Дискретная математика. Часть 2

Теория алгоритмов и теория графов

Лабораторный практикум

Факультет компьютерных наук ВГУ

Воронеж 2015

О нашем курсе

1. Введение в теорию алгоритмов
2. Математические основы анализа алгоритмов
3. Поиск
4. Сортировка
5. Порядковые статистики
6. Алгоритмы на графах

Литература

Литература


С.В. Борзунов, С.Д. Кургалин.
Практикум по дискретной математике (для программистов).
Воронеж : ИПЦ ВГУ, 2012.

Литература

С.В. Борзунов, С.Д. Кургалин.
Практикум по дискретной математике (для программистов).
Воронеж : ИПЦ ВГУ, 2012.

Дж. Макконелл.
Анализ алгоритмов. Активный обучающий подход.
3-е дополненное издание. М. : Техносфера, 2009.

Литература

С.В. Борзунов, С.Д. Кургалин.
Практикум по дискретной математике (для программистов).
Воронеж : ИПЦ ВГУ, 2012.

Дж. Макконелл.
Анализ алгоритмов. Активный обучающий подход.
3-е дополненное издание. М. : Техносфера, 2009.

Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн.
Алгоритмы. Построение и анализ.
3-е издание. М., СПб., Киев : Вильямс, 2013.

Литература


С.В. Борзунов, С.Д. Кургалин.
Практикум по дискретной математике (для программистов).
Воронеж : ИПЦ ВГУ, 2012.


Дж. Макконелл.
Анализ алгоритмов. Активный обучающий подход.
3-е дополненное издание. М. : Техносфера, 2009.


Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн.
Алгоритмы. Построение и анализ.
3-е издание. М., СПб., Киев : Вильямс, 2013.

В учебниках встречаются опечатки.
Можно решать дополнительные задачи из «Практикума».

Цели курса

О нашем курсе

Цели курса

- Познакомиться с основными алгоритмами.
  - Рекурсивные алгоритмы
  - Поиск
  - Сортировка
  - Алгоритмы на графах

Цели курса

- Познакомиться с основными алгоритмами.
  - Рекурсивные алгоритмы
  - Поиск
  - Сортировка
  - Алгоритмы на графах
- Научиться анализировать алгоритмы. Различия по:
  - Эффективности
  - Требованиям к ресурсам

Определение алгоритма

Определение

Алгоритм (algorithm) — это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

Определение алгоритма

Определение

Алгоритм (algorithm) — это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

Многие исследователи пользуются разными формулировками определения понятия алгоритма, отличающимися друг от друга.

Альтернативные определения алгоритма

Алгоритм по Колмогорову

Алгоритм (Колмогоров) — это всякая система вычислений, выполняемых по строго определённым правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

Альтернативные определения алгоритма

Алгоритм по Колмогорову

Алгоритм (Колмогоров) — это всякая система вычислений, выполняемых по строго определённым правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

Алгоритм по Кормену и др.

Алгоритм (Кормен и др.) — корректно определённая вычислительная процедура, на вход которой подаётся некоторая величина или набор величин, и результатом выполнения которой является выходная величина или набор значений.

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность
- элементарность шагов

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность
- элементарность шагов
- детерминированность

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность
- элементарность шагов
- детерминированность
- направленность

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность
- элементарность шагов
- детерминированность
- направленность
- массовость

Основные свойства

Понятие алгоритма относится к базовым, фундаментальным понятиям математики. Во всех формулировках явно или неявно подразумеваются следующие **свойства алгоритма**:

- дискретность
- элементарность шагов
- детерминированность
- направленность
- массовость

Возможность применения конкретного алгоритма в компьютерной программе требует обоснования правильного решения задачи для всех входных данных, т. е. **доказательства корректности алгоритма**. С математической точки зрения речь идёт об установлении истинностных значений некоторых предикатов, описывающих переменные величины.

Разделы теории алгоритмов

1. классическая теория алгоритмов (формулировка задач в терминах формальных языков, описание классов задач, проблема $P \stackrel{?}{=} NP$)
2. теория асимптотического анализа алгоритмов (понятие сложности алгоритма, критерии оценки алгоритмов, методы получения асимптотических оценок)
3. теория практического анализа вычислительных алгоритмов (практически значимые критерии качества алгоритмов)
4. методы создания эффективных алгоритмов (динамическое программирование, «жадные» алгоритмы, специальные структуры данных)

Рекурсия

Рекурсивным называется алгоритм, который обращается к самому себе. Такие алгоритмы обычно просты для понимания и удобны в практической реализации.

Рекурсия

Рекурсивным называется алгоритм, который обращается к самому себе. Такие алгоритмы обычно просты для понимания и удобны в практической реализации.

Пример. Рекурсивное вычисление определителя матрицы A.

$$\det A = \begin{cases} a_{11}a_{22} - a_{12}a_{21}, & n = 2; \\ a_{11}\det \mathcal{A}_{[11]} - a_{12}\det \mathcal{A}_{[12]} + \cdots + a_{1n}\det \mathcal{A}_{[1n]}, & n > 2. \end{cases}$$

Знаки чередуются!
$\mathcal{A}_{[ij]}$ — алгебраический минор элемента a_{ij}, т. е. матрица размера $(n-1)\times(n-1)$, получающаяся из A вычёркиванием i-й строки и j-го столбца.

Пример вычисления $\det A$

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}, \quad n = 4.$$

$$\begin{aligned} \det A &= 1 \cdot \det \begin{pmatrix} 6 & 7 & 8 \\ 10 & 11 & 12 \\ 14 & 15 & 16 \end{pmatrix} - 2 \cdot \det \begin{pmatrix} 5 & 7 & 8 \\ 9 & 11 & 12 \\ 13 & 15 & 16 \end{pmatrix} + \\ &+ 3 \cdot \det \begin{pmatrix} 5 & 6 & 8 \\ 9 & 10 & 12 \\ 13 & 14 & 16 \end{pmatrix} - 4 \cdot \det \begin{pmatrix} 5 & 6 & 7 \\ 9 & 10 & 11 \\ 13 & 14 & 15 \end{pmatrix} = \\ &= 1 \cdot [6 \cdot \det \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix} - 7 \cdot \det \begin{pmatrix} 10 & 12 \\ 14 & 16 \end{pmatrix} + 8 \cdot \det \begin{pmatrix} 10 & 11 \\ 14 & 15 \end{pmatrix}] - 2 \cdot \ldots = \\ &= 1 \cdot [6 \cdot (11 \cdot 16 - 12 \cdot 15 + \cdots)] + \ldots = 0. \end{aligned}$$

Есть более эффективные методы вычисления $\det A$. $\square$

Задача

Задача 1. Вычисление определителя матрицы

В текстовом файле `input.txt` записаны подряд по строкам элементы целочисленной квадратной матрицы A. Используя рекурсию, вычислите определитель $\det A$. Результат выведите в текстовый файл `output.txt`.

Математические вопросы

1. Асимптотические соотношения
2. Конечные суммы
3. Факториал и кофакториал
4. Рекуррентные соотношения

Асимптотические соотношения

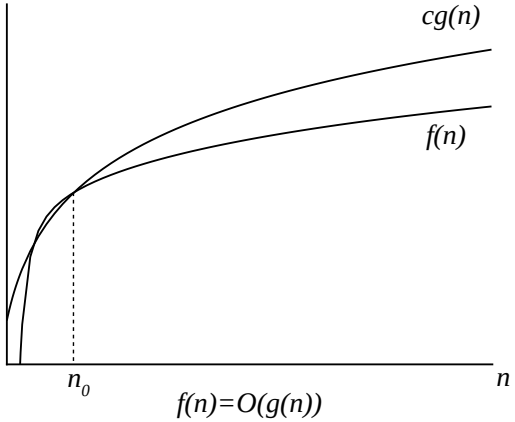
Пусть размер входных данных определяется переменной n. Исследуемый алгоритм совершает $T(n)$ элементарных операций (например, умножений).

Основная идея — оценить скорость роста $T(n)$ при больших n.

Асимптотические соотношения: $O(f)$ и $\Omega(f)$

Если $\exists n_0$ и c, что $\forall n \geqslant n_0$ выполняются неравенства $0 \leqslant f(n) \leqslant cg(n)$, $c > 0$, то пишут

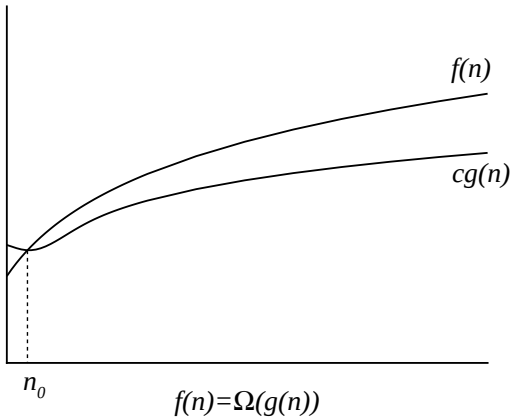
$$\boxed{f(n) = O(g(n)).}$$



Пример O-символики.

Если $\exists n_0$ и c, что $\forall n \geqslant n_0$ выполняются неравенства $0 \leqslant cg(n) \leqslant f(n)$, $c > 0$, то пишут

$$\boxed{f(n) = \Omega(g(n)).}$$

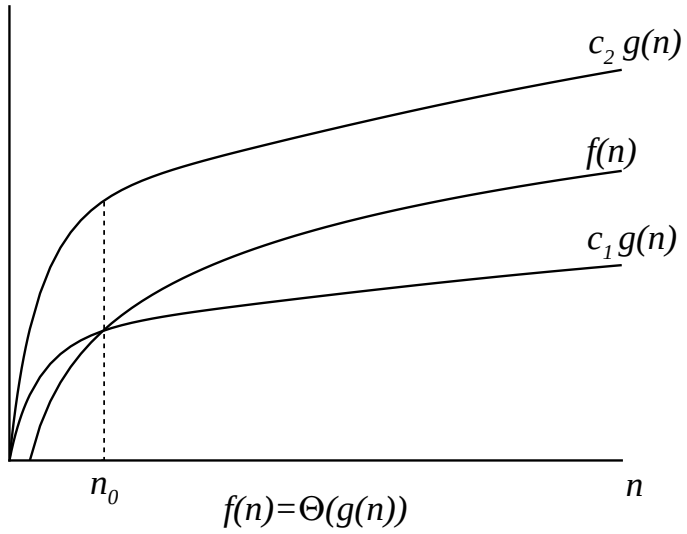


Пример Ω-символики.

Асимптотические соотношения: $\Theta(f)$

Если $\exists n_0, c_1, c_2$, что $\forall n \geqslant n_0$ выполняются неравенства $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$, $c_1, c_2 > 0$, то пишут

$$\boxed{f(n) = \Theta(g(n)).}$$



Пример Θ-символики.

Асимптотические соотношения

Пример. Пусть $f_1(n) = \frac{n^2}{2} - 3n$, покажем, что $f_1(n) = \Theta(n^2)$.

Асимптотические соотношения

Пример. Пусть $f_1(n) = \frac{n^2}{2} - 3n$, покажем, что $f_1(n) = \Theta(n^2)$.

Решение.

По определению $\Theta(n)$, для всех достаточно больших $n \geqslant n_0$ должны выполняться неравенства:

$$c_1 n^2 \leqslant \frac{n^2}{2} - 3n \leqslant c_2 n^2.$$

Разделим на n^2: $c_1 \leqslant \frac{1}{2} - \frac{3}{n} \leqslant c_2$. Возьмём любое n_0, например, $n_0 = 7$. Тогда существуют такие c_1, c_2, что неравенства верны, например, $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}$. **Вывод**: $\frac{n^2}{2} - 3n = \Theta(n^2)$.

Но, например, $f_2(n) = \frac{1}{1000} n^3 \neq \Theta(n^2)$, т. к. для любых n_0 и c_2 существует такое $n > n_0$, что $\frac{1}{1000} n^3 > c_2 n^2$. □

Асимптотические соотношения

Полиномом степени d от аргумента n называется функция вида $p(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_0$.

Лемма

Для любого полинома $p(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_0$, $a_d > 0$, справедлива оценка

$$p(n) = \Theta(n^d),$$

т. е. скорость роста полинома определяется старшим членом.

Запись $f(n) = \Theta(g(n))$ подразумевает две оценки — сверху и снизу.

Асимптотические соотношения

Условные взаимоотношения с отношениями порядка:

$$\begin{array}{lcl} f(n) = O(g(n)) & \sim & \text{“}\leqslant\text{”} \\ f(n) = \Theta(g(n)) & \sim & \text{“}=\text{”} \\ f(n) = \Omega(g(n)) & \sim & \text{“}\geqslant\text{”} \end{array}$$

$$\begin{array}{lcl} f(n) = o(g(n)) & \sim & \text{“}<\text{”} \\ f(n) = \omega(g(n)) & \sim & \text{“}>\text{”} \end{array}$$

Стандартные определения из математического анализа:

$$f(n) = o(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f}{g} = 0, \text{ или } 0 < f(n) < cg(n);$$

$$f(n) = \omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{g}{f} = 0, \text{ или } 0 < cg(n) < f(n).$$

Асимптотические соотношения

$O(f)$ — класс функций, растущих не быстрее $f(n)$,
$\Theta(f)$ — класс функций, растущих с той же скоростью, что и $f(n)$,
$\Omega(f)$ — класс функций, растущих по крайней мере так же быстро, как и $f(n)$.

Пример. Расположим функции в порядке возрастания скорости роста: $n^3 + \log_2 n$, 2^{n-1}, $n \log_2 n$, 6, $\left(\frac{3}{2}\right)^n$.

Асимптотические соотношения

$O(f)$ — класс функций, растущих не быстрее $f(n)$,
$\Theta(f)$ — класс функций, растущих с той же скоростью, что и $f(n)$,
$\Omega(f)$ — класс функций, растущих по крайней мере так же быстро, как и $f(n)$.

Пример. Расположим функции в порядке возрастания скорости роста: $n^3 + \log_2 n,\ 2^{n-1},\ n \log_2 n,\ 6,\ \left(\frac{3}{2}\right)^n$.

Ответ: $6,\ n \log_2 n,\ n^3 + \log_2 n,\ \left(\frac{3}{2}\right)^n,\ 2^{n-1}$. □

Конечные суммы

$$\sum_{i=1}^{n} 1 = n, \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \tag{1}$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1, \qquad \sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2, \tag{2}$$

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1), \qquad \sum_{i=1}^{n} \log_2 i = n \log_2 n + O(n), \tag{3}$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right). \tag{4}$$

Соотношение (4) называется формулой Стирлинга.

Факториал и кофакториал

Определение

Факториалом числа n называется произведение натуральных чисел до n включительно. Обозначение — $n!$.
Кофакториалом числа n называется произведение натуральных чисел до n включительно одинаковой с n чётности. Обозначение — $n!!$.

Факториал и кофакториал

Определение

Факториалом числа n называется произведение натуральных чисел до n включительно. Обозначение — $n!$.
Кофакториалом числа n называется произведение натуральных чисел до n включительно одинаковой с n чётности. Обозначение — $n!!$.

Пример. Оценим асимптотическое поведение величины $(4n)!!$.

Решение.
По определению имеем:

$$(4n)!! = \underbrace{2 \cdot 4 \cdot \ldots \cdot (4n-2) \cdot (4n)}_{2n \text{ сомножителей}}.$$

Вынесем 2^{2n}:

$$(4n)!! = 2^{2n} \cdot (1 \cdot 2 \cdot \ldots \cdot (2n-1)(2n)) = 2^{2n} \cdot (2n)!.$$

Осталось воспользоваться формулой Стирлинга.

$$(4n)!! = 2^{2n} \cdot \sqrt{2\pi(2n)} \left(\frac{2n}{e}\right)^{2n} \left(1 + \Theta\left(\frac{1}{n}\right)\right) =$$

$$= 2\sqrt{\pi} \left(\frac{4}{e}\right)^{2n} n^{2n+1/2} \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

□

Рекуррентные соотношения

Часто сложность алгоритма можно выразить в терминах сложности той же задачи, но для меньшего размера входных данных.

Теорема (Bentley, Haken, Saxe — 1980)

Пусть $T(n) = aT(\frac{n}{b}) + f(n)$, где под $\frac{n}{b}$ понимается либо $\lfloor \frac{n}{b} \rfloor$, либо $\lceil \frac{n}{b} \rceil$, $a \geqslant 1$, $b > 1$. Тогда:

Рекуррентные соотношения

Часто сложность алгоритма можно выразить в терминах сложности той же задачи, но для меньшего размера входных данных.

Теорема (Bentley, Haken, Saxe — 1980)

Пусть $T(n) = aT(\frac{n}{b}) + f(n)$, где под $\frac{n}{b}$ понимается либо $\lfloor \frac{n}{b} \rfloor$, либо $\lceil \frac{n}{b} \rceil$, $a \geqslant 1$, $b > 1$. Тогда:

1. если $f(n) = O(n^{\log_b a - \varepsilon})$ для некоторого $\varepsilon > 0$, то $T(n) = \Theta\left(n^{\log_b a}\right)$,

Рекуррентные соотношения

Часто сложность алгоритма можно выразить в терминах сложности той же задачи, но для меньшего размера входных данных.

Теорема (Bentley, Haken, Saxe — 1980)

Пусть $T(n) = aT(\frac{n}{b}) + f(n)$, где под $\frac{n}{b}$ понимается либо $\lfloor \frac{n}{b} \rfloor$, либо $\lceil \frac{n}{b} \rceil$, $a \geqslant 1$, $b > 1$. Тогда:

1. если $f(n) = O(n^{\log_b a - \varepsilon})$ для некоторого $\varepsilon > 0$, то $T(n) = \Theta\left(n^{\log_b a}\right)$,
2. если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta\left(n^{\log_b a} \log_2 n\right)$,

Рекуррентные соотношения

Часто сложность алгоритма можно выразить в терминах сложности той же задачи, но для меньшего размера входных данных.

Теорема (Bentley, Haken, Saxe — 1980)

Пусть $T(n) = aT(\frac{n}{b}) + f(n)$, где под $\frac{n}{b}$ понимается либо $\lfloor \frac{n}{b} \rfloor$, либо $\lceil \frac{n}{b} \rceil$, $a \geqslant 1$, $b > 1$. Тогда:

1. если $f(n) = O(n^{\log_b a - \varepsilon})$ для некоторого $\varepsilon > 0$, то $T(n) = \Theta\left(n^{\log_b a}\right)$,
2. если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta\left(n^{\log_b a} \log_2 n\right)$,
3. если $f(n) = \Omega(n^{\log_b a + \varepsilon})$ для некоторого $\varepsilon > 0$, и $af(\frac{n}{b}) \leqslant cf(n)$ для некоторого $c < 1$ и больших n (**условие регулярности**), то $T(n) = \Theta(f(n))$.

Задача 2. Сравнение скоростей роста функций

Расположите функции

$(\sqrt{2})^{\log_2 n}$; n^2; $n!$; $(\log_2 n)!$; $\left(\frac{3}{2}\right)^n$; n^3;
$(\log_2 n)^2$; $\log_2(n!)$; 2^{2^n}; $n^{1/\log_2 n}$; $\ln\ln n$; $n \cdot 2^n$;
$n^{\log_2 \log_2 n}$; $\ln n$; 1; $2^{\log_2 n}$; $(\log_2 n)^{\log_2 n}$; e^n;
$(2n)!!$; $\sqrt{\log_2 n}$; $2^{\sqrt{2\log_2 n}}$; $n \log_2 n$; 2^n; n

в порядке увеличения скорости их асимптотического роста. Отдельно выделите группы функций, растущих с одинаковой скоростью.

Задачи поиска и выборки

В задачах поиска и выборки применяют:

1. Последовательный поиск
2. Двоичный поиск
3. Интерполяционный поиск
4. Алгоритмы определения порядковых статистик

Задача о порядковых статистиках (задача о выборке) будет рассмотрена после изучения алгоритмов сортировки.

Последовательный поиск

Просматриваем массив по одному элементу, пока не найдём элемент, равный ключевому.

```
SequentialSearch (list, target, N)
// list       массив для просмотра
// target     целевое значение
// N          число элементов в массиве

  for i=1 to N do
     if (target=list[i])
        return i
     end if
  end for
  return 0
```

Анализ алгоритма последовательного поиска

- Наихудший случай. Число сравнений равно $W(N) = N$.

Анализ алгоритма последовательного поиска

- Наихудший случай. Число сравнений равно $W(N) = N$.
- Наилучший случай. Число сравнений равно $B(N) = 1$.

Анализ алгоритма последовательного поиска

- Наихудший случай. Число сравнений равно $W(N) = N$.
- Наилучший случай. Число сравнений равно $B(N) = 1$.
- Средний случай. Если целевое значение присутствует в массиве, то $A(N) = \frac{1}{N} \sum_{i=1}^{N} i = \frac{1}{N} \frac{N(N+1)}{2} = \frac{N+1}{2}$.
  Если `target` может не оказаться в массиве, то
  $A(N) = \frac{1}{N+1} \left(\sum_{i=1}^{N} i + N \right) = \frac{1}{N+1} \left(\frac{N(N+1)}{2} + N \right) =$
  $= \frac{N}{2} + 1 - \frac{1}{N+1} \approx \frac{N+2}{2}$.

Двоичный поиск

```
BinarySearch (list, target, N)
// list      массив для просмотра
// target    целевое значение
// N         число элементов в массиве
  start=1
  end=N
  while start<=end do
    middle=(start+end)/2
    select(Compare(list[middle],target)) from
       case -1: start=middle+1
       case  0: return middle
       case  1: end=middle-1
    end select
  end while
  return 0
```

Анализ алгоритма двоичного поиска

1	2	3	4	5	6	7

$N = 7,\ k = 3$

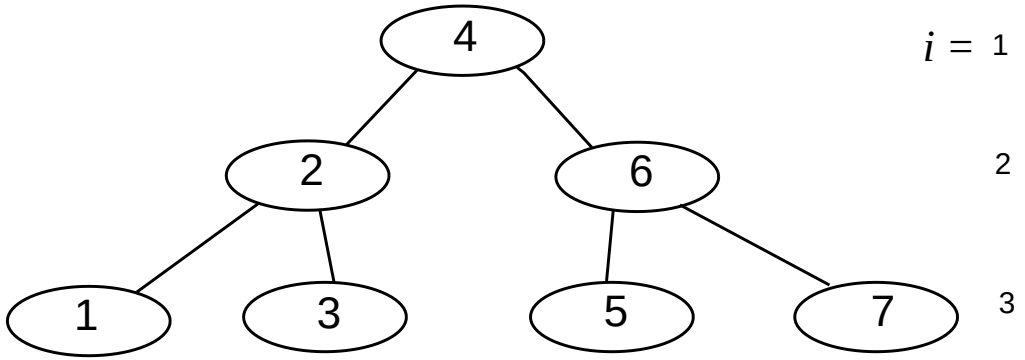
$$Compare(x, y) = \begin{cases} -1,\ x < y; \\ 0,\ x = y; \\ 1,\ x > y. \end{cases}$$

Анализ алгоритма двоичного поиска

1	2	3	4	5	6	7

$N = 7,\ k = 3$

$$Compare(x, y) = \begin{cases} -1, & x < y; \\ 0, & x = y; \\ 1, & x > y. \end{cases}$$



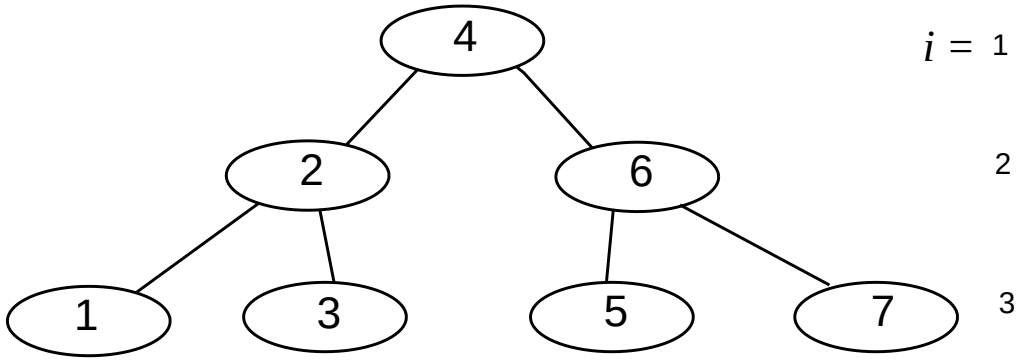
Дерево сравнений

Анализ алгоритма двоичного поиска

1	2	3	4	5	6	7

$N = 7,\ k = 3$

$$Compare(x, y) = \begin{cases} -1, & x < y; \\ 0, & x = y; \\ 1, & x > y. \end{cases}$$



Дерево сравнений

Наихудший случай. Пусть $N = 2^k - 1$, при каждом проходе степень уменьшается на 1.

Число проходов равно $\leqslant k \Rightarrow W(N) = \log_2(N + 1)$.

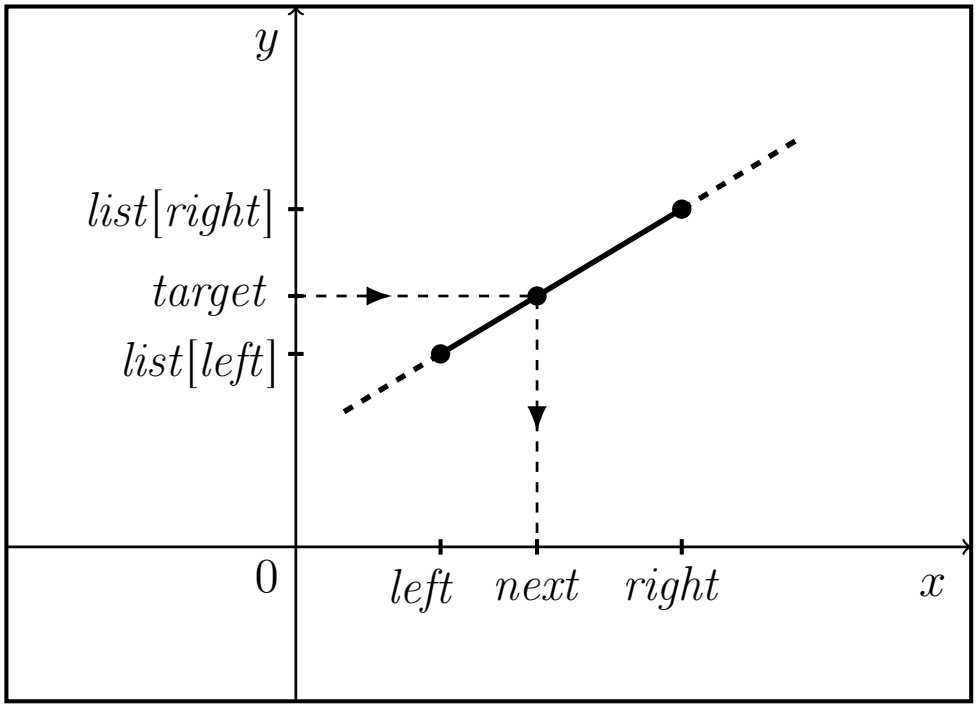
Анализ алгоритма двоичного поиска

Средний случай. Пусть целевое значение присутствует в массиве. На i-м уровне расположены 2^{i-1} узлов. Полное число сравнений получим, просуммировав числа сравнений на каждом уровне: $A(N) = \frac{1}{N}\sum_{i=1}^{k} i2^{i-1} = \frac{1}{2N}\sum_{i=1}^{k} i2^{i}$.
По формуле (2) из раздела «Конечные суммы» имеем: $A(N) = \frac{k2^k}{N} - 1$. Поскольку $N = 2^k - 1$, то среднее число сравнений, выполняемых алгоритмом двоичного поиска, равно $A(N) = \log_2(N+1) - 1$.

Если `target` нет в массиве, то
$A(N) = \frac{1}{2N+1}\left(\sum_{i=1}^{k} i2^{i-1} + (N+1)k\right) = \log_2(N+1) - \frac{1}{2}$.

Интерполяционный поиск

Интерполяционный поиск основан на предположении, что значения элементов в массиве растут линейно. Искомое значение сравнивается со значением элемента, индекс которого вычисляется как координата точки на прямой, соединяющей точки $(left, list[left])$ и $(right, list[right])$.



Интерполяционный поиск

По данному значению *target* получаем индекс элемента, с которым произойдёт следующее сравнение (см. рисунок):

$$next = left + \left\lfloor \frac{right - left}{list[right] - list[left]} (target - list[left]) \right\rfloor.$$

После сравнения *target* и $list[next]$ алгоритм интерполяционного поиска продолжает поиск среди элементов $left \ldots next - 1$ или $next + 1 \ldots right$ (или останавливается, если $list[next] = target$).

Число сравнений, необходимых для работы алгоритма в среднем случае, равно $A(N) = \Theta(\log_2 \log_2 N)$, однако в худшем случае $W(N) = O(N)$.

Задача

Задача 3. Поиск

1) В текстовом файле `input.txt` представлен массив из N целых чисел от 1 до N, расположенных в произвольном порядке без повторений. Реализуйте функцию поиска в этом массиве на основе алгоритма последовательного поиска. Головная программа должна вызывать функцию поиска для каждого элемента массива от 1 до N. В текстовый файл `output.txt` выведите среднее число сравнений, проведённых программой последовательного поиска.
2) Выполните то же для двоичного поиска в упорядоченном массиве.
3) Выполните то же для интерполяционного поиска в упорядоченном массиве.
Сравните вычислительную сложность рассмотренных в задаче алгоритмов.

Задача сортировки

Быстрые алгоритмы поиска требуют упорядоченного представления данных.

Задача сортировки состоит в перестановке (изменении порядка) входной последовательности, чтобы на выходе последовательность была упорядочена.

Вход: $(a_1, a_2, \ldots, a_n)$.

Выход: $(a'_1, a'_2, \ldots, a'_n)$, $\quad a'_1 \leqslant a'_2 \leqslant \ldots \leqslant a'_n$.

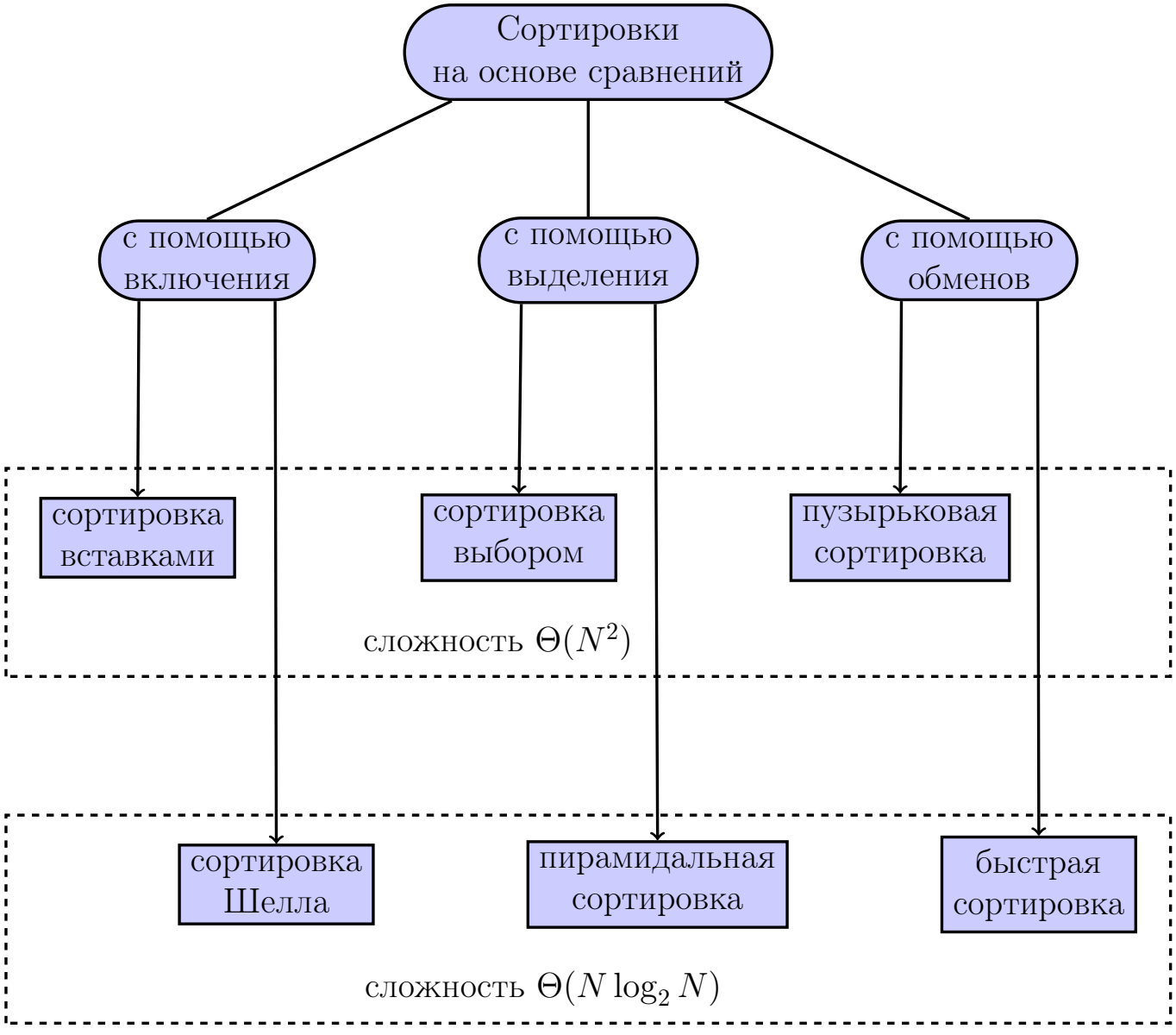
Различают внутренние и внешние алгоритмы сортировки.

Теорема о нижней границе

Теорема о нижней границе сложности сортировки гласит, что любая сортировка, основанная на использовании сравнений, имеет сложность в наихудшем случае $W(N) = \Omega(N \log_2 N)$.

Алгоритм сортировки называется **асимптотически оптимальным**, если для его сложности в наихудшем случае верна оценка $W(N) = \Theta(N \log_2 N)$.

Классификация сортировок



Алгоритмы сортировок, основанные на операциях сравнения

Сортировка вставками

```
InsertionSort (list, N)
// list  сортируемый массив
// N     число элементов в массиве

for i=2 to N do
   newElement=list[i]
   location=i-1
   while (location>=1) and (list[location]>newElement) do
      // сдвигаем все элементы, большие очередного
      list[location+1]=list[location]
      location=location-1
   end while
   list[location+1]=newElement
end for
```

Пример работы алгоритма сортировки вставками

Исходный массив:	6	2	4	7	1	3	8	5

Пример работы алгоритма сортировки вставками

Исходный массив: | **6** 2 4 7 1 3 8 5

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8
Проход 3 :	**2**	**4**	**6**	**7**	1	3	8	5

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8
Проход 3 :	**2**	**4**	**6**	**7**	1	3	8	5
Проход 4 :	**1**	**2**	**4**	**6**	**7**	3	8	5

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8
Проход 3 :	**2**	**4**	**6**	**7**	1	3	8	5
Проход 4 :	**1**	**2**	**4**	**6**	**7**	3	8	5
Проход 5 :	**1**	**2**	**3**	**4**	**6**	**7**	8	5

Пример работы алгоритма сортировки вставками

Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8
Проход 3 :	**2**	**4**	**6**	**7**	1	3	8	5
Проход 4 :	**1**	**2**	**4**	**6**	**7**	3	8	5
Проход 5 :	**1**	**2**	**3**	**4**	**6**	**7**	8	5
Проход 6 :	**1**	**2**	**3**	**4**	**6**	**7**	**8**	5

Пример работы алгоритма сортировки вставками

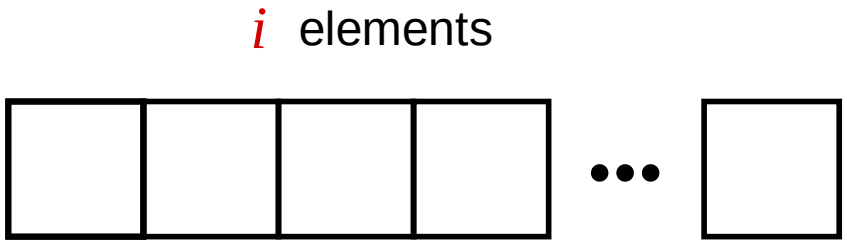
Исходный массив:	**6**	2	4	7	1	3	8	5
Проход 1 :	**2**	**6**	4	7	1	3	8	5
Проход 2 :	**2**	**4**	**6**	7	1	3	8	8
Проход 3 :	**2**	**4**	**6**	**7**	1	3	8	5
Проход 4 :	**1**	**2**	**4**	**6**	**7**	3	8	5
Проход 5 :	**1**	**2**	**3**	**4**	**6**	**7**	8	5
Проход 6 :	**1**	**2**	**3**	**4**	**6**	**7**	**8**	5
Проход 7 :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Анализ алгоритма сортировки вставками

- Наихудший случай.
  Пусть добавляемый элемент меньше всех остальных, тогда он всегда попадает в начало. Реализуется, например, когда элементы исходного массива расположены в убывающем порядке.

$$W(N) = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = O(N^2).$$

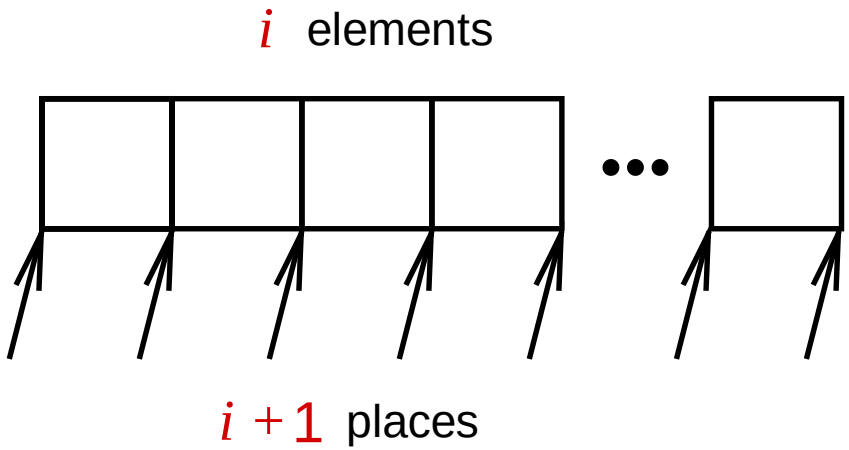
Анализ алгоритма сортировки вставками



- Средний случай. Необходимо просуммировать операции сравнения для вставки каждого i-го элемента:

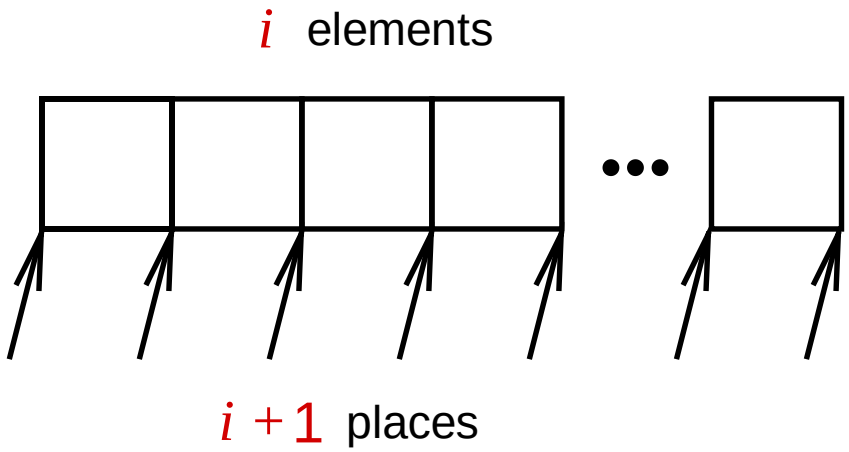
$$A(N) = \sum_{i=1}^{N-1} A_i.$$

Анализ алгоритма сортировки вставками



- Средний случай. Необходимо просуммировать операции сравнения для вставки каждого i-го элемента: $A(N) = \sum_{i=1}^{N-1} A_i$.

Анализ алгоритма сортировки вставками



- Средний случай.
  Среднее число сравнений для определения положения очередного элемента: $A_i = \frac{1}{i+1}\left(\sum_{i=1}^{N-1} p + i\right) = \frac{i}{2} + 1 - \frac{1}{i+1}$
  Суммируем A_i для каждого из $N-1$ элементов массива.
  $A(N) = \sum_{i=1}^{N-1} A_i = \sum_{i=1}^{N-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) =$
  $= \frac{1}{4}(N^2 + 3N - 4) - (\ln N - 1) = \frac{1}{4}N^2 + O(N).$

Задача

Задача 4. Сортировка вставками

В текстовом файле `input.txt` находится массив из N целых чисел. Выполните сортировку данных по возрастанию с помощью алгоритма сортировки вставками. В текстовый файл `output.txt` выведите отсортированный массив (в 1-ю строку файла) и число сравнений, проведённых программой (во 2-ю строку файла).

Пузырьковая сортировка

```
BubbleSort (list, N)
// list    сортируемый массив
// N       число элементов в массиве

numberOfPairs=N
swappedElements=true
while swappedElements do
   numberOfPairs=numberOfPairs-1
   swappedElements=false
   for i=1 to numberOfPairs do
      if list[i] > list[i+1] then
         Swap(list[i],list[i+1])
         swappedElements=true
      end if
   end for
end while
```

Пример работы алгоритма пузырьковой сортировки

Исходный массив: | 6 2 4 7 1 3 8 5

Пример работы алгоритма пузырьковой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Проход 1 :	2	4	6	1	3	7	5	**8**

Пример работы алгоритма пузырьковой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Проход 1 :	2	4	6	1	3	7	5	**8**
Проход 2 :	2	4	1	3	6	5	**7**	**8**

Пример работы алгоритма пузырьковой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Проход 1 :	2	4	6	1	3	7	5	**8**
Проход 2 :	2	4	1	3	6	5	**7**	**8**
Проход 3 :	2	1	**3**	**4**	**5**	**6**	**7**	**8**

Пример работы алгоритма пузырьковой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Проход 1 :	2	4	6	1	3	7	5	**8**
Проход 2 :	2	4	1	3	6	5	**7**	**8**
Проход 3 :	2	1	**3**	**4**	**5**	**6**	**7**	**8**
Проход 4 :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Пример работы алгоритма пузырьковой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Проход 1 :	2	4	6	1	3	7	5	**8**
Проход 2 :	2	4	1	3	6	5	**7**	**8**
Проход 3 :	2	1	**3**	**4**	**5**	**6**	**7**	**8**
Проход 4 :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Проход 5 :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Анализ алгоритма пузырьковой сортировки

- Наилучший случай.
  При первом проходе цикл `for` должен был выполнен полностью, $N - 1$ сравнение. Если перестановок не было, то массив отсортирован. Значит, $B(N) = N - 1$.

Анализ алгоритма пузырьковой сортировки

- Наилучший случай.
  При первом проходе цикл `for` должен был выполнен полностью, $N - 1$ сравнение. Если перестановок не было, то массив отсортирован. Значит, $B(N) = N - 1$.
- Наихудший случай.
  При первом проходе $N - 1$ сравнение. Если перестановки были, то цикл повторится с $N - 2$ сравнениями и т.д.

$$W(N) = \sum_{i=N-1}^{1} i = \frac{N^2 - N}{2} = O(N^2).$$

Анализ алгоритма пузырьковой сортировки

- Средний случай.
  Появление прохода без перестановок равновероятно в любой из $N-1$ проходов. Обозначим через $C(i)$ число сравнений на первых i проходах.

$$A(N) = \frac{1}{N-1} \sum_{i=1}^{N-1} C(i),$$

$$C(i) = \sum_{j=N-1}^{i} j = \sum_{j=i}^{N-1} j = \sum_{1}^{N-1} j - \sum_{1}^{i-1} j = \frac{N(N-1)}{2} - \frac{i(i-1)}{2},$$

$$A(N) = \frac{1}{N-1} \sum_{i=1}^{N-1} \frac{N^2 - N - i^2 + i}{2} = \frac{N^2}{3} + O(N).$$

Шейкерная сортировка

Можно внести следующие улучшения:

Шейкерная сортировка

Можно внести следующие улучшения:

1. Запоминать место последнего обмена элементов и при следующем проходе на заходить за это место.
2. Не только «лёгкие» элементы поднимать, но и параллельно «тяжёлые» опускать — совершать проходы в противоположных направлениях.

Шейкерная сортировка

Можно внести следующие улучшения:

1. Запоминать место последнего обмена элементов и при следующем проходе на заходить за это место.
2. Не только «лёгкие» элементы поднимать, но и параллельно «тяжёлые» опускать — совершать проходы в противоположных направлениях.

Реализация указанных улучшений приведёт к *шейкерной* сортировке. Как изменится сложность по сравнению с пузырьковой сортировкой?

Идея сортировки Шелла

В 1959 г. Д. Л. Шелл предложил усовершенствованный вариант сортировки вставками. Недостаток — эффективная работа только с небольшими массивами — превращён в достоинство. Весь массив длины N рассматривается как совокупность перемешанных подмассивов. Сортировка сводится к многократному применению сортировки вставками.

Разбивать массив на подмассивы можно по-разному!

Сортировка Шелла

```
Shellsort (list, N)
// list     сортируемый массив
// N        число элементов в массиве

passes = [log_2 N]
while (passes >= 1) do
   increment = 2^passes - 1
   for start = 1 to increment do
      InsertionSort (list,N,start,increment)
   end for
   passes = passes - 1
end while
```

Сортировка Шелла

```
Shellsort (list, N)
// list     сортируемый массив
// N        число элементов в массиве

passes = [log_2 N]
while (passes >= 1) do
   increment = 2^passes - 1
   for start = 1 to increment do
      InsertionSort (list,N,start,increment)
   end for
   passes = passes - 1
end while
```

Пример работы сортировки Шелла

Исходный массив:	6 2 4 7 1 3 8 5

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :								

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :	5	2	4	7	1	3	8	6

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :	5	2	4	7	1	3	8	6
После прохода со смещением 3 :								

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :	5	2	4	7	1	3	8	6
После прохода со смещением 3 :	5	1	3	7	2	4	8	6

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :	5	2	4	7	1	3	8	6
После прохода со смещением 3 :	5	1	3	7	2	4	8	6
После прохода со смещением 1 :								

Пример работы сортировки Шелла

Исходный массив:	6	2	4	7	1	3	8	5
После прохода со смещением 7 :	5	2	4	7	1	3	8	6
После прохода со смещением 3 :	5	1	3	7	2	4	8	6
После прохода со смещением 1 :	1	2	3	4	5	6	7	8

Анализ алгоритма сортировки Шелла

Анализ основывается на понятии *инверсии*.
Инверсия — пара элементов, расположенных в неправильном порядке. Напр., в массиве $(4, 3, 2, 1)$ всего 6 инверсий.
Выбор последовательности смещений влияет на сложность.
Для приведённого выбора значений смещений $\ldots, 15, 7, 3, 1$ сложность в наихудшем случае равна $O(N^{3/2})$.

Анализ алгоритма сортировки Шелла

Оценка сложности для заданной последовательности смещений представляет серьёзную математическую задачу. Например, количество сравнений в случае $h_s = 1, 2, 4, 8, 16, 32, \ldots$ и N, равного степени двойки, выражается формулой:

$$W(N) = \frac{N}{16} \sum_{i=1}^{\log_2 N} \frac{\Gamma(2^{i-1})}{2^i \Gamma(2^i)} \sum_{r=1}^{2^{i-1}} r(r+3)2^r \frac{\Gamma(2^i - r + 1)}{\Gamma(2^{i-1} - r + 1)} + \\ + N \log_2 N - \frac{3}{2}(N-1),$$

где $\Gamma(x)$ — гамма-функция.

Теорема об h- и k-упорядочении. Результат h-сортировки k-упорядоченного массива образует h- и k-упорядоченный массив.

Задача 5. Сортировка Шелла

В текстовом файле `input.txt` находится массив из N целых чисел. Выполните сортировку элементов массива по возрастанию с помощью сортировки Шелла для следующих значений смещений:

1) значения смещений равны h_s, где $h_{s+1} = 2h_s + 1$, $h_0 = 1$, причём $0 \leqslant s < \lfloor \log_2 N \rfloor$ (последовательность $1, 3, 7, 15, 31, \ldots$);
2) значения смещений равны h_s, где $h_{s+1} = 3h_s + 1$, $h_0 = 1$, причём $0 \leqslant s < \lfloor \log_3(2N + 1) \rfloor - 1$ (последовательность $1, 4, 13, 40, \ldots$).

В текстовый файл `output.txt` выведите отсортированный массив (в 1-ю строку файла) и число сравнений, проведённых двумя вариантами сортировки (во 2-ю строку файла).

Идея быстрой сортировки

Усовершенствование сортировки, основанной на обмене, даёт один из лучших методов. Основное достоинство отражено даже в названии — *быстрая сортировка* (Ч. Хоар, 1962 г.).
Выберем элемент в массиве (т. н. *осевой*) и разделим массив на две части, в первой окажутся элементы, меньшие выбранного, во второй — бóльшие. Затем алгоритм вызываем рекурсивно на полученных частях массива.

Быстрая сортировка

```
Quicksort (list, first, last)
// list    сортируемый массив
// first   номер первого элемента в сортируемой части
// last    номер последнего элемента в сортируемой части

if first<last then
   pivot=PivotList(list,first,last)
   Quicksort(list,first,pivot-1)
   Quicksort(list,pivot+1,last)
end if
```

Пример работы быстрой сортировки

Исходный массив: | 6 2 4 7 1 3 8 5

Пример работы быстрой сортировки

Исходный массив: | 6 2 4 7 1 3 8 5

Ось в ячейке 6:

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:								

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:								

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:								

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:	**1**	2	**3**	4	**5**	**6**	8	7

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:	**1**	2	**3**	4	**5**	**6**	8	7
Ось в ячейке 8:								

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:	**1**	2	**3**	4	**5**	**6**	8	7
Ось в ячейке 8:	**1**	2	**3**	4	**5**	**6**	7	**8**

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:	**1**	2	**3**	4	**5**	**6**	8	7
Ось в ячейке 8:	**1**	2	**3**	4	**5**	**6**	7	**8**
Окончательно:								

Пример работы быстрой сортировки

Исходный массив:	6	2	4	7	1	3	8	5
Ось в ячейке 6:	3	2	4	5	1	**6**	8	7
Ось в ячейке 3:	1	2	**3**	5	4	**6**	8	7
Ось в ячейке 1:	**1**	2	**3**	5	4	**6**	8	7
Ось в ячейке 5:	**1**	2	**3**	4	**5**	**6**	8	7
Ось в ячейке 8:	**1**	2	**3**	4	**5**	**6**	7	**8**
Окончательно:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Выбор осевого элемента

```
PivotList (list, first, last)

PivotValue = list[first]
lower = first
upper = last+1
do
   do upper = upper-1 until list[upper] <= PivotValue
   do lower = lower+1 until list[lower] >= PivotValue
   Swap(list[lower], list[upper])
until lower >= upper
// устранение лишнего обмена
Swap(list[lower], list[upper])
// перенос оси в нужное место
Swap(list[first], list[upper])
return upper
```

Анализ алгоритма быстрой сортировки

- Наихудший случай. PivotList либо больше всех элементов массива, либо меньше. Тогда разбиение происходит на максимально неравные части: из 1 и $N-1$ элементов.

1	$N-1$

При каждом рекурсивном вызове из рассмотрения удаляется один элемент:
$W(N) = (n+1) + n + \ldots + 3 = O(N^2)$.

Это выполняется для отсортированного в требуемом порядке массива.

Анализ алгоритма быстрой сортировки

- Средний случай. PivotList делает $N + 1$ сравнение, разбивая массив из N элементов (N сравнений для массива с повторяющимися значениями). Если функция вернёт значение P, то рекурсивный вызов Quicksort будет для массивов длины $P - 1$ и $N - P$. Соответствующее рекуррентное соотношение:

$$\begin{cases} A(N) = N + 1 + \frac{1}{N} \sum\limits_{i=1}^{N} \left(A(i-1) + A(N-i) \right), \quad N \geqslant 2, \\ A(0) = A(1) = 0. \end{cases}$$

Можно показать, что $A(N) = 2N \ln N + \Theta(N)$.

Замечания по реализации

Цикл по `lower` выйдет за пределы массива, если и только если опорный элемент больше всех остальных. Можно добавить «барьер» `list[last+1]`, больший всех допустимых значений ключей, который не позволит выйти за пределы массива.

Замечания по реализации

Цикл по `lower` выйдет за пределы массива, если и только если опорный элемент больше всех остальных. Можно добавить «барьер» `list[last+1]`, больший всех допустимых значений ключей, который не позволит выйти за пределы массива. Среди множества усовершенствований алгоритма отметим следующие:

1. разбиение на основе медианы трёх элементов `list[first]`, `list[last]`, `list[(first+last)/2]`
2. использование простой сортировки для малых подмассивов
3. нерекурсивный вариант

Генерация всех перестановок

Все перестановки элементов {1,...,n} можно получить, например, так [Новиков. Дискретная математика для программистов (2011)]:

```
begin
      for i = 1 to n do P[i] = i
      Antylex(n)
end
```

В результате получим последовательность перестановок множества {1,...,n} в антилексикографическом порядке.

Генерация всех перестановок. Продолжение

```
procedure Antylex(m)

if  m = 1 then print(P[1],..., P[n]) // новая перестановка
          else
            for i = 1 to m do
                  Antylex(m - 1)
                  if i < m then
                        Swap(P[i], P[m])
                        Reverse(m - 1)
                  end if
            end for
end if
```

Генерация всех перестановок. Окончание

Простая процедура обращения последовательности (P[1],..., P[n]).

```
procedure Reverse(m)
  i = 1
  j = m
  while i < j do
        Swap(P[i], P[j])
        i = i + 1
        j = j - 1
  end while
```

Задача 6. Быстрая сортировка

В текстовом файле `input.txt` записано натуральное число N, определяющее размер массива *list*, причём $N < 15$.
Сформируйте всевозможные варианты массива *list* и в текстовый файл `output.txt` выведите массив, на котором алгоритм быстрой сортировки выполняет максимальное количество сравнений.

Сравнение алгоритмов сортировки

Название сортировки	Автор и дата создания	Устойчивость	Естественное поведение	Асимптотическая сложность	Преимущества/ недостатки
1. Простыми вставками	Дж. фон Нейман, 1945 г.	Да	Да	$A(N) = N^2/4 + O(N)$, $W(N) = N^2/2 + O(N)$	Простота идеи и реализации. Неэффективна для массивов больших размеров
2. Пузырьковая	...	...	...	...	...

В таблицу войдут сортировки: пузырьковая, шейкерная, Шелла, быстрая, пирамидальная, сортировка слиянием и др.

Порядковые статистики

Поиск минимума, максимума или медианы нескольких чисел — частные случаи задачи о порядковых статистиках.

Определение

k-й порядковой статистикой непустого конечного множества $A = \{a_1, a_2, \ldots, a_N\}$ с введённым на нём антисимметричным и транзитивным отношением, все элементы которого попарно сравнимы, называется k-я компонента вектора $(a_{j_1}, \ldots, a_{j_k}, \ldots, a_{j_N})$, компоненты которого упорядочены $(1 \leqslant k \leqslant N)$.

Например, минимум чисел $a_1, a_2, \ldots, a_N$ представляет собой первую порядковую статистику этого множества.

Порядковые статистики

Пусть необходимо определить k-й по величине элемент массива $list[1..N]$, обозначим такой элемент через t.
Один из способов решения задачи о k-й порядковой статистике заключается в использовании процедуры Partition.

Предположим, что осевой элемент v занял в массиве $list$ окончательную позицию $list[i]$. Тогда число элементов массива $list$, по величине меньших или равных v, равно $i - 1$.
Аналогично, $N - i$ элементов имеют значение, не меньшее v.

Далее реализуется одна из трех возможностей:

Порядковые статистики

Пусть необходимо определить k-й по величине элемент массива $list[1..N]$, обозначим такой элемент через t.
Один из способов решения задачи о k-й порядковой статистике заключается в использовании процедуры Partition.

Предположим, что осевой элемент v занял в массиве $list$ окончательную позицию $list[i]$. Тогда число элементов массива $list$, по величине меньших или равных v, равно $i - 1$.
Аналогично, $N - i$ элементов имеют значение, не меньшее v.

Далее реализуется одна из трех возможностей:

1. $k < i$ и искомый элемент t расположен в подмассиве $list[1..i - 1]$

Порядковые статистики

Пусть необходимо определить k-й по величине элемент массива $list[1..N]$, обозначим такой элемент через t.
Один из способов решения задачи о k-й порядковой статистике заключается в использовании процедуры Partition.

Предположим, что осевой элемент v занял в массиве $list$ окончательную позицию $list[i]$. Тогда число элементов массива $list$, по величине меньших или равных v, равно $i - 1$.
Аналогично, $N - i$ элементов имеют значение, не меньшее v.

Далее реализуется одна из трех возможностей:

1. $k < i$ и искомый элемент t расположен в подмассиве $list[1..i - 1]$
2. $k = i$ и $t = list[i]$

Порядковые статистики

Пусть необходимо определить k-й по величине элемент массива $list[1..N]$, обозначим такой элемент через t.
Один из способов решения задачи о k-й порядковой статистике заключается в использовании процедуры Partition.

Предположим, что осевой элемент v занял в массиве $list$ окончательную позицию $list[i]$. Тогда число элементов массива $list$, по величине меньших или равных v, равно $i - 1$.
Аналогично, $N - i$ элементов имеют значение, не меньшее v.

Далее реализуется одна из трех возможностей:

1. $k < i$ и искомый элемент t расположен в подмассиве $list[1..i - 1]$
2. $k = i$ и $t = list[i]$
3. $k > i$ и t расположен в подмассиве $list[i + 1..N]$

Порядковые статистики

Функция SelectPart определяет индекс k-й порядковой статистики в исходном массиве с использованием процедуры Partition. Вспомогательная функция compare(k, v) осуществляет сравнение (k ? v).

```
function SelectPart(var list:Mass; k:integer):integer;
var found:boolean;
    left,right,v:integer;
begin
   left:=1;
   right:=N+1;
   list[N+1]:=MaxInt;
   found:=false; // изменение логической переменной
                 // found укажет на завершение работы
```

Порядковые статистики

```
    repeat
       v:=Partition(list,left,right);
       case compare(k,v) of
          -1: right:=v-1;
           0: begin
                 found:=true;
                 result:=v;
              end;
          +1: left:=v+1;
       end;
    until found;
end;
```

Асимптотический анализ алгоритма SelectPart в целом повторяет исследование сложности быстрой сортировки. Сложность алгоритма SelectPart в среднем случае $A(N) = \Theta(N)$, в наихудшем случае $W(N) = \Theta(N^2)$.

Порядковые статистики

Второй способ решения задачи о k-й порядковой статистике приводит к алгоритму SelectOpt сложности $O(N)$. Рассмотрим этот алгоритм.

Пусть требуется определить k-й по величине элемент массива $list[1..N]$, где $N > 1$. Для простоты предполагаем, что все элементы массива $list$ являются попарно различными целыми числами. Введем вспомогательные переменные $left$ и $right$ с начальными значениями $left = 1$, $right = N$. Представим работу алгоритма SelectOpt в виде следующих шагов.

Порядковые статистики

1. Элементы массива $list[left..right]$ разделим на $\left\lfloor \frac{N}{w} \right\rfloor$ подмассивов по w элементов в каждом, где w — целое число, большее единицы.

Порядковые статистики

1. Элементы массива $list[left..right]$ разделим на $\left\lfloor \frac{N}{w} \right\rfloor$ подмассивов по w элементов в каждом, где w — целое число, большее единицы.
2. Находим медианы m_i каждого из подмассивов, где $1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor$.

Порядковые статистики

1. Элементы массива $list[left..right]$ разделим на $\left\lfloor \frac{N}{w} \right\rfloor$ подмассивов по w элементов в каждом, где w — целое число, большее единицы.
2. Находим медианы m_i каждого из подмассивов, где $1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor$.
3. Определим медиану μ множества медиан $M = \left\{ m_i : 1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor \right\}$.

Порядковые статистики

1. Элементы массива $list[left..right]$ разделим на $\left\lfloor \frac{N}{w} \right\rfloor$ подмассивов по w элементов в каждом, где w — целое число, большее единицы.
2. Находим медианы m_i каждого из подмассивов, где $1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor$.
3. Определим медиану μ множества медиан $M = \left\{ m_i : 1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor \right\}$.
4. Величину μ используем в качестве осевого элемента для процедуры Partition, которая применяется на этом шаге к массиву $list[left..right]$.

Порядковые статистики

1. Элементы массива $list[left..right]$ разделим на $\left\lfloor \frac{N}{w} \right\rfloor$ подмассивов по w элементов в каждом, где w — целое число, большее единицы.
2. Находим медианы m_i каждого из подмассивов, где $1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor$.
3. Определим медиану μ множества медиан $M = \left\{ m_i : 1 \leqslant i \leqslant \left\lfloor \frac{N}{w} \right\rfloor \right\}$.
4. Величину μ используем в качестве осевого элемента для процедуры Partition, которая применяется на этом шаге к массиву $list[left..right]$.
5. Если искомый элемент t не найден, то сужаем область его поиска путем изменения значения переменных *left* или *right* и возвращаемся к первому шагу. Если элемент t найден — алгоритм завершает работу.

Порядковые статистики

Отметим особенности алгоритма SelectOpt.

Во-первых, разбиение анализируемого массива $list[left..right]$ на подмассивы приведёт к тому, что ровно $d - w\left\lfloor \frac{d}{w} \right\rfloor$ элементов, где $d = right - left + 1$ — размер массива, не попадут ни в один из подмассивов. Во время выполнения шагов 2–3 данные элементы не используются, однако в процедуре разбиения Partition на четвертом шаге их необходимо учитывать.

Во-вторых, медианы m_i на втором шаге можно определить путем сортировки каждого из подмассивов, после чего медианы будут расположены на местах с индексом $\left\lceil \frac{w}{2} \right\rceil$.

В-третьих, еще одна особенность связана с определением величины μ. Элементы множества медиан M можно сохранять во вспомогательном массиве, но это потребует выделения дополнительной памяти. Альтернативный вариант заключается в переносе элементов m_i в начало массива $list$.

Порядковые статистики

С учётом указанных особенностей представим одну из возможных реализаций алгоритма SelectOpt. При этом величина w является параметром алгоритма и задается в виде глобальной константы.

```
const w=5;
function SelectOpt(var list:Mass;
                       k,left,right:integer):integer;
var i,d,dd,v,temp:integer;
begin
   while (true) do
      begin
         d:=right-left+1;
         if (d<=w) then
            begin
               InsertionSort(list,left,right);
               result:=left+k-1;
               break;
            end;
```

Порядковые статистики

```
dd:=Floor(d/w);  // количество подмассивов
for i:=1 to dd do
   begin
      InsertionSort(list,
                    left+(i-1)*w,left+i*w-1);
      swap(list[left+i-1],
           list[left+(i-1)*w+Ceil(w/2)-1]);
   end;
v:=SelectOpt(list,Ceil(dd/2),
               left,left+dd-1);
swap(list[left],list[v]);
v:=Partition(list,left,right);
temp:=v-left+1;
```

Порядковые статистики

```
            case compare(k,temp) of
               -1: right:=v-1;
                0: begin
                      result:=v;
                      break;
                   end;
               +1: begin
                      k:=k-temp;
                      left:=v+1;
                   end;
            end;
         end;
end;
```

Можно показать, что при $w \geqslant 5$ алгоритм SelectOpt имеет сложность, линейную по количеству элементов исходного массива.

Задача

Задача 7. Порядковые статистики

1) В первой строке текстового файла `input.txt` записано натуральное число k. Во второй строке данного текстового файла записан целочисленный массив *list*. С помощью алгоритма SelectPart определите k-ю порядковую статистику массива *list* и выведите её в текстовый файл `output.txt`.
2) Выполните то же с помощью алгоритма SelectOpt.
Сравните вычислительную сложность рассмотренных в задаче алгоритмов.

Основные определения

Основное определение

Графом $G(V, E)$ называется упорядоченная пара двух множеств — непустого множества V (множества вершин) и множества E неупорядоченных пар различных элементов множества V (E — множество рёбер).

- Пусть v_1, v_2 — вершины, $e = (v_1, v_2)$ — соединяющее их ребро. Тогда вершина v_1 и ребро e *инцидентны*, вершина v_2 и ребро e также *инцидентны*.
- Два ребра, инцидентные одной вершине, называются *смежными*; две вершины, инцидентные одному ребру, также называются *смежными*.

Основные определения. Продолжение

- *Ориентированный граф (орграф)*. Элементы множества V — узлы, элементы множества E — дуги.
- *Псевдограф* (граф с петлями).
- *Мультиграф* (граф с кратными рёбрами).
- Если задана функция $F\colon V \to M$ и/или $F\colon E \to M$, то множество M называется множеством пометок, а граф называется *помеченным* (или *нагруженным*).

Маршруты

- *Маршрутом* в графе называется чередующаяся последовательность вершин и рёбер $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, в которой любые два соседних элемента инцидентны. Для простого графа достаточно указать только последовательность вершин или рёбер.
- Если все рёбра различны, то маршрут называется *цепью*. Если все вершины (а значит, и рёбра) различны, то маршрут называется *простой цепью*.
- Замкнутая цепь называется *циклом*; замкнутая простая цепь называется *простым циклом*. Граф без циклов называется *ациклическим*.
- Две вершины в графе *связаны*, если существует соединяющая их (простая) цепь. Граф, в котором все вершины связаны, называется *связным*.

Специальные виды графов

Наиболее важные типы графов:

- в *полном* графе K_N любая пара вершин соединена ребром. Число рёбер в таком графе $|E| = \frac{1}{2}N(N-1)$
- *эйлеровы*
- *гамильтоновы*
- *планарные*
- *регулярные*

Представление графов

1. **Матрица смежности**. Матрица смежности графа $G(V, E)$ с числом вершин $|V| = N$ записывается в виде двумерного массива размером $N \times N$:

$$AdjMat[i, j] = \begin{cases} 1, & \text{если } v_i v_j \in E; \\ 0, & \text{если } v_i v_j \notin E. \end{cases}$$

Необходимый объем памяти составляет $O(|V|^2)$.

Представление графов

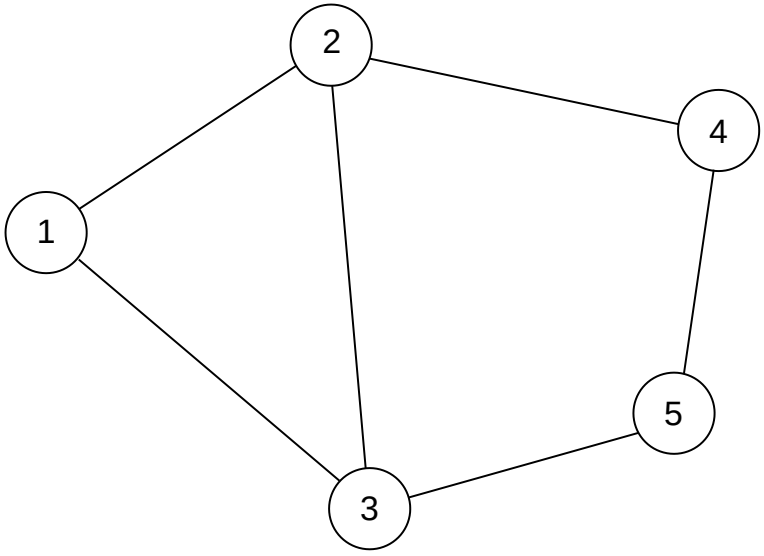
1. **Матрица смежности**. Матрица смежности графа $G(V, E)$ с числом вершин $|V| = N$ записывается в виде двумерного массива размером $N \times N$:

$$AdjMat[i, j] = \begin{cases} 1, & \text{если } v_i v_j \in E; \\ 0, & \text{если } v_i v_j \notin E. \end{cases}$$

   Необходимый объем памяти составляет $O(|V|^2)$.

2. **Список смежности**. Список смежности графа $G(V, E)$ записывается в виде одномерного массива длины N, каждый элемент которого представляет собой ссылку на список, в котором содержатся смежные вершины. Необходимый объем памяти в случае неориентированных графов $O(|V| + 2|E|)$, в случае ориентированных графов $O(|V| + |E|)$.

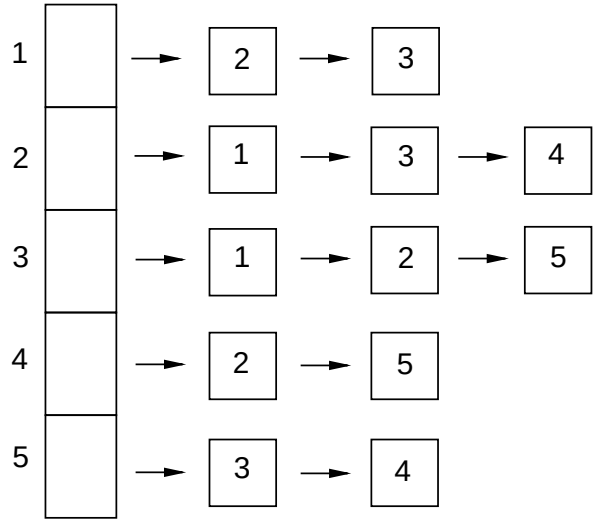
Пример графа



$$G = (\{1,2,3,4,5\}, \{\{1,2\},\{1,3\},\{2,4\},\{3,5\},\{2,3\},\{4,5\}\})$$

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Матрица смежности



Список смежности

Обходы графов

Обход графа

Обход графа — это систематическое перечисление его вершин (и/или рёбер).

- Обход в глубину.
- Обход по уровням (обход в ширину).

Если граф G является связным (и конечным), то поиск в ширину и поиск в глубину обойдут все вершины по одному разу.

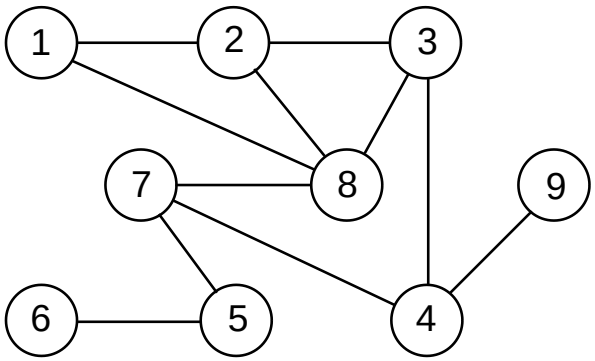
Обход в глубину

```
DepthFirstTraversal(G, v)
// G  граф
// v  текущий узел

Visit(v)
Mark(v)
for каждого ребра vw графа G do
   if вершина w не помечена then
      DepthFirstTraversal(G, w)
   end if
end for
```

При обходе в глубину мы посещаем первый узел, а затем идём вдоль рёбер графа, пока не упрёмся в тупик. Для отслеживания текущей вершины графа используется системный стек.

Пример работы алгоритма обхода в глубину

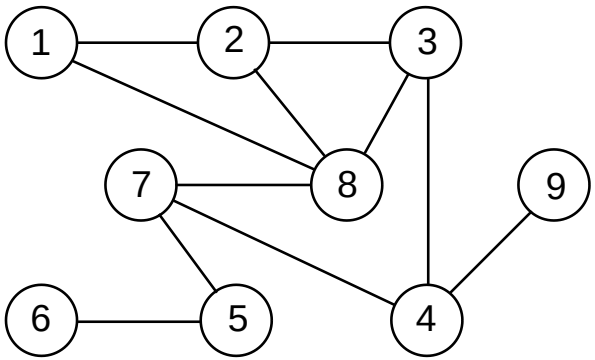


Порядок посещения вершин:
1, 2, 3, 4, 7, 5, 6, нет подх., 5, нет подх., 7, 8, нет подх., 7, нет подх., 4, 9, нет подх., 4, нет подх., 3, нет подх., 2, нет подх., 1
Все вершины посещены.

Обход по уровням

```
BreadthFirstTraversal(G, v)
// G  граф
// v  текущий узел
Visit(v)
Mark(v)
Enqueue(v)
while очередь непуста do
   Dequeue(x)
   for каждого ребра xw в графе G do
      if вершина w не помечена then
         Visit(w)
         Mark(w)
         Enqueue(w)
      end if
   end for
end while
```

Пример работы алгоритма обхода по уровням



Порядок посещения вершин:

$$1;\ \underbrace{2, 8}_{\text{1й проход}}\ ;\ \underbrace{3, 7}_{\text{2й проход}}\ ;\ \underbrace{4, 5}_{\text{3й проход}}\ ;\ \underbrace{6, 9}_{\text{4й проход}}\ .$$

Все вершины посещены.

Анализ алгоритмов обхода

Алгоритмы обхода гарантируют посещение всех узлов графа. Время работы алгоритмов обхода пропорционально размеру используемой для представления графа структуры данных. Асимптотическая сложность алгоритма обхода равна $O(|V|^2)$ для представления с использованием матрицы смежности и $O(|V| + |E|)$ для представления с использованием списков смежности.

Задача

Задача 8. Обход графа

Граф G задан матрицей смежности, записанной в текстовом файле `input.txt`. Выполните обход графа G в глубину и по уровням, начиная с первой вершины. В текстовый файл `output.txt` выведите количество посещений вершин для двух вариантов обхода.

Поиск минимального остовного дерева

МОД

Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его рёбер, причём сумма весов рёбер минимально возможная.

Поиск минимального остовного дерева

МОД

Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его рёбер, причём сумма весов рёбер минимально возможная.

- *Алгоритм Дейкстры–Прима*. «Жадный» алгоритм. На каждом шаге рассматриваем множество рёбер, допускающих присоединение к уже построенной части остовного дерева, и выбирать из них ребро с наименьшим весом.

Поиск минимального остовного дерева

МОД

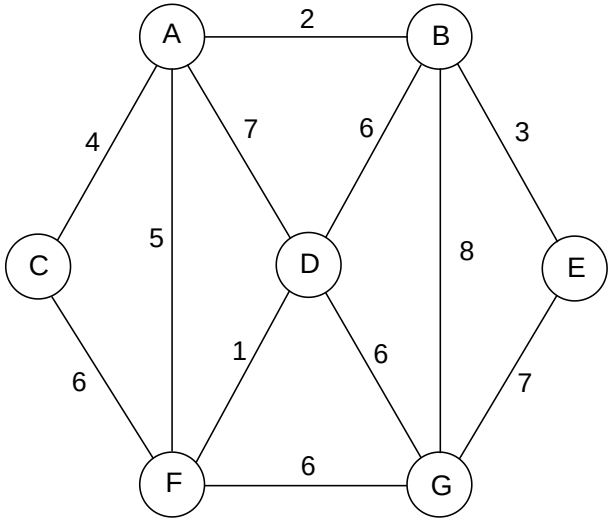
Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его рёбер, причём сумма весов рёбер минимально возможная.

- *Алгоритм Дейкстры–Прима*. «Жадный» алгоритм. На каждом шаге рассматриваем множество рёбер, допускающих присоединение к уже построенной части остовного дерева, и выбирать из них ребро с наименьшим весом.
- *Алгоритм Краскала*. Начинаем с пустого дерева и добавляем к нему рёбра в порядке возрастания их весов пока не получим набор рёбер, объединяющий все вершины графа.

Алгоритм Дейкстры–Прима

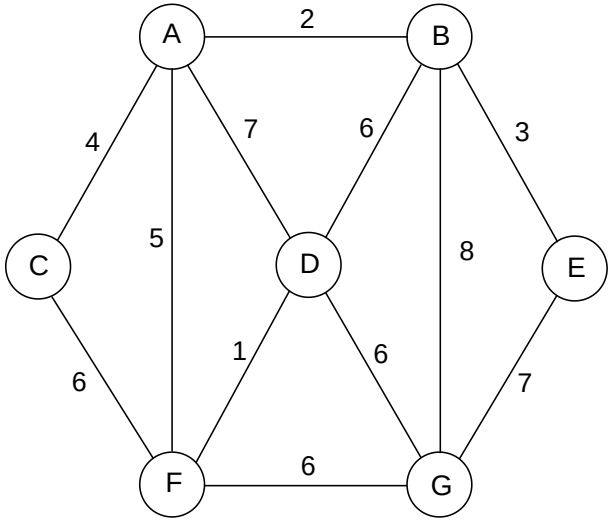
```
выбрать начальный узел
сформировать начальную кайму, состоящую из вершин,
                              соседних с начальным узлом
while  в графе есть вершины, не попавшие в дерево  do
   выбрать ребро из дерева в кайму с наименьшим весом
   добавить конец ребра к дереву
   изменить кайму, для чего
      добавить в кайму вершины, соседние с новой
      обновить список рёбер из дерева в кайму так,
         чтобы он состоял из рёбер наименьшего веса
end while
```

Пример работы алгоритма Дейкстры–Прима

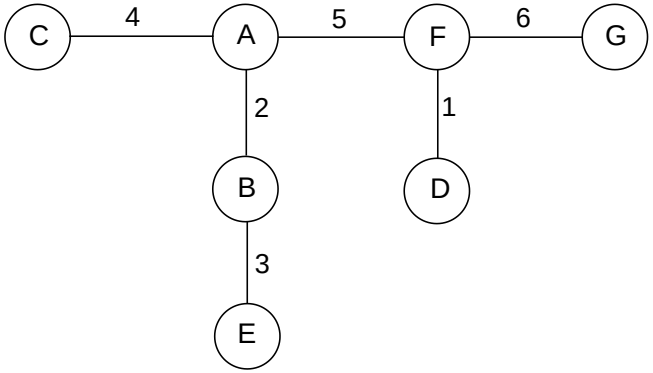


Начальный узел — A.

Пример работы алгоритма Дейкстры–Прима



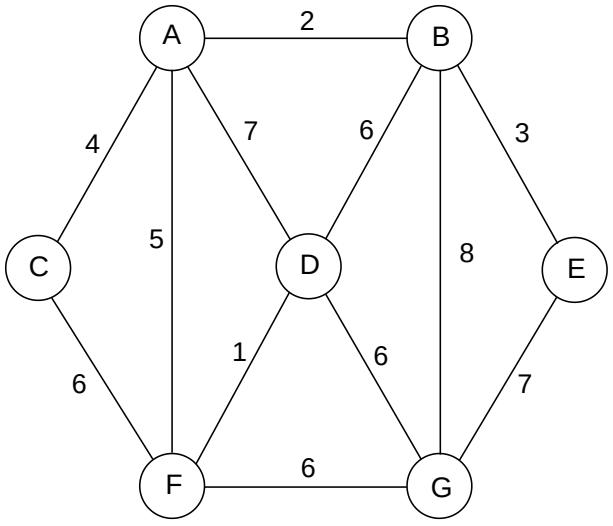
Начальный узел — A.



Алгоритм Краскала

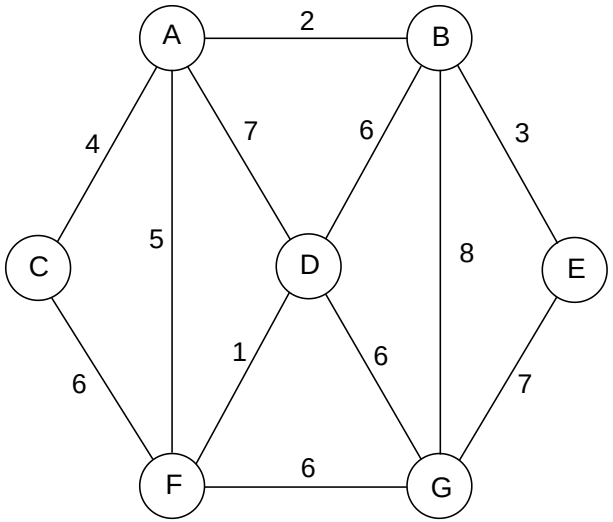
```
отсортировать рёбра в порядке возрастания весов
инициализировать структуру разбиений
edgeCount = 1
while edgeCount <= E and includeCount <= N - 1 do
   parent1 = FindRoot(edge[edgeCount].start)
   parent2 = FindRoot(edge[edgeCount].end)
   if parent1 /= parent2 then
      добавить edge[edgeCount] в остовное дерево
      includedCount = includedCount + 1
      Union(parent1, parent2)
   end if
    edgeCount = edgeCount + 1
end while
```

Пример работы алгоритма Краскала

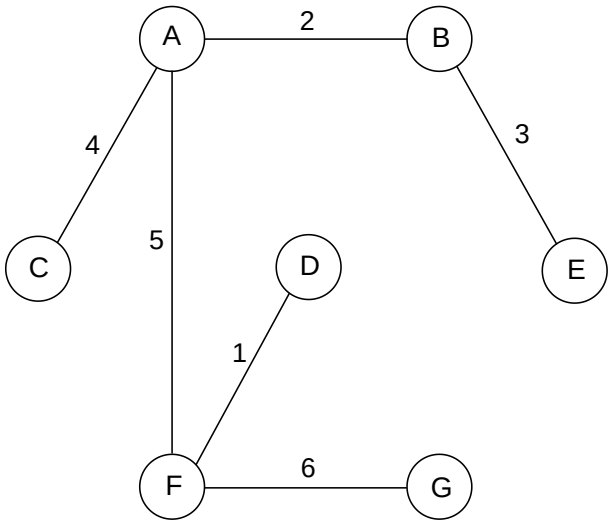


Начальное ребро — DF.

Пример работы алгоритма Краскала



Начальное ребро — DF.



Поиск кратчайшего маршрута

Задача состоит в поиске последовательности рёбер, соединяющей заданные 2 вершины и имеющей минимальную длину.
«Жадный» алгоритм построения МОД не пригоден!

Поиск кратчайшего маршрута

Задача состоит в поиске последовательности рёбер, соединяющей заданные 2 вершины и имеющей минимальную длину.
«Жадный» алгоритм построения МОД не пригоден!

- Алгоритм Флойда–Уоршелла. Находит кратчайшие маршруты между всеми парами вершин в орграфе.

Поиск кратчайшего маршрута

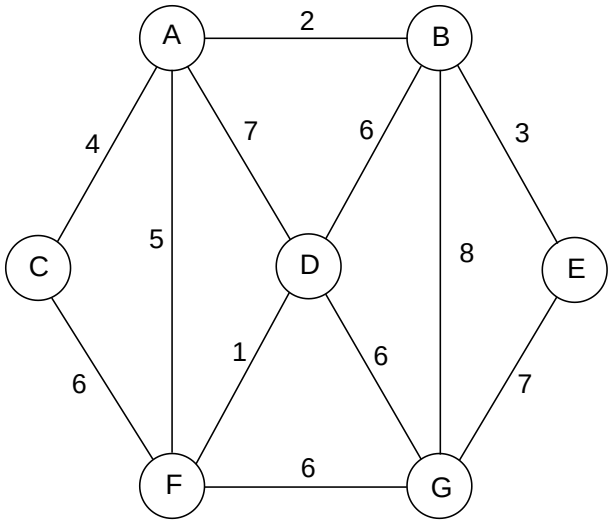
Задача состоит в поиске последовательности рёбер, соединяющей заданные 2 вершины и имеющей минимальную длину.
«Жадный» алгоритм построения МОД не пригоден!

- Алгоритм Флойда–Уоршелла. Находит кратчайшие маршруты между всеми парами вершин в орграфе.
- Алгоритм Дейкстры. Находит кратчайший маршрут между двумя вершинами орграфа. Для работы достаточно выполнения *неравенства треугольника*:
  $\forall v_1, v_2, v_3 \in V \; d(v_1, v_2) \leqslant d(v_1, v_3) + d(v_3, v_2)$.

Алгоритм Дейкстры

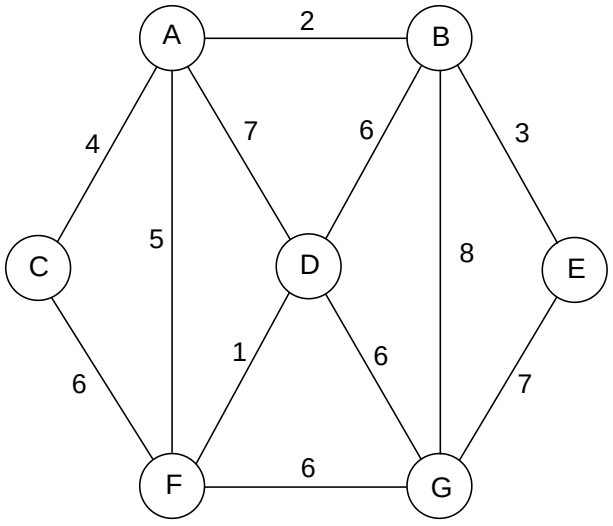
```
выбрать начальную вершину
создать начальную кайму из вершин, соединённых с начальной
while вершина назначения не достигнута do
   выбрать вершину каймы с кратчайшим расстоянием до начальной
   добавить эту вершину и ведущее в неё ребро к дереву
   изменить кайму путём добавления к ней вершин,
                соединённых с вновь добавленной:
      for  всякой вершины каймы  do
         приписать к ней ребро, соединяющее её с деревом и
             завершающее кратчайший маршрут к начальной вершин
   end for
end while
```

Пример работы алгоритма Дейкстры



Результат для вершин A и G:

Пример работы алгоритма Дейкстры



Результат для вершин A и G: