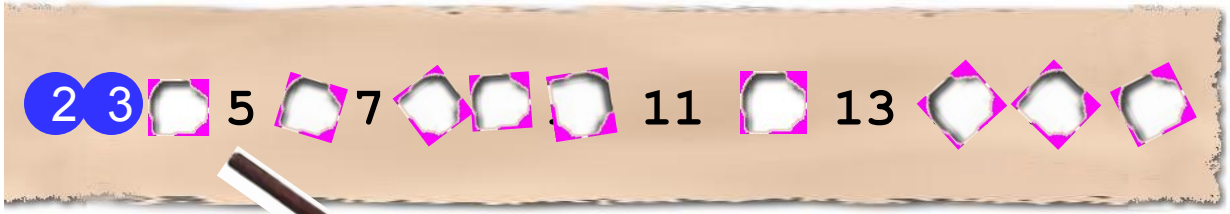
Алгоритмизация и программирование. Язык C++

Алгоритмизация и программирование. Язык C++

§ 38. Целочисленные алгоритмы

Решето Эратосфена



Эратосфен Киренский
(Eratosthenes, Ερατοσθδνη)
(ок. 275-194 до н.э.)

Алгоритм:

1) начать с **k = 2**
2) «выколоть» все числа через **k**, начиная с **k·k**
3) перейти к следующему «невыколотому» **k**
4) если **k·k <= N**, то перейти к шагу 2
5) напечатать все числа, оставшиеся «невыколотыми»

Новая версия – решето Аткина.

? Как улучшить?

⊕ высокая скорость, количество операций

$$O((N \cdot \log N) \cdot \log \log N)$$

⊖ нужно хранить в памяти все числа от **1** до **N**

Решето Эратосфена

Задача. Вывести все простые числа от 2 до N.

Объявление переменных:

```
const int N=100;
bool A[N+1];
int i, k;
```

выделяем на 1 элемент больше, чтобы начать с `A[1]`

Сначала все невычеркнуты:

```
for ( i=2; i<=N; i++ )
  A[i]=true;
```

Решето Эратосфена

Вычёркивание непростых:

```
k = 2;
while ( k*k <= N ) {
   if ( A[k] ) {
      i = k*k;
      while ( i <= N )
         {
         A[i] = false;
         i += k;
         }
      }
   k ++;
   }
```

Решето Эратосфена

Вывод результата:

```
for ( i = 2; i <= N; i++ )
  if ( A[i] )
    cout << i << " ";
```

«Длинные» числа

Ключи для шифрования: **≥ 256** битов.

Целочисленные типы данных: ≤ 64 битов.

Как хранить?

Длинное число – это число, которое не помещается в переменную одного из стандартных типов данных языка программирования.

«**Длинная арифметика**» – алгоритмы для работы с длинными числами.

«Длинные» числа

A = 12345678

	0	1	2	3	4	5	6	7	8	9
A	1	2	3	4	5	6	7	8	0	0

? Что плохо?

- нужно хранить длину числа
- неудобно вычислять (с младшего разряда!)
- неэкономное расходование памяти

Обратный порядок элементов:

	9	8	7	6	5	4	3	2	1	0
A	0	0	1	2	3	4	5	6	7	8

«Длинные» числа

Упаковка элементов: `A = 12345678`

	9	8	7	6	5	4	3	2	1	0
A	0	0	0	0	0	0	0	12	345	678

$12345678 = 12 \cdot 1000^2 + 345 \cdot 1000^1 + 678 \cdot 1000^0$

? На что похоже?

система счисления с основанием 1000!

`long int`:

от $-2^{31} = -2\,147\,483\,648$ до $2^{31} - 1 = 2\,147\,483\,647$.

? Какие основания можно использовать?

должны помещаться все промежуточные результаты!

Вычисление факториала

Задача 1. Вычислить точно значение факториала

100! = 1·2·3·…·99·100

? Как оценить количество цифр?

1·2·3·…·99·100 <

201 цифра

основание 1000000

6 цифр в ячейке ⇒ 34 ячейки

```
const int N=33;
long int A[N+1];
```

Основной алгоритм:

длинное число

```
[A]=1;
for ( k=2; k<=100; k++ )
   [A]=[A]*k;
```

Вычисление факториала

основание d = 1 000 000

`[A] = 1234567890173456 7`

	3	2	1	0
A	0	12345	678901	734567

*3

734 567 ·3 = 2 203 701

остаётся в A[0]

r = перенос в A[1]

? Как найти перенос?

```
s = A[0]*k;
A[0] = s % d;
r = s / d;
```

? Что изменится для A[1]?

```
s = A[1]*k + r;
```

Вычисление факториала

Умножение «длинного» числа на k:

```
r = 0;
for ( i = 0; i <= N; i++ ) {
   s = A[i] * k + r;
   A[i] = s % d;
   r = s / d;
   }
```

все разряды

Вычисление 100!:

```
for ( k = 2; k <= 100; k++ )
   {
   ...
   }
```

Вывод длинного числа

	3	2	1	0
A	0	1	2	3

[A] = 1000002000003

Какое число?

- найти старший ненулевой разряд

```
i=N;
while( !A[i] )
  i--;
```

- вывести этот разряд

```
cout << A[i];
```

- вывести все следующие разряды, добавляя лидирующие нули до 6 цифр

Вывод длинного числа

Вывод остальных разрядов:

```
for ( k=i-1; k>=0; k-- )
  Write6 ( A[k] );
```

со старшего

Write6:

x = 12345

x / 100000

012345

x % 100000

x	M	x / M
12345	100000	0

Вывод длинного числа

Вывод числа с лидирующими нулями:

```
void Write6 ( long int x )
{
  long int M=100000;
  while ( M>0 )
    {
    cout << x / M;
    x %= M;
    M /= 10;
    }
}
```

Алгоритмизация и программирование. Язык C++

§ 39. Структуры

Зачем нужны структуры?

Книги в библиотеках:

- **автор**
- **название**
- **количество экземпляров**
- **...**

символьные строки

целое число

Как хранить данные?

Несколько массивов:

```
string authors[N];
string titles[N];
int  count[N];
...
```

неудобно работать (сортировать и т.д.), ошибки

Задача: объединить разнотипные данные в один блок.

Структуры

Структура – это тип данных, который может включать в себя несколько **полей** – элементов разных типов (в том числе и другие структуры).

новый тип данных

```
typedef struct
    {
    string author;   // автор, строка
    string title;    // название, строка
    int count;       // количество, целое
    } TBook;
```

структура

название типа данных

Объявление структур

```
const int N=100;
TBook B;
TBook Books[N];
```

? Сколько места занимает в памяти?

```
cout << sizeof(TBook) << endl; // 12
cout << sizeof(B) << endl;     // 12
cout << sizeof(Books) << endl; // 1200
```

```
typedef struct
    {
    string author;
    string title;
    int count;
    } TBook;
```

4 байта

4 байта

4 байта

! Это указатели!

Обращение к полям структур

Точечная нотация:

```
B.author // поле author структуры B
```

```
Books[5].count // поле count структуры
               // Books[5]
```

```
cout << sizeof(B.author) << endl; // 4
cout << sizeof(B.title) << endl;  // 4
cout << sizeof(B.count) << endl;  // 4
```

```
cin >> B.author;
cin >> B.title;
cin >> B.count;
```

```
cout << B.author << " " << B.title << ". "
     << B.count << " шт.";
```

Обращение к полям структур

Присваивание:

```
B.author = "Пушкин А.С.";
B.title = "Полтава";
B.count = 1;
```

Использование:

```
B.count--;              // одну книгу взяли
if( B.count == 0 )
    cout << "Этих книг больше нет!";
```

Запись структур в файлы

Текстовые файлы:

символ-разделитель

```
'Пушкин А.С.';'Полтава';12
'Лермонтов М.Ю.';'Мцыри';8
```

! Сложно читать, ошибки!

Двоичные файлы:

поток вывода

двоичный

```
ofstream Fout;
Fout.open ( "books.dat", ios::binary );
Fout.write ( (char*) &B, sizeof(TBook) );
Fout.write ( (char*) Books
                        eof(T
Fout.close ();
```

адрес в памяти

сколько байтов

Чтение структур из файла

Одна структура:

```
ifstream *Fin;
Fin.open ( "books.dat", ios::binary );
Fin.read ( (char*) &B, sizeof(B) );
cout << B.a                 << " "          itle
     << ".                  count           r.";
Fin.close ();
```

адрес в памяти

сколько байтов

Сразу несколько структур:

```
Fout.read ( (char*) Books,
            5*sizeof(TBook) );
```

Чтение структур из файла

Число структур неизвестно:

```
const int N=100;
int M;
...
Fin.read ( (char*) Books,
             N*sizeof(TBook) );
M = Fin.gcount() / sizeof(TBook);
cout << "Прочитано " << M << " структур.";
```

`gcount` возвращает число успешно прочитанных **байтов**!

Сортировка структур

Ключ – фамилия автора:

Какой метод?

```
for ( i=0; i<N-1; i++ )
  for ( j=N-2; j>=i; j-- )
    if ( Books[j].author >
         Books[j+1].author) )
      {
      B = Books[j];
      Books[j] = Books[j+1];
      Books[j+1] = B;
      }
```

```
TBook B;
```

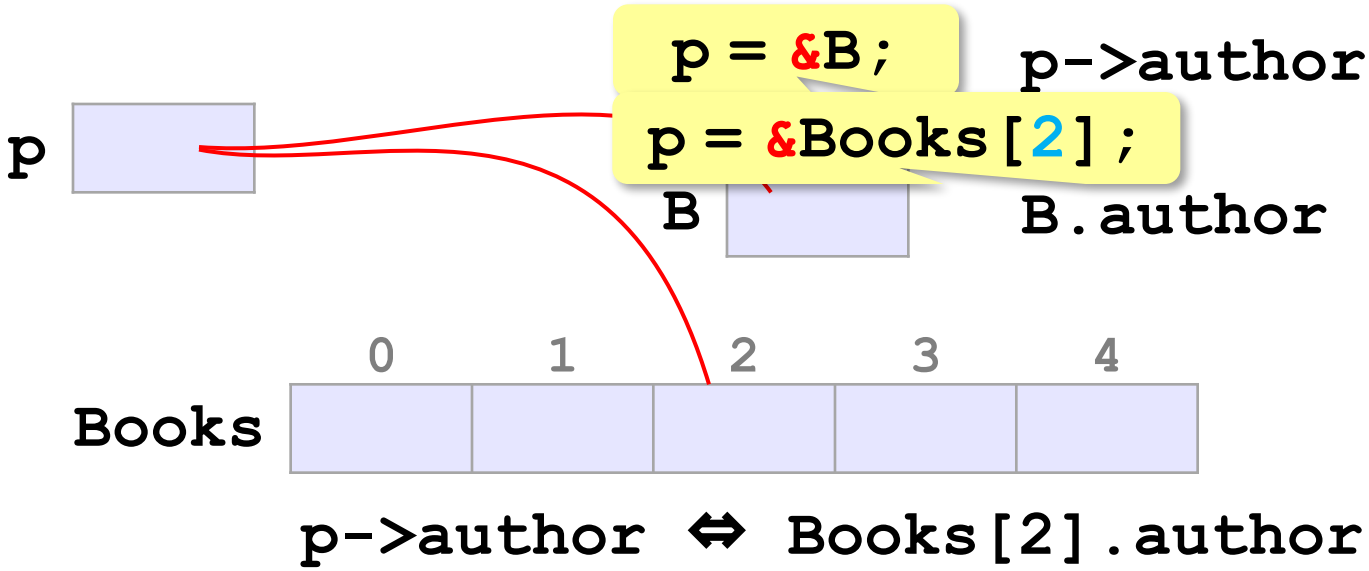
структуры перемещаются в памяти

Указатели

Указатель – это переменная, в которой можно сохранить адрес любой переменной заданного типа.

```
TBook *p;
```

указатель на переменную типа **TBook**

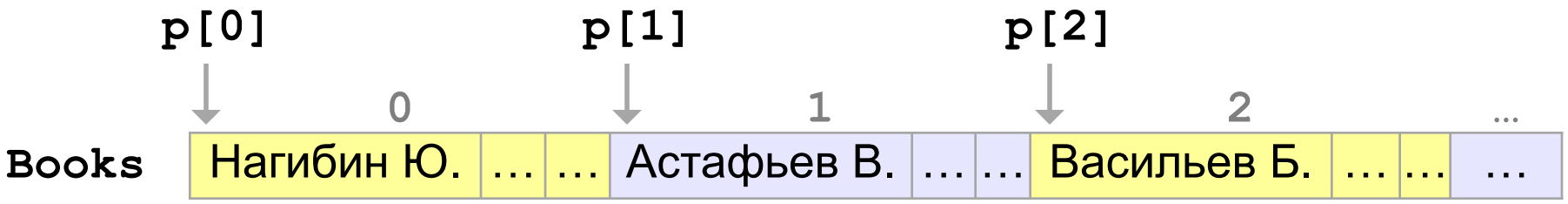



```
p->author ⇔ Books[2].author
```

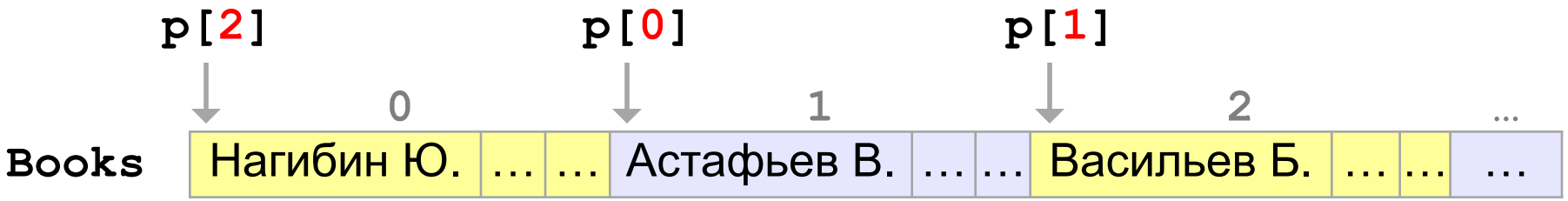
Сортировка по указателям

```
TBook *p[N], *p1;
```

```
for ( i=0; i<N; i++ )
  p[i] = &Books[i];
```



Задача – переставить указатели:



Сами структуры не перемещаются!

Сортировка по указателям

```
for ( i=0; i<M-1; i++ )
  for ( j=M-2; j>=i; j-- )
    if ( p[j]->author > p[j+1]->author )
      {
      p1=p[j]; p[j]=p[j+1];
      p[j+1]=p1;
      }
```

обращение к полям через указатели

переставляем **указатели**!

```
TBook *p1;
```

Вывод результата:

```
for ( i=0; i<M; i++ )
  cout << p[i]->author << " " << p[i]->title
       << ". " << p[i]->count
       << " шт." << endl;
```

Алгоритмизация и программирование. Язык C++

§ 40. Динамические массивы

Чем плох обычный массив?

```
const int N = 100;
int A[N];
```

статический массив

- память выделяется при трансляции
- нужно заранее знать размер
- изменить размер нельзя

Задача. В файле записаны фамилии (сколько – неизвестно!). Вывести их в другой файл в алфавитном порядке.

- выделить заранее большой блок (с запасом)
- выделять память во время работы программы (**динамически**!)

Динамические структуры данных

… **позволяют**

- создавать новые объекты в памяти
- изменять их размер
- удалять из памяти, когда не нужны

Задача. Ввести с клавиатуры целое значение N, затем – N целых чисел, и вывести на экран эти числа в порядке возрастания.

```
// прочитать данные из файла в массив
// отсортировать их по возрастанию
// вывести массив на экран
```

В чём проблема?

Динамические массивы

Объявление:

```
int *A;
```

Память не выделяется!

Выделение памяти:

```
A=new int[N];
```

количество элементов

Динамические массивы

Использование массива:

```
for ( i=0; i<N; i++ )
   cin >> A[i];
...
for ( i=0; i<N; i++ )
   {
   A[i]=i;
   cout << A[i] << " ";
   }
```

Освобождение памяти:

```
delete [] A;
```

удаление массива

Тип vector (библиотека STL)

STL = *Standard Template Library*

Вектор – это массив переменного размера!

Заголовочный файл:

```
#include <vector>
```

Объявление:

```
vector <int> A;
```

пустой массив типа `int`

Размер:

```
cout << A.size();
```

Заполнение (добавление в конец):

```
for ( i = 0; i < N; i++ )
  A.push_back ( i + 1 );
```

Тип vector (библиотека STL)

Обработка :

```
for ( i = 0; i < A.size(); i++ )
   cout << A[i] << " ";
```

Так же, как с обычным массивом!

Динамические матрицы

Матрица – это массив из массивов!

Указатель на матрицу:

```
typedef int *pInt;
pInt *A;
```

новый тип данных: указатель

указатель на указатель

Выделение памяти под массив указателей:

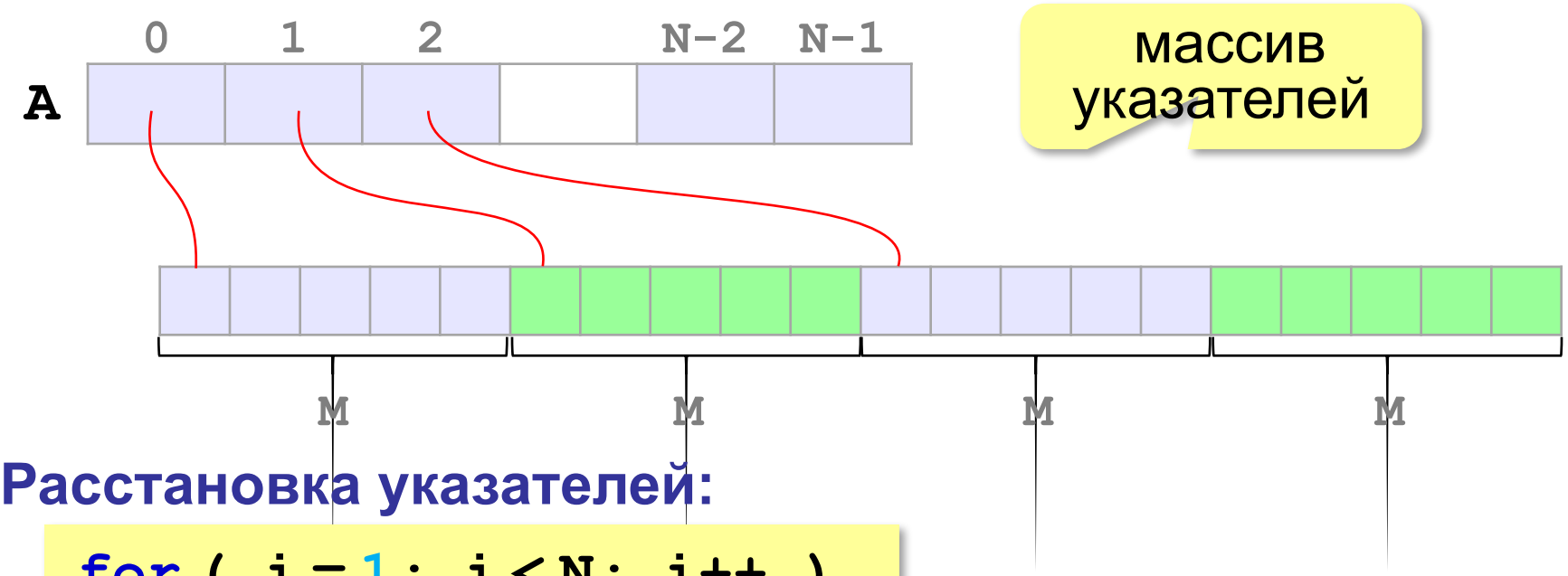
```
A = new pInt[N];
```

Выделение памяти под элементы матрицы:

```
A[0] = new int[M*N];
```

число элементов матрицы

Динамические матрицы



Расстановка указателей:

```
for ( i=1; i<N; i++ )
  A[i]=A[i-1]+M;
```

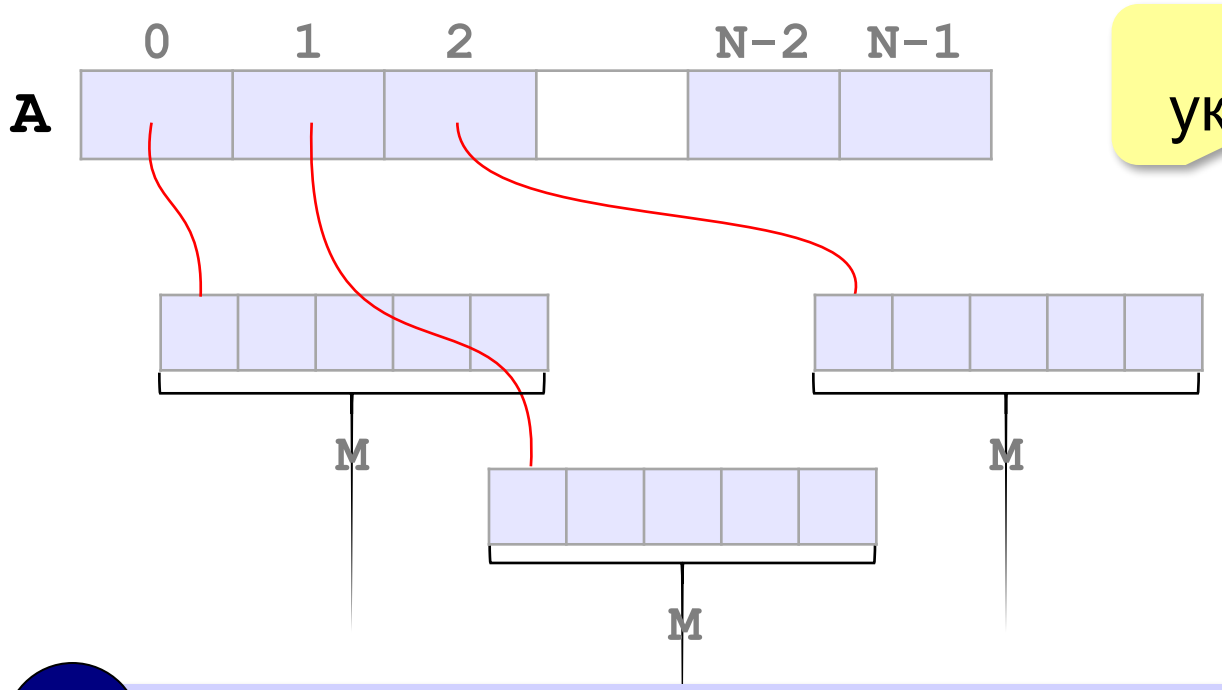
Работа с матрицей:

```
for ( i=0; i<N; i++ )
  for ( j=0; j<M; j++ )
    A[i][j]=i+j;
```

Удаление:

```
delete [] A[0];
delete [] A;
```

Динамические матрицы



массив указателей

Строки могут быть разной длины!

Динамические матрицы

Выделение памяти:

```
for ( i=0; i<N; i++ )
  A[i]=new int[M];
```

Освобождение памяти:

```
for ( i=0; i<N; i++ )
  delete [] A[i];
```

освободить память для отдельных строк

```
delete [] A;
```

освободить массив указателей

Динамические матрицы (vector)

Объявление:

```
typedef vector<int> vint;
vector <vint> A;
```

вектор из векторов

Изменение размера (число строк):

```
A.resize ( N );
```

Установка размера строк:

```
for ( i=0; i<N; i++ )
  A[i].resize ( M );
```

! Строки могут быть разной длины!

Использование:

```
for ( i=0; i<N; i++ )
  for ( j=0; j<M; j++ )
    A[i][j]=i+j;
```

Расширение массива

Задача. С клавиатуры вводятся натуральные числа, ввод заканчивается числом **0**. Нужно вывести на экран эти числа в порядке возрастания.

Какой размер массива нужен?

Ввод данных:

```
cin >> x;
while ( x != 0 )
   {
   A.push_back(x);
   cin >> x;
   }
```

автоматическое расширение

Алгоритмизация и программирование. Язык C++

§ 41. Списки

Зачем нужны списки?

Задача. В файле находится список слов, среди которых есть повторяющиеся. Каждое слово записано в отдельной строке. Построить **алфавитно-частотный словарь**: список слов в алфавитном порядке, справа от каждого слова должно быть указано, сколько раз оно встречается в исходном файле.

Нужно вставлять новые слова в список!

Список – это упорядоченный набор элементов одного типа, для которого введены операции вставки (включения) и удаления (исключения).

Алгоритм (псевдокод)

```
пока есть слова в файле
  {
  прочитать очередное слово
  если оно есть в списке то
    увеличить на 1 счётчик для этого слова
  иначе
    {
    добавить слово в список
    записать 1 в счетчик слова
    }
  }
```

Использование контейнера map (STL)

Map («отображение») – это словарь (ассоциативный массив). Индексы элементов – любые данные.

Объявление:

```
#include <map>
...
map <string,int> L;
```

индекс – строка

данные – целые

Размер словаря:

```
int p=L.count ( s );
```

Увеличение счётчика слова s:

```
L[s] ++;
```

Использование контейнера map (STL)

Вставка слова:

```
L.insert ( pair <string,int> (s,1) );
```

пара «строка – счётчик»

Заполнение словаря:

```
while ( Fin >> s ) {
  int p;
  p = L.count ( s );
  if ( p==1 )
    L[s] ++;
  else
    L.insert ( pair <string,int> (s, 1) );
  }
```

пока есть данные в файле

сколько раз встречается слово?

```
while ( Fin >> s ) L[s] ++;
```

Вывод результата

Итератор (или курсор) – специальный объект, который позволяет перебрать все элементы контейнера.

Объявление:

тип контейнера

```
map <string,int>::iterator it;
```

На первый элемент:

```
it=L.begin();
```

Вывод данных по текущему элементу:

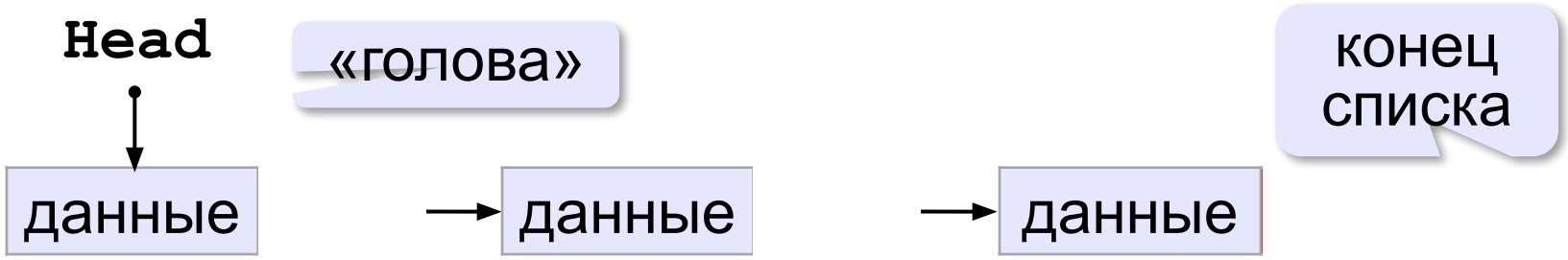
```
Fout << it->first << ": "
     << it->second;
```

```
автомат: 1
ананас: 12
...
```

Все элементы:

```
for ( it=L.begin(); it != L.end(); it++ )
  Fout << it->first << ": "
       << it->second << endl;
```

Связные списки (`list`)



узлы могут размещаться в разных местах в памяти

только последовательный доступ

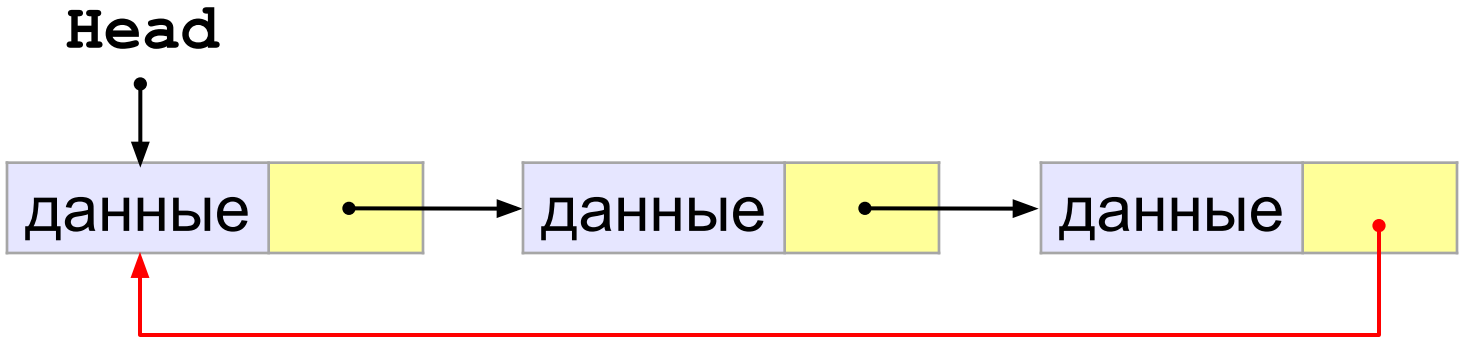
Рекурсивное определение:

- пустой список – это список
- список – это узел и связанный с ним список

Применение: много вставок в середину и удалений элементов!

Связные списки

Циклический список:



Двусвязный список:

обход в двух направлениях

сложнее вставка и удаление

Алгоритмизация и программирование. Язык C++

§ 42. Стек, дек, очередь

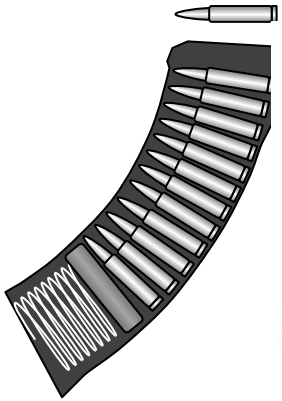
Что такое стек?

Стек (англ. *stack* – стопка) – это линейный список, в котором элементы добавляются и удаляются только с одного конца («последним пришел – первым ушел»).

LIFO = *Last In – First Out*.







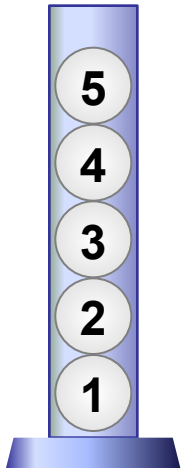

Системный стек:

- адреса возврата из подпрограмм
- передача аргументов подпрограмм
- хранение локальных переменных

Реверс массива

Задача. В файле записаны целые числа. Нужно вывести их в другой файл в обратном порядке.

```
пока файл не пуст
   {
   прочитать x
   добавить x в стек
   }
```




```
пока стек не пуст
   {
   вытолкнуть число из стека в x
   записать x в файл
   }
```

Использование контейнера stack (STL)

```
#include <stack>
...
stack <int> S;
```

стек целых чисел

Основные операции со стеком:

- **push** – добавить элемент на вершину стека
- **pop** – удалить элемент с вершины стека
- **top** – вернуть элемент с вершины стека (без удаления)
- **empty** – вернуть **true**, если стек пуст, и **false** в противном случае.

Использование контейнера stack (STL)

Переменные:

```
ifstream Fin;
ofstream Fout;
stack <int> S;
int x;
```

Чтение данных и загрузка в стек:

```
Fin.open ( "input.dat" );
while ( Fin >> x )
  S.push ( x );
Fin.close();
```

Использование контейнера stack (STL)

Вывод в обратном порядке:

```
Fout.open( "output.dat" );
while ( !S.empty() )
   {
   Fout << S.top() << endl;
   S.pop();
   }
Fout.close();
```

Вычисление арифметических выражений

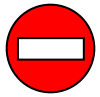
Как компьютер вычисляет арифметические выражения?

`(5+15)/(4+7-1)` **инфиксная форма** (знак операции между данными)

1920 (*Я. Лукашевич*): **префиксная форма** (знак операции перед данными)

```
/ + 5 15 - + 4 7 1
/ 20 - + 4 7 1
/ 20 - 11 1
/ 20 10
2
```

не нужны скобки



первой стоит последняя операция (вычисляем с конца)

Вычисление арифметических выражений

```
(5+15)/(4+7-1)
```

1950-е: **постфиксная форма**
(знак операции после данных)

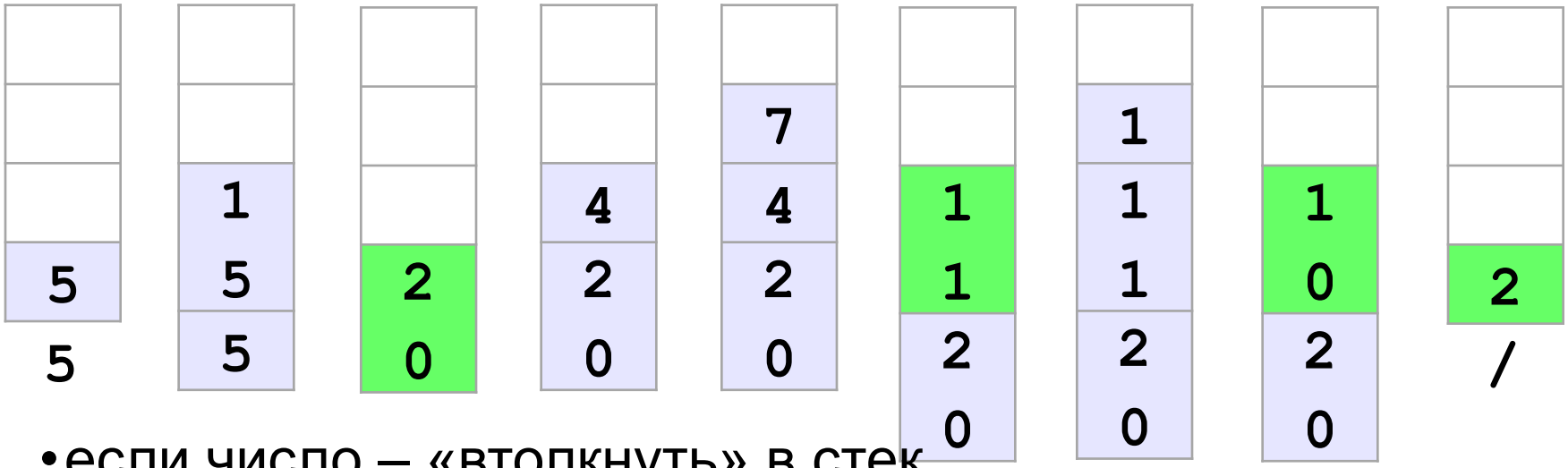
```
 5 15 + 4 7 + 1 - /
20 4 7 + 1 - /
20 11 1 - /
20 10 /
2
```

- не нужны скобки
- вычисляем с начала

! Вычисляем с помощью стека!

Использование стека

```
5 15 + 4 7 + 1 - /
```



- если число – «втолкнуть» в стек
- если операция – выполнить с верхними элементами стека

В стеке остается результат!

Скобочные выражения

Задача. Вводится символьная строка, в которой записано некоторое (арифметическое) выражение, использующее скобки трёх типов: **( )**, **[ ]** и **{ }**. Проверить, правильное ли расставлены скобки.

( ) [{ ( ) [ ] }] ✓ **[( )]** ✗ **[( ) }** ✗ **) (** ✗ **([ )]** ✗

Для одного типа скобок:

		(	)	(	(	)	(	(	)	)	)
счётчик	0	1	0	1	2	1	2	3	2	1	0

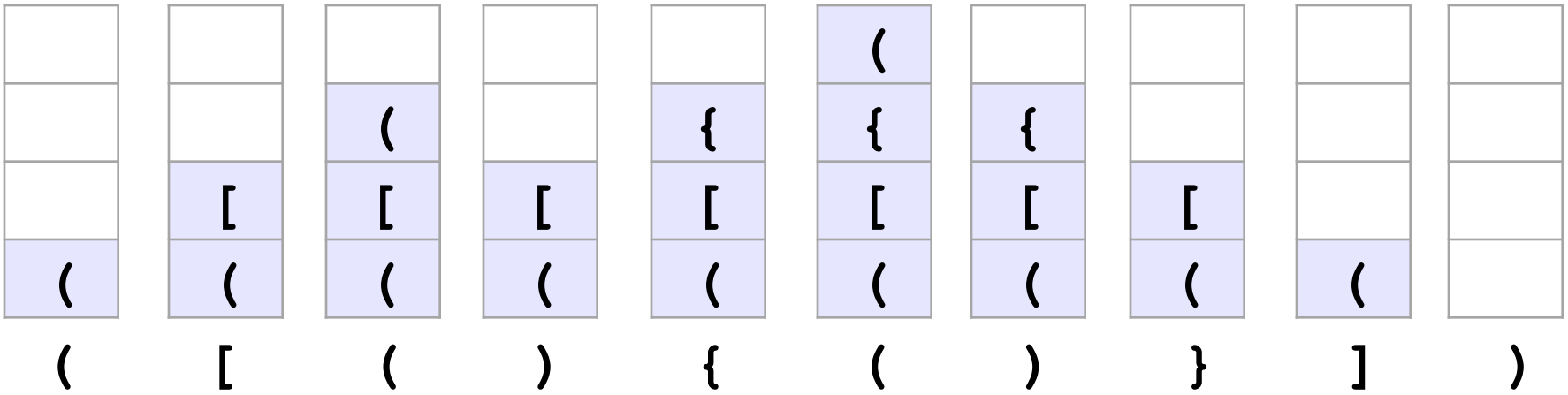
? Когда выражение правильное?

- счётчик всегда ≥ 0
- в конце счётчик = 0

({ [ ) }]

! Для разных скобок не работает!

Скобочные выражения (стек)



- если открывающая скобка – «втолкнуть» в стек
- если закрывающая скобка – снять парную со стека

? Когда выражение правильное?

- когда встретили закрывающую скобку, на вершине стека лежит соответствующая открывающая
- в конце работы стек пуст

Скобочные выражения (стек)

Константы и переменные:

```
const string L = "([{", // открывающие
               R = ")]}"; // закрывающие
string str;      // рабочая строка
stack <char> S; // стек
bool err;        // была ли ошибка?
int i, p;
char c;
```

Вывод результата:

```
if ( !err )
  cout << "Скобки расставлены верно.";
else
  cout << "Скобки расставлены неверно.";
```

Скобочные выражения (стек)

```
for ( i=0; i<str.size(); i++ ) {
  p=L.find ( str[i] );
  if ( p>=0 )
    S.push ( str[i] );
  p=R.find ( str[i] );
  if ( p>=0 ) {
    if ( S.empty() )
      err=true;
    else {
      c=S.top(); S.pop();
      if ( p!=L.find(c) )
        err=true;
      }
    if ( err ) break;
    }
  }
```

открывающую скобку в стек

если закрывающая скобка…

если не та скобка…

? Что ещё?

Что такое очередь?

Очередь – это линейный список, для которого введены две операции:

- добавление элемента в конец
- удаление первого элемента

FIFO = *Fist In – First Out*.

Применение:

- очереди сообщений в операционных системах
- очереди запросов ввода и вывода
- очереди пакетов данных в маршрутизаторах
- …

Заливка области

Задача. Рисунок задан в виде матрицы **A**, в которой элемент **A[y][x]** определяет цвет пикселя на пересечении строки **y** и столбца **x**. Перекрасить в цвет **2** одноцветную область, начиная с пикселя (x_0, y_0).

	0	1	2	3	4
0	0	1	0	1	1
1	1	1	1	2	2
2	0	1	0	2	2
3	3	3	1	2	2
4	0	1	1	0	0

(1,2) →

	0	1	2	3	4
0	0	2	0	1	1
1	2	2	2	2	2
2	0	2	0	2	2
3	3	3	1	2	2
4	0	1	1	0	0

Заливка: использование очереди

```
добавить в очередь точку (x0,y0)
запомнить цвет начальной точки
пока очередь не пуста
  {
  взять из очереди точку (x,y)
  если A[y][x] = цвету начальной точки то
    {
    A[y][x] = 2;
    добавить в очередь точку (x-1,y)
    добавить в очередь точку (x+1,y)
    добавить в очередь точку (x,y-1)
    добавить в очередь точку (x,y+1)
    }
  }
```

Очередь queue (STL)

```
#include <queue>
```

```
typedef struct {
  int x, y;
  } TPoint;
```

структура «точка»

```
queue <TPoint> Q;
```

контейнер «очередь» из точек

Построение структуры «точка»:

```
TPoint Point ( int x, int y )
{
  TPoint P;
  P.x=x; P.y=y;
  return P;
}
```

Очередь `queue` (STL)

Основные операции:

- **`push`** – добавить элемент в конец очереди
- **`pop`** – удалить первый элемент в очереди
- **`front`** – вернуть первый элемент в очереди (без удаления)
- **`empty`** – вернуть **`true`**, если очередь пуста, и **`false`** в противном случае.

Заливка

Константы и переменные:

```
const int XMAX=5, YMAX=5,
          NEW_COLOR=2;
```

```
int A[YMAX][XMAX];    // матрица
queue <TPoint> Q;     // очередь
int i, j, x0, y0, color;
TPoint pt;
```

Начало программы:

```
// заполнить матрицу A
y0=0; x0=1;       // начать заливку отсюда
color=A[y0][x0]; // цвет начальной точки
Q.push ( Point(x0,y0) );
```

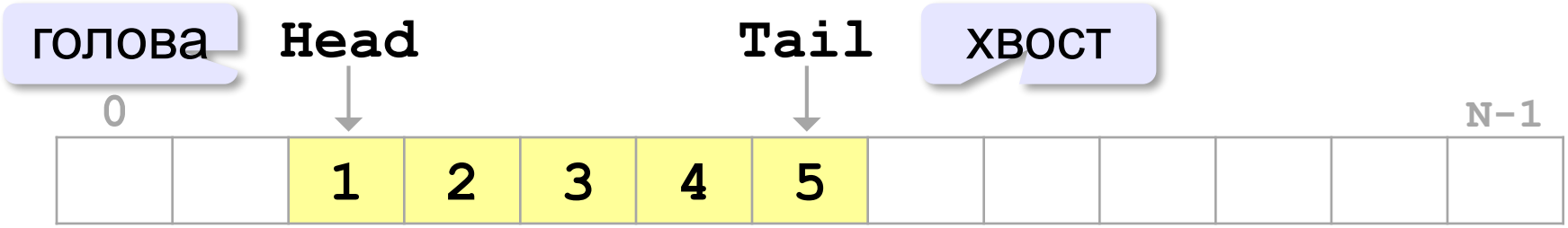
Заливка (основной цикл)

пока очередь не пуста

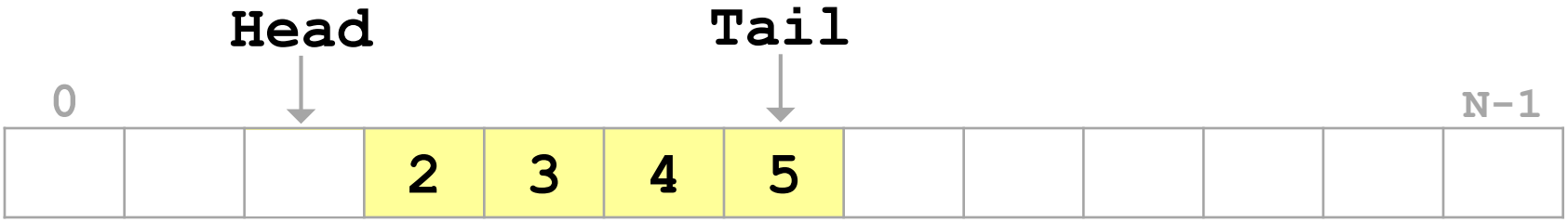
```
while ( !Q.empty() ) {
  pt=Q.front(); Q.pop();
  if ( A[pt.y][pt.x]==color ) {
    A[pt.y][pt.x]=NEW_COLOR;
    if ( pt.x>0 )
        Q.push ( Point(pt.x-1,pt.y) );
    if ( pt.y>0 )
        Q.push ( Point(pt.x,pt.y-1) );
    if ( pt.x<XMAX-1 )
        Q.push( Point(pt.x+1,pt.y) );
    if ( pt.y<YMAX-1 )
        Q.push( Point(pt.x,pt.y+1) );
    }
  }
```

? Что можно улучшить?

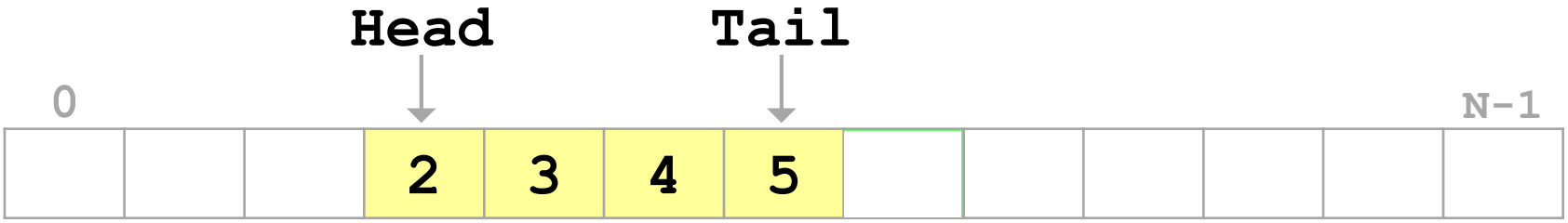
Очередь: статический массив



Удаление элемента:



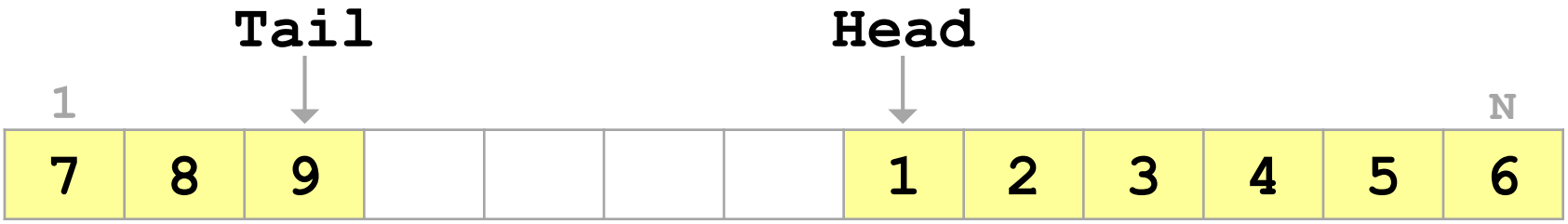
Добавление элемента:



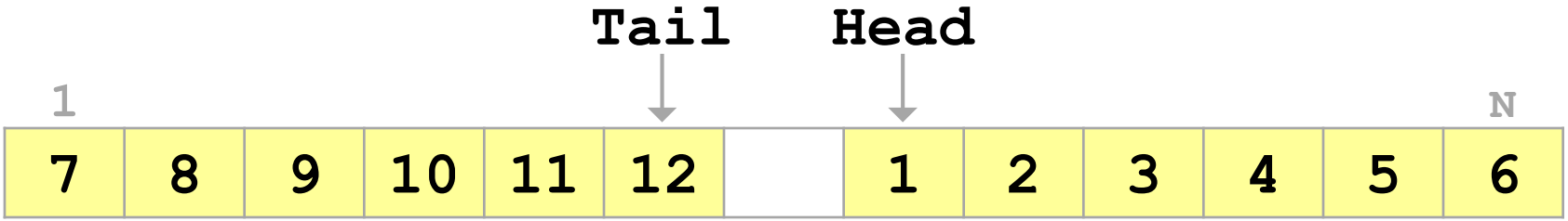
- не двигаем элементы
- нужно знать размер

Очередь: статический массив

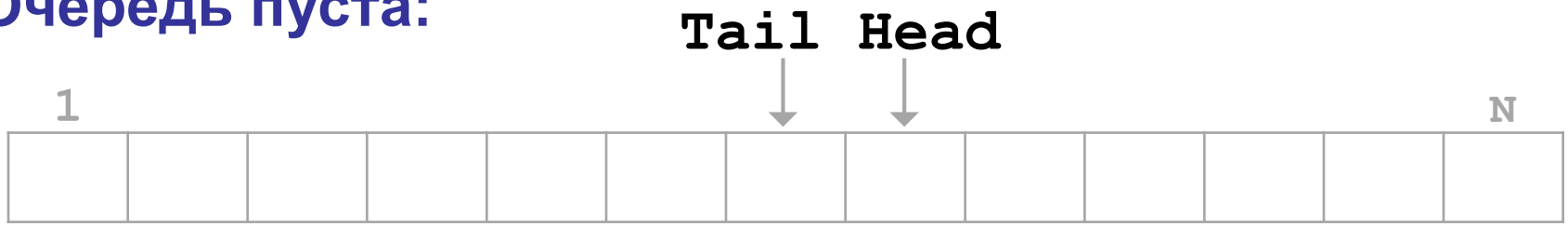
Замыкание в кольцо:



Очередь заполнена:



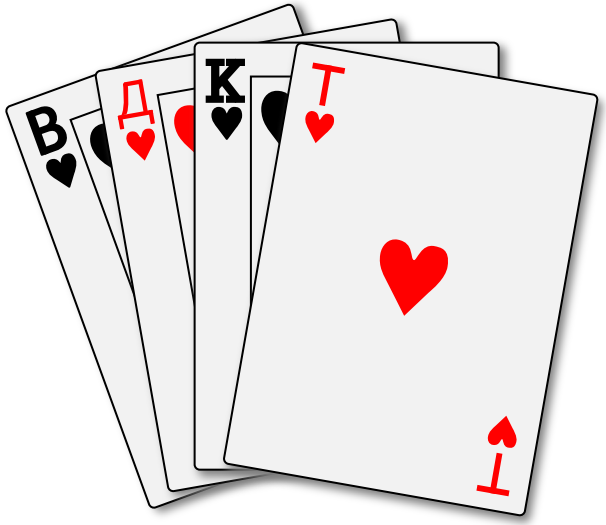
Очередь пуста:



Вариант: хранить размер очереди в переменной!

Что такое дек?

Дек – это линейный список, в котором можно добавлять и удалять элементы как с одного, так и с другого конца.



Моделирование:

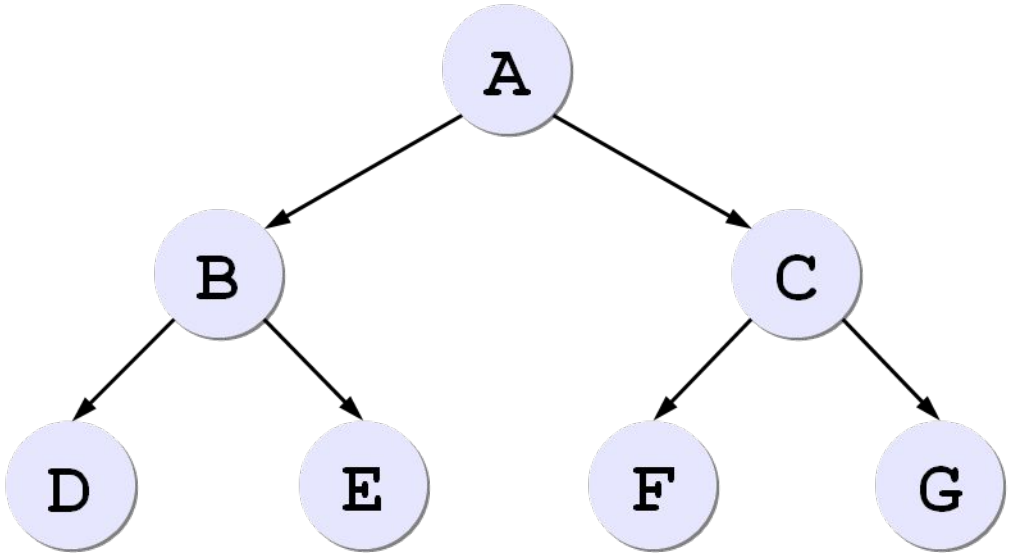
- статический массив (кольцо)
- динамический массив
- связный список

! STL: **deque**!

Алгоритмизация и программирование. Язык C++

§ 43. Деревья

Что такое дерево?



«Сыновья» A: B, C. **«Родитель» B**: A.

«Потомки» A: B, C, D, E, F, G. **«Предки» F**: A, C.

Корень – узел, не имеющий предков (A).

Лист – узел, не имеющий потомков (D, E, F, G).

Рекурсивные определения

1) пустая структура – это **дерево**
2) дерево – это корень и несколько связанных с ним отдельных (не связанных между собой) деревьев

Двоичное (бинарное) дерево:

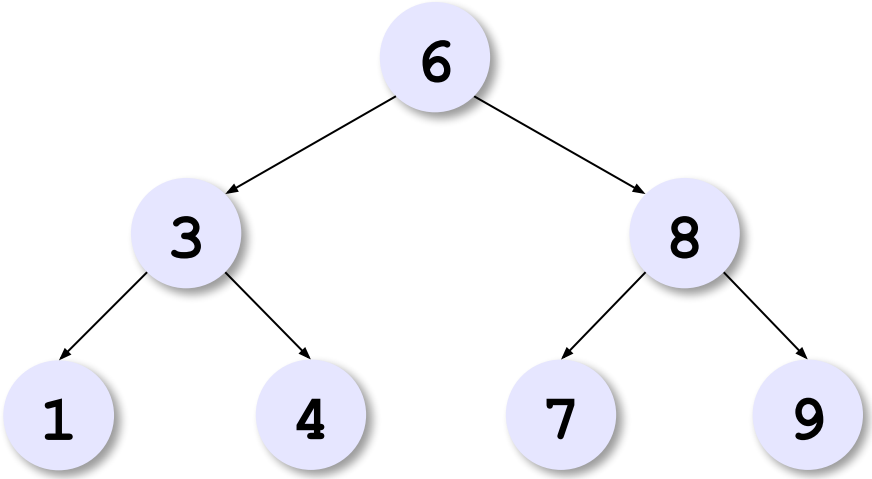
1) пустая структура – это **двоичное дерево**
2) двоичное дерево – это корень и **два** связанных с ним отдельных двоичных дерева («левое» и «правое» поддеревья)

Применение:

- поиск в большом массиве неменяющихся данных
- сортировка данных
- вычисление арифметических выражений
- оптимальное сжатие данных (метод Хаффмана)

Деревья поиска

Ключ – это значение, связанное с узлом дерева, по которому выполняется поиск.



- слева от узла – узлы с *меньшими* или равными ключами
- справа от узла – узлы с *большими* или равными ключами

$O(\log N)$

? Сложность поиска?

Двоичный поиск $O(\log N)$

Линейный поиск $O(N)$

Обход дерева

Обойти дерево ⇔ «посетить» все узлы по одному разу.

⇒ список узлов

КЛП – «корень-левый-правый» (в прямом порядке):

```
посетить корень
обойти левое поддерево
обойти правое поддерево
```

ЛКП – «левый-корень-правый» (симметричный):

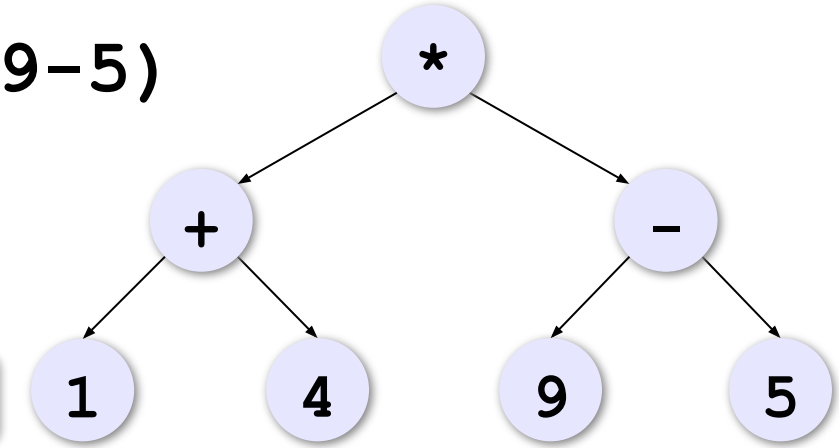
```
посетить корень
обойти левое поддерево
обойти правое поддерево
```

ЛПК – «левый-правый-корень» (в обратном порядке):

```
посетить корень
обойти левое поддерево
обойти правое поддерево
```

Обход дерева

(1+4)*(9-5)



«в глубину»

КЛП: * + 1 4 – 9 5 **префиксная форма**

ЛКП: 1 + 4 * 9 – 5 **инфиксная форма**

ЛПК: 1 4 + 9 5 – * **постфиксная форма**

Обход «в ширину»: «сыновья», потом «внуки», …

* + – 1 4 9 5

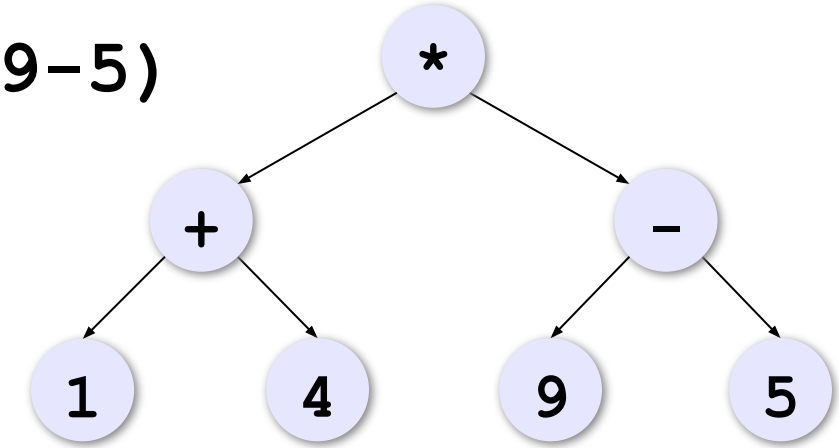
Обход КЛП – обход «в глубину»

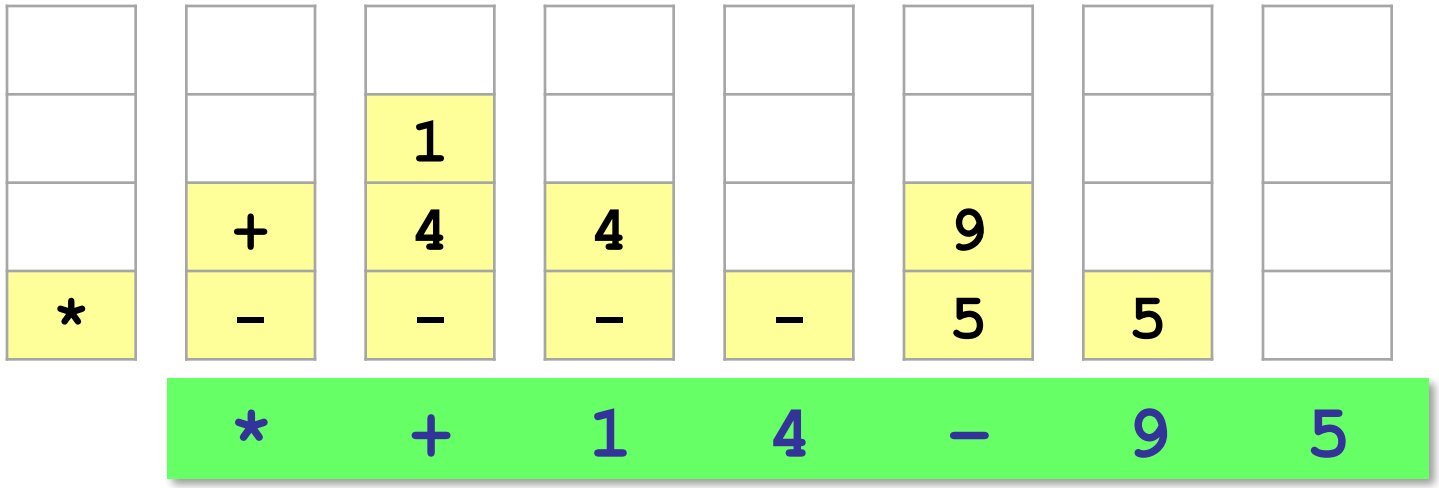
```
записать в стек корень дерева
пока стек не пуст
  {
  выбрать узел V с вершины стека
  посетить узел V
  если у узла V есть правый сын то
    добавить в стек правого сына V
  если у узла V есть левый сын то
    добавить в стек левого сына V
  }
```

? Почему сначала добавить правого сына?

Обход КЛП – обход «в глубину»

(1+4)*(9-5)





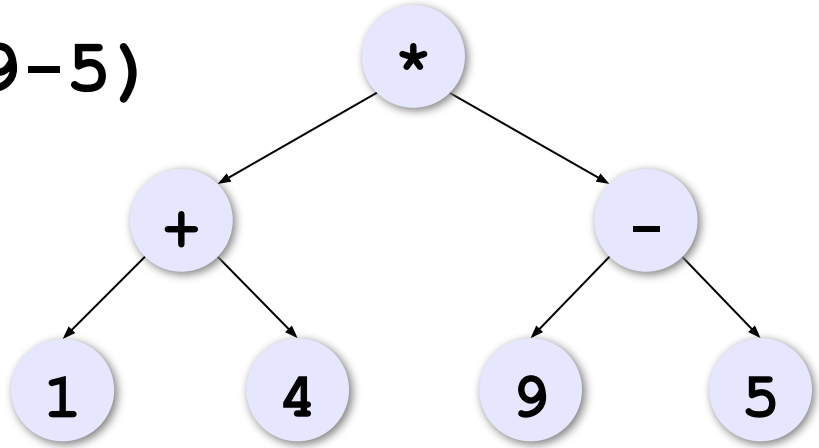
Обход «в ширину»

```
записать в очередь корень дерева
пока очередь не пуста
  {
  выбрать узел V из очереди
  посетить узел V
  если у узла V есть левый сын то
    добавить в очередь левого сына V
  если у узла V есть правый сын то
    добавить в очередь правого сына V
  }
```

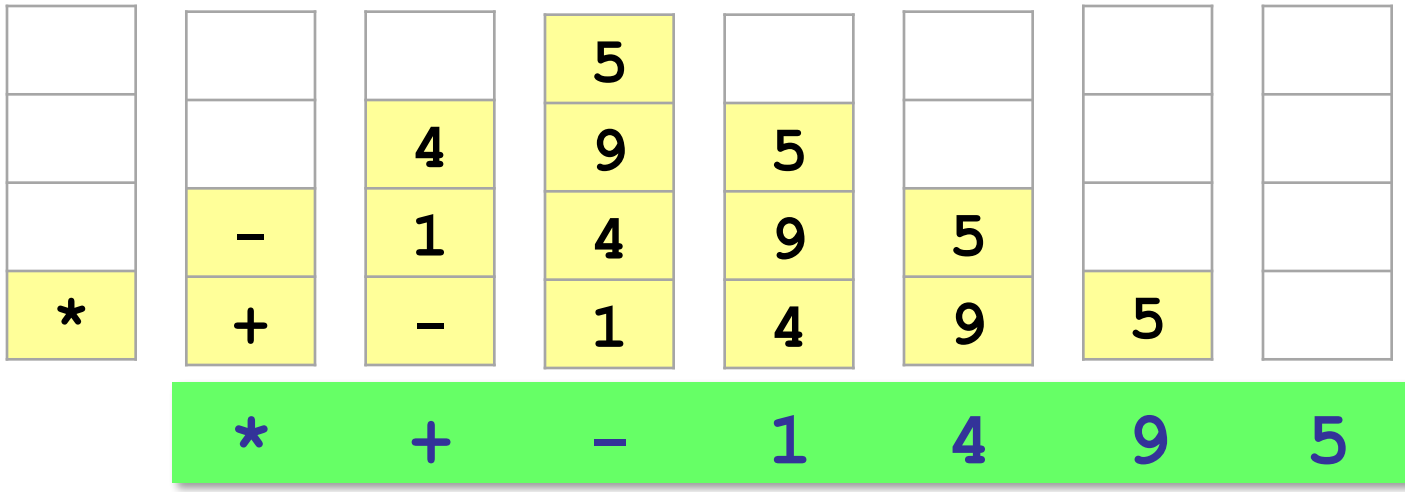
? Почему сначала добавить левого сына?

Обход «в ширину»

(1+4)*(9-5)



голова очереди

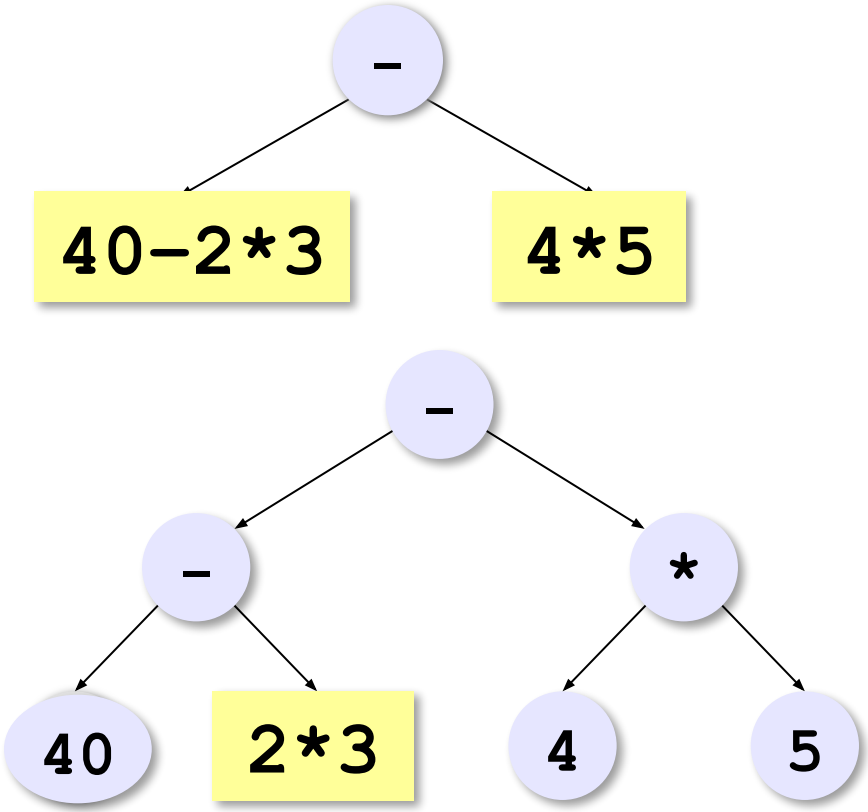


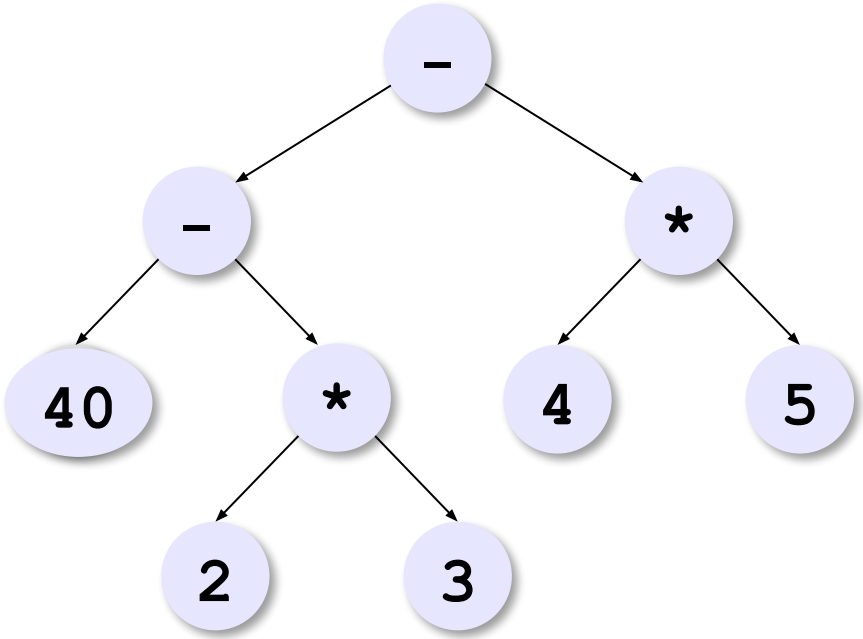
Вычисление арифметических выражений

`40-2*3-4*5`

? Что будет в корне дерева?

В корень дерева нужно поместить последнюю из операций с наименьшим приоритетом.





Вычисление арифметических выражений

Построение дерева:

```
найти последнюю выполняемую операцию
если операций нет то
   {
   создать узел-лист
   выход
   }
поместить операцию в корень дерева
построить левое поддерево
построить правое поддерево
```

Рекурсия!

Вычисление арифметических выражений

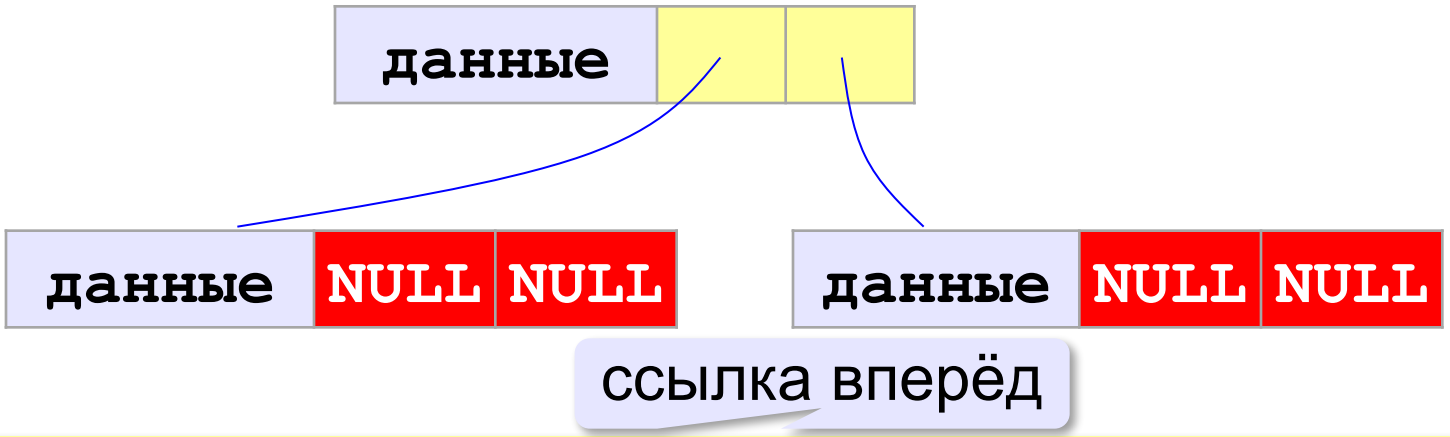
Вычисление по дереву:

```
n1 = значение левого поддерева
n2 = значение правого поддерева
результат = операция(n1, n2)
```

Рекурсия!

Использование связанных структур

Дерево – **нелинейная** структура ⇒ динамический массив неудобен!




```
typedef struct TNode *PNode;
typedef struct TNode
  {
  string data;
  PNode left;
  PNode right;
  } TNode;
```

новый тип: адрес узла

ссылки на сыновей

Работа с памятью

```
PNode p;  // указатель на узел
```

Выделить память для узла:

```
p = new TNode;
```

Обращение к новому узлу (по указателю):

```
p->data = s;
p->left = NULL;
p->right = NULL;
```

Освобождение памяти:

```
delete p;
```

не массив,
поэтому нет []

Основная программа

```
main()
{
   PNode T;
   string s;
      // ввести строку s
   T=MakeTree ( s );
   cout << "Результат: ", Calc(T);
}
```

Нужно построить **MakeTree** и **Calc**!

Построение дерева

```
PNode MakeTree ( string s )
{
  int k;
  PNode Tree;
  Tree = new struct TNode;
  k = LastOp ( s );
  if ( k == -1 ) {
    // новый узел – лист (число)
    }
  else {
    // новый узел – операция
    // построить поддеревья
    }
  return Tree;
}
```

вернёт адрес нового дерева

Построение дерева

Новый узел – лист:

```
Tree->data = s;
Tree->left = NULL;
Tree->right = NULL;
```

нет сыновей!

Новый узел – операция:

один символ!

```
Tree->data = s.substr(k,1);
Tree->left = MakeTree ( s.substr(0,k) );
Tree->right = MakeTree ( s.substr(k+1) );
```

Рекурсия!

до конца строки

Вычисление по дереву

```
int Calc ( PNode Tree )
{
  int n1, n2, res;
  if ( Tree->left == NULL )
    res = atoi ( Tree->data.c_str() );
  else {
    n1 = Calc ( Tree->left );
    n2 = Calc ( Tree->right );
    switch ( Tree->data[0] ) {
      case '+': res = n1 + n2; break;
      case '-': res = n1 - n2; break;
      case '*': res = n1 * n2; break;
      case '/': res = n1 / n2; break;
      default: res = 99999;
      }
    }
  return res;
}
```

ЭТО ЧИСЛО (ЛИСТ)

! Рекурсия!

Приоритет операции

```
int Priority ( char op )
{
  switch ( op )
    {
    case '+':
    case '-': return 1;
    case '*':
    case '/': return 2;
    }
  return 100;
}
```

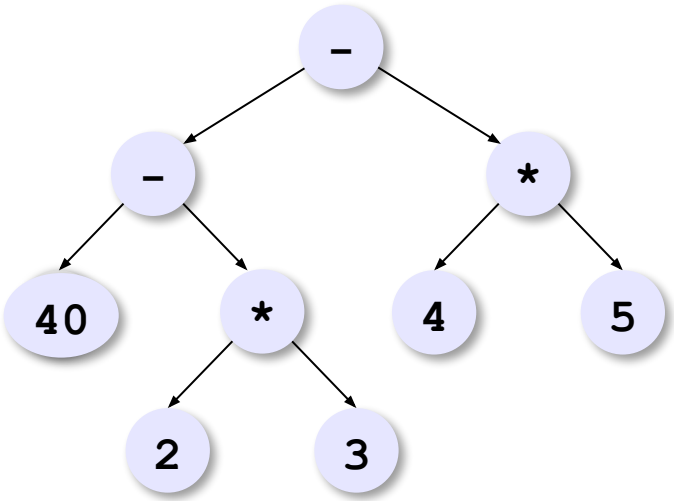
Последняя выполняемая операция

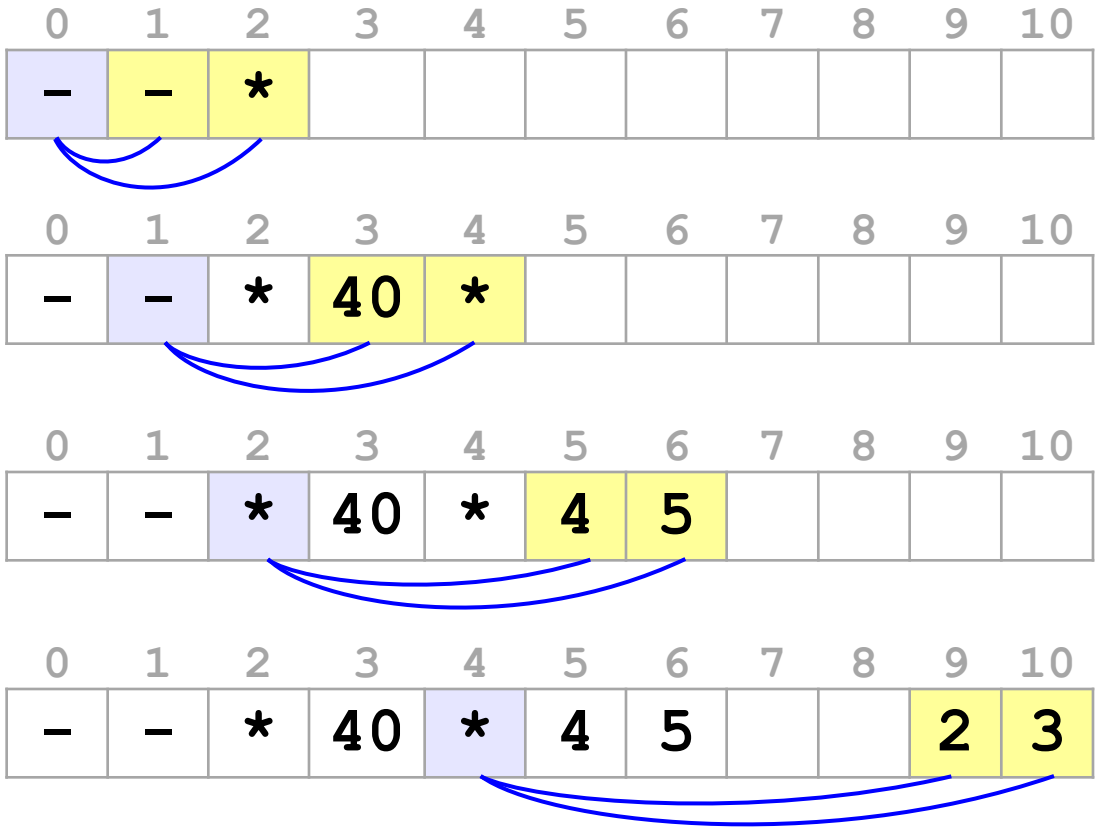
```
int LastOp ( string s )
{
  int i, minPrt, res;
  minPrt = 50; // любое между 2 и 100
  res = -1;
  for ( i = 0; i < s.size(); i++ )
    if ( Priority(s[i]) <= minPrt )
      {
      minPrt = Priority(s[i]);
      res = i;
      }
  return res;
}
```

вернёт номер символа

? Почему <=?

Двоичное дерево в массиве





A[0] → A[1], A[2]

A[1] → A[3], A[4]

A[2] → A[5], A[6]

A[4] → A[9], A[10]

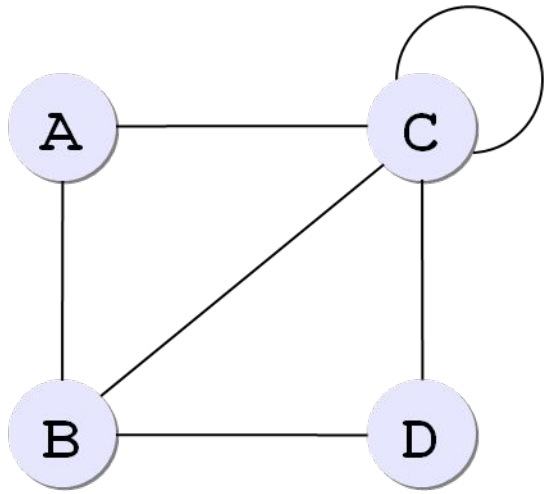
A[i] → A[?], A[?]

Алгоритмизация и программирование. Язык C++

§ 44. Графы

Что такое граф?

Граф – это набор вершин и связей между ними (рёбер).



Матрица смежности:

	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	1	1
D	0	1	1	0

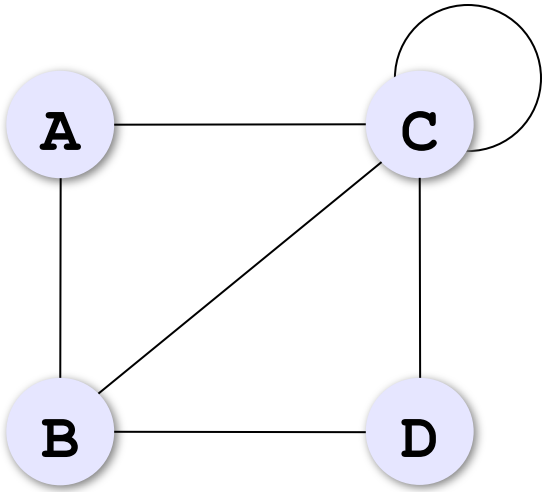
петля

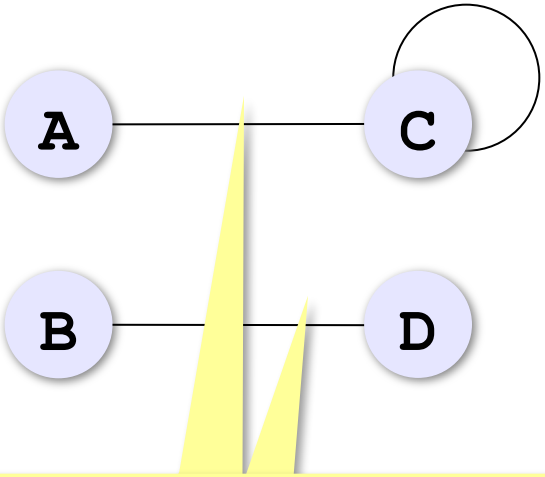
Список смежности:

```
( A(B, C),
  B(A, C, D),
  C(A, B, C, D),
  D(B, C) )
```

Связность графа

Связный граф – это граф, между любыми вершинами которого существует путь.

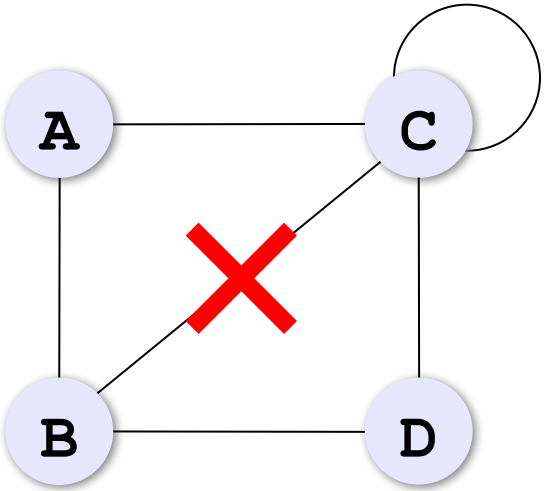




компоненты связности

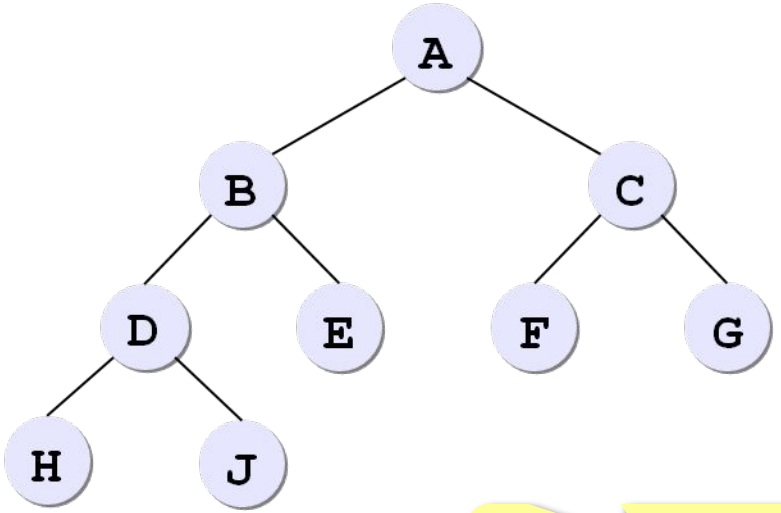
Дерево – это граф?

Дерево – это связный граф без циклов (замкнутых путей).



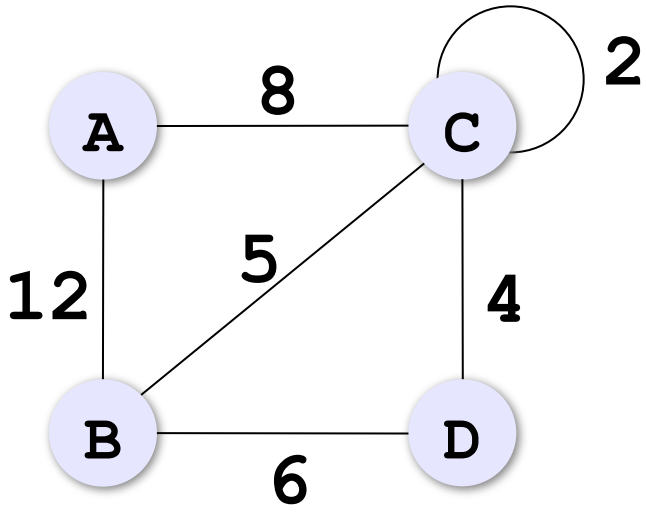
ABC ABDC

BCD CCC…



дерево

Взвешенные графы



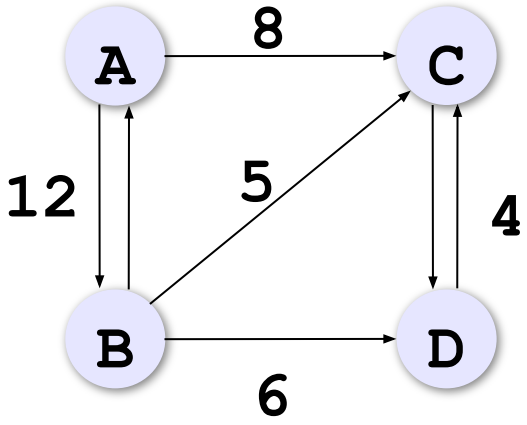
вес ребра

Весовая матрица:

	A	B	C	D
A		12	8	
B	12		5	6
C	8	5	2	4
D		6	4	

Ориентированные графы (орграфы)

Рёбра имеют направление (начало и конец), рёбра называю **дугами**.



	A	B	C	D
A		12	8	
B	12		5	6
C				4
D			4	

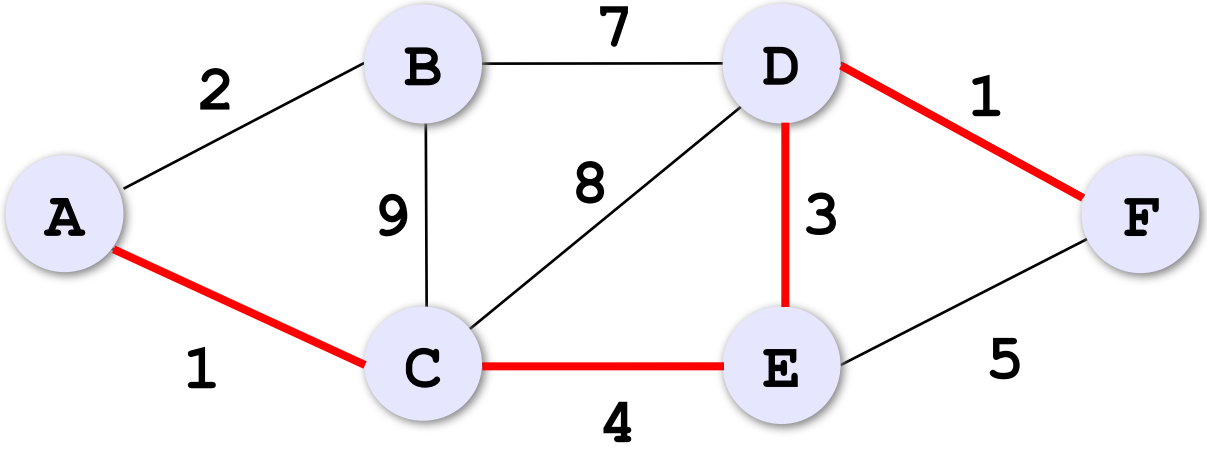
! Весовая матрица может быть несимметрична!

Жадные алгоритмы



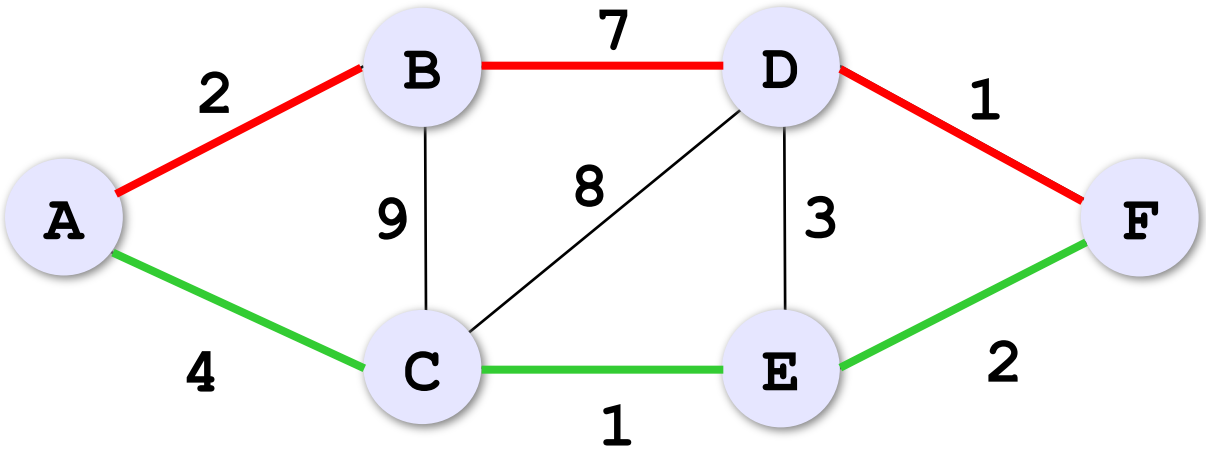
Жадный алгоритм – это многошаговый алгоритм, в котором на каждом шаге принимается решение, лучшее в данный момент.

Задача. Найти кратчайший маршрут из **A** в **F**.



Жадные алгоритмы

Задача. Найти кратчайший маршрут из **A** в **F**.

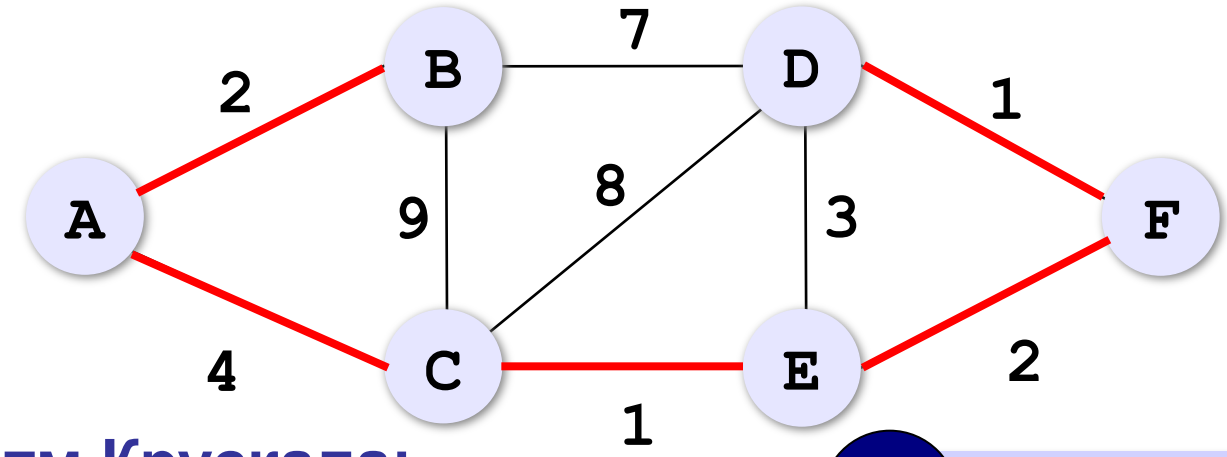


? Это лучший маршрут?

! Жадный алгоритм не всегда даёт наилучшее решение!

Задача Прима-Крускала

Задача. Между какими городами нужно проложить линии связи, чтобы все города были связаны в одну систему и общая длина линий связи была наименьшей? (**минимальное остовное дерево**)

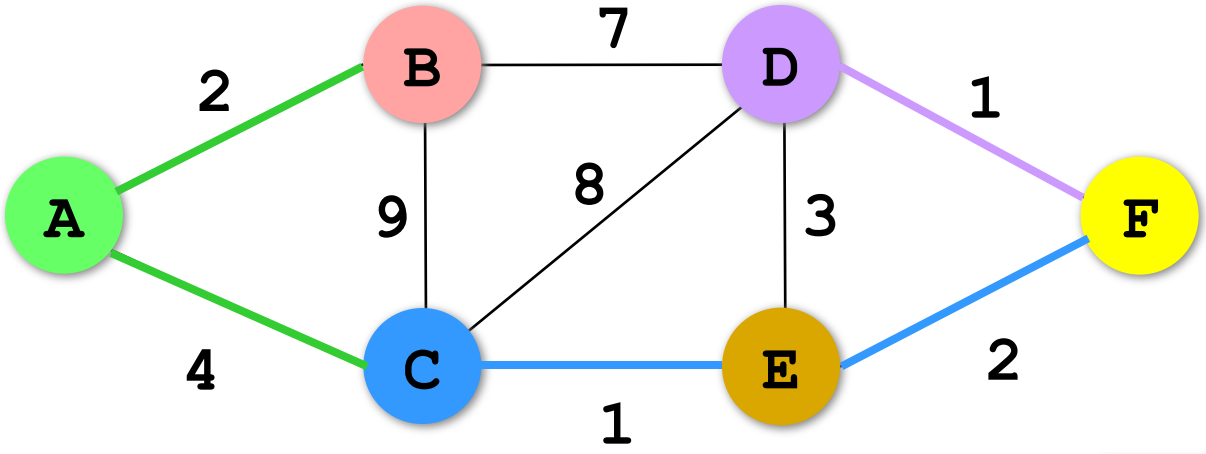


Алгоритм Крускала:

- начальное дерево – пустое
- на каждом шаге добавляется ребро минимального веса, которое ещё не входит в дерево и не приводит к появлению цикла

! Лучшее решение!

Раскраска вершин




```
for (i = 0; i < N; i ++) col[i] = i;
```

каждой вершине свой цвет

Сделать N-1 раз:

- ищем ребро минимальной длины среди всех рёбер, ***концы которых окрашены в разные цвета***;
- найденное ребро `(iMin,jMin)` добавляется в список выбранных, и все вершины, имеющие цвет `col[jMin]`, перекрашиваются в цвет `col[iMin]`.

Раскраска вершин

Данные:

```
const int N = 6;
int W[N][N]; // весовая матрица
int col[N];  // цвета вершин
    // номера вершин для выбранных ребер
int ostov[N-1][2];
int i, j, k, iMin, jMin, min, c;
```

Вывод результата:

```
for ( i = 0; i < N-1; i ++ )
  cout << "(" << ostov[i][0] << ","
       << ostov[i][1] << ")" << endl;
```

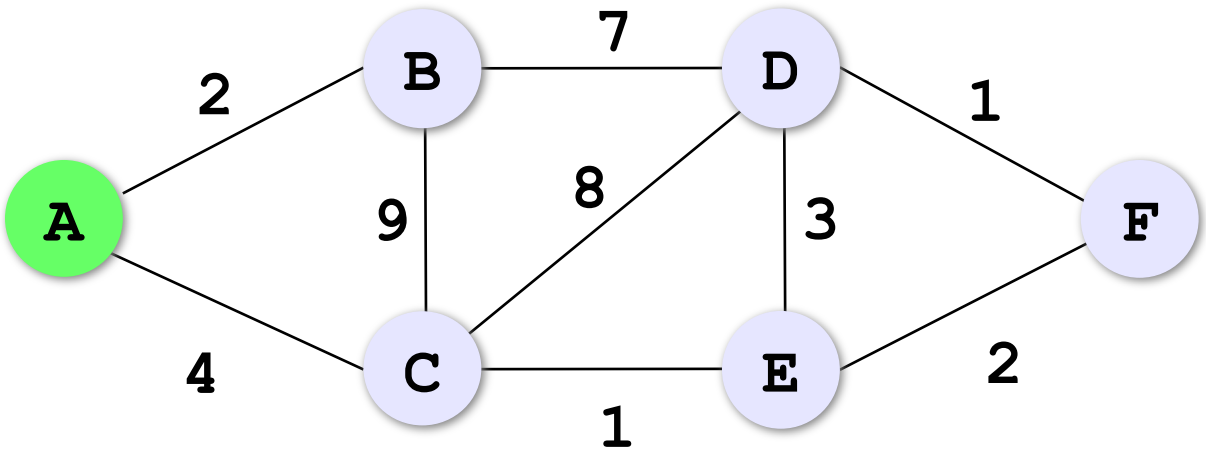
Раскраска вершин

```
for ( k=0; k<N-1; k++ ) {
    // поиск ребра с минимальным весом
  minDist=99999;
  for ( i=0; i<N; i++ )
    for ( j=0; j<N; j++ )
      if ( col[i]!=col[j] &&
           W[i][j]<minDist ) {
        iMin=i; jMin=j; minDist=W[i][j];
        }
    // добавление ребра в список выбранных
  ostov[k][0]=iMin; ostov[k][1]=jMin;
    // перекрашивание вершин
  c=col[jMin];
  for ( i=0; i<N; i++ )
    if ( col[i]==c ) col[i]=col[iMin];
  }
```

нет цикла

Кратчайший маршрут

Алгоритм Дейкстры (1960):



Э.В. Дейкстра

	A	B	C	D	E	F
R	0	2	4	∞	∞	∞
P	×	A	A	A	A	A

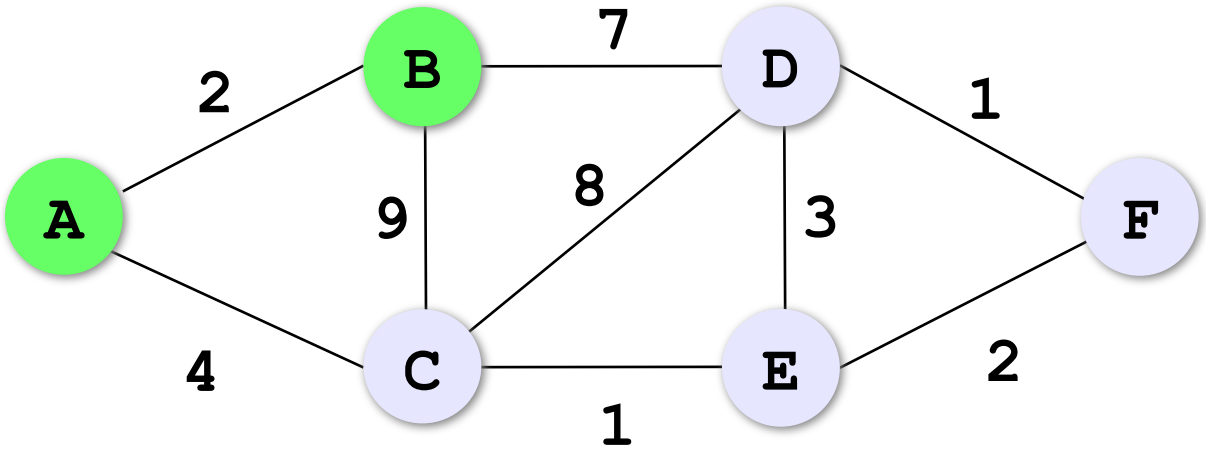
кратчайшее расстояние

откуда ехать

ближайшая от A невыбранная вершина

Кратчайший маршрут

Алгоритм Дейкстры (1960):

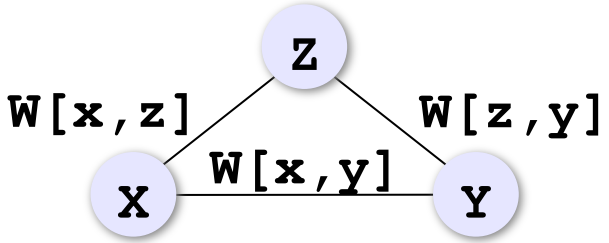


Э.В. Дейкстра

	A	B	C	D	E	F
R	0	2	4	9	∞	∞
P	x	A	A	B	A	A

кратчайшее расстояние

откуда ехать

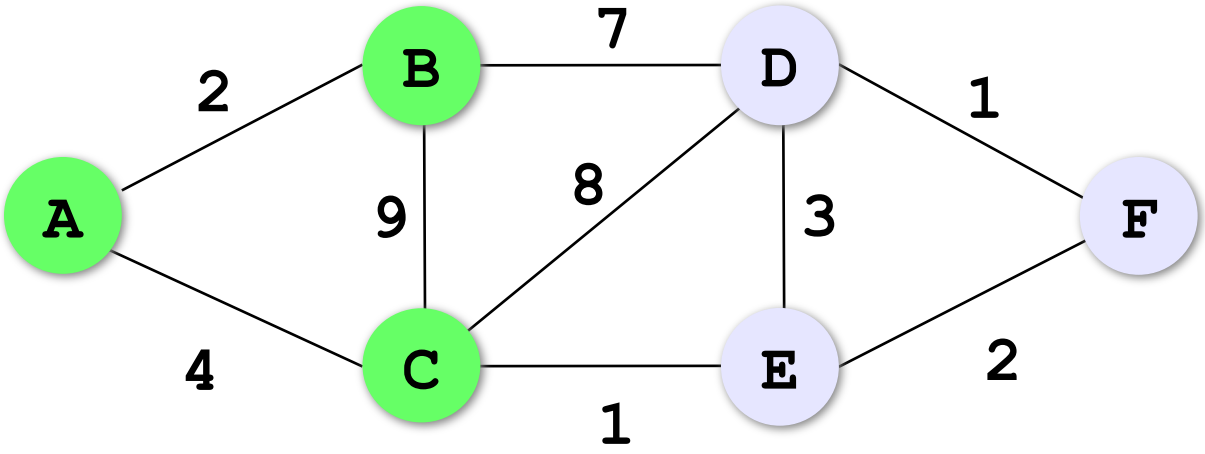


может быть так, что

$$W[x,z] + W[z,y] < W[x,y]$$

Кратчайший маршрут

Алгоритм Дейкстры (1960):

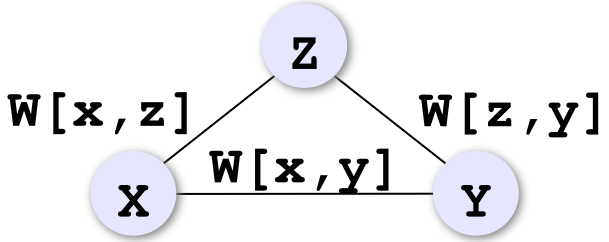


Э.В. Дейкстра

	A	B	C	D	E	F
R	0	2	4	9	5	∞
P	x	A	A	B	C	A

кратчайшее расстояние

откуда ехать

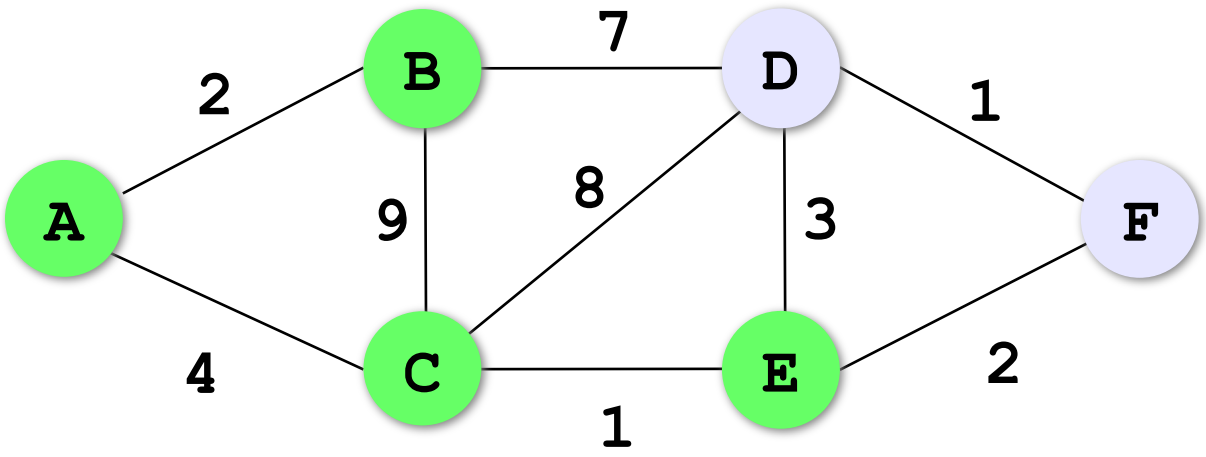


может быть так, что

$$W[x,z] + W[z,y] < W[x,y]$$

Кратчайший маршрут

Алгоритм Дейкстры (1960):



Э.В. Дейкстра

	A	B	C	D	E	F
R	0	2	4	8	5	7
P	×	A	A	E	C	E

кратчайшее расстояние

откуда ехать

При рассмотрении вершин **F** и **D** таблица не меняется!

Кратчайший маршрут

	A	B	C	D	E	F
R	0	2	4	8	5	7
P	×	A	A	E	C	E

длины кратчайших маршрутов из A в другие вершины

? Как найти сам маршрут?

	A	B	C	D	E	F
R	0	2	4	8	5	7
P	×	A	A	E	C	E

A → C → E → F

Алгоритм Дейкстры

Данные:

```
const int N = 6;
int W[N][N];     // весовая матрица
bool active[N]; // вершина не выбрана?
int R[N], P[N];
int i, j, min, kMin;
```

Начальные значения (выбор начальной вершины):

```
for ( i = 0; i < N; i ++ ) {
  active[i] = true; // все вершины не выбраны
  R[i] = W[0][i];   // рёбра из вершины 0
  P[i] = 0;
  }
active[0] = false; // вершина уже выбрана
P[0] = -1;         // это начальная вершина
```

Алгоритм Дейкстры

Основной цикл:

```
for ( i=0; i<N-1; i++ ) {
  minDist=99999;
  for ( j=0; j<N; j++ )
    if ( active[j] && R[j]<minDist) {
      minDist=R[j];
      kMin=j;
      }
  active[kMin]=false;
  for ( j=0; j<N; j++ )
    if ( R[kMin]+W[kMin][j]<R[j] ) {
      R[j]=R[kMin]+W[kMin][j];
      P[j]=kMin;
      }
  }
```

выбор следующей вершины, ближайшей к **A**

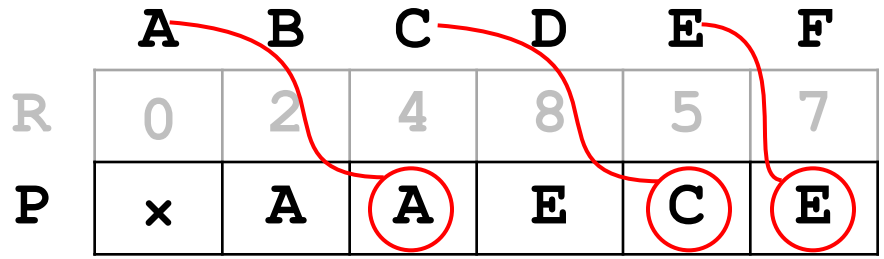
проверка маршрутов через вершину **kMin**

Алгоритм Дейкстры

Вывод результата (маршрут 0 → N-1):

```
i = N-1;
while ( i != -1 )
   {
   cout << i << " ";
   i = P[i];   // к следующей вершине
   }
```

для начальной вершины P[i]=-1

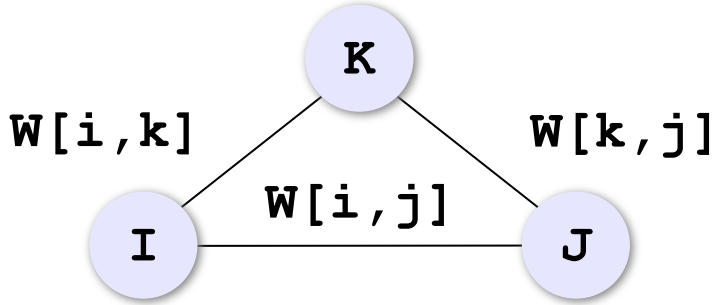


A → C → E → F

Алгоритм Флойда

Все кратчайшие пути (из любой вершины в любую):

```
for ( k=0; k<N; k++ )
  for ( i=0; i<N; i++ )
    for ( j=0; j<N; j++ )
      if ( W[i][k]+W[k][j]<W[i][j] )
         W[i][j]=W[i][k]+W[k][j];
```



? Как найти сам маршрут?

Алгоритм Флойда + маршруты

Дополнительная матрица:

```
for ( i = 0; i < N; i++ ) {
  for ( j = 0; j < N; j++ )
    P[i][j] = i;
  P[i][i] = -1;
  }
```

Кратчайшие длины путей и маршруты:

```
for ( k = 0; k < N; k++ )
  for ( i = 0; i < N; i++ )
    for ( j = 0; j < N; j++ )
      if ( W[i][k] + W[k][j] < W[i][j] ) {
        W[i][j] = W[i][k] + W[k][j];
        P[i][j] = P[k][j];
        }
```

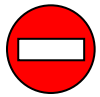
Задача коммивояжера

Коммивояжер (бродячий торговец) должен выйти из города **1** и, посетив по разу в неизвестном порядке города **2,3,...N**, вернуться обратно в город **1**. В каком порядке надо обходить города, чтобы путь коммивояжера был кратчайшим?

Это NP-полная задача, которая строго решается только перебором вариантов (пока)!

Точные методы:

1) простой перебор;
2) метод ветвей и границ;
3) метод Литтла;
4) …

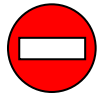


большое время счета для больших **N**

O(N!)

Приближенные методы:

5) метод случайных перестановок (*Matlab*)
6) генетические алгоритмы
7) метод муравьиных колоний
8) …



не гарантируется оптимальное решение

Некоторые задачи

Задача на минимум суммы. Имеется **N** населенных пунктов, в каждом из которых живет p_i школьников **(i=1,...,N)**. Надо разместить школу в одном из них так, чтобы общее расстояние, проходимое всеми учениками по дороге в школу, было минимальным.

Задача о наибольшем потоке. Есть система труб, которые имеют соединения в **N** узлах. Один узел **S** является источником, еще один – стоком **T**. Известны пропускные способности каждой трубы. Надо найти наибольший поток от источника к стоку.

Задача о наибольшем паросочетании. Есть **M** мужчин и **N** женщин. Каждый мужчина указывает несколько (от **0** до **N**) женщин, на которых он согласен жениться. Каждая женщина указывает несколько мужчин (от **0** до **M**), за которых она согласна выйти замуж. Требуется заключить наибольшее количество моногамных браков.

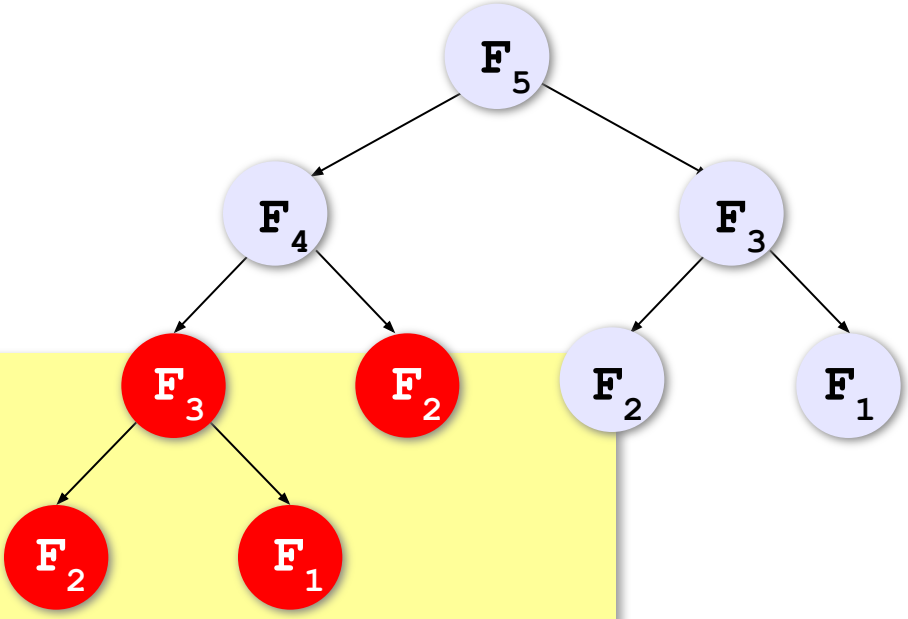
Алгоритмизация и программирование. Язык C++

§ 44. Динамическое программирование

Что такое динамическое программирование?

Числа Фибоначчи:

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \text{ при } n > 2$$




```
int Fib ( int N )
{
  if ( N<3 )
        return 1;
  else return Fib(N-1) + Fib(N-2);
}
```

повторное вычисление тех же значений

Запоминать то, что вычислено!

Динамическое программирование

F_1	F_2	F_3	F_4	F_5
1	1			

$$F_1 = F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \text{ при } n > 2$$

Объявление массива:

```
const int N=10;
int F[N+1]; // чтобы начать с 1
```

Заполнение массива:

```
F[1]=1; F[2]=1;
for ( i=3; i<=N; i++ )
  F[i]=F[i-1]+F[i-2];
```

F_{45}: рекурсия: **8 с**

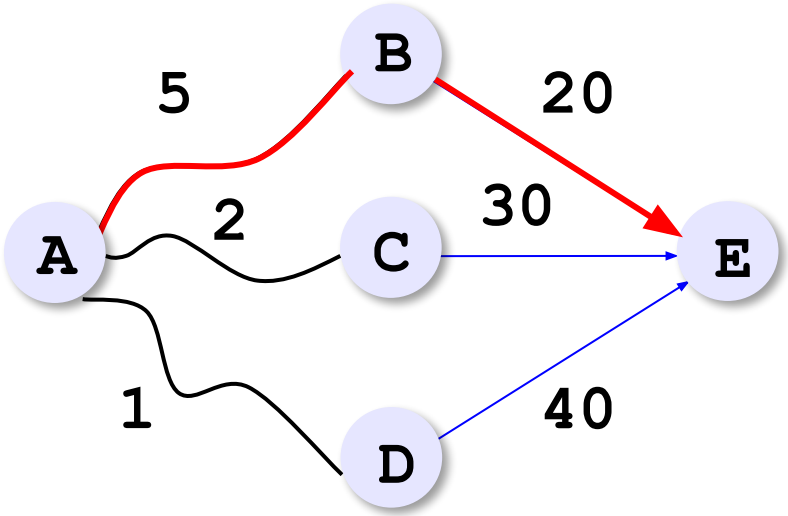
дин. программирование: **< 0,01 с**

? Можно ли обойтись без массива?

нужны только два последних!

Динамическое программирование

Динамическое программирование – это способ решения сложных задач путем сведения их к более простым задачам того же типа.

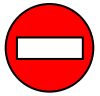


ABE: 5 + 20 = 25

ACE: 2 + 30 = 32

ADE: 1 + 40 = 41


увеличение скорости

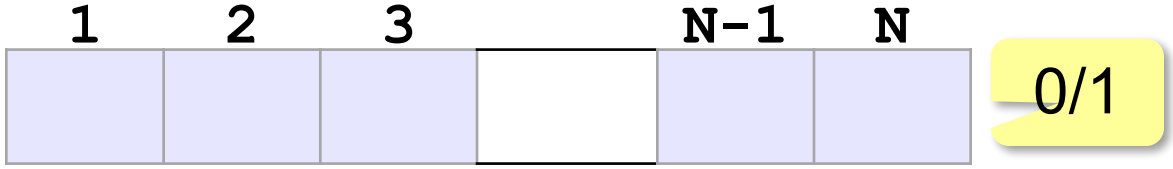

дополнительный расход памяти

Количество вариантов

Задача. Найти количество K_N цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд единиц.

Решение «в лоб»:

битовые цепочки



- построить все возможные цепочки
- проверить каждую на «правильность»

? Сколько возможных цепочек? 2^N

Сложность алгоритма $O(2^N)$

Количество вариантов

Задача. Найти количество K_N цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд единиц.

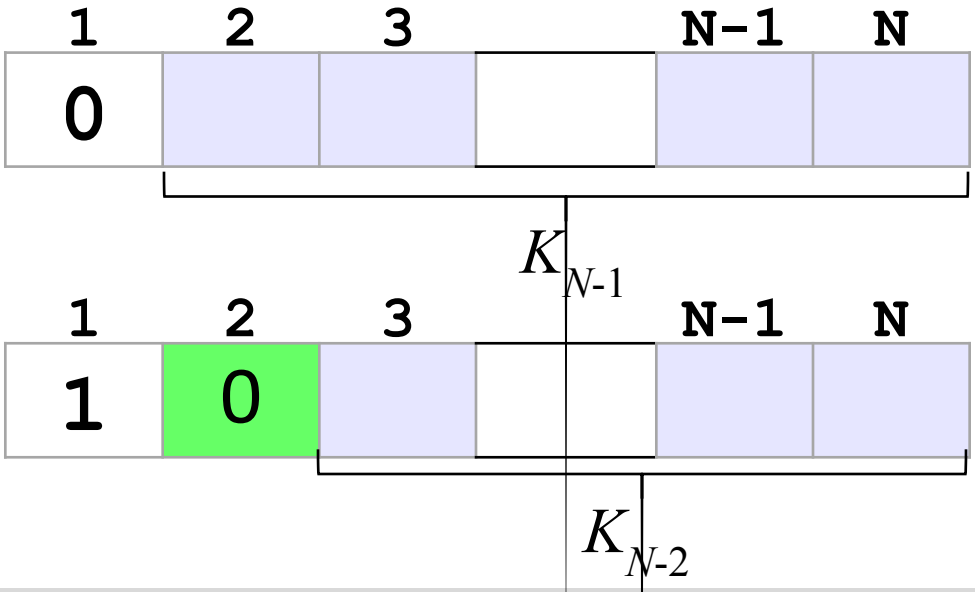
Простые случаи:

$N = 1$: **0** **1** $K_1 = 2$

$N = 2$: **00** **01** **10** $K_2 = 3$

$$K_N = K_{N-1} + K_{N-2} = F_{N+2}$$

Общий случай:



K_{N-1} «правильных» цепочек начинаются с нуля!

K_{N-2} «правильных» цепочек начинаются с единицы!

Оптимальное решение

Задача. В цистерне N литров молока. Есть бидоны объемом 1, 5 и 6 литров. Нужно разлить молоко в бидоны так, чтобы все бидоны были заполнены и количество используемых бидонов было **минимальным**.

Перебор?

при больших N – очень долго!

«Жадный алгоритм»?

$N = 10$: 10 = 6 + 1 + 1 + 1 + 1 $K = 5$

10 = 5 + 5 $K = 2$

! Не даёт оптимального решения!

Оптимальное решение

K_N – минимальное число бидонов для N литров

Сначала выбрали бидон…

1 л: $K_N = 1 + K_{N-1}$

5 л: $K_N = 1 + K_{N-5}$

6 л: $K_N = 1 + K_{N-6}$

} min

Рекуррентная формула:

$$K_N = 1 + \min(K_{N-1}, K_{N-5}, K_{N-6}) \quad \text{при } N \geq 6$$

$$K_N = 1 + \min(K_{N-1}, K_{N-5}) \quad \text{при } N = 5$$

$$K_N = 1 + K_{N-1} \quad \text{при } N < 5$$

Оптимальное решение (бидоны)

$$K_N = 1 + \min(K_{N-1}, K_{N-5}, K_{N-6})$$

N	0	1	2	3	4	5	6	7	8	9	10
K_N	0	1	2	3	4	1	1	2	3	4	2
P	0	1	1	1	1	5	6	1	1	1	5

объём бидона, взятого последним

N	0	1	2	3	4	5	6	7	8	9	10
K_N	0	1	2	3	4	1	1	2	3	4	2
P	0	1	1	1	1	5	6	1	1	1	5

2 бидона 5 + 5

Похоже на алгоритм Дейкстры!

Задача о куче

Задача. Из камней весом p_i (i=1, …, N) набрать кучу весом ровно W или, если это невозможно, максимально близкую к W (но меньшую, чем W).

Решение «в лоб»:

1	2	3		N-1	N
1	0	0		1	0

камень взят

камень не взят

! Выбрать лучшую цепочку!

? Сколько возможных цепочек? 2^N

Сложность алгоритма $O(2^N)$

Задача о куче

Задача. Из камней весом p_i (i=1, …, N) набрать кучу весом ровно W или, если это невозможно, максимально близкую к W (но меньшую, чем W).

Идея: сохранять в массиве решения всех более простых задач этого типа (при меньшем количестве камней N и меньшем весе W).

Пример: W = 8, камни 2, 4, 5 и 7

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0								
3	5	0								
4	7	0								

w

базовые случаи

i

p_i

`T[i][w]` – оптимальный вес, полученный для кучи весом `w` из `i` первых по счёту камней.

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0								
4	7	0								

Добавляем камень с весом 4:

для `w < 4` ничего не меняется!

для `w ≥ 4`:

если его не брать: `T[2][w] = T[1][w]`

если его взять: `T[2][w] = 4 + T[1][w-4]`

? Какой вариант выбрать?

`max`

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0	0	2	2	4	5	6	7	7
4	7	0								

Добавляем камень с весом 5:

для `w < 5` ничего не меняется!

для `w ≥ 5`:

если его не брать: `T[3][w] = T[2][w]`

если его взять: `T[3][w] = 5 + T[2][w-5]`

`max`

Задача о куче

		0	1	2	3	4	5	6	7	8
1	2	0	0	2	2	2	2	2	2	2
2	4	0	0	2	2	4	4	6	6	6
3	5	0	0	2	2	4	5	6	7	7
4	7	0	0	2	2	4	5	6	7	7

Добавляем камень с весом 7:

для `w < 7` ничего не меняется!

для `w ≥ 7`:

если его не брать: `T[4][w] = T[3][w]`

если его взять: `T[4][w] = 7 + T[3][w-7]`

max

Задача о куче

Добавляем камень с весом p_i:

для $w < p_i$ ничего не меняется!

для $w \geq p_i$:

если его не брать: `T[i][w] = T[i-1][w]`

если его взять: $T[i][w] = p_i + T[i-1][w-p_i]$

max

Рекуррентная формула:

при $w < p_i$: `T[i][w] = T[i-1][w]`

при $w \geq p_i$:

```
T[i][w] = max(T[i-1][w],
              p_i+T[i-1][w-p_i])
```

Задача о куче

? Какие камни нужно взять?

	0	1	2	3	4	5	6	7	8
2	0	0	2	2	2	2	2	2	2
4	0	0	2	2	4	4	6	6	6
5	0	0	2	2	4	5	6	7	7
7	0	0	2	2	4	5	6	7	7

Оптимальный вес 7 5 + 2

Задача о куче

? Какова сложность алгоритма?

Заполнение таблицы:

W+1

N	0	1	2	3	4	5	6	7	8
2	0	0	2	2	2	2	2	2	2
4	0	0	2	2	4	4	6	6	6
5	0	0	2	2	4	5	6	7	7
7	0	0	2	2	4	5	6	7	7

! Сложность $O(N \cdot W)$!

псевдополиномиальный

Количество программ

Задача. У исполнителя Утроитель есть команды:

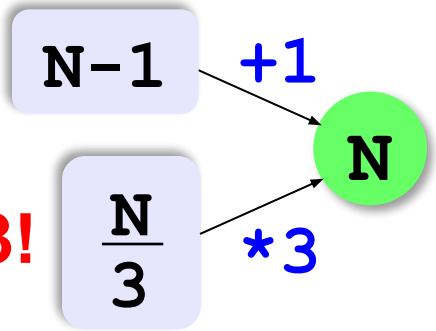
1. **прибавь 1**
2. **умножь на 3**

Сколько есть разных программ, с помощью которых можно из числа **1** получить число **20**?

Как решать, не выписывая все программы?

Количество программ

Как получить число N:



последняя команда

если делится на 3!

Рекуррентная формула:

$K_N = K_{N-1}$ если N не делится на 3

$K_N = K_{N-1} + K_{N/3}$ если N делится на 3

Количество программ

Рекуррентная формула:

$K_N = K_{N-1}$ если N не делится на 3

$K_N = K_{N-1} + K_{N/3}$ если N делится на 3

Заполнение таблицы:

N	1	2	3	4	5	6	7	8	9	10
K_N	1	1	2	2	2	3	3	3	5	5

одна пустая!

K_2+K_1

K_5+K_2

K_8+K_3

N	11	12	13	14	15	16	17	18	19	20
K_N	5	7	7	7	9	9	9	12	12	12

Количество программ

20

Только точки изменения:

N	1	3	6	9	12	15	18	21
K_N	1	2	3	5	7	9	12	15

Программа:

```
K[1] = 1;
for ( i = 2; i <= N; i ++ )
   {
   K[i] = K[i-1];
   if ( i % 3 == 0 )
      K[i] = K[i] + K[i/3];
   }
```

? Где ответ?

? Как объявить массив K?

Размен монет

Задача. Сколькими различными способами можно выдать сдачу размером W рублей, если есть монеты достоинством p_i (i=1, …, N)? В наборе есть монета достоинством 1 рубль ($p_1 = 1$).

Перебор?

при больших N и W – очень долго!

Динамическое программирование:

запоминаем решения всех задач меньшей размерности: для меньших значений W и меньшего числа монет N.

Размен монет

Пример: W = 10, монеты 1, 2, 5 и 10

		0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	1										
3	5	1										
4	10	1										

w

базовые случаи

i

p_i

`T[i][w]` – количество вариантов для суммы `w` с использованием `i` первых по счёту монет.

Рекуррентная формула (добавили монету p_i):

при $w < p_i$: `T[i][w] = T[i-1][w]`

при $w \geq p_i$: `T[i][w] = T[i-1][w] + T[i][w-p_i]`

без этой монеты

все варианты размена остатка

Размен монет

Пример: W = 10, монеты 1, 2, 5 и 10

		0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	1	1	2	2	3	3	4	4	5	5	6
3	5	1	1	2	2	3	4	5	6	7	8	10
4	10	1	1	2	2	3	4	5	6	7	8	11

Рекуррентная формула (добавили монету p_i):

при $w < p_i$: `T[i,w] = T[i-1][w]`

при $w \geq p_i$: `T[i,w] = T[i-1][w] + T[i][w-`p_i`]`

Конец фильма

ПОЛЯКОВ Константин Юрьевич
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

ЕРЕМИН Евгений Александрович
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

Источники иллюстраций

1. wallpaperscraft.com
2. www.mujerhoy.com
3. www.pinterest.com
4. www.wayfair.com
5. www.zchocolat.com
6. www.russiantable.com
7. www.kursachworks.ru
8. ebay.com
9. centrgk.ru
10. www.riverstonellc.com
11. 53news.ru
12. 10hobby.ru
13. ru.wikipedia.org
14. иллюстрации художников издательства «Бином»
15. авторские материалы