать до тех пор, пока не будет выбрано последнее http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image441.gif число. В результате на месте первого числа окажется наибольшее (наименьшее) число с указанным его номером в ряду чисел. [2]

Блок – схема алгоритма поиска наибольшего и наименьшего элемента на рис.18.

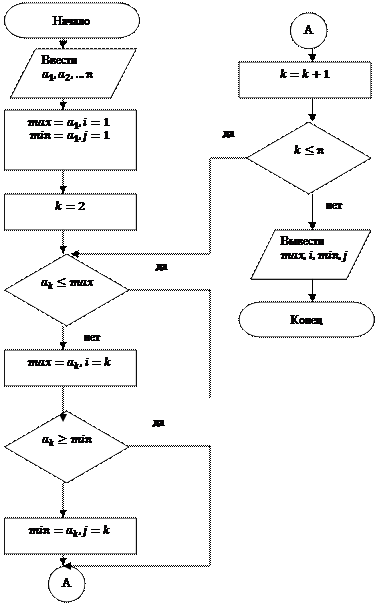


Рис. 1 Алгоритм нахождения наибольшего и наименьшего элемента в линейном массиве

Программа на языке Pascal представлена в Приложении 1, MaxMin.pas.

*Бинарный поиск.*

Метод бинарного поиска можно применять уже в отсортированном массиве. Допустим, что массив А отсортирован в порядке не убывания. Это позволяет по результату сравнения со средним элементом массива исключить из рассмотрения одну из половин. С оставшейся частью процедура повторяется. И так до тех пор, пока не будет найден искомый элемент или не будет построен весь массив. [6,7]

Рассмотрим алгоритм бинарного поиска на примере.

Пример. Пусть X = 6, а массив А состоит из 10 элементов:

3 5 6 8 12 15 17 18 20 25.

*1-й шаг.* Найдем номер среднего элемента среднего элементов: *m*= http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image467.gif = 5.

Так как 6 < А[5], то далее рассматриваются только элементы, индексы которых меньше 5.

3 5 6 8 **~~12~~**~~15 17 18 20 25.~~

*2-й шаг.* Рассматриваем лишь первые 4 элемента массива, находим индекс среднего элемента этой части : *m*= http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image469.gif = 2.

6 > А[2], следовательно, первый и второй элементы из рассмотрения исключаются:

~~3~~**~~5~~**6 8 ~~12 15 17 18 20 25~~;

*3-й шаг.*Рассматриваем два элемента, значение *m*= http://konspekta.net/lektsiiorgimg/baza9/6503561298357.files/image471.gif = 3.

~~3 5~~**6** 8 ~~12 15 17 18 20 25~~;

А[3] = 6. Элемент найден, его номер – 3.

Блок - схема алгоритма бинарного поиска на рис.19:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Случайный поиск.

Организация поиска k-го элемента в неупорядоченном массиве X возможна следующим образом. Выби­рается случайным образом элемент с номером q. Массив X раз­бивается на три части: элементы, меньшие X[q], равные X[q]и большие X[q]. А затем, в зависимости от количества элементов в каждой части, выбирается одна из частей для дальнейшего поиска. Теоретическая оценка числа сравнений имеет порядок k\*N, т. е. для худшего случая N2, но на практике он значительно быстрее.

Метод перебора в задачах поиска

Вариант 1. Дана вещественная матрица А(N,M). Составить программу замены всех положительных элементов матрицы на элемент, имеющий минимальное значение.

Вариант 2. Дана вещественная матрица А(N,M). Составить программу нахождения максимального отрицательного элемента матрицы и нахождения его местоположения.

Вариант 3. Дана вещественная матрица А(N,M). Составить программу замены всех отрицательных элементов матрицы на элемент, имеющий максимальное значение.

Вариант 4. Составить программу замены всех отрицательных элементов матрицы А(N,N) на элемент этой матрицы, имеющий минимальное значение. Скорректированную матрицу напечатать.

Вариант 5. Дана вещественная матрица А(N,M).Составить программу нахождения минимального положительного элемента матрицы и нахождения его местоположения.

Вариант 6. Составить программу замены всех отрицательных элементов матрицы А(6,6) на 0, если сумма минимального и максимального элементов этой матрицы окажется меньше Р.

# Практическое занятие № 7-8 Реализация методов сортировки данных

**Цель занятия**: изучить различные методы сортировки, научиться проводить сравнительный анализ методов друг с другом и выбирать наиболее подходящий для конкретных целей алгоритм сортировки

**Норма времени**: 2 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

*Сортировка*  – это процесс упорядочения некоторого множества элементов, на котором определены отношения порядка >, <,  ,  . Когда говорят о сортировке, подразумевают упорядочение множества элементов по возрастанию или убыванию. Рассмотрим различные алгоритмы сортировки и выясним, почему возникла необходимость появления такого большого числа алгоритмов решения одной и той же задачи.

Вообще, алгоритмы сортировки – одна из самых хорошо исследованных областей информатики. Тем не менее не исключены открытия и в этой области, потому что наверняка существуют еще какие-то пока неизвестные методы сортировки, основанные на новых принципах и идеях.

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные упорядочены. Примеры таких задач мы рассмотрим в этой лабораторной работе.

Интересным моментом в исследовании этой задачи является то, что переход от тривиальных алгоритмов к базирующимся на тех же принципах алгоритмам повышенной эффективности (от простого включения к методу Шелла; от простого извлечения к древесной сортировке; от пузырьковой сортировки к быстрой) требует значительного «концептуального прыжка». В таких случаях менее интересен поиск «микроскопических» улучшений, дающих экономию нескольких тактов процессора, чем занимаются многие программирующие на ассемблере.

**Методы сортировки**

Традиционно различают *внутреннюю*сортировку, в которой предполагается, что данные находятся в оперативной памяти, и важно оптимизировать число действий программы (для методов, основанных на сравнении, число сравнений, обменов элементов и пр.), и *внешнюю*, в которой данные хранятся на внешнем устройстве с медленным доступом (магнитные лента, барабан, диск) и прежде всего надо снизить число обращений к этому устройству.

В этой работе рассмотрим алгоритмы внутренней сортировки массивов элементов, так как они более важны большинству программистов в повседневной практике.

Стоит заметить, что алгоритм сортировки слиянием удобно применять и при сортировке внешних данных. При этом речь будет идти о слиянии файлов.

**О сложности алгоритмов**

При рассмотрении методов будем оперировать понятиями временной T и пространственной O теоретической сложности (в дальнейшем будем опускать слово «теоретическая») алгоритма, поэтому вкратце упомянем, что это такое.

*Сложность алгоритма* - одночлен, отражающий порядок величины требуемого ресурса (времени/дополнительной памяти) в зависимости от размерности задачи.

Например, если число тактов (действий), необходимое для работы метода, выражается как 11\*n2+19\*n\*log(n)+3\*n+4, где n-размерность задачи, то мы имеем дело с алгоритмом со сложностью T(n2). Фактически, мы из всех слагаемых оставляем только слагаемое, вносящее наибольший вклад при больших n (в этом случае остальными слагаемыми можно пренебречь) и игнорируем коэффициент перед ним.

В большинстве случаев оказывается довольно трудно найти точную практическую сложность алгоритма (функцию от n, позволяющую точно определить требуемое время работы/объем памяти). Однако, теоретическую сложность найти можно и это достаточно просто.

Временная (пространственная) сложность не позволяет определить время (необходимый объем дополнительной памяти) работы алгоритма, но она позволяет оценить динамику роста времени (объема дополнительной памяти), необходимого для работы метода, при увеличении размерности задачи. Например, для алгоритма с временной сложностью T(n2) при достаточно больших n можно утверждать, что при увеличении размера задачи (при сортировке - размера массива) в 3 раза время работы алгоритма увеличится в 32=9 раз.

Если операция выполняется за фиксированное число шагов, не зависящее от размера задачи, то принято писать T(1).

Надо помнить, что перед одночленом, отражающим сложность, стоит некоторый коэффициент C. Поэтому алгоритмы с одинаковой сложностью могут сильно отличаться временем исполнения.

Практически время выполнения алгоритма зависит не только от объема множества данных (размера задачи), но и от их значений, например, время работы некоторых алгоритмов сортировки значительно сокращается, если первоначально данные частично упорядочены, тогда как другие методы оказываются нечувствительными к этому свойству. Чтобы учитывать этот факт, полностью сохраняя при этом возможность анализировать алгоритмы независимо от данных, различают:

* *максимальную сложность*, или сложность наиболее неблагоприятного случая, когда метод работает дольше всего;
* *среднюю сложность* - сложность метода в среднем;
* *минимальную сложность* - сложность в наиболее благоприятном случае, когда метод справляется быстрее всего.

Алгоритмы, рассматриваемые здесь, имеют временные сложности T(n2), T(n\*log(n)), T(n). Минимальная сложность всякого алгоритма сортировки не может быть меньше T(n). Максимальная сложность метода, оперирующего сравнениями, не может быть меньше T(n\*log(n)). Доказательство последнего факта можно найти в многочисленной серьезной теоретической литературе, посвященной проблемам сортировки. Однако есть методы, которые не сравнивают элементы между собой. Мы рассмотрим один из них - сортировку распределением.

Для дальнейшего изложения нам понадобится следующее тождество:

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image162.gif, n m

*Доказательство:*

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image163.gif

Посмотрим, как получаются сложности, с которыми нам придется столкнутся при выполнении лабораторных работ этого учебного пособия. Рассмотрим их в порядке возрастания.

Сложность T(log(n)).

Из всех рассматриваемых здесь сложностей эта самая малая. Действительно, при достаточно большом n: log(n)<n<n\*log(n)<n2.

Эта сложность получается, если на каждом шаге метода выполняется некоторое постоянное число действий C и задача размерности n сводится к аналогичной задаче размерностью n/m, где m – целая положительная константа:

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image164.gif  http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image165.gif

*Доказательство:*

(При доказательстве этой и нижеследующих формул сложности используется метод математической индукции)

*F*(1)=С\*logm(1)=0;

Пусть http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image166.gif

Тогда http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image167.gif, что и требовалось доказать.

Примеры методов с такой сложностью вы можете найти в лабораторных работах по темам «Методы поиска», «Управление сбалансированными деревьями», «Управление b-деревьями».

Сложность T(n).

Такая сложность получается в обычных итерационных процессах, когда на каждом шаге метода задача размерности n сводится к задаче размерности n-1, и при этом на каждом шаге выполняется некоторое постоянное число действий C, не зависящее от n:

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image168.gif  *F*(n)=C\*(n-1)=C\*n-C

*Доказательство:*

*F*(1)=С\*(n-1)=0;

Пусть *F*(n-1)=C\*(n-2)

Тогда *F*(n)=C+*F*(n-1)=C+C\*(n-2)=C\*(1+n-2)=C\*(n-1), что и требовалось доказать.

Примером такой сложности может служит любой цикл, в котором число итераций прямо зависит от n. В этой работе такие методы можно найти при рассмотрении лабораторной работы по теме «Методы поиска». Также некоторые методы сортировки в лучшем случае (если массив уже отсортирован) выполняют порядка n действий, а сортировка распределением всегда требует порядка n действий.

Сложность T(n\*log(n)).

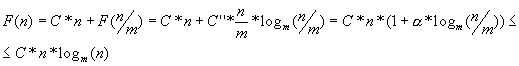
Эта сложность получается, если на каждом шаге метода выполняется некоторое количество действий C\*n, то есть зависящее от размера задачи прямопропорционально, и задача размерности n сводится к задаче размерностью n/m, где m – целая положительная константа:

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image169.gif  *F*(n)=C’\*n\*logm(n)

*Доказательство:*

*F*(1)=С\*logm(1)=0;

Пусть http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image170.gif

Тогда (0< <1), что и требовалось доказать.

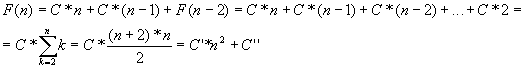
Примеры алгоритмов с такой сложностью можно найти в лабораторной работе, посвященной сортировкам.

Сложность T(n2).

Такая сложность получается, когда мы имеем два вложенных цикла, число итераций которых зависит от n прямопропорционально. Или другими словами, на каждом шаге метода задача размерности n сводится к задаче размерности n-1, и при этом выполняется некоторое количество действий C\*n, то есть зависящее от n прямо пропорционально:

http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image172.gif  *F*(n)=C’\*n2+C’’

*Доказательство:*

, что и требовалось доказать.

Примеры алгоритмов с квадратичной сложностью вы можете найти в лабораторной работе по теме «Методы сортировки». Это все простые и малоэффективные при больших n методы сортировок (в том числе получившая широкую известность благодаря рассмотрению во многих школьных учебниках информатики пузырьковая сортировка).

При рассмотрении алгоритмов сортировки прежде всего нас будет интересовать максимальная и средняя сложности метода, так как именно они, как правило, важны на практике. Максимальная сложность показывает поведение алгоритма в худшем случае, а средняя сложность – наиболее вероятное поведение при увеличении размера сортируемого массива. Сводные данные о сложности рассматриваемых методов можно найти в приложении 2 в конце работы.

Начнем разбор алгоритмов.



**Сортировка подсчетом**

Суть метода заключается в том, что на каждом шаге мы подсчитываем, в какую позицию результирующего массива надо записать очередной элемент исходного массива. Выглядит это так:

for i := 1 to N do

begin

<вычислить\_место\_k+1\_элемента\_A[i]

в\_результирующем\_массиве\_B>

B[k+1] := A[i];

end;

Слева от B[k+1] должны стоять элементы большие или равные B[k+1]. Значит число k складывается из количества элементов меньших A[i] и, возможно, некоторого числа элементов, равных A[i]. Условимся, что из равных мы будем учитывать только те элементы, которые в исходном массиве стоят левее A[i]. Теперь вычисление k можно записать следующим образом:

k := 0;

for j := 1 to N do

if (A[i]>A[j])or((A[i]=A[j])and(i>j)) then

Inc(k);

Легко видеть, что этот алгоритм всегда имеет сложности T(n2) (два вложенных цикла, зависящих от n линейно) и O(n) (результирующий массив).

**Сортировка включением**

В этой сортировке массив делится на 2 части: отсортированную и неотсортированную. На каждом шаге берется очередной элемент из неотсортированной части и «включается» в отсортированную часть массива.

Простое включение

Пусть отсортировано начало массива A[1], A[2], ..., A[i-1], а остаток массива A[i], ...,A[n] содержит неотсортированную часть. На очередном шаге будем включать элемент A[i] в отсортированную часть, ставя его на соответствующее место. При этом придется сдвинуть часть элементов, больших A[i], (если таковые есть) на одну позицию правее, чтобы освободить место для элемента A[i]. Но при сдвиге будет потеряно само значение A[i], поскольку в эту позицию запишется первый (самый правый - с самым большим индексом) сдвигаемый элемент. Поэтому прежде чем производить сдвиг элементов необходимо сохранить значение A[i] в промежуточной переменной.

Так как массив из одного элемента можно считать отсортированным, начнем с i=2.

Программа будет выглядеть так:

for i := 2 to n do

begin

Tmp := A[i];

j := i-1;

while (A[j]>Tmp)and(j>1) do

begin

A[j+1] := A[j];

Dec(j)

end;

A[j+1] := Tmp;

end;

Этот алгоритм также имеет максимальную и среднюю сложности T(n2), но в случае исходно отсортированного массива внутренний цикл не будет выполняться ни разу, поэтому метод имеет Tmin(n). Можно заметить, что метод использует любой частичный порядок, и чем в большей степени массив исходно упорядочен, тем быстрее он закончит работу. В отличие от предыдущего метода, этот не требует дополнительной памяти.

2.2.3.2. Метод Шелла

Метод Шелла является усовершенствованием *простого включения*, которое основано на том, что включение использует любой частичный порядок. Но недостатком простого включения является то, что во внутреннем цикле элемент A[i] фактически сдвигается на одну позицию. И так до тех пор, пока он не достигнет своего места в отсортированной части. (На самом деле мы избавлялись от сдвига A[i], сохраняя его в промежуточной переменной, но сути метода это не изменяло, так как передвигалось место, оставленное под сохраненное значение). Метод Шелла позволяет преодолеть это ограничение следующим интересным способом.

Вместо включения A[i] в подмассив предшествующих ему элементов, его включают в подсписок, содержащий элементы A[i-h], A[i-2h], A[i-3h] и так далее, где h - положительная константа. Таким образом формируется массив, в котором «h- серии» элементов, отстоящих друг от друга на h, сортируются отдельно.

Конечно, этого недостаточно: процесс возобновляется с новым значением h, меньшим предыдущего. И так до тех пор, пока не будет достигнуто значение h=1.

В настоящее время неизвестна последовательность hi, hi-1, hi-2, ..., h1, оптимальность которой доказана. Для достаточно больших массивов рекомендуемой считается такая последовательность, что hi+1=3hi+1, а h1=1. Начинается процесс с hm-2, где m - наименьшее целое такое, что hmn. Другими словами hm-2 первый такой член последовательности, что hm-2[n/9].

Теперь запишем алгоритм:

Step := 1;

while Step < N div 9 do Step := Step\*3 + 1;

Repeat

for k := 1 to Step do

begin

i := Step + k;

while i <= N do

begin

x := A^[i];

j := i - Step;

while (j >= 1) and (A^[j]>x) do

begin

A^[j + Step] := A^[j];

Dec(j);

end;

A^[j + Step] := x;

Inc(i, Step);

end;

end;

Step := Step div 3;

until Step=0;

Как показывают теоретические выкладки, которые мы приводить не будем, сортировке методом Шелла требуется в среднем 1,66n1,25 перемещений.

**Сортировка извлечением**

В этих методах массив также делится на уже отсортированную часть A[i+1], A[i+1], ..., A[n] и еще не отсортированную A[1], A[2], ..., A[i]. Но здесь из неотсортированной части на каждом шаге мы извлекаем максимальный элемент. Этот элемент будет минимальным элементом отсортированной части, так как все большие его элементы мы извлечем на предыдущих шагах, поэтому ставим извлеченный элемент в начало отсортированной части.

Простое извлечение

При простом извлечении мы будем искать максимальный элемент неотсортированной части, просматривая ее заново на каждом шаге. Найдя положение максимального элемента поменяем его с A[i] местами.

for i := N downto 2 do

begin

MaxIndex := 1;

for j := 2 to i do

if A[j]>A[MaxIndex] then

MaxIndex := j;

Tmp := A[i]; A[i] := A[MaxIndex];

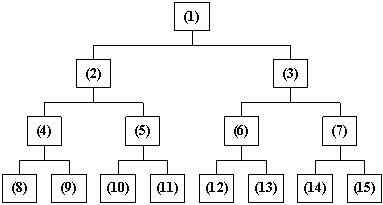
A[MaxIndex] := Tmp;

end;

Простое извлечение во всех случаях имеет временную сложность T(n2) (два вложенных цикла, зависящих от n линейно).

Древесная сортировка

Древесная сортировка избегает необходимости каждый раз находить максимальный элемент. При следовании этому методу постоянно поддерживается такой порядок, при котором максимальный элемент всегда будет оказываться в A[1]. Сортировка называется древесной, потому что в этом методе используется структура данных, называемая *двоичным деревом*.



Двоичное дерево имеет элемент, называемый *корнем*. Корень, как и любой другой элемент дерева (называемый *вершиной*), может иметь до двух элементов-сыновей. Все элементы, кроме корневого имеют элемент-родитель.

Условимся нумеровать вершины числами 1, 2, ...: сыновьями вершины с номером k будут вершины с номерами 2\*k и 2\*k+1, - как это сделано на рисунке. Как можно заметить, двоичное разложение номера вершины указывает путь к этой вершине от корня (0 - к левому сыну, 1 - к правому сыну). Например, 910=10012. Первую 1 двоичного разложения пропускаем (в начале всегда стоит 1). Затем идет 0, то есть от корня переходим к левому сыну (вершина номер 210=102), потом снова 0 - переходим к левому сыну (вершина номер 410=1002), последней следует цифра 1 - переходим к правому сыну и как раз попадаем в вершину номер 910=10012. Значит номер элемента однозначно определяет его положение в дереве.

Введенный способ нумерации вершин дерева позволяет хранить деревья в массиве, где индекс массива будет номером вершины.

Пусть A[1]..A[n] - массив, подлежащий сортировке. Вершинами дерева будут числа от 1 до n; о числе A[i] мы будем говорить как о числе, стоящем в вершине i. В процессе сортировки количество вершин дерева будет сокращаться. Число вершин текущего дерева будем хранить в переменной k. Таким образом, в процессе работы алгоритма массив A[1]..A[n] делится на две части: в A[1]..A[k] хранятся числа на дереве, а в A[k+1] .. A[n] хранится уже отсортированная в порядке возрастания часть массива - элементы, уже занявшие свое законное место.

На каждом шаге алгоритм будет изымать максимальный элемент дерева и помещать его в отсортированную часть, на освободившееся в результате сокращения дерева место.

Договоримся о терминологии. Вершинами дерева считаются числа от 1 до текущего значения переменной k. У каждой вершины s могут быть сыновья 2\*s и 2\*s+1. Если оба этих числа больше k, то сыновей нет; такая вершина называется *листом*. Если 2\*s=k, то вершина s имеет ровно одного сына (2\*s).

Для каждого s из 1..k рассмотрим "поддерево" с корнем в s: оно содержит вершину s и всех ее потомков (сыновей, сыновей сыновей и т.д. - до тех пор, пока мы не выйдем из отрезка 1..k).

Вершину s будем называть *регулярной*, если стоящее в ней число - максимальный элемент s-поддерева; s-поддерево назовем *регулярным*, если все его вершины регулярны. (В частности, любой лист образует регулярное одноэлементное поддерево.)

Схема алгоритма такова:

k:= n

... Сделать 1-поддерево регулярным;

while k<>1 do

begin

... обменять местами A[1] и A[k];

k := k - 1;

{A[1]..A[k-1] <= A[k] <=...<= A[n];

1-поддерево регулярно везде, кроме, возможно,

самого корня }

... восстановить регулярность 1-поддерева всюду

end;

В качестве вспомогательной процедуры нам понадобится процедура восстановления регулярности s-поддерева в корне. Вот она:

{s-поддерево регулярно везде, кроме, возможно, корня}

t := s;

{s-поддерево регулярно везде, кроме, возможно,

вершины t}

while ((2\*t+1<=k) and (A[2\*t+1]>A[t])) or

((2\*t<=k) and (A[2\*t]>A[t])) do

begin

if (2\*t+1<=k) and (A[2\*t+1]>=A[2\*t]) then

begin

... обменять A[t] и A[2\*t+1];

t := 2\*t + 1;

end else

begin

... обменять A[t] и A[2\*t];

t := 2\*t;

end;

end;

Чтобы убедиться в правильности этой процедуры, посмотрим на нее повнимательнее. Пусть в s-поддереве все вершины, кроме разве что вершины t, регулярны. Рассмотрим сыновей вершины t. Они регулярны, и потому содержат наибольшие числа в своих поддеревьях. Таким образом, на роль наибольшего числа в t-поддереве могут претендовать число в самой вершине t и числа, содержащиеся в ее сыновьях. (В первом случае вершина t регулярна, и все в порядке.) В этих терминах цикл можно записать так:

while наибольшее число не в t, а в одном из сыновей do

begin

if оно в правом сыне then

begin

...поменять t с ее правым сыном; t:= правый сын

end else

begin {наибольшее число - в левом сыне}

...поменять t с ее левым сыном; t:= левый сын

end

end

После обмена вершина t становится регулярной (в нее попадает максимальное число t-поддерева). Не принявший участия в обмене сын остается регулярным, а принявший участие может и не быть регулярным. В остальных вершинах s-поддерева не изменились ни числа, ни поддеревья их потомков (разве что два элемента поддерева переставились), так что регулярность не нарушилась.

Эта же процедура может использоваться для того, чтобы сделать 1-поддерево регулярным на начальной стадии сортировки:

k := n;

u := n;

{все s-поддеревья с s>u регулярны }

while u<>0 do

begin

{u-поддерево регулярно везде, кроме разве что корня}

... восстановить регулярность u-поддерева в корне;

u:=u-1;

end;

Теперь запишем процедуру сортировки на паскале:

procedure Sort;

Var u, k: Integer;

procedure Exchange(i, j: Integer);

Var Tmp: Integer;

begin

Tmp := x[i]; A[i] := A[j]; A[j] := Tmp;

end;

procedure Restore(s: Integer);

Var t: Integer;

begin

t:=s;

while ((2\*t+1<=k) and (A[2\*t+1]>A[t])) or

((2\*t<=k) and (A[2\*t]>A[t])) do

begin

if (2\*t+1<=k) and (A[2\*t+1]>=A[2\*t]) then

begin

Exchange(t, 2\*t+1);

t := 2\*t+1;

end else

begin

Exchange(t, 2\*t);

t := 2\*t;

end;

end;

end;

begin

k:=n;

u:=n;

while u<>0 do

begin

Restore(u);

Dec(u);

end;

while k<>1 do

begin

Exchange(1, k);

Dec(k);

Restore(1);

end;

end;

Преимущество этого метода перед другими в том, что он, имея Tmax(n\*log(n)) (внутри внешнего цикла зависящего от n линейно вызывается процедура Restore, требующая порядка log(n) действий), при сортировке не требует дополнительной памяти порядка n.

**Сортировка обменами**

Как следует из названия метода он сортирует массив путем последовательных обменов пар элементов местами.

Пузырьковая сортировка

Пузырьковая сортировка - один из наиболее широко известных алгоритмов сортировки. В этом методе массив также делится на две части: отсортированную и неотсортированную. На каждом шаге метода мы пробегаем от меньших индексов к большим по неотсортированной части, каждый раз сравнивая два соседних элемента: если они не упорядочены между собой (меньший следует за большим), то меняем их местами. Тем самым за один проход путем последовательных обменов наибольший элемент неотсортированной части сдвинется к ее концу.

Алгоритм называют пузырьковой сортировкой, потому что на каждом шаге наибольший элемент неотсортированной части подобно пузырьку газа в воде всплывает вверх.

Запишем алгоритм на языке Паскаль, заметив, что в том случае, когда за очередной проход не было сделано ни одного обмена, массив уже отсортирован и следующие проходы можно пропустить. Для отслеживания такой ситуации введем логическую переменную Flag - признак совершения обмена на очередном проходе:

for i := N-1 downto 1 do

begin

Flag := false;

for j := 1 to i do

if a[j]>a[j+1] then

begin

Tmp := A[j]; A[j] := A[j+1]; A[j+1] := Tmp;

Flag := true;

end;

if not Flag then Break;

end;

Этот алгоритм имеет среднюю и максимальную временные сложности T(n2) (два вложенных цикла, зависящих от n линейно) и не требует дополнительной памяти за исключением памяти для фиксированного числа переменных (i, j, Tmp, Flag). Введение переменной Flag и прерывание работы в случае отсортированного массива позволяет свести минимальную временную сложность к T(n).

Быстрая сортировка (Хоара)

Эту сортировку называют быстрой, потому что на практике она оказывается самым быстрым методом сортировки из тех, что оперируют сравнениями.

Этот метод является ярким примером реализации принципа «разделяй и властвуй». Как показывают теоретические выкладки наиболее эффективным в общем случае оказывается разделение задачи на две равные по сложности части, что мы и будем пытаться делать в этом методе.

На каждом шаге метода мы сначала выбираем «средний» элемент, затем переставляем элементы массива так, что он делится на три части: сначала следуют элементы, меньшие «среднего», потом равные ему, а в третьей части - большие. После такого деления массива остается только отсортировать первую и третью его части, с которыми мы поступим аналогично (разделим на три части). И так до тех пор, пока эти части не окажутся состоящими из одного элемента, а массив из одного элемента всегда отсортирован.

Запишем то, что мы сейчас сформулировали:

procedure Sort;

procedure QuickSort(a, b: Longint);

Var

x: Longint; {индекс «среднего» элемента}

begin

x := элемент\_выбранный\_средним;

...деление на три части:

1) A[a]..A[i]; 2) A[i+1]..A[j-1]; 3) A[j]..A[b]

if i>a then QuickSort(a, i);

if b>j then QuickSort(j, b);

end;

begin

Sort(1, N);

end;

Выбор «среднего» - задача непростая, так как требуется, не производя сортировку, найти элемент со значением максимально близким к среднему. Здесь, конечно, можно просто выбрать произвольный элемент (обычно выбирают элемент, стоящий в середине сортируемого подмассива), но пойдем чуть дальше: из трех элементов (самого левого, самого правого и стоящего посередине) выберем средний:

i := A[l]; y := A[(l+r)div 2]; j := A[r];

if i<=y then

if y<=j then

x := (l+r)div 2

else

if i<=j then

x := r

else

x := l

else

if y>=j then

x := (l+r)div 2

else

if i>=j then

x := r

else

x := l;

Теперь нам надо разделить массив относительно элемента A[x] так, чтобы сначала следовали элементы, меньшие A[x], затем сам элемент A[x], а потом уже элементы, большие или равные A[x]. Для этого мы, двигаясь слева найдем первый элемент A[i], больший A[x], и, двигаясь справа, - первый элемент A[j], меньший A[x]. Если i окажется меньше j, то это значит, что массив еще не поделен на три искомых части, в таком случае обменяем местами элементы A[i] и A[j], а затем продолжим поиск слева от A[i] и справа от A[j].

i := l; j := r; x := A[x];

repeat

while A[i] < x do Inc(I);

while x < A[j] do Dec(j);

if i <= j then

begin

y := A[i]; A[i] := A[j]; A[j] := y;

Inc(i); Dec(j);

end;

until i > j;

Пойдем дальше. Заметим, что все-таки наш способ нахождения «среднего» элемента подмассива в худшем случае приведет к тому, что после деления, например, правая и средняя части поделенного массива будут содержать по одному элементу, а левая все остальные. В этом случае, если мы сначала вызовем QuickSort для левой части, то (опять же в худшем случае) это может породить порядка N рекурсивных вызовов. А значит нам надо будет завести дополнительную память размером пропорциональным N и пространственная сложность будет O(N). Этого можно избежать, если сначала вызывать QuickSort для меньшей части. Тогда можно гарантировать пространственную сложность O(log(N)).

Теперь осталось только собрать вместе части программы:

procedure Sort;

procedure QuickSort(a, b: Longint);

Var

x: Longint; {индекс «среднего» элемента}

begin

i := A[l]; y := A[(l+r)div 2]; j := A[r];

if i<=y then

if y<=j then

x := (l+r)div 2

else

if i<=j then

x := r

else

x := l

else

if y>=j then

x := (l+r)div 2

else

if i>=j then

x := r

else

x := l;

i := l; j := r; x := A[x];

repeat

while A[i] < x do Inc(i);

while x < A[j] do Dec(j);

if i <= j then

begin

y := A[i]; A[i] := A[j]; A[j] := y;

Inc(i); Dec(j);

end;

until i > j;

if l-j<i-r then

begin

if l < j then QuickSort(l, j);

if i < r then QuickSort(i, r);

end else

begin

if i < r then Sort(i, r);

if l < j then Sort(l, j);

end;

end;

begin

QuickSort(1, N);

end;

В худшем случае этот алгоритм дает временную сложность Tmax(N2) (для случая, когда все выборки «среднего» элемента оказались неудачны), но как показывают теоретические исследования вероятность такого случая очень мала. В среднем же и в лучшем

**Сортировка слиянием**

Этот метод сортирует массив последовательным слиянием пар уже отсортированных подмассивов, поэтому его и назвали сортировкой слиянием.

Пусть k - положительное целое число. Разобьем массив A[1]..A[n] на отрезки длины k. (Первый - A[1]..A[k], затем A[k+1]..A[2k] и т.д.) Последний отрезок будет неполным, если n не делится нацело на k. Назовем массив k-упорядоченным, если каждый из этих отрезков длины k упорядочен.

Ясно, что любой массив 1-упорядочен, так как его подмассивы длиной 1 можно считать упорядоченными. Если массив k-упорядочен и n<=k, то он упорядочен.

Теперь запишем общую схему алгоритма:

k:=1;

while k < n do

begin

... преобразовать k-упорядоченный массив

в 2k-упорядоченный;

k := 2 \* k;

end;

Рассмотрим процедуру преобразования k-упорядоченного массива в 2k-упорядоченный. Сгруппируем все подмассивы длины k в пары подмассивов. Теперь пару упорядоченных подмассивов сольем в один упорядоченный подмассив. Проделав это со всеми парами, мы получим 2k-упорядоченный массив:

t:=0;

while t + k < n do

begin

p := t;

q := t+k;

...r := min(t+2\*k, n); {в паскале нет функции min }

...сливаем два подмассива A[p..q-1] и A[q..r]

t := r;

end;

Слияние требует вспомогательного массива для записи результатов слияния - обозначим его B. Через p0 и q0 обозначим номера последних элементов участков, подвергшихся слиянию, s0 - последний записанный в массив B элемент. На каждом шаге слияния производится одно из двух действий:

B[s0+1]:=A[p0+1];

Inc(p0);

Inc(s0);

или

B[s0+1]:=A[q0+1];

Inc(q0);

Inc(s0);

Первое действие (взятие элемента из первого отрезка) может производиться при двух условиях:

(1) первый отрезок не кончился (p0 < q);

(2) второй отрезок кончился (q0 = r) или не кончился, но элемент в нем не меньше [(q0 < r) и (A[p0+1] <= A[q0+1])].

Аналогично для второго действия. Итак, получаем

p0 := p; q0 := q; s0 := p;

while (p0<>q)or(q0<>r) do

begin

if (p0<q)and((q0=r)or((q0 < r) and

(A[p0+1]<=A[q0+1]))) then

begin

B[s0+1] := A[p0+1]; Inc(p0); Inc(s0);

end else

begin

B[s0+1] := A[q0+1]; Inc(q0); Inc(s0);

end;

end;

(Если оба отрезка не кончены и первые невыбранные элементы в них равны, то допустимы оба действия; в программе выбрано первое.)

Осталось собрать программу в одно целое:

k := 1;

while k < N do

begin

t := 0;

while t+k < N do

begin

p := t; q := t+k;

if t+2\*k>N then r := N else r := t+2\*k;

p0 := p; q0 := q; s0 := p;

while (p0<>q) or (q0<>r) do

begin

if (p0<q) and ((q0=r)or((q0<r)and

(A[p0+1]<=A[q0+1]))) then

begin

B[s0+1] := A[p0+1];

Inc(p0);

end else

begin

B[s0+1] := A[q0+1];

Inc(q0);

end;

Inc(s0);

end;

t := r;

end;

k := k shl 1;

A := B;

end;

Сразу же бросается в глаза недостаток метода – он требует дополнительную память размером порядка N (для хранения вспомогательного массива). Но как и для древесной сортировки, его временная сложность всегда пропорциональна N\*log(N) (так как преобразование k-упорядоченного массива в 2k-упорядоченный требует порядка N действий и внешний цикл по k совершает порядка log(N) итераций).

**Сортировка распределением**

Сортировка распределением интересна тем, что она сортирует массив, не сравнивая элементы друг с другом.

Рассмотрим сначала вырожденный случай сортировки распределением, а затем более общий.

Вырожденное распределение

Пусть каждый элемент массива может принимать M (например, от 1 до M) фиксированных значений. Заведем массив Amount, первоначально обнулив его. Затем для каждого i подсчитаем количество элементов массива A, равных i, и занесем это число в Amount[i]:

for i := 0 to M do Amount[i] := 0;

for i := 1 to N do

Inc(Amount[A[i]]);

Теперь в первые Amount[1] элементов массива A запишем 1, в следующие Amount[2] элементов массива A запишем 2 и т.д. до тех пор, пока не дойдем до конца массива A (заметим, что в то же время мы окажемся в конце массива Amount):

p := 1;

for i := 0 to M do

for j := 1 to Amount[i] do

begin

A[p] := i;

Inc(p);

end;

Несмотря на то, что здесь два внешних цикла на этом участке алгоритма выполняется порядка N действий. Это следует из того, что http://altim.narod.ru/Docs/Russian/Manuals/Lab/Chapter1/Image175.gif. Поэтому временную сложность метода можно оценить как T(M+N) (M появляется в сумме, так как изначально надо обнулить массив Amount, а это требует M действий). Пространственная сложность в этом случае равна O(M), поскольку требуется дополнительная память размером порядка M.

Недостатком этого метода является то, что требуется дополнительная память размером порядка M, а это может оказаться недопустимым из-за большого значения M. Но, если M>>N, то есть способ уменьшить объем требуемой дополнительной памяти, который мы сейчас и рассмотрим.

Сортировка распределением

Пусть мы можем завести дополнительную память размером M+N, а элементы массива могут принимать значения от 0 до S, причем S>>M.

Каждый элемент этого массива можно представить в M-ичной системе счисления и разбить на K цифр этой системы счисления.

Заведем M списков общей суммарной длиной порядка N (это можно сделать, ограничившись дополнительной памятью O(M+N)).

Наш алгоритм можно представить следующим образом:

for i := K downto 1 do

begin

for j := 1 to N do

begin

L:=<i-ой цифре A[j] в M-ной системе счисления>;

... занести A[j] в L-ный список;

end;

... очистить массив A

for j := 1 to M do

... дописать элементы j-ого списка в массив A

end;

Итак, как видно из приведенной выше программы, на каждом шаге метода производится сортировка элементов массива по значению i-ого разряда. При этом производится промежуточное распределение элементов массива по спискам в зависимости от значения соответствующего разряда этих элементов. Во время распределения очень важно сохранить при записи в списки порядок следования элементов, чтобы не нарушить порядок, достигнутый на предыдущих шагах.

Индукцией по i легко доказать, что после i шагов любые два числа, отличающиеся только в i последних разрядах, идут в правильном порядке.

Достигнув i=1, мы получим полностью отсортированный массив.

Как нетрудно заметить, если положить S=M, то отпадает необходимость заводить списки и производить запись в них: в j–ый список будут попадать только числа, равные j. В этом случае достаточно хранить лишь размеры списков, то есть подсчитать количество элементов, равных j, для всех j от 1 до S. А потом просто заново заполнить массив A в соответствии с этими количествами. Что мы и делали в случае вырожденной сортировки.

Интересно, что временная сложность этого метода T(K\*N), а если учесть, что K фактически является константой, то мы получаем гарантированную (минимальную, среднюю и максимальную) линейную сложность. Но недостатком этого метода является необходимость выделять дополнительную память размером порядка M+N. Если бы не это ограничение, можно было бы считать этот метод самым эффективным при больших значениях N.

**Сравнительный анализ**

Для проведения экспериментального сравнительного анализа различных методов сортировки была разработана программа, которая в автоматическом режиме подсчитывала время, требуемое для каждого метода сортировки. Для чистоты эксперимента сортировка всеми методами проводилась на одинаковых наборах входных данных, затем формировался новый набор данных, и они опять подвергались сортировке различными методами. Таким образом было выполнено 60 циклов сортировки, подсчитано среднее время, которое потребовалось каждому методу, чтобы отсортировать входной массив. Для более полного анализа методов в каждом цикле сортировки сортировка проводилась для размерностей 500, 1000, 3000, 5000, 8000, 10000, 30000, 60000. Это дало возможность проследить динамику роста требуемого для сортировки времени. Также проверялось, как ведут себя методы на различных входных данных: упорядоченных в прямом порядке, упорядоченных в обратном порядке и случайных.

Ниже приведены таблицы, которые выдала составленная программа.

а) Для массива, заполненного случайными элементами:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 3000 | 5000 | 8000 | 10000 | 30000 | 60000 |
| Подсчетом | 0.08 | 0.31 | 2.76 | 7.67 | 19.67 |  |  |  |
| Включением | 0.03 | 0.06 | 0.60 | 1.66 | 4.22 | 6.58 | 59.62 |  |
| Шелла | 0.02 | 0.05 | 0.43 | 1.12 | 2.72 | 4.24 | 35.62 |  |
| Слиянием | 0.03 | 0.02 | 0.06 | 0.09 | 0.11 | 0.13 | 0.38 | 0.77 |
| Извлечением | 0.01 | 0.10 | 0.92 | 2.54 | 6.51 | 10.17 |  |  |
| Древесная | 0.02 | 0.02 | 0.11 | 0.20 | 0.33 | 0.42 | 1.39 | 3.02 |
| Пузырьковая | 0.07 | 0.27 | 2.43 | 6.75 | 17.28 |  |  |  |
| Быстрая | 0.00 | 0.00 | 0.01 | 0.03 | 0.03 | 0.04 | 0.13 | 0.27 |
| Двоичным распределением | 0.00 | 0.00 | 0.02 | 0.03 | 0.05 | 0.06 | 0.20 | 0.38 |
| Вырожденным распределением | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 |

б) Для исходно упорядоченного массива:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 3000 | 5000 | 8000 | 10000 | 30000 | 60000 |
| Подсчетом | 0.08 | 0.32 | 2.65 | 7.26 | 18.46 |  |  |  |
| Включением | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.03 |
| Шелла | 0.00 | 0.00 | 0.01 | 0.02 | 0.02 | 0.05 | 0.15 | 0.29 |
| Слиянием | 0.02 | 0.02 | 0.04 | 0.08 | 0.09 | 0.12 | 0.31 | 0.63 |
| Извлечением | 0.02 | 0.11 | 0.92 | 2.54 | 6.53 | 10.20 |  |  |
| Древесная | 0.01 | 0.03 | 0.11 | 0.19 | 0.34 | 0.43 | 1.45 | 3.11 |
| Пузырьковая | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 |
| Быстрая | 0.00 | 0.00 | 0.01 | 0.02 | 0.03 | 0.04 | 0.11 | 0.22 |
| Двоичным распределением | 0.00 | 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.18 | 0.36 |
| Вырожденным распределением | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.02 |

в) Для массива исходно упорядоченного в обратном порядке:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 500 | 1000 | 3000 | 5000 | 8000 | 10000 | 30000 | 60000 |
| Подсчетом | 0.09 | 0.32 | 2.64 | 7.26 | 18.47 |  |  |  |
| Включением | 0.02 | 0.13 | 1.16 | 3.26 | 8.37 | 13.08 |  |  |
| Шелла | 0.01 | 0.00 | 0.03 | 0.16 | 0.21 | 0.27 | 2.09 | 8.02 |
| Слиянием | 0.02 | 0.03 | 0.06 | 0.06 | 0.10 | 0.11 | 0.32 | 0.64 |
| Извлечением | 0.02 | 0.11 | 0.91 | 2.54 | 6.51 | 10.19 |  |  |
| Древесная | 0.00 | 0.02 | 0.10 | 0.18 | 0.30 | 0.39 | 1.32 | 2.85 |
| Пузырьковая | 0.06 | 0.29 | 2.95 | 8.35 | 21.60 |  |  |  |
| Быстрая | 0.00 | 0.00 | 0.01 | 0.01 | 0.03 | 0.04 | 0.11 | 0.24 |
| Двоичным распределением | 0.01 | 0.01 | 0.03 | 0.03 | 0.05 | 0.07 | 0.19 | 0.37 |
| Вырожденным распределением | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.01 | 0.03 |

Примечание. Пусты ячейки, для которых испытание не проводилось. Время засекалось с точностью до одной пятидесятой секунды, поэтому нули в таблице означают, что было затрачено меньше времени, чем одна пятидесятая доля секунды. Испытание проводилось на AMD-K5-PR133/32Mb EDO/850Mb под управлением MS-DOS 7.0, входящую в состав WINDOWS 95. Массивы размещались в динамической памяти.

Теоретические сложности рассмотренных методов сортировки:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tmax | Tmid | Tmin | Omax |
| Подсчетом | n2 | | | n |
| Включением | n2 | | n | 1 |
| Шелла | n2 | n1,25 | n | 1 |
| Извлечением | n2 | | | 1 |
| Древесная | n\*log(n) | | | 1 |
| Пузырьковая | n2 | | n | 1 |
| Быстрая | n2 | n\*log(n) | | log(n) |
| Слиянием | n\*log(n) | | | n |
| Распределением | n | | | n |

Эти таблицы позволяют сделать ряд выводов.

На небольших наборах данных целесообразнее использовать сортировку включением, так как из всех методов, имеющих очень простую программную реализацию, этот на практике оказывается самым быстрым и при размерностях<~3000 дает вполне приемлемую для большинства случаев скорость работы. Еще одно преимущество этого метода заключается в том, что он использует полную или частичную упорядоченность входных данных и на упорядоченных данных работает быстрее, а на практике данные, как правило, уже имеют хотя бы частичный порядок.

Алгоритм пузырьковой сортировки, причем в той его модификации, которая не использует частичный порядок данных исходного массива, хотя и часто используется, но имеет плохие показатели даже среди простых методов с квадратичной сложностью.

Сортировка Шелла оказывается лишь красивым теоретическим методом, потому что на практике использовать его нецелесообразно: он сложен в реализации, но не дает такой скорости, какую дают сравнимые с ним по сложности программной реализации методы.

При сортировке больших массивов исходных данных лучше использовать быструю сортировку.

Если же добавляется требование гарантировать приемлемое время работы метода (как вы помните, быстрая сортировка в худшем случае имеет сложность T(N2), хотя вероятность такого случая очень мала), то надо применять либо древесную сортировку, либо сортировку слиянием. Как видно из таблиц, сортировка слиянием работает быстрее, но следует помнить, что она требует дополнительную память размером порядка N.

В тех же случаях, когда мы можем себе позволить использовать дополнительную память размером порядка n, имеет смысл воспользоваться сортировкой распределением.

**Контрольные вопросы**

1. Что такое *сортировка*? Чем отличаются *внешняя*и *внутренняя*сортировка?
2. Что такое *практическая и теоретическая сложности*? Можно ли из практической сложности вывести теоретическую? Можно ли из теоретической сложности вывести практическую?
3. Что такое *максимальная*, *средняя* и *минимальная сложности*?
4. Что означает T(1)?
5. Общее в методах сортировки включением (сортировки включением и метода Шелла)?
6. Доказать, что метод Шелла действительно сортирует массив.
7. Общее в методах сортировки извлечением (сортировки извлечением и древесная)?
8. Описать способ хранения двоичного дерева, используемый в древесной сортировке.
9. Что такое регулярная вершина дерева? регулярное поддерево?
10. Доказать, что древесная сортировка действительно сортирует массив.
11. Почему нельзя сделать так, чтобы быстрая сортировка давала гарантированную сложность T(n\*log(n))? (Подсказка: причина в алгоритме выбора среднего элемента)
12. Как мы можем гарантировать пространственную сложность O(log(n)) при реализации метода быстрой сортировки?
13. Доказать, что сортировка слиянием действительно сортирует массив.
14. Доказать, что сортировка распределением действительно сортирует массив.

**Методические указания.**

Перед выполнением индивидуального задания ознакомиться с понятиями временной и пространственной сложности, методами сортировки, их сравнительным анализом.

При выполнении индивидуального задания придерживаться следующей последовательности действий:

* изучить словесную постановку задачи;
* выбрать метод сортировки, который лучше всего подходит для решения поставленной задачи;
* разработать программу, решающую поставленную задачу;
* оттестировать и отладить программу;
* написать и представить к защите отчет по работе.

**Содержание отчета**

* Тема и цель
* Словесная постановка задачи.
* Алгоритм решения задачи в текстуальном виде.
* Обоснование правильности работы алгоритма и правильности выбора метода сортировки исходя из постановки задачи.
* Листинг программы.
* Ответы на контрольные вопросы по согласованию с преподавателем.

**Варианты индивидуальных заданий**

Сортировка имеет множество практических применений: прямых, когда прямо ставится задача отсортировать данные; и косвенных, когда при решении какой-либо задачи требуется промежуточная сортировка данных. Здесь будут приведены ряд задач, при решении которых удобно использовать сортировку.

Во всех задачах надо обязательно показать, почему Ваш алгоритм обеспечивает заданную временную сложность.

Задача 1. Найти количество различных чисел среди элементов данного массива. Обеспечить число действий порядка n\*log n.

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

Задача 2. Найти k-ое по порядку число среди элементов данного массива. Обеспечить число действий порядка n\*log n.

*Указание.* Отсортировать массив, а затем взять число, хранящееся в элементе массива с индексом k.

Задача 3. Дано n отрезков [A[i], B[i]] на прямой (i=1..n), где A[i] – одномерная координата начала отрезка, а B[i] – конца отрезка.

Найти максимальное k, для которого существует точка прямой, покрытая k отрезками ("максимальное число слоев"). Обеспечить число действий - порядка n\*log n.

*Указание.* Упорядочим все левые и правые концы отрезков вместе (при этом левый конец считается меньше правого конца, расположенного в той же точке прямой). Далее двигаемся 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).

Задача 4. Дано n точек на плоскости. Указать (n-1)-звенную несамопересекающуюся незамкнутую ломаную, проходящую через все эти точки. (Соседним отрезкам ломаной разрешается лежать на одной прямой.) Обеспечить число действий порядка n\*log n.

*Указание.* Упорядочим точки по x-координате, а при равных x-координатах - по y-координате. В таком порядке и можно проводить ломаную.

Задача 5. Та же задача, если ломаная должна быть замкнутой.

*Указание.* Возьмем самую левую точку (т.е. точку с наименьшей x-координатой) и проведем из нее лучи во все остальные точки. Теперь упорядочим эти лучи, а точки на одном луче поместим в порядке увеличения расстояния от начала луча.

Задача 6. Дано n точек на плоскости. Построить их выпуклую оболочку - минимальную выпуклую фигуру, их содержащую. (Форму выпуклой оболочки примет резиновое колечко, если его натянуть на гвозди, вбитые в точках.) Обеспечить число операций порядка n\*log n.

*Указание.* Упорядочим точки - годится любой из порядков, использованных в двух предыдущих задачах. Затем, рассматривая точки по очереди, будем строить выпуклую оболочку уже рассмотренных точек. (Для хранения выпуклой оболочки полезно использовать дек – смотрите главу «Стеки, очереди, деки»)

Задача 7. Дан массив, состоящий из чисел 0, 1 и 2. Переместить все 0 в начало массива, а 2 – в конец. Обеспечить число действий порядка n.

*Указание.* Воспользоваться вырожденной сортировкой распределением.

Задача 8. В неупорядоченном массиве А могут быть совпадающие элементы. Из каждой группы одинаковых элементов оставить только один, удалив остальные и «поджав» массив к его началу. Обеспечить число операций порядка n\*log n.

*Указание.* Можно сначала отсортировать массив, а затем произвести его «поджатие» с удалением повторяющихся. При удалении очередного повтора не надо сразу сдвигать весь массив (это может привести к сложности T(n2)) – достаточно, рассматривая массив поэлементно и помня последний рассмотренный элемент, либо пропускать очередной, либо приписывать его к уже просмотренной части.

Задача 9. Турнирная таблица соревнований представлена квадратной матрицей A, каждый элемент которой aij есть число голов, забитых i-ой командой в ворота j-ой команды. По диагонали расположить место каждой команды (по числу побед за вычетом числа поражений; в случае равенства – по разности забитых и пропущенных голов).

Задача 10. В целочисленном массиве найти наибольшее число одинаковых элементов.

Указание. Удобно предварительно отсортировать массив.

Задача 11. Отсортировать список по неубыванию. Гарантировать число действий порядка n. Элементы списка могут содержать числа от 1 до 1000000000.

Указание. Воспользоваться сортировкой распределением.

# Практическое занятие № 9-10 Определение сложности алгоритмов

**Цель занятия**: научиться оценивать сложность алгоритмов

**Норма времени**:4 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

Для решения той или иной задачи может существовать несколько алгоритмов. Вопрос выбора конкретного алгоритма является одним из важнейших вопросов программирования.

В общем случае конечный пользователь программного обеспечения выбирает приложение, которое является максимально эффективным: во-первых, потребляет предельно малое количество системных ресурсов (память, процессорное время), во-вторых, решает вычислительную задачу за наименьшее время. Таким образом, при разработке алгоритма необходимо дать ему оценку, т.е. представить количество потребляемых ресурсов и времени работы при решении вычислительной задачи.

Часто, в практике создания программного обеспечения, применяется практический подход, согласно которому реализованное ПО тестируется на конкретной программно-аппаратной платформе (например, корпоративный стандарт компании, стандратный офисный компьютер и стандартный набор офисных приложений). В результате данного тестирования производится мониторинг потребления памяти, количества процессорного времени и ведется учет времени, в процессе которого выполняется вычислительная задача. Далее, если полученные результаты удовлетворяют конечного пользователя (или, например, заказчика, если данные параметры были оговорены в техническом задании), установлено, что функционирование данного ПО не существенно (допустимо) влияет на другое программное обеспечение и т.д., то ПО принимается к использованию.

Такой подход вполне находит свое применение в практике создания корпоративных и пользовательских приложений, однако в теории программирования принято давать более строгую оценку “качества” алгоритма.

Данной оценкой является математическая оценка количества операций, или тредоемкость (сложность) алгоритма в зависимости от количества входных данных или длины входной последовательности.

Рассмотрим алгоритм нахождения значения многочлена в точке *x*:

= + + ... +  + ... + + .

Для вычисления трудоемкости решения данной задачи может быть предложен следующий алгоритм: для каждого i-го слагаемого (кроме ) возвести x в степень последовательным умножением, а затем домножить на коэффициент. Для вычисления i-го (i=1..n) слагаемого требуется

 =

операций умножения, кроме того требуется  - сложение, таким образом для решения данной задачи потребуется выполнение

= + + 1

элементарных операций.

Как правило, подобная оценка в определении точного числа элементарных операций не требуется. Вместо нее приводят асимптотическую скорость возрастания количества операций в зависимости от значения n.

Функция приблизительно возрастает как (отбрасываем сравнительно медленно возрастающаю часть ). Далее, избавляемся от константного множителя , так как он не играет существенной роли. Получаем следующую асимптотическую оценку: функция  возрастает приблизительно как . Записывается как  (“О-большое от эн-квадрат”).

Таким образом, мы получили верхнюю оценку, т.е. количество операций (от которого прямо пропорционально зависит время работы) растет не быстрее, чем квадрат количества входных элементов.

Далее приведена таблица скорости роста некоторых основных функций:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 | 0 | 0 | 1 |
| 16 | 4 | 64 | 256 |
| 256 | 8 | 2,048 | 65,536 |
| 4,096 | 12 | 49,152 | 16,777,216 |
| 65,536 | 16 | 1,048,565 | 4,294,967,296 |
| 1,048,576 | 20 | 20,969,520 | 1,099,301,922,576 |
| 16,775,616 | 24 | 402,614,784 | 281,421,292,179,456 |

Представим себе, что значения в столбцах таблицы - это значение в миллисекундах. Т.е. для  алгоритму с временем работы  для решения данной задачи потребуется порядка 20мс, алгоритму с временем работы  порядка 16 минут, алгоритму со сложностью ) - порядка 13 дней.

Наилучшей оценкой является , т.е. количество операций не зависит от длины входа. Далее приводятся два основных правила выполнения расчета оценки :

При оценке за функцию принимается количество операций, возрастающее быстрее всего. Т.е. при некотором достаточно большом n основное влияние на скорость работы будет оказывать именно наибольшее число операций данного типа (например, если данный алгоритм выполняет  сложений и  умножения, наибольшее влияние на время работы будут оказывать регулярные операции сложения, чем редкие операции умножения).

При оценке  константы не учитываются. Т.е. 2000n+10000 и 3n+1 возрастают линейно и имеют сложность . При этом второй алгоритм может быть в 2000 раз эффективнее первого, но важно лишь то, что их сложность возрастатает линейно.

Важно помнить что учитывается лишь асимптотика. Понятно, что при малых n сложность алгоритма  может быть меньше чем *10000000n*, однако при достаточно больших n первый алгоритм растет гораздо быстрее.

Внутри  основание логарифма не пишется (это следует из правила 2, т.к. по правилу преобразования логарифмов можно перейти к любому основанию, а выносимая константа отбрасывается).

Дадим математическое толкование символа :  - множество функций  для которых существуют такие константы C и N, что    для всех . Запись  дословно обозначает, что  принадлежит множеству . При этом обратное выражение  не имеет смысла.

Другие виды оценок. Наряду с оценкой  используется оценка  (читается как "Омега большое от эн"). Обозначает нижнюю оценку роста функции. Например, пусть количество операций алгоритма описывает функция  = (). Это значит, что даже в самом удачном случае будет произведено не менее порядка  операций. При этом оценка  =  гарантирует, что в самом худшем случае будет произведено не более  операций.

Так же используется оценка . Представляет собой гибрид и . является одновременно верхней и нижней асимптотической оценкой - всегда будет выполняться порядка  операций. Оценка  существует только тогда, когда  и совпадают и равна им.

**Задания для практического занятия:**

1. Какая из функций растет быстрее?

Варианты ответов: f = , f = , f = .

1. n − 100 n − 200

2.  

3.  n + 

4.  

5.  

6.  

7.  

8.  

9.  

2. Составьте программу для решения следующих задач. Значения вводятся с клавиатуры. Выполнять проверку области определения и области значения, пороговых значений, деления на 0.

1. а) Даны два числа: вычислить их среднее арифметическое, их среднее геометрическое.

б) Даны основание и высота равнобедренной трапеции, найти периметр.

2. а) Составить программу решения линейного уровнения ax + b = 0.

б) Даны координаты двух точек на плоскости. Составить программу, вычисляющую расстояния между ними.

3. а) Дано двухзначное число. Определить: число десятков, число единиц, сумму и произведение цифр.

б) Треугольник задан координатами трех вершин. Найти длины сторон и периметр.

4. а) Даны катеты прямоугольного треугольника. Найти гипотенузу.

б) Дано трехзначное число. Определить: число сотен, десятков, единиц, сумму и произведение его цифр.

# Практическое занятие № 11 Определение сложности рекурсивных алгоритмов

**Цель занятия**: научиться вычислять эффективность и сложность исследуемого алгоритма

**Норма времени**2 часа

**Методическое обеспечение:** методические указания к практической работе

**Теоретические сведения**

**Сложность алгоритма** – характеристика алгоритма, определяющая зависимость времени выполнения программы, описывающей этот алгоритм, от объёма обрабатываемых данных. Сложность можно оценить по содержанию программы. Так, если в программе выполняется вложенный цикл с числом шагов внешнего цикла m и вложенного цикла n, то сложность будет пропорциональна mn. Формально определяется как порядок функции, выражающей время работы алгоритма.

**Эффективность алгоритма** – временная сложность в самом худшем случае **O(f(n)) или просто f(n)**. Функция от n **f(n)** равна максимальному числу шагов, выполняемых алгоритмом и имеет порядок роста O(f(n)), причём максимум берётся по всем входным данным длины n. Существует константа ***c***, такая, что для достаточно больших **n** величина **c•f(n)** является верхней границей количества шагов, выполняемых алгоритмом для любых входных данных длины **n**. Анализ рекурсивных программ значительно сложнее обычных, так как зачастую требуется решать дифференциальные уравнения. Для их анализа необходимо применять методы похожие на методы решения дифференциальных уравнений. При анализе **рекурсивных процедур** составляются **рекуррентные соотношения**, которые получают исходя из структуры рекурсивного алгоритма, отражающие характер рекурсивного вызова алгоритма и зависящие от величины входных данных.

**Решение рекуррентных соотношений**

Существуют **3 способа решения рекуррентных соотношений**:

Находится функция **f(n)**, которая **мажорирует T(n)** для всех значений **n**, т.е. для всех n выполняется неравенство **T(n)<=f(n)**. Иногда только лишь предположительно определяется вид функции **f(n)**, зависящей от некоторых неопределённых параметров. Далее подбираются такие параметры, что для всех **n** будет выполняться **T(n)<=f(n)**.

Последовательно подставляются в рекуррентное соотношение зависимости **T(m)**, где **m<n**, так, чтобы из правой части были исключены все **T(m)** с **m>1**, оставив толь ко **T(1)**. Но так как **T(1)** всегда является константой, то получится под конец зависимость от констант и **n**.

Использование общих решений.

**Оценка решений рекуррентных соотношений**

Рассмотрим пример процедуры-функции **mergesort** сортировки слиянием, входные данные которой – это список элементов длиной **n**, а выходные – это отсортированный список. Эта функция так же использует процедуру слияния **merge**, входные данные которой – это два отсортированных списка **L1** и **L2**. Данная процедура просматривает эти списки поэлементно, начиная с больших. На каждом шаге наибольший элемент из двух сравниваемых удаляется из своего списка и помещается в выходные данные. Получается тем самым единый отсортированный список, содержащий все элементы **L1** и **L2**. Процедура на списках **merge**, длиной **n/2**, выполняется за время порядка **O(n)**.

**function** mergesort(L:LIST; n:integer):LIST;{L - список типа LIST длиной n, где n является степенью числа 2}

**var** L1,L2:LIST;

**begin**

**if** n=1 **then** return(L)

**else** **begin**

разбиение L на две части L1 и L2, каждая длиной n/2;

return(merge(mergesort(L1, n/2),(mergesort(L2, n/2)));

**end**;

**end**; {mergesort}

Пусть **T(n)** - время выполнения процедуры **mergesort** в самом худшем случае. Анализируя текст программы, запишем рекуррентное неравенство, которое ограничивает сверху **T(n)**:

 (4.1)

В данном неравенстве **c1** – это количество шагов выполняемых алгоритмом над списком **L** длиной 1. Время работы процедуры можно разбить на две части, если **n>1**. Первая часть состоит из: **1)** проверки n<>1, **2)** разбивки L, на две равные части и **3)** вызова процедуры **merge**. Эти три операции требуют или фиксированное время для выполнения первой части или пропорционального n для второй и третьей. Следовательно, можно выбрать такую константу **c2**, которая будет создавать ограничение для выполнения данной части процедуры равное **c2\*n**. Вторая часть процедуры **mergesort** состоит из двух рекурсивных вызовов этой процедуры для списков длины **n/2**, которые будут требовать время **2T(n/2)**. Так было получено второе неравенство.

Формулу верхней границы в замкнутой форме можно получить лишь, если **n** является степенью числа **2**. При выполнении этого условия **T(n)** можно оценить для любых **n**. Другими словами, если **n** лежит в промежутке , то значение **T(n)** располагается между **T(2i)…T(2i+1)**. Нетрудно заметить, что выражение **2T(n)** можно заменить на **T((n+1)/2)+T((n-1)/2)** для нечётных **n>1**. Таким образом, можно найти решение рекуррентного соотношения в замкнутой форме для любых n. **Замкнутая форма** - это вид функции **T(n)**, не включающей в себя никаких выражений **T(m)** для **m<n**.

Произведём оценку рекуррентного соотношения (4.1).

Заменим в этом соотношении ***n*** на ***n/2*** и получим

 (4.2)

Подставим правую часть (7. 2) в (7. 1)

 (4.3)

Заменяя аналогичным образом в (7. 1) ***n*** на ***n/4***, получаем оценку для ***T(n/4)***: ***T(n/4)≤2T(n/8)+c2\*n/4***. Подставим эту оценку в (7.3) и получим такое выражение:

 (4.4)

Проанализировав характер изменения ***T(n)*** преобразуем (7.1) к виду:

 (4.5)

Предположим, что , тогда при ***i=k*** в правой части (7.5) находится ***T(1)***:

 (4.6)

Так как , то ,а ***T(1)≤ c1***, то (7.6) можно преобразовать

. (4.7)

Неравенство (7.7) демонстрирует верхнюю границу для ***T(n)***, а порядок роста ***T(n)*** не более ***O(n logn)***.

**Задания для практического занятия:**

Для своего варианта – столбец ***A***, выбрать рекуррентное уравнение и значение ***T(1)***. Необходимо решить данное рекуррентное соотношение и определить эффективность алгоритма, описанного функцией ***T(n).***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **Уравнение** | ***T(1)*** | **A** | **Уравнение** | ***T(1)*** |
| 1 | T(n)=3T(n/2)+n | 2 | 18 | T(n)=3T(n-1)+ | 9 |
| 2 | T(n)=2T(n-1)+2 | 2 | 19 | T(n)=3T(n/2)+n | 9 |
| 3 | T(n)=T(n/2)+ | 5 | 20 | T(n)=3T(n-1)+9 | 1 |
| 4 | T(n)=2T(n/2)+n | 2 | 21 | T(n)=2T(n/2)+ | 2 |
| 5 | T(n)=T(n/2)+*log*n | 1 | 22 | T(n)=2T(n/2)+n | 1 |
| 6 | T(n)=9T(n/2)+ | 9 | 23 | T(n)=T(n/2)+3*log*n | 3 |
| 7 | T(n)=2T(n/2)+5 | 3 | 24 | T(n)=8T(n/2)+ | 2 |
| 8 | T(n)=3T(n/2)+n | 3 | 25 | T(n)=T(n/2)+9 | 3 |
| 9 | T(n)=16T(n-1)+4 | 3 | 26 | T(n)=3T(n/2)+n | 4 |
| 10 | T(n)=T(n-1)+3n | 3 | 27 | T(n)=2T(n-1)+9 | 1 |
| 11 | T(n)=2T(n/2)+n | 8 | 28 | T(n)=2T(n-1)+3n | 6 |
| 12 | T(n)=4T(n/2)+2 | 8 | 29 | T(n)=2T(n/2)+n | 4 |
| 13 | T(n)=3T(n/2)+ | 3 | 30 | T(n)=(T(n-1))2 | 4 |
| 14 | T(n)=2T(n/2)+ | 4 | 31 | T(n)=T(n/2)+2 | 1 |
| 15 | T(n)=2T(n/2)+*log*n | 2 | 32 | T(n)=2T(n/2)+ | 16 |
| 16 | T(n)=(T(n-1))2 | 4 | 33 | T(n)=T(n/2)+2*log*n | 2 |
| 17 | T(n)=4T(n/2)+4 | 4 | 34 | T(n)=2T(n-1)+2 | 2 |

# Список рекомендуемой литературы

1. Тихомирова, А.Н. Практикум по теории алгоритмов: учеб. пособие [Текст] / А.Н. Тихомирова, Н.В. Сафоненко.– М.: НИЯУ МИФИ, 2011.– 132 с.
2. Тихомирова, А.Н. Теория алгоритмов: Учеб. пособие [Текст] / А.Н. Тихомирова.– М. : НИЯУ МИФИ, 2010 .– 176 с.
3. Поляков В.И. Основы теории алгоритмов [Текст] / В.И. Поляков, В.И. Скорубский. – СПб.: СПб НИУ ИТМО, 2012. – 51 с.
4. Кнут, Д.Э. Искусство программирования, том 1. Основные алгоритмы [Текст] / Д.Э. Кнут. — М.:, «Вильямс», 2010. – 720 с.