Алгоритмы и анализ сложности

Лекции 16 часов

Лабораторные работы 16 часов

Форма отчетности – зачет

Литература

- Абрамов С.А. Лекции о сложности алгоритмов.- М.: МЦНМО, 2009.- 256 с.
- *Дональд Кнут* Искусство программирования, том 1. Основные алгоритмы— 3-е изд. — М.: «Вильямс», 2006. — С. 720.
- Кнут В. Искусство программирования. Т.3. Сортировка и поиск. – 2-е изд. - Издательский дом “Вильямс”, 2000.
- Вирт Н. Алгоритмы и структуры данных. Пер. с англ. – М.: Мир, 1989. – 360.
- Носов В.А. Основы теории алгоритмов и анализа их сложности. – М., 1992.

Алгоритмы

Алгоритм - обозначение произвольных процессов, в которых искомые величины решаемых задач находятся последовательно из исходных данных по определенным правилам и инструкциям.

Свойства алгоритмов

- ***Массовость и общность***. Алгоритм может применятся к целому классу исходных данных и к широкому классу задач.
- ***Результативность***. При любых допустимых данных работа алгоритма должна быть закончена за конечное число шагов.
- ***Правильность***. Способность получать правильный результат.

Свойства алгоритмов 2

- ***Дискретность***. Рассматриваемы процесс может быть разделен на элементарные части.
- ***Детерминированность***. Каждый этап алгоритма должен быть сформулирован так, что бы предусматриваемые им действия определялись однозначно.

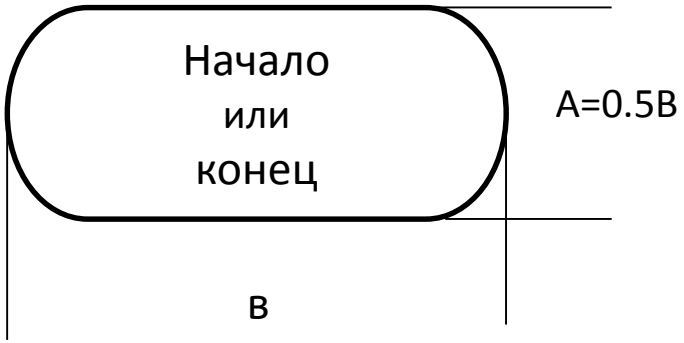
Описательный алгоритм

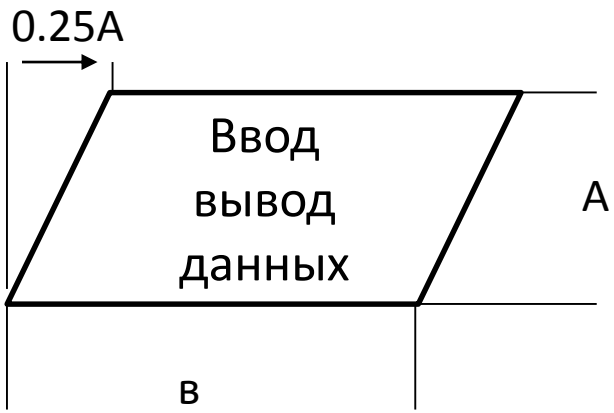
это алгоритм составленный на естественном, в частности математическом, языке.

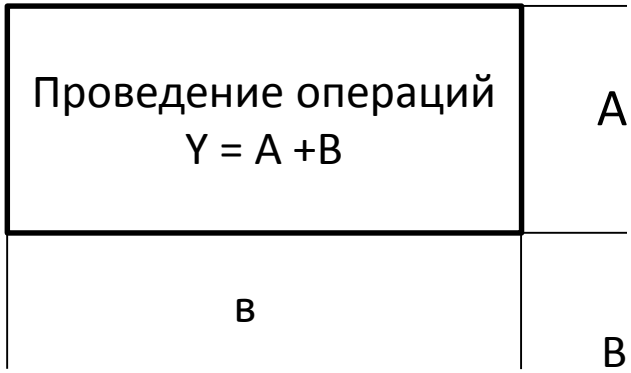
Графический алгоритм

это компактная форма записи алгоритма в виде специальных графических знаков с указанием связей между ними.

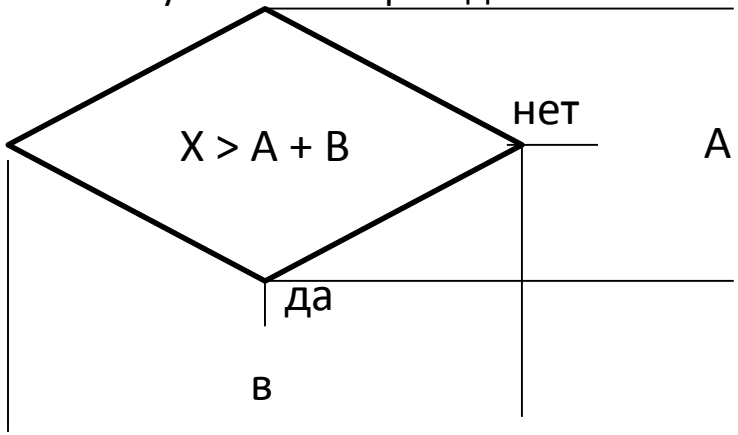
Основные блоки графического алгоритма







Выполнение условных переходов



Изображение циклического алгоритма

Алгоритм в виде программы

это конечный продукт разработки алгоритма в виде программы, записанный на языке программирования. Текст программы называют

листинг.

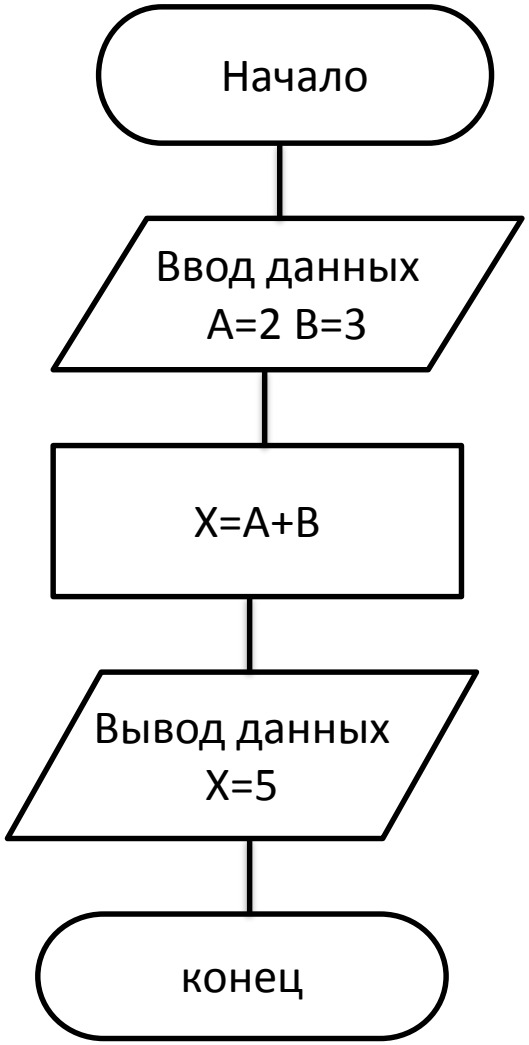
Схема - алгоритм

- Графическое изображение структуры алгоритма в котором каждый этап вычислений изображается геометрической фигурой (блоком), а связи изображаются стрелками.

Линейный алгоритм

- Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых исходных данных. Операторы задействованы последовательно, один за другим, в соответствии с их расположений в тексте программы.

Линейная схема - алгоритм

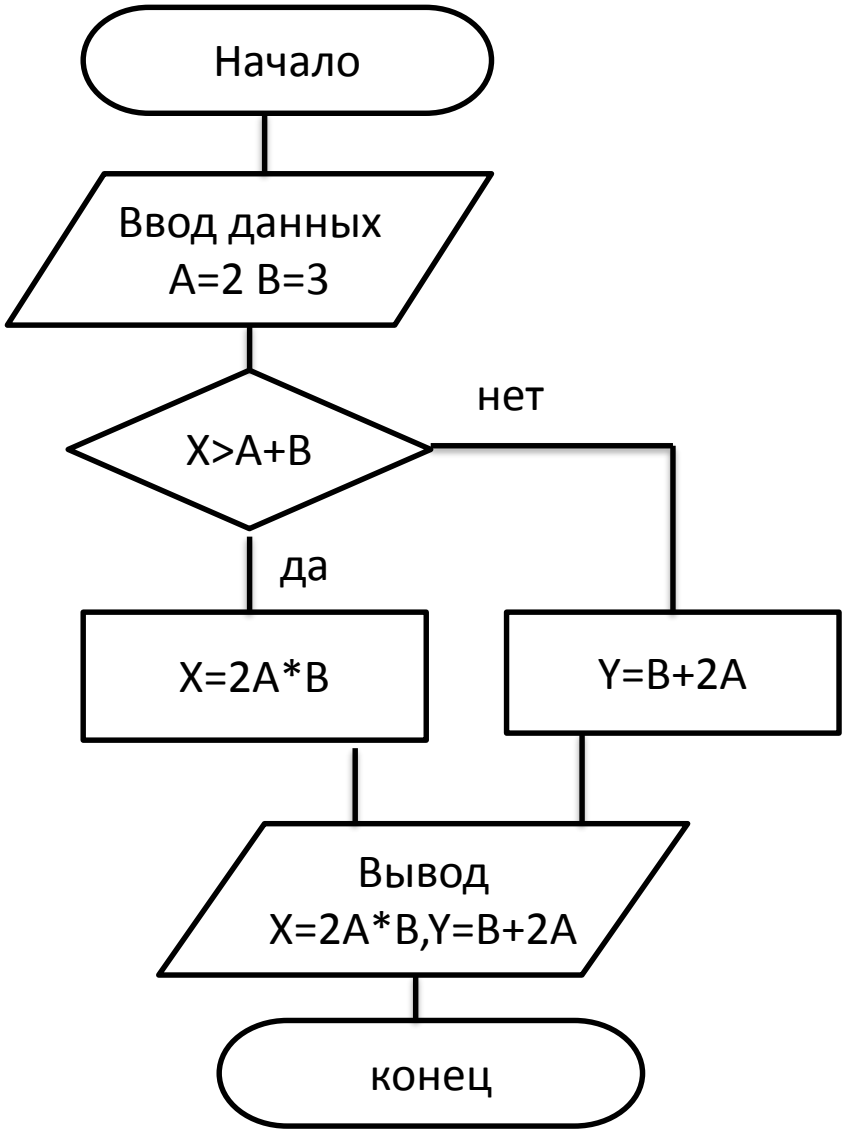


Разветвляющийся алгоритм

- Алгоритм называется разветвляющимся,

если он содержит несколько ветвей выполнения программы отличающихся друг от друга содержанием вычислений.

Разветвляющаяся схема-алгоритм

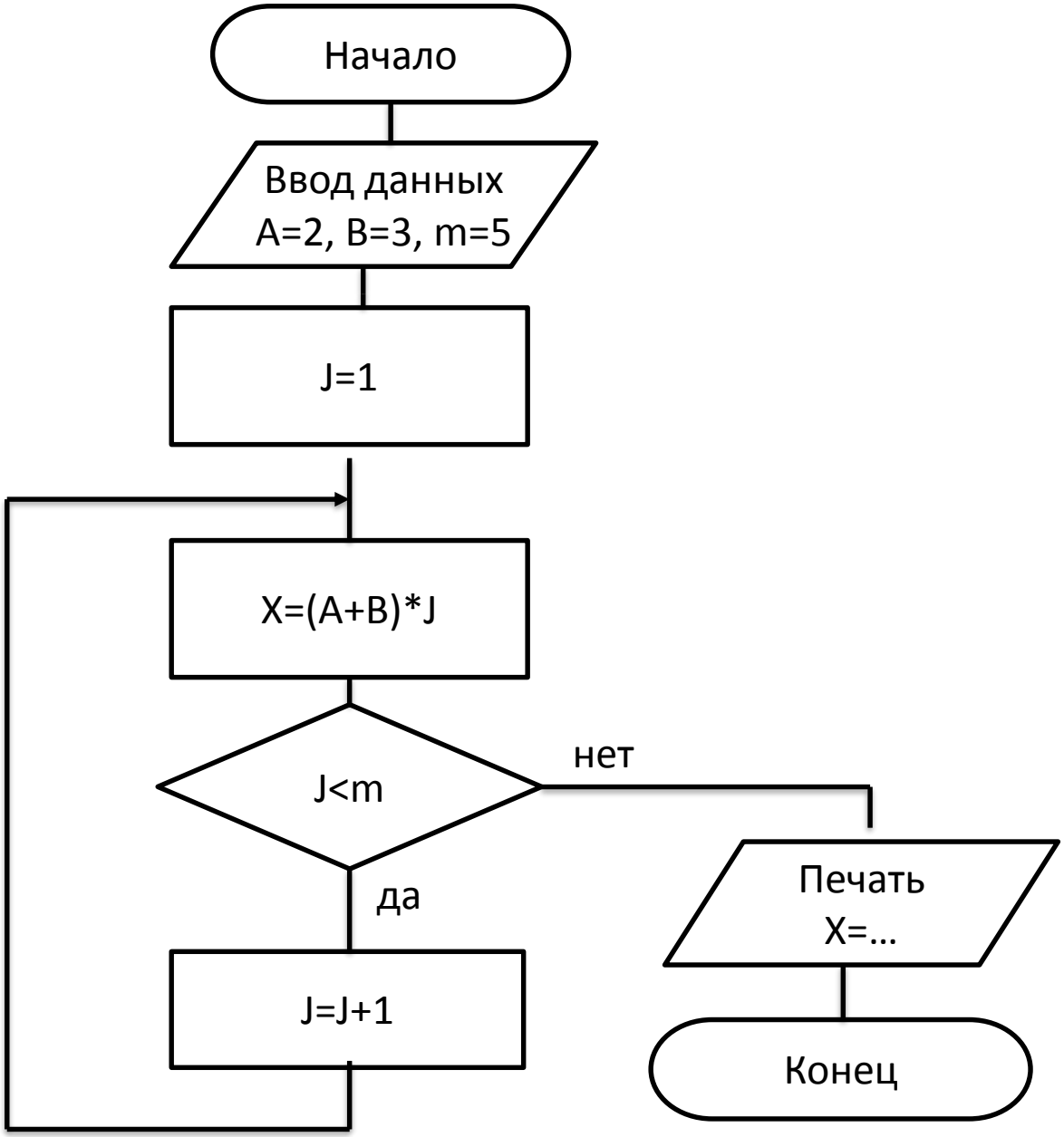


Циклический алгоритм

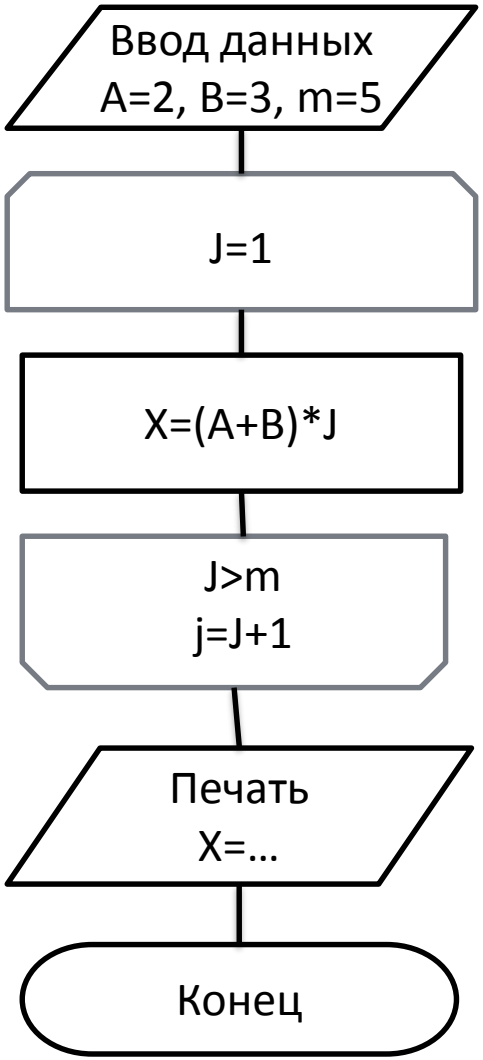
- Алгоритм называется циклическим,

если он содержит многократное выполнение одних и тех же ветвей при различных значениях промежуточных данных.

Циклическая схема - алгоритм



Циклическая схема – алгоритм 2



Основные требования к алгоритмам

1. Каждый алгоритм имеет дело с данными - входными, промежуточными, выходными. Для того, чтобы уточнить понятие данных, фиксируется конечный алфавит исходных символов (цифры, буквы и т.п.) и указываются правила построения алгоритмических объектов.

- Например, определение идентификатора

приписана справа либо буква, либо цифра. Слова конечной длины в конечных алфавитах - наиболее обычный тип алгоритмических данных, а число

символов в слове - естественная мера объема данных.

Другой случай алгоритмических объектов – формулы.

2. Алгоритм для размещения данных требует памяти . Память обычно считается однородной и дискретной, т.е. она состоит из одинаковых ячеек , причем каждая ячейка может содержать один символ данных, что позволяет согласовать единицы измерения объема данных и памяти.

3. Алгоритм состоит из отдельных элементарных шагов, причем множество различных шагов, из которых составлен алгоритм, конечно .

- Типичный пример множества элементарных шагов - система команд ЭВМ.

4. Последовательность шагов алгоритма детерминированна, т.е. после каждого шага указывается, какой шаг следует выполнять дальше, либо указывается, когда следует работу алгоритма считать законченной.

5. Алгоритм должен обладать результативностью, т.е. останавливаться после конечного числа шагов (зависящего от исходных данных) с выдачей результата.

Данное свойство иногда называют сходимостью алгоритма.

Анализ трудоёмкости алгоритмов

- В качестве критерия оптимальности алгоритма выбирается трудоемкость алгоритма, понимаемая как количество элементарных операций, которые необходимо выполнить для решения задачи с помощью данного алгоритма. Функцией трудоемкости называется отношение, связывающие входные данные алгоритма с количеством элементарных операций.

- Одним из упрощенных видов анализа, используемых на практике, является асимптотический анализ алгоритмов. Целью асимптотического анализа является сравнение затрат времени и других ресурсов различными алгоритмами, предназначенными для решения одной и той же задачи, при больших объемах входных данных.

- Используемая в асимптотическом анализе оценка функции трудоёмкости, называемая **сложностью алгоритма**, позволяет определить, как быстро растет трудоёмкость алгоритма с увеличением объема данных. В асимптотическом анализе алгоритмов используются обозначения, принятые в математическом асимптотическом анализе. Ниже перечислены основные оценки сложности.

T(n) – временная оценка сложности алгоритма.

Самый простой способ оценки сложности алгоритма – экспериментальный, т.е. запрограммировать алгоритм и выполнять полученную программу на нескольких задачах, оценивая время выполнения программы.

Метод имеет недостатки:

- дорогостоящий процесс;
- нельзя применить типовые единицы измерения временной сложности алгоритма (секунды, миллисекунды), так как можно получить самые различные оценки для одного и того же алгоритма

(разные компиляторы, ЭВМ и программисты).

Временная сложность алгоритма зависит от количества входных данных «n» и имеет порядок T(n). Точно определить порядок T(n) на практике трудно и поэтому прибегают к асимптотическим отношениям с использованием O - символики. Когда используют обозначение O(), имеют в виду не точное время исполнения, а только его предел сверху с точностью до постоянного множителя.

Например, если число тактов (действий), необходимое для работы алгоритма выражается как

$$3 \cdot n^2 + 5 \cdot n \cdot \log n + 7 \cdot n,$$

то это алгоритм, для которого T(n) имеет порядок

$$O(n^2).$$

Фактически остается только то слагаемое, которое вносит наибольший вклад при больших «n».

Время выполнения алгоритма зависит не только от количества входных данных, но и от их значений. Для анализа алгоритма независимо от входных данных и их значений различают:

- максимальную сложность – $T_{max}(n)$, или сложность наиболее неблагоприятного случая, когда алгоритм работает дольше всего;
- среднюю сложность - $T_{mid}(n)$;
- минимальную сложность - $T_{min}(n)$ – сложность в наиболее благоприятном случае.

Оценка временной сложности алгоритма

1. O(1) – время выполнения операций присваивания, чтения, записи. Исключением являются операторы присваивания, в которых операнды представляют собой массивы или вызовы функций;

2. время выполнения последовательности операций совпадает с наибольшим временем выполнения операции в данной последовательности (правило сумм: $T_1(n)$ имеет $O(f(n))$, а $T_2(n)$ – порядок $O(g(n))$, то $T_1(n)+ T_2(n)$ имеет порядок $O(\max(f(n), g(n)))$;

3. время выполнения конструкции ветвления (if – then – else) состоит из времени вычисления логического выражения O(1) и наибольшего из времени выполнения операций при истинном значении логического выражения и при ложном значении логического выражения;

4. время выполнения цикла состоит из времени вычисления условия прекращения цикла и имеет порядок O(1) и произведения количества выполненных итераций цикла на наибольшее возможное время выполнения операций тела цикла.

Основной оценкой функции сложности алгоритма f(n) является верхняя оценка O. Здесь

n — величина объёма данных или длина входа. Мы говорим, что оценка сложности алгоритма

$$f(n)=O(g(n)),$$

если при g > 0 при n > 0 существуют положительные c_1, c_2, n_0, такие, что:

$$c_1 q(n) \le f(n) \le c_2(n) q(n)$$

при $n > n_0$, иначе говоря, можно найти такие c_1 и c_2, что при достаточно больших n f(n) будет заключена между

$$c_1 q(n) \text{ è } \tilde{n}_2 q(n).$$

В таком случае говорят еще, что функция g(n) является асимптотически точной оценкой функции f(n), так как по определению функция f(n) не отличается от функции g(n) с точностью до постоянного множителя.

Датчик случайных чисел

- Для формирования одномерного или двухмерного массивов при программировании используется равномерный датчик случайных чисел

  random(n),

  n – задает диапазон случайных чисел от

  0 ÷ n-1.

  Для того, что бы датчик случайных чисел начинал работать с различных начальных значениях перед ним ставят процедуру

  randomize.

Упорядочивание (сортировка) массива

- Упорядочить массив $X_1, X_2, \ldots, X_n$ – это значит расположить все числа массива в возрастающем порядке т. е.

$$X_1 < X_2 < \ldots < X_n.$$

На первом месте должен быть наименьший элемент, на втором месте – наименьший из всех остальных элементов и т. д.

Пузырьковая сортировка

- Реализация данного метода не требует дополнительной памяти. Метод очень прост и состоит в следующем: берется пара рядом стоящих элементов, и если элемент с меньшим индексом оказывается больше элемента с большим индексом, то мы меняем их местами.

- Эти действия продолжаем, пока есть такие пары. Легко понять, что когда таких пар не останется, то данные будут отсортированными. Для упрощения поиска таких пар данные просматриваются по порядку от начала до конца. Из этого следует, что за такой просмотр находится максимум, который помещается в конец массива, а потому следующий раз достаточно просматривать уже меньшее количество элементов.

- Максимальный элемент как бы всплывает вверх, отсюда и название алгоритма так как каждый раз на свое место становится по крайней мере один элемент, то не потребуется более N проходов, где N — количество элементов. Вот как это можно реализовать:

Как только наименьший элемент занял свое место, он сразу выводится из массива.

Для перестановки A[i] с A[k] привлекается дополнительная переменная V

V=A[i]; A[i]=A[k]; A[k]=V;

Сортировка вставками

Суть алгоритма сортировки вставками состоит в следующем: создается ***новый массив***, в который мы последовательно вставляем элементы из исходного массива так, чтобы новый массив был упорядоченным. Вставка происходит следующим образом:

в конце нового массива выделяется свободная ячейка, далее анализируется элемент, стоящий перед пустой ячейкой и, если этот элемент больше вставляемого, то подвигаем элемент в свободную ячейку (при этом на том месте, где он стоял, образуется пустая ячейка) и сравниваем следующий элемент.

Так мы прейдем к ситуации, когда элемент перед пустой ячейкой меньше вставляемого, или пустая ячейка стоит в начале массива. Помещаем вставляемый элемент в пустую ячейку .

Таким образом, по очереди вставляем все элементы исходного массива. Так как порядок элементов в новом массиве не меняется, то сформированный массив будет упорядоченным после каждой вставки. А значит, после последней вставки мы получим упорядоченный исходный массив.

Анализ сложности

- Эти методы объединяет не только то, что они сортируют данные, но также и время их работы. В каждом из алгоритмов присутствуют вложенные циклы, время выполнения которых зависит от размера входных данных. Значит, общее время выполнения программ есть O(2n) (константа, умноженная на 2n). Следует отметить, что эти два алгоритма используют O(2n) перестановок.

Сортировка Шейкером

- Этот алгоритм уменьшает количество перемещений, действуя следующим образом: за один проход из всех элементов выбирается минимальный и максимальный. Потом минимальный элемент помещается в начало массива, а максимальный, соответственно, в конец.

Далее алгоритм выполняется для остальных данных. Таким образом, за каждый проход два элемента помещаются на свои места, а значит, понадобится n/2 проходов, где n — количество элементов.

В каждом из алгоритмов присутствуют вложенные циклы, время выполнения которых зависит от размера входных данных. Значит, общее время выполнения программ есть O(2n), т.е. константа, умноженная на 2n.

Эти алгоритмы используют также O(2n) перестановок, в то время как метод Шейкера использует их O(n). Отсюда следует, что метод Шейкера является более выгодным для сортировки данных.

Сортировка слиянием

- Эта сортировка использует следующую подзадачу: есть два отсортированных массива, нужно сделать (слить) из них один отсортированный. Алгоритм сортировки работает по такому принципу:

разбить массив на две части, отсортировать каждую из них, а потом слить обе части в одну отсортированную, что обеспечивает быстродействие и корректность данного метода.

Алгоритм слияния

6 1| 3 5 | 4 2 | 8 7

Проводят частичную упорядочивание массивов

1 6 3 5 | 2 4 7 8

Упорядочивание внутри массива

|1 3 5 6 2 4 7 8 |

и затем в этом массиве проводят слияние двух массивов

1 2 3 4 5 6 7 8

В программе использованы обозначения:

p – индекс начала массива;

q – индекс конца сортируемой части;

r:=(p+q) div 2 – массив разбивается на две части и рассматривается массив A и берется вспомогательный массив B.

Время работы алгоритма слияния

Для оценки времени работы алгоритма слияния, составим рекуррентное соотношение. Пускай T(n) — время сортировки массива длины n, тогда для сортировки слиянием справедливо $T(n)=2T(n/2)+O(n)$ ($O(n)$ — это время, необходимое на то, чтобы слить два массива).

Распишем это соотношение:

$$T(n)=2T(n/2)+O(n)=4T(n/4)+2O(n/2)+O(n)=$$

$$4T(n/4)+2O(n)= \ldots = 2kT(1)+kO(n)$$

Осталось оценить k. Мы знаем, что 2=n, а значит k=log2n. Уравнение примет вид

$$T(n)=nT(1)+ \log 2nO(n).$$

Так как T(1) — константа, то

$$T(n)=O(n)+\log 2nO(n)=O(n\log 2n).$$

Т. е., оценка времени работы сортировки слиянием меньше, чем у предыдущих алгоритмов

Сортировка массива метод Хоара

Он основан на сравнениях и обменах местами элементов, стоящих на возможно больших расстояниях друг от друга. Массив разделяют на два фрагмента так, что в левом фрагменте оказались элементы некоторого эталонного значения. В правом фрагменте все остальные элементы.

- Два индекса проходят по массиву с разных сторон и ищут элементы, которые попали не в свою группу. Найдя такие элементы, их меняют местами. Тот элемент, на котором индексы пересекутся, и определяет разбиение на группы.

Алгоритм можно определить как рекурсивную процедуру, параметрами которой являются нижняя и верхняя границы изменения индексов сортируемой части исходного массива.

Для дальнейшей сортировки необходимо применить алгоритм для каждой из этих частей. И так до тех пор, пока не останутся подмассивы, состоящие из одного элемента, то есть пока не будет отсортирован весь массив.

Выбор эталонного значения

При выборе эталонного значения x необходимо, чтобы оно было из элементов фрагмента массива A.

Можно выбирать:

A[l] – первый элемент массива;

A[r] – последний элемент массива;

A[(l+r)/2] – средний элемент массива;

A[l+random(r-l+1)] – выбор случайного элемента массива.

При выборе эталонного значения A[l] или A[r] можно встретится с вариантом переполнения стека рекурсии. Поэтому рекомендуется выбор эталонного элемента проводить для середины массива:

A[(l+r)/2]

или выбирать случайным образом из элементов массива по соотношению

A[l+random(r-l+1)].

Пример с эталонным значением A[l]

Дано множество

{09, 06, 03, 04, 10, 0 8, 02, 07}

Берем 09 в качестве эталонного элемента. Сравниваем 09 с противоположностоящим элементом, в данном случае это 07. 07 меньше, чем 09, следовательно элементы меняются местами:

{**07**, 06, 03, 04, 10, 08, 02, **09**}.

В дальнейшем последовательно будем сравнивать элементы с 09, и менять их местами в зависимости от сравнения:

{07, **06**, 03, 04, 10, 08, 02, **09**}

{07, 06, **03**, 04, 10, 08, 02, **09**}

{07, 06, 03, **04**, 10, 08, 02, **09**}

{07, 06, 03, 04, **09**, 08, 02, **10**} –

- 09 и 10 меняем местами.

{07, 06, 03, 04, **08**, **09**, 02, 10} –

- 09 и 08 меняем местами.

{07, 06, 03, 04, 08, **02**, **09**, 10} –

- 02 и 09 меняем местами.

После такого перебрасывания элементов весь массив разбивается на два подмножества, разделенных элементом 09:

{07, 06, 03, 04, 08, 02} 09 {10}.

Далее по уже отработанному алгоритму сортируются эти подмножества. Подмножество из одного элемента естественно можно не сортировать.

Выбираем в первом подмножестве базовый элемент 07:

{07, 06, 03, 04, 08, 02}.

{**02**, 06, 03, 04, 08**, 07**} –

- меняем местами 02 и 07.

{02, 06, 03, 04, **07**, **08**} –

- меняем местами 07 и 08.

Получили снова два подмножества:

{02, 06, 03, 04} 07 {08}.

По этому алгоритму сортируется первое подмножества и окончательно получим отсортированный массив.

Анализ сложности метода Хоара

Худший случай, когда в одном из фрагментов массива только один элемент,

а во втором фрагменте остальные n-1 элементов:

{ 1 } { n-1 }.

Составляем рекуррентное соотношение для описания сложности алгоритма сортировки Хоара:

$$T(n)=O(n)+T(1)+T(n-1),$$

где O(n) – время, необходимое для разделения массива на два фрагмента;

T(1)=const – это время, необходимое для сортировки одного элемента массива.

Распишем полученное соотношение на всю процедуру сортировки:

$$T(n)=O(n)+(O(n-1)+T(n-2)+T(1))=$$

$$=O(n)+O(n-1)+O(n-2)+\ldots+O(1)+nT(1).$$

После преобразований аргументов функции O, получим следующее соотношение:

$T(n)=O(n+(n-1)+(n-2)+\ldots+2+1)+nT(n)$.

Аргументы функции O представляют собой убывающую арифметическую прогрессию, сумма которой определяется по соотношению $S=n*(n-1)/2$. Окончательно найдем время сортировки массива:

$T(n)=O(n(n-1)/2)+nT(1)=O(n\cdot n)$, т. е., время сортировки массива по методу Хоара в худшем случае пропорционально квадрату количества элементов массива.

Анализ сложности метода Хоара в общем случае

Разделение на фрагменты массива примет следующий вид:

{ g } { n-g }.

После разделения на фрагменты эталонный элемент окажется в позиции g с вероятностью p=1/n, тогда время сортировки массива определится по соотношению:

$$T(n) = O(n) + \frac{1}{n} \sum_{g=1}^{n-1} (T(g) + T(n-g)) = O(n) + \frac{2}{n} \sum_{g=1}^{n-1} T(g).$$

После преобразований получим:

T(n)=O(n log n).

В лучшем случае, когда массив делится на два одинаковых фрагмента:

{ n/2 } { n/2 },

время определится по соотношению:

T(n)=O(n)+2 T(n/2)=O(n log n).

Метод сортировки массива Шелла

Суть состоит в том, что здесь задается последовательность шагов, отстоящих один от другого на некотором расстоянии :

$h_0, h_1, \ldots, h_{t-1} : h_0=1, h_i < h_{i+1}$.

Исходный размер шага обычно выбирается соизмеримым с половиной общего размера сортируемой последовательности.

Выполняется пузырьковая сортировка с интервалом сравнения h. Затем величина h уменьшается и вновь выполняется пузырьковая сортировка, далее h уменьшается еще и т.д. Последняя пузырьковая сортировка выполняется при h=1.

Алгоритм сортировки состоит из t раундов. В k-ом раунде используется шаг h_k : при этом k изменяется от t-1 до 0.

Сортировка методом Шелла с уменьшающимися расстояниями.

Дан массив: 44 55 12 42 94 18 06 67 из восьми элементов – провести сортировку методом Шелла. Выбираем для первого раунда h_k равное четырем, т. е. попарно рассматриваем элементы массива:

44 и 94 – оставляем на месте;

55 и 18 – меняем местами: 18 55;

12 и 06 – меняем местами: 06 12;

42 и 67 – оставляем на месте.

В результате получим частично отсортированный массив:

44 18 06 42 94 55 12 67.

Для следующего раунда выбираем h_k=2 и проводим сортировку полученного массива с шагом два:

44 и 06 – меняем местами 06 44;

18 и 42 – оставляем на месте;

94 и 12 – меняем местами 12 94:

55 и 67 – оставляем на месте – получим частично отсортированный массив:

06 18 12 42 44 55 94 67.

И раунд сортировки с шагом равный единице дает окончательно отсортированный массив.

18 и 12 меняем местами, затем сравниваем остальные элементы и из них 94 и 67 меняем местами, получим окончательно отсортированный массив:

{06, 12, 18, 42, 44, 55, 67, 94}.

Анализ сортировки Шелла. В общем случае не известно, какие расстояния для h дают наилучший результат. Но они не должны быть множителями один другого. Справедлива такая теорема: если *k*-отсортированную последовательность

i-отсортировать, то она остается

k-отсортированной. что имеет смысл использовать такую последовательность, в которой $h_{k-1} = 3h_k + 1$, $h_t = 1$ и t= $[\log_2 n] - 1$.

Пирамида
(сортировка с помощью кучи)

Для ускорения выбора минимального элемента по сравнению с методом прямого выбора воспользуемся алгоритмом Флоида:

разбиваем массив на подмассивы и из каждого подмассива выбираем наименьший элемент, полученный подмассив снова разбиваем на подмассивы и выбираем соответствующие минимальные элементы, и так до последнего элемента, который будет минимальным из всех элементов:

6 1 | 2 8 | 4 3 | 5 9

1 2 | 3 5

1 3

1

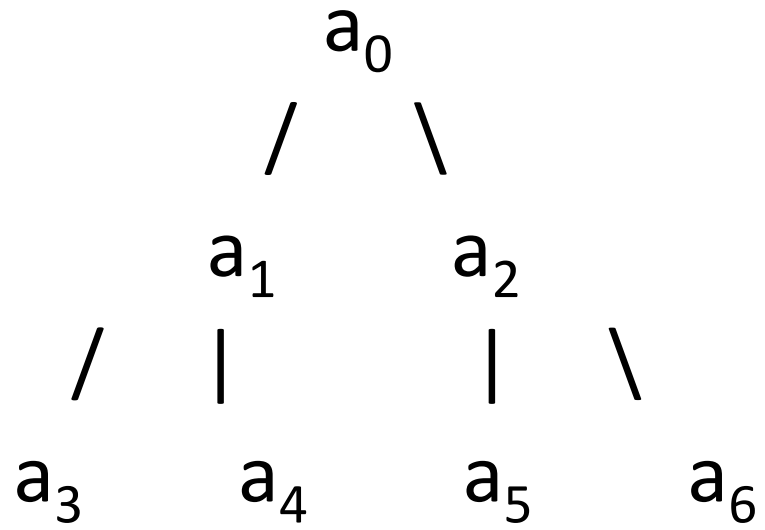
Пирамидой называется последовательность элементов

$$a_l, a_{l+1}, \ldots a_i : a_i \le a_{2i+1}; a_i \le a_{2i+2};$$

Рассмотрим последовательность элементов

$a_0, a_1, a_2, a_3, a_4, a_5, a_6$,

тогда пирамида примет вид



Из определения пирамиды можно увидеть, что в вершине пирамиды находится минимальный элемент.

Алгоритм создания пирамиды

Элемент a_i помещают в вершину пирамиды, потом сравнивают его со своими «потомками»,

1. если он меньше обоих своих «потомков», то процесс завершен;

2. иначе меняем этот элемент с меньшим из двух «потомков».

Повторяем процедуру до тех пор, пока либо:

1. элемент не станет меньше «потомка»;
2. не будет «потомков».

a_i

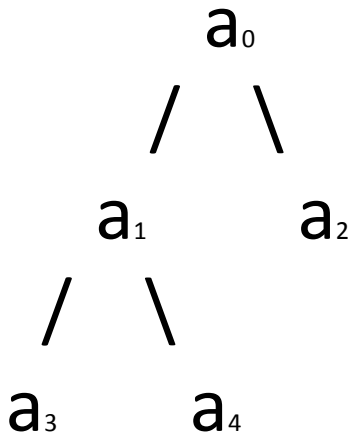
/ \

a_{2i+1} a_{2i+2}

Таким образом проводится просеивание i-го элемента через подчиненную ему пирамиду-это свойство пирамиды.

Анализ сложности пирамидальной сортировки

Наибольшее число операций при просеивание через пирамиду выполняется тогда, когда элементы просеиваются через левую часть пирамиды. Разница в высоте пирамиды равна единице



Введем T(n) время процедуры просеивания элемента через пирамиду, состоящую из n элементов. Это время для левой части пирамиды определяется по соотношению

$$T(n)=O(1)+T(2/3n),$$

где O(1) – время, которое определяется процедурой обмена элемента с одним из своих «потомков».

Анализ времени построения пирамиды

Для просеивания элемента, расположенного на высоте h требуется O(h) операций. Число вершин на высоте h меньше или равно величине

$$2^{n/h} - 1$$

Время построения пирамиды определяется по соотношению:

$$T1(n) = \sum_{h=1}^{\log_2 n} (2^{n/h} - 1) \bullet O(h) = n \bullet O(\sum_{h=1}^{\log_2 n} (2^{n/h} - 1)) \leq n.$$

Алгоритмы поиска элемента в упорядоченном массиве

Дан упорядоченный массив целых чисел:

$a_0, a_1, \ldots, a_{n-1}, a_i <= a_{i+1}$, x – целое число:

Необходимо найти такое I для которого

a_i = x, либо такого элемента нет.

Линейный поиск

При линейном поиске в упорядоченном массиве просматривается лишь та часть массива, в которой элементы меньше или равны x, если x больше последнего элемента, то такого элемента нет.

IF(i<n и a[i]<x) то i=i+1,

Иначе нет такого элемента.

Поиск элемента с барьером

При поиске элемента в упорядоченном массиве вводится барьер $a_n = x$ и рассматриваются элементы меньше введенного барьера.

Для линейного алгоритма и алгоритма с барьером в худшем случае равно n.

Бинарный поиск
Метод дихотомии, деления пополам

Идея метода состоит в следующем:

выбирается k (эффективнее выбирать k в середине массива) . Пусть переменная i - индекс левого элемента, значение j - индекс правого элемента, для элементов массива, среди которых осуществляется поиск.

Далее k-ый элемент массива a[k] сравнивается с образцом x. Если окажется, что x <= a[k], то поиск следует продолжать среди элементов с индексами [i,k]., если же x > a[k], искать надо среди элементов [k+1,j]. Процесс поиска продолжается до тех пор, пока исследуемая часть массива не станет равной одному элементу, тогда результат будет зависеть от того, равен этот элемент образцу или нет.

Поиск медианы в массиве

Медианой массива, содержащего N элементов, называется элемент, значение которого меньше (или равно) половины N элементов и больше (или равно) другой половины.

В общем случае – это задача поиска k – го наименьшего элемента массива.

Например, медианой массива, содержащего элементы 16, 22, 99, 95, 18, 87, 10 является 18.

Задачу поиска медианы можно связать с сортировкой следующим образом: вначале произвести сортировку массива, а затем выбрать "средний" элемент.

Дан массив A размерности n и целое число k, которое находится в середине массива A (0 <= k <= n). Необходимо Найти элемент X, и если элемент после сортировки окажется на k-ом месте, т. е.

X=A[k] и k=n/2, то задача поиска медианы массива завершена.

Метод Хоара

По методу Хоара сортируются те фрагменты массива, в котором содержится k-ый элемент.

Рекурсивный алгоритм (обращается сам к себе):

1. Устанавливаются значения переменных:

i=l, j=r, x=A[(l+r)/2];

2. пока i<=j, то

2.1 Пока A[i] < x, то увеличиваем переменную i на единицу;

2.2 После, когда A[j] > x уменьшаем переменную j на единицу;

2.3 Если l <= j, то меняем местами содержимое A[i] и A[j], с использованием дополнительной ячейки v и увеличиваем переменную i на единицу, переменную j уменьшаем на единицу:

(i<=j, то A[i] ⇔ A[j], i+1, j-1;)

3. Если k в левом фрагменте массива

(l <= k <= j), то вернуть результат рекурсии для этого фрагмента [l, j].

Если k между фрагментами массивов

(i <= k <= j), то вернуть значение A[k].

Если k в правом фрагменте массива, то вернуть тот результат, для которого k находился интервале [i,r].

Пример поиска медианы (k-го элемента)

Дан массив: {07, 08, 03, 07 , 07, 09, 07, 01}

для которого l=1, r=8, k=4 A=07.

{**01,** 08, 03, 07, 07, 09, 07, **07** } -

- меняем местами 07 и 01, затем меняем местами 08 и 07, получим следующий массив:

{01, **07**, 03, 07, 07, 09, **08**, 07}.

Получим два подмножества для которых применим алгоритм сортировки Хоара:

{01, 07, 03, 07} {07, 09, 08, 07}.

В результате сортировки в первом и во втором фрагментах массива A получим два массива разделенных элементом 07:

{ } 07 {08}

и медианой массива A будет элемент 08.

Оценка сложности метода

Худший случай:

$T(n)=n+T(n-1)=n+(n-1)+(n-2)+\ldots+1=Q(n*n)$.

Лучший приблизительно равен среднему случаю:

$T(n)=n+T(n/2)=n+n/2+n/4+\ldots+1<=$

$<=n(1+1/2+1/4+\ldots)=n*1/(1-1/2)=2n$.

Поразрядная сортировка

Сортировка чисел осуществляется последовательно от младшего разряда к старшему. При таком разбиении получается, что для всех элементов одной группы и для всех элементов другой группы устанавливается один знак сравнения (любой элемент одной группы меньше любого элемента другой группы).

При этом используются столько вспомогательных списков, сколько значений могут принимать старшие разряды.

Рассмотрим массив:

357 127 399 004 017 249 444 222 577.

Для данного массива составляем список младшего разряда:

h[0]
h[1]
h[2] 222
h[3]
h[4] 444 004
h[5]
h[6]
h[7] 577 017 127 357
h[8]
h[9] 249 399.

Перемещаем элементы массива по порядку в общий список:

399 249 357 127 017 557 004 444 222

Для полученного массива составляем список для второго разряда:

h[0] 004 h[5] 357

h[1] 017 h[6]

h[2] 222 127 h[7] 577

h[3] h[8]

h[4] 444 249 h[9] 399.

Запишем полученный массив:

399 577 357 249 444 127 222 017 004.

Составляем список старшего разряда:

h[0] 004 017	h[5] 577
h[1] 127	h[6]
h[2] 222 249	h[7]
h[3] 357 399	h[8]
h[4] 444	h[9].

Окончательно получим отсортированный массив: 577, 444, 399, 357, 249, 222, 127, 017, 004.

Сложность определяется по соотношению:

$T(n)=Q(n \log_2 n)$.

Побитовая сортировка

Побитовая сортировка базируется на методе Хоара: рассмотрим массив:

12 5 8 1 7 3

В двоичной системе счисления:

1100 0101 1000 0001 | 0111 0011.

Разбиваем массив на два подмассива и проводим обмен элементов 12 и 3; 7 и 8, получим следующий массив:

3 5 | 7 1 | 8 12:

0011 0101 0111 0001 1000 1100.

Разбиваем на три подмассива и меняем местами 5 и 1 получим частично отсортированный массив вида:

3 1 7 5 8 12.

В полученном массиве меняем местами

3 и 1; 7 и 5: получим отсортированный массив:

0001 0011 0101 1000 1100.

Алгоритм побитовой сортировки

Исходные данные: l=0, r=n-1, k приравниваем к номеру старшего разряда.

Устанавливаем начальные значения: i=l, j=r.

Пока i<=j и в k – ом разряде элемента A[i] ноль, то увеличиваем i на единицу;

Пока i<=j и в k – ом разряде элемента A[j] единица, то уменьшаем j на единицу;

Если i<j, то меняем местами A[i] и A[j], увеличиваем i на единицу, а уменьшаем j на единицу.

Выполняем эту же процедуру для фрагмента массива [l, j] по k-1 разряду, затем выполняем эту же процедуру для фрагмента [l,r] по k-1 разряду.

Сложность алгоритма побитовой сортировки массива определяется по соотношению:

$$T(n)=O(n \log_2 max],$$

где max – максимальное значение элемента в массиве. Т.е., побитовую сортировку хорошо использовать для чисел небольших по значению.

Топологическая сортировка

Содержательная постановка:

Рассеянный джентльмен: известно, какие элементы гардероба нельзя подавать раньше других. Требуется указать такой линейный порядок, чтобы не нарушать введенные запреты. Для технических систем:

1. В списке операций не должно встречаться термина: ранее не определялось;

2. Не должно нарушаться расписание работ в проекте;

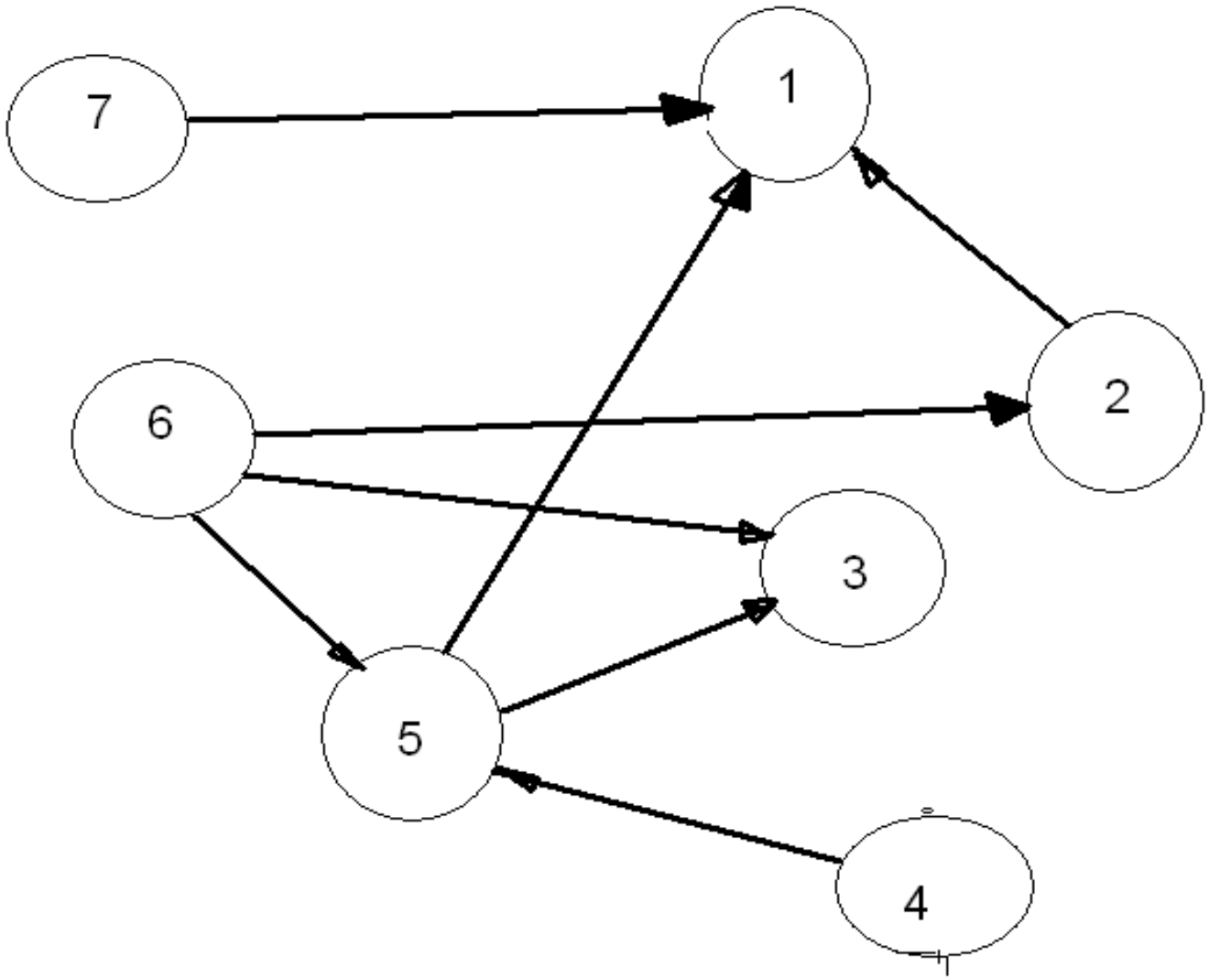
3. Не должен нарушаться список функций.

Математическая постановка

Дано конечное множество A с заданным отношением частичного порядка. Требуется указать единственный линейный порядок элементов в массиве A, не нарушая исходного частичного порядка.

Задан ориентированный граф без циклов. Требуется указать такую последовательность всех вершин графа, что для каждой пары вершин (a_i, a_j) такой, что для i<j, нет дуги, ведущей из a_j в a_i .

Дан массив из семи элементов и построен направленный граф.

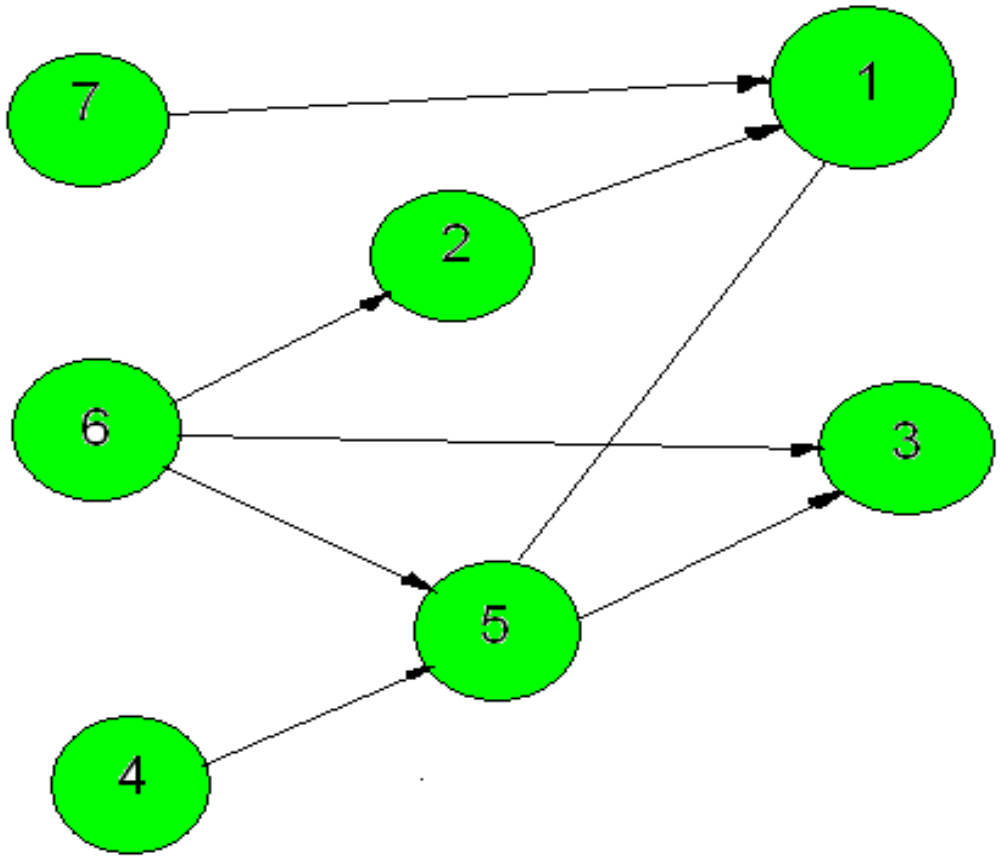

7
1
6
2
3
5
4

Разработка алгоритма

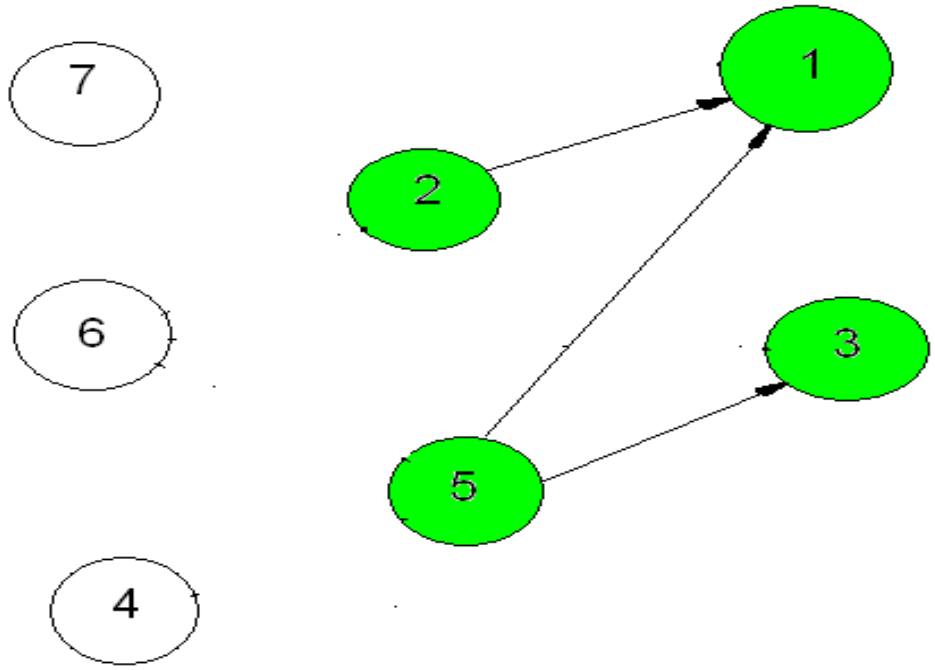
Располагают вершины графа на прямой так, чтобы все дуги вели с лева на право.

1. Выписать в произвольном порядке все вершины, которые не имеют входных дуг:

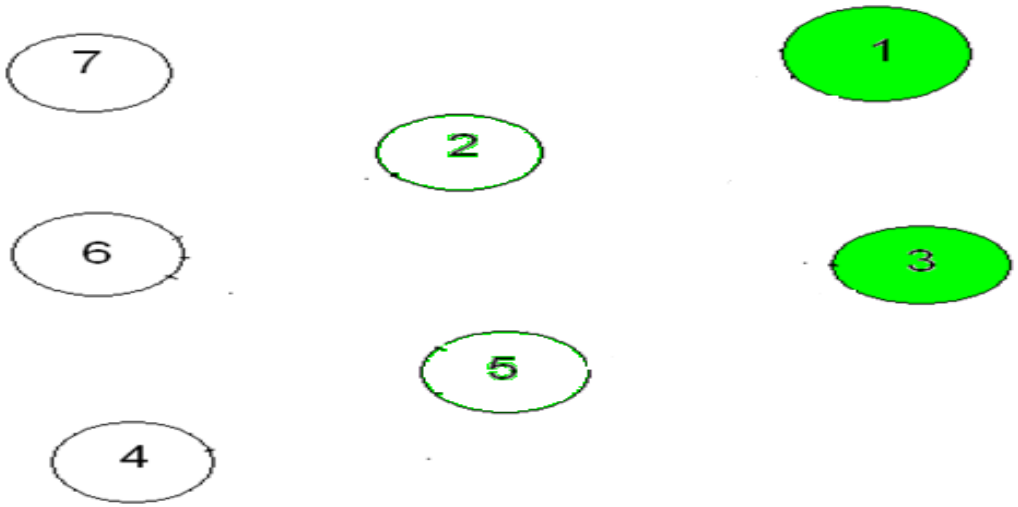
4 6 7.

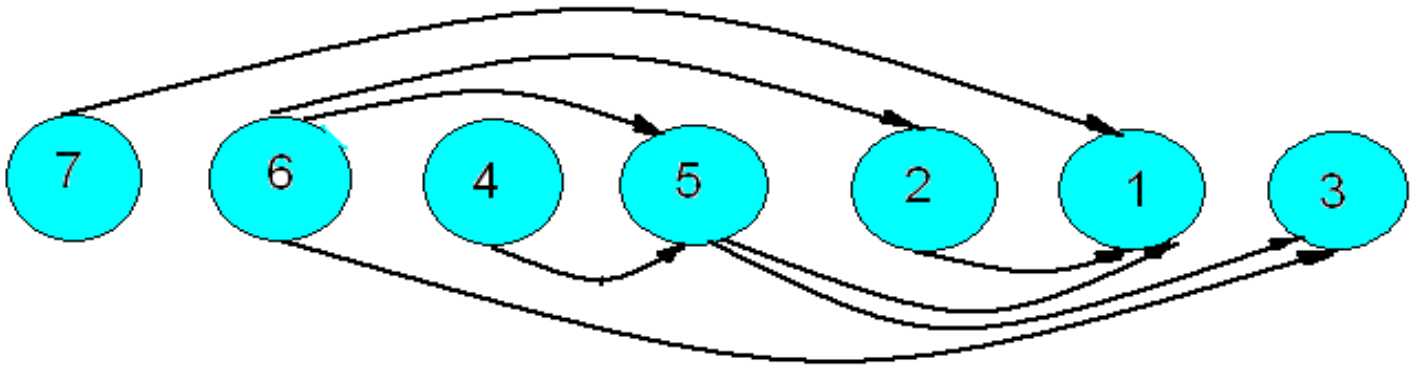

7
1
2
6
3
5
4

2. Удалить эти вершины и исходящие из них дуги.

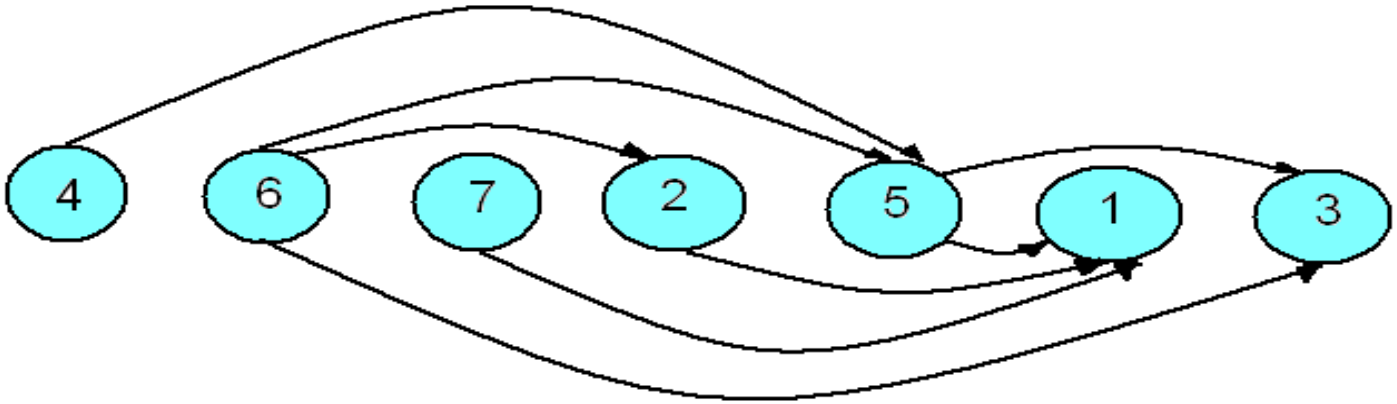


3. Если в графе ещё есть вершины, перейти в пункт один. Получим алгоритм топологической сортировки исходного графа: 7 6 4 5 2 1 3. По этому алгоритму строим граф с полученным порядком сортировки.




7
6
4
5
2
1
3

Выпишем все вершины, которые не имеют входных дуг, получим другой порядок топологической сортировки исходного графа: 4 6 7 2 5 1 3.



Представление графа в виде булевых функций

Составим булеву таблицу состояния исходного графа: A – множество номеров необработанных столбцов: A={1, 2,…,n}.

	1	2	3	4	5	6	7
1							
2	1						
3							
4					1		
5	1		1				
6		1	1		1		
7	1						

1. Найти в A номера столбцов, не содержащих единиц и выписать их в решение в произвольном порядке: 4 6 7.

2. Обнулить все строки с найденными номерами.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1							
2	1						
3							
4					0		
5	1		1				
6		0	0		0		
7	0						

3. Удалить эти номера из таблицы множества A.
4. Если таблица множества A непустая, то переходим в пункт 1, иначе заканчивается работа алгоритма.
В результате работы алгоритма получим следующий порядок топологической сортировки: 4 6 7 2 5 1 3.

	1	**2**	**3**	**5**
1				
2	1			
3				
5	1		1	

Применение топологической сортировки

Топологическая сортировка применяется в самых разных ситуациях, например при распараллеливании алгоритмов, когда по некоторому описанию алгоритма нужно составить граф зависимостей его операций и, отсортировав его топологически, определить, какие из операций являются независимыми и могут выполняться параллельно (одновременно).

Сложность метода

Сложность метода топологической сортировки определяется по соотношению:

$$T(n)=O(n\ m),$$

где n – число вершин заданного графа;

m – число дуг в графе.

Сортировка с подсчетом

Алгоритм сортировки в котором используется диапазон чисел сортируемого массива или списка для подсчёта совпадающих элементов. Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством.

Эффективность алгоритма падает, если при попадании нескольких различных элементов в одну ячейку, их надо дополнительно сортировать.

Предположим, что входной массив состоит из n целых чисел в диапазоне от 0 до $k - 1$, где k Є n. Далее алгоритм будет обобщён для произвольного целочисленного диапазона. Существует несколько модификаций сортировки подсчётом:

Простой алгоритм

Для реализации простого алгоритма необходимо создать вспомогательный массив C[0..k - 1], состоящий из нулей, затем последовательно прочитать элементы входного массива A, для каждого A[i] увеличить C[A[i]] на единицу. Теперь достаточно пройти по массиву C, для каждого $j \in \{0,\ldots, k\text{-}1\}$ в массив A последовательно записать число j C[j] раз.

Алгоритм сортировки со списком

Этот вариант используется, когда на вход подается массив структур данных, который следует отсортировать по символам (ключам).

Дан текст S= s_0, s_1, …, - 256 символов, и имеется массив A, содержащий элементы S,

необходимо определить число различных символов в тексте.

Для сортировки подсчетом создается вспомогательный нулевой массив C:

$C=c_0, c_1, \ldots, c_{255}=0 \ldots 0.$

Просматриваем текст с лева на право и для каждого i выполняем операцию подсчета и внесения количества найденных элементов в соответствующие ячейки массива C:

C[S[i]] + +.

Сложность метода определяется по соотношению:

$$T(n)=O(n+M),$$

где M – количество символов для подсчета в данном тексте;

n – количество символов в тексте.

Сортировку с подсчетом хорошо использовать там, где диапазон значений сортируемых элементов небольшой.

Организация очереди

Очередь представляет собой линейный список данных, доступ к которому осуществляется по принципу "первый вошел, первый вышел" /иногда сокращенно его называют методом доступа FIFO (FIFO, First In — First Out) /. Элемент, который был первым поставлен в очередь, будет первым получен при поиске. Элемент, поставленный в очередь вторым, при поиске будет получен также вторым и т.д.

Этот способ является единственным при постановке элементов в очередь и при поиске элементов в очереди. Применение очереди не позволяет делать прямой доступ к любому конкретному элементу.

Рассмотрим две процедуры: постановка в очередь и выборка из очереди. При выполнении процедуры постановки в очередь элемент помещается в конец очереди.

При выполнении процедуры выборки из очереди из нее удаляется первый элемент, который является результатом выполнения данной процедуры. Следует помнить, что при выборке из очереди из нее действительно удаляется один элемент. Если этот элемент нигде не будет сохранен, то в последствии к нему нельзя будет осуществить доступ.

	Операция	Содержимое очереди
1	Постановка (A) в очередь	A
1	Постановка (B) в очередь	A B
1	Постановка (C) в очередь	A B C
2	Выборка из очереди (A)	B C
1	Постановка (D) в очередь	B C D
2	Выборка из очереди (B)	C D
2	Выборка из очереди (C)	D

Алгоритм постановки элемента в очередь

b – номер элемента массива, в котором хранится первый элемент очереди;

L – номер элемента массива, в котором записывается новый элемент массива;

Если массив имеет n элементов, то максимальное число элементов должно содержатся в очереди (n-1), т. к. иначе не удастся отличить пустая или полная очередь.

Пустая очередь: b=L;

Полная очередь: (L+1) mod n = b.

Сначала обнуляем ячейки:

b=0, L=0.

Добавление элемента x в очередь:

1. Если очередь полная, то нельзя добавить элемент в очередь: завершение программы;

2. В ячейку L записываем новый элемент очереди: a[L]=x.

3. Следующем этапом очередь проверяется на полноту заполнения по соотношению:

$$L=(L+1) \bmod n,$$

и, если в очереди есть место, то к L прибавляется единица; в противном случае, когда L=n ячейка L обнуляется (L=0).

Удаление элемента из очереди в переменную x:

1. Если очередь пустая (b=L), то нельзя удалить элемент из очереди;

2. В переменную x записывается удаленный элемент: x=a[b];

3. Очередь проверяется на наличие свободных ячеек:

$$b=(b+1) \bmod n.$$

Организация очереди с приоритетами

Объекты в очереди с приоритетами подчинены закону пирамиды, т. е. при удалении и добавлении элементов в очередь, просеиваются элементы через пирамиду.

a_i – приоритет объекта – чем меньше a_i, тем выше его приоритет.

Организация очереди с приоритетом на базе массива из n элементов.

k – число элементов в очереди.

Удаление элемента из очереди:

1. Если очередь пуста, то нельзя удалить элемент из очереди;

2. Удаляют элемент a_0 из очереди в x, а k уменьшают на единицу:

$x=a_0$; k - - ;

3. В освободившуюся ячейку записывают k элемент:

a[0]=a[k];

4. Просеивают элемент a_0 через пирамиду.

Добавление элемента в очередь:

Добавление осуществляется с «низу» в «верх»:

1. Если очередь полная, то нельзя добавить элемент в очередь;

2. Из x записывают k элемент в очередь, при этом i приравнивают к «k», а «k» увеличивают на единицу:

a_k = x; i=k; k + +;

3. Подъем элемента на вершину пирамиды:

3.1. Если i = = 0, то конец работы алгоритма;

3.2. Номер родительской вершины r пирамиды определяется по соотношению:

r=[(i-1)/2], i=2r+1;

3.3. Если $a_i < a_r$, то производим обмен a_i с a_r и переходим в пункт один и i=r.

Сложность алгоритма определяется соотношением: T(n)=maxT(x)=

=O(|x| log2 |x|),

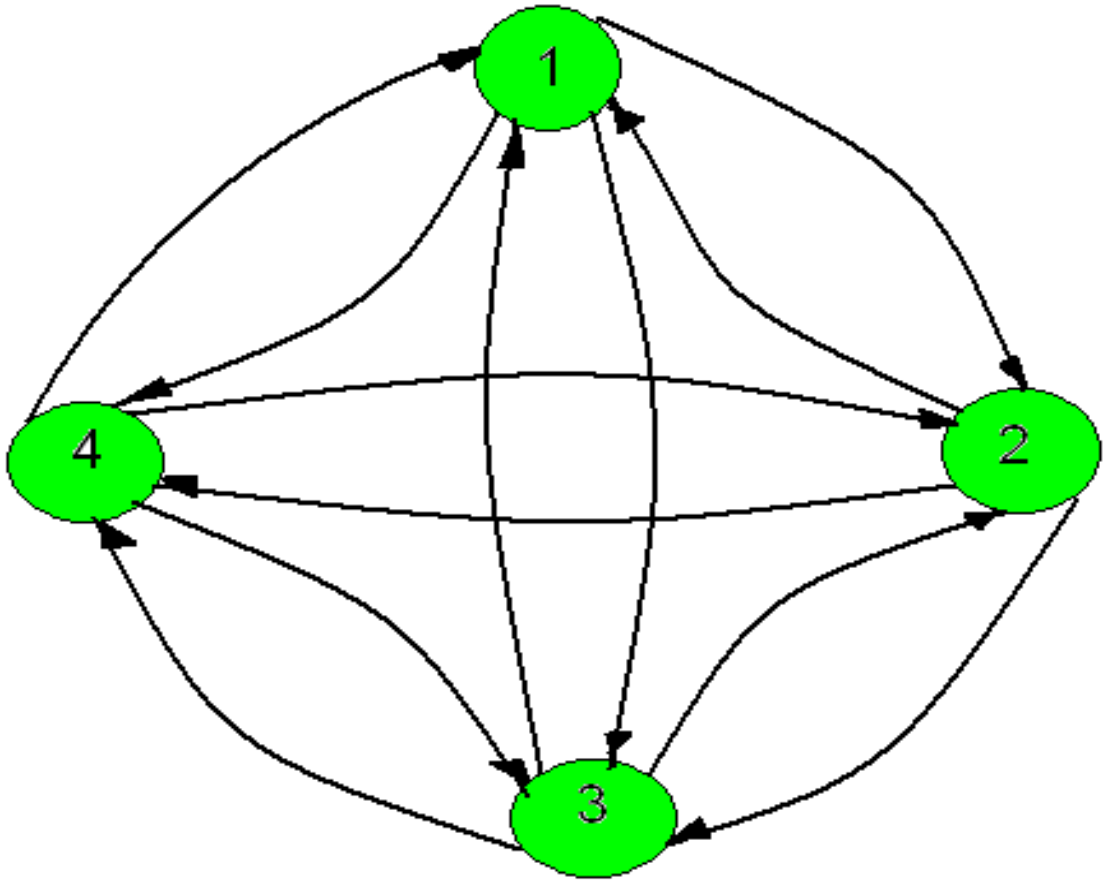
где |x|<= n - размерность входных данных.

Задача коммивояжёра

Постановка задачи:

Коммивояжёр Джек (коммивояжёр — разъездной сбытовой посредник) должен объехать n городов, заезжая в каждый город ровно один раз и при этом потратить минимальное количество денег на путевые расходы. Джеку известна стоимость проезда из каждого города в каждый, при этом он должен вернуться туда, откуда выехал.

Рассмотрим ориентированный, полный, взвешенный граф, предполагаемого пути следования коммивояжера. Известны стоимость путей любого пункта следования i в пункт j. Необходимо проехать через все вершины по одному разу и вернуться в исходный пункт за минимальную стоимость проезда.


1
2
3
4

Представление графа C[n x n],
C i j ={стоимость пути из i в j, i<>j;

Таблица стоимости пути из i в j:

	1	2	3	4
1		**10**	**5**	**17**
2	**6**		**1**	**1**
3	**9**	**10**		**4**
4	**16**	**3**	**7**	

Алгоритм перебора (перестановок, Дейкстры)

Перебор в лексиконе программирования – это порядок. Порядок определяется по формуле перестановок:

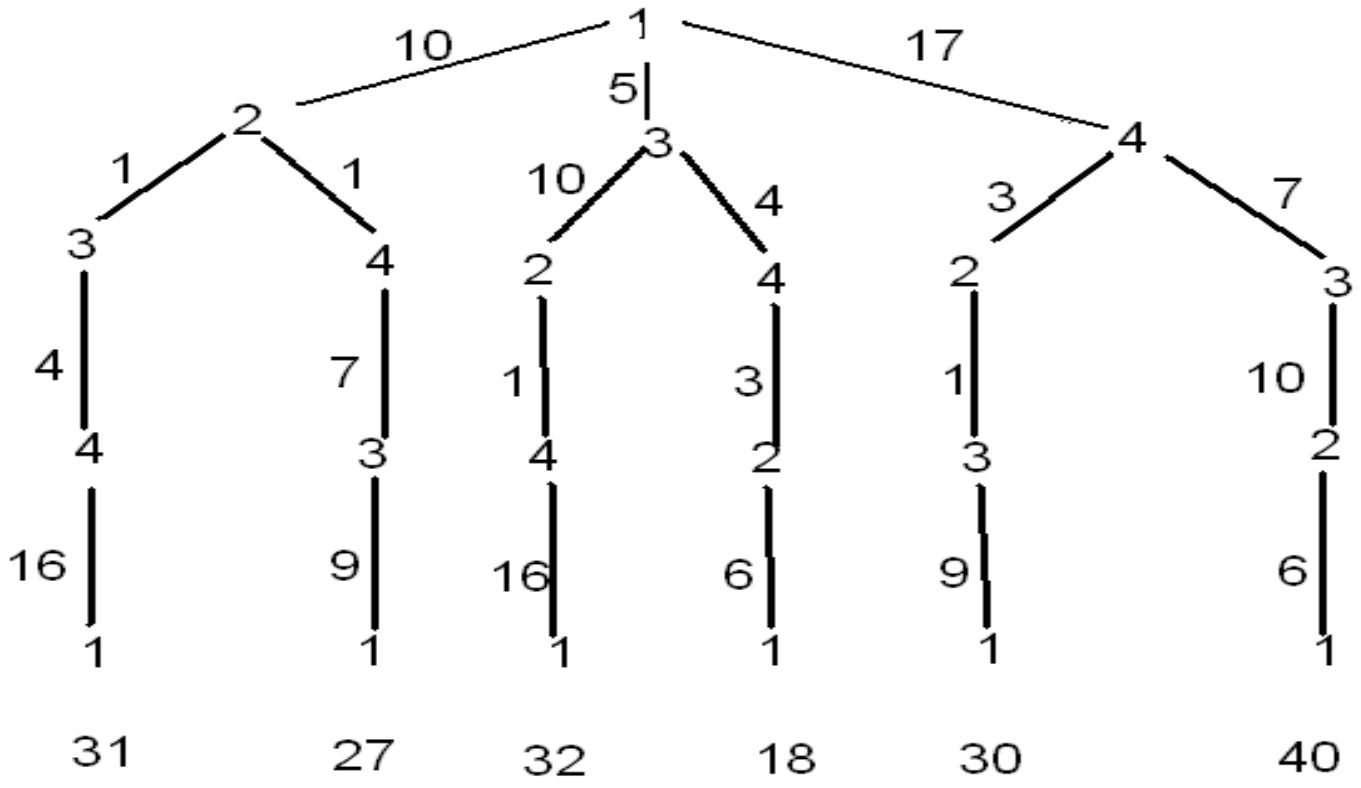
$$P_n = n!.$$

Перестановки из n=3 равняются 6 комбинациям: $P_3 = 3! = 6$.

123, 132, 213, 231, 312, 321.

Дерево перестановок

Строим дерево перестановок для задачи 4 города, с указанием на ребрах стоимости проезда.



Из дерева перестановок видно, что лучший путь включает в себя следующие ветвь перестановок: 1 – 3 – 4 – 2 – 1.

Эвристические методы решения задачи коммивояжера.

«Жадный алгоритм»

На каждом шаге выбирается ребро наименьшей стоимости из множества рёбер, не нарушающих корректности решения: выбираем ребро 2 – 3, стоимость – 1 у. е. (условная единица).

Таблица 1

	1	2	3	4
1		10	5	17
2	6		1	1
3	9	10		4
4	16	3	7	

Из таблицы 1 удаляем все дуги, входящие в вершину три. Удаляем все дуги, выходящие из вершины два. Маршрут 2 – 3.

Таблица 2

	1	2	4
1		10	17
3	9		4
4	16	3	

Из таблицы 2 выбираем маршрут 4 – 2 и, соответственно, удаляем все дуги, входящие в вершину 2 и выходящие из вершины 4.

Таблица 3

	1	4
1		17
3	9	

Из таблицы 3 выбираем маршрут 3 –1 и, соответственно, удаляем все дуги, входящие в вершину 3 и выходящие из вершины 1.

Таблица 4

	4
1	17

Из таблицы 4 выбираем маршрут 1 – 4 и соединяя все отдельные маршруты в единую цепь, получим окончательный алгоритм: 1 – 4 – 2 – 3 – 1 , стоимость маршрута 30 у. е. (условные единицы).

«Жадный алгоритм - 2»

В таблице проезда в начале подсчитываем суммарную стоимость проезда по всем строкам матрицы стоимости проезда. Из худшей строки выбираем элемент с минимальной стоимостью проезда.

Из первой строки, имеющую максимальную стоимость, выбираем минимальный элемент «5», а соответствующая дуга (путь проезда) (1 – 3) включаются в решение задачи.

	1	2	3	4
1		10	5	17
2	6		1	1
3	9	10		4
4	16	3	7	

Суммарная стоимость проезда

32

8

23

26

Из соответствующих ячеек матрицы удаляем элементы строк, выбранного пути. По строкам таблицы находим суммарную стоимость проезда. Из четвертой строки (с максимальной суммой проезда) выбираем дугу (путь) 4–2.

	1	2	3	4	
-		-	-	-	
2	6		-	1	7
3	-	10		4	14
4	16	3	-		19

Удаляем соответствующие элементы из матрицы стоимости проезда и из второй строки выбираем путь 2 – 1.

	1	2	3	4
-		-	-	-
2	6		-	-
3	-	-		4
4	-	-	-	

6

4

Из полученной таблицы выбираем путь 3 – 4. Соединяем полеченные результаты в единый путь, получим 1 – 3 – 4 – 2 – 1, стоимостью проезда 18 у. е.

	1	2	3	4
-		-	-	-
2	-		-	-
3	-	-		4
4	-	-	-	

4

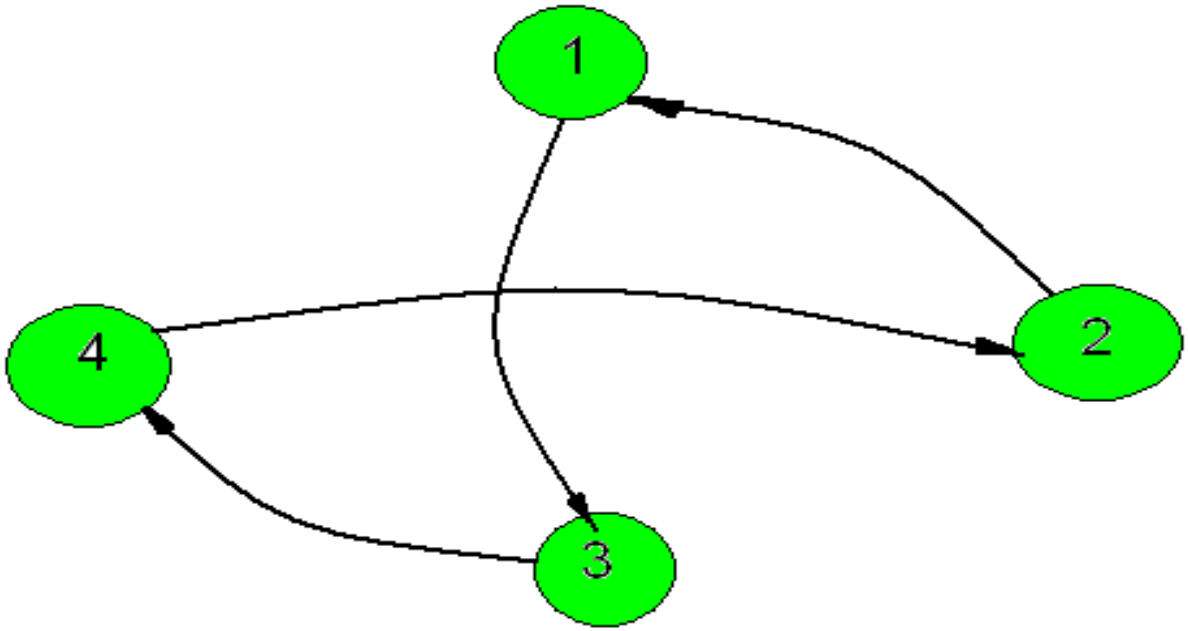
«Жадный алгоритм - 3»

Из текущей вершины один выбираем дугу (путь) с минимальной стоимости проезда 1 – 3. Из вершины 3 выбираем путь с минимальной стоимостью 3 – 4. Из вершины 4 выбираем опять путь с наименьшей стоимостью 4 – 2 . Из вершины 2 попадаем в исходную точку пути. Окончательно получим следующий путь движения коммивояжёра:

1 – 3 – 4 – 2 – 1

с суммарной стоимостью проезда – 18 у. е.

Граф движения коммивояжера имеет следующий вид:



Поиск подстроки в строке

Постановка задачи

Дана строка S, содержащая n символов:

$S = s_0 s_1 \ldots s_{n-1}$.

Под *строкой* понимается вся последовательность символов текста. Речь не обязательно должна идти именно о тексте. В общем случае строка – это любая последовательность байтов.

Задан некоторый образец P, содержащий m символов:

$P=p_0\ p_1\ \ldots\ p_{m-1}$, при этом $m \ll n$.

Поиск подстроки в строке осуществляется по заданному *образцу*, т. е. некоторой последовательности байтов, длина которой не превышает длину строки. Задача заключается в том, чтобы определить, содержит ли строка заданный образец.

Необходимо найти все номера i, в которых есть совпадения следующего вида:

$s_i = p_0, s_{i+1} = p_1, ..., s_{i+m-1} = p_{m-1}$.

Либо сказать, что заданные совпадения отсутствуют.

Прямой поиск

Это посимвольное сравнение строки с подстрокой.

Сравнение образца и строки выполняются с лева на право. Сравнение всегда начинается с нулевого символа строки:

S = abcaefabc, p = bc.

Как только встречается несовпадение символов образец P сдвигают на единицу вправо:

S = abcaefabc, p = bc.

Записывается i номер совпадения (i=1) и образец передвигают на единицу вправо.

Проводится вся поверка строки на совпадение символов строки и образца и записывают номера совпадения:

S = abcaefabc, p = bc,

i=1, 7.

Поиск подстроки в строке. Алгоритм Бойера, Мура (БМ поиск)

Поиск ведется от начала строки s, но с конца искомой подстроки (образца) P. Далее мы совмещаем начало строки и образца и начинаем проверку с последнего символа образца. Если последний символ образца и соответствующий ему при наложении символ строки не совпадают, образец сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа образца. Если же символы совпадают, производится сравнение предпоследнего символа образца и т. д.

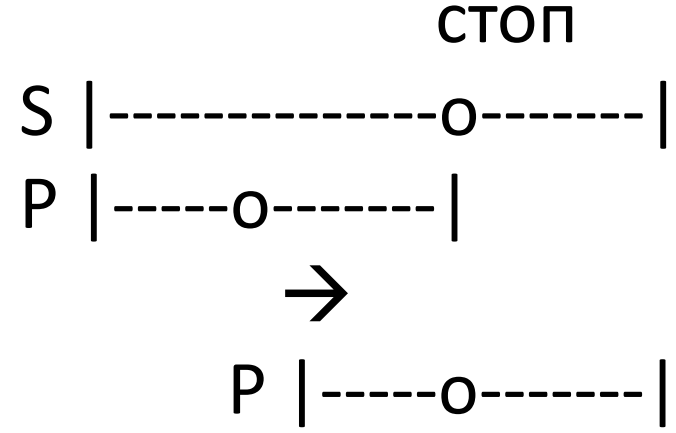
Сравнение по стоп - символу

Стоп – символом называется символ строки S, который сравнивается с символом образца P_{m-1} (последний элемент образца).

а) Если среди элементов образца:

$P_0, P_1, \ldots, P_{m-2}$,

нет стоп – символа, то осуществляется полный сдвиг образца (т. е. сдвиг на величину m).



b) Если элемент образца P_j совпал со стоп символом и 0 <= j <= m-2, то j принимает максимальное значение и образец сдвигают так, чтобы символ j оказался под стоп – символом.

Таблица смещений

Поиск ведется от начала строки S, но с конца искомой подстроки P, для которой формируется таблица, размерность которой равна 256 - количеству всех символов в машинном алфавите. В таблице записываются расстояния от последнего символа искомой подстроки P до каждого ее символа. (Если в P встречаются одинаковые символы, то в таблицу заносится расстояние до ближайшего из них.) Если символ не входит в P, то в соответствующую ячейку таблицы заносится m - длина подстроки P.

Построение вспомогательной таблицы d

d[c]=m, если символа «c» нет среди символов образца: P_0, P_1, …, P_{m-2} .

d[c]=m-j-1, если c = P_j , где j максимальный индекс образца.

d[c] – количество символов на которое надо сдвинуть P, если символ «c» является стоп – символом.

Вычисление для j от 0 до m-2 проводится по соотношению

d[P[i]] = m – 1 – j.

Пример построения одномерного массива

Пусть имеется набор символов из пяти символов: *a, b, c, d, e* и необходимо найти вхождение образца P:“abbad” в строке S: “abeccacbadbabbad”.

Определяем таблицу смещений для образца P: “abbad” .

a	b	c	d	e
1	2	5	0	5

Символы «c, e» не входят в символы образца, поэтому в таблице m=5.

a b e c c a c b a d b a b b a d

a b b a d

Начало поиска. Последний символ образца не совпадает с наложенным символом строки. Сдвигаем образец вправо на 5 позиций:

a b e c c a c b a d b a b b a d

a b b a d

Три символа образца совпали, а четвертый – нет. Сдвигаем образец вправо на одну позицию:

a b e c c a c b a d b a b b a d

a b b a d.

Последний символ снова не совпадает с символом строки. В соответствии с таблицей смещений сдвигаем образец на 2 позиции:

a b e c c a c b a d b a b b a d

a b b a d.

Еще раз сдвигаем образец на 2 позиции:

a b e c c a c b a d b a b b a d

a b b a d.

Теперь, в соответствии с таблицей, сдвигаем образец на одну позицию, и получаем искомое вхождение образца:

a b e c c a c b a d b a b b a d

a b b a d.

Алгоритм Бойера, Мура (БМ поиск)

1. Построить таблицу смещений «d»;
2. i=m-1;
3. j=m-1, k=I;
4. Пока (j>=0 и P[j]==S[k]), j--, k--;
5. Если j<0, то выводится промежуточный ответ, k увеличивают на единицу и (выход в пункт два);
6. i=i+d[S[i]], сдвиг образца вправо;
7. Если i<n то переход в пункт три, иначе выход (окончание программы).

В самом худшем случае, число сравнений приблизительно равно nxm.

Поиск подстроки в строке. Алгоритм Кнута, Морриса, Пратта (КМП поиск)

Это алгоритм поиска подстроки в строке, при котором сдвиг подстроки выполняется на некоторое переменное количество символов.

Образец P сдвигается с лева на право по строке символов S. Сравнение символов образца также проводится с лева на право.

В КМП-поиске не происходит возврата по строке S.

Поиск в алгоритме КМП исполняется до одного вхождения.

Префикс – это подстрока, начинающаяся с первого символа строки.

Префикс строки **S[**..i**]** — это строка из i первых символов строки **S**.

Назовем префиксом длины i строки s ее подстроку с 1 по i-й символ. В частности, префикс длины len(s) - это сама строка s, префикс длины 0 - пустая строка. Префикс длины i будем обозначать pref(s,i), а суффикс - suf(s,i).

Суффикс – это подстрока, заканчивающаяся на последний символ строки.

Суффикс строки **S[**j..**]** — это строка из

|**S**|-j+1 последних символов.

Символ P_j образца P является **ошибочным,**

если при очередном сравнении образца со строкой S:

$$P_0 = S_{i-j}\ ,\ P_1 = S_{i-j+1}\ ,\ \ldots,\ P_{j-1} = S_{i-1}\ ,$$

$$P_j <> S_i\ .$$

Сдвиг образца

Правило, позволяющее сдвинуть образец вправо, при этом не потеряв решение. Пусть P_j ошибочный символ, если в образце P есть префикс, совпадающий с суффиксом, то образец сдвигают так, чтобы префикс пришел на место суффикса. Сравнение символов продолжается с символа, следующего за префиксом.

Если в строке P_j нет префикса равного суффиксу, то можно выполнить «полный» сдвиг образца так, чтобы сравнивать

P_0 и S_i.

Сдвиг образца КМП поиске выполняется за счет изменения значения j – номера ошибочного символа, а именно – уменьшение номера ошибочного символа.

В самом худшем случае, число сравнений приблизительно равно n+m.

Алгоритм Кнута, Морриса и Пратта основывается на том, что после частичного совпадения начальной части подстроки с соответствующими символами строки можно, вычислить сведения, с помощью которых быстро продвинуться по строке.

Задачи поиска слова в тексте используются в криптографии, различных разделах физики, сжатии данных, распознавании речи и других сферах человеческой деятельности.

Список вопросов, курс «алгоритмы и анализ сложности»

1. Алгоритм. Свойства алгоритмов.
2. Алгоритм. Типы алгоритмов.
3. Алгоритм. Основные требования к алгоритмам.
4. Анализ трудоемкости алгоритмов.
5. Алгоритм сортировки вставками, анализ сложности алгоритма.
6 .Алгоритм слияния, анализ сложности алгоритма.
7. Алгоритм сортировки «пирамида», анализ сложности.
8. Алгоритм поиска элемента в упорядоченном массиве.
9. Поиск медианы в массиве, анализ сложности.
10. Алгоритм сортировки, метод Хоара: выбор эталонного значения.

11. Поразрядная сортировка.
12. Топологическая сортировка.
13. Сортировка с подсчетом.
14. Организация очереди: без приоритета.
15. Организация очереди: с приоритетом.
16. Задача коммивояжера, дерево перестановок.
17. Задача коммивояжера, «жадный алгоритм».
18. Поиск подстроки в строке, прямой поиск.
19. Поиск подстроки в строке, алгоритм Бойера, Мура (БМ поиск).
20. Поиск подстроки в строке, метод Кнута, Мориса, Пратта (КМП поиск).

Список лабораторных работ «алгоритмы и анализ сложности»

- Создание одномерного массива с датчиком случайных чисел.
- Сортировка массива: метод пузырька.
- Сортировка массива: метод Шейкера.
- Сортировка массива: метод Хоара.
- Сортировка массива: метод Шелла.
- Сортировка массива: сортировка с помощью «кучи»;«пирамида».