

# Алгоритмизация и программирование

§ 38. Целочисленные алгоритмы

§ 39. Структуры (записи)

2-е изд.

§ 40. Множества

§ 40. Динамические массивы

§ 41. Списки

§ 42. Стек, очередь, дек

§ 43. Деревья

§ 44. Графы

§ 45. Динамическое программирование

# Алгоритмизация и программирование

## **§ 38. Целочисленные алгоритмы**

# Решето Эратосфена



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

## Алгоритм:

- 1) начать с  $k = 2$
- 2) «выколоть» все числа через  $k$ , начиная с  $k \cdot k$
- 3) перейти к следующему «невыколотому»  $k$
- 4) если  $k \cdot k \leq N$ , 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:

```
цел i, k, N = 100  
логтаб A[2:N]
```

```
const N = 100;  
var i, k: integer;  
    A: array[2..N]  
      of boolean;
```

Сначала все невычеркнуты:

```
нц для i от 2 до N  
    A[i] := да  
кц
```

```
for i := 2 to N do  
    A[i] := True;
```

# Решето Эратосфена

## Вычёркивание непростых:

```
к := 2
нц пока к*к <= N
  если А[к] то
    і := к*к
    нц пока і <= N
      А[і] := нет
      і := і + к
    кц
  все
  к := к + 1
кц
```

```
к := 2;
while к*к <= N do begin
  if А[к] then begin
    і := к*к;
    while і <= N do begin
      А[і] := False;
      і := і + к
    end
  end;
  к := к + 1
end;
```

# Решето Эратосфена

---

## Вывод результата:

```
нц для i от 2 до N
  если A[i] то
    вывод i, нс
  все
кц
```

```
for i:=2 to N do
  if A[i] then
    writeln ( i );
```

## «Длинные» числа

---

Ключи для шифрования:  $\geq 256$  битов.

Целочисленные типы данных:  $\leq 64$  битов.



Как хранить?

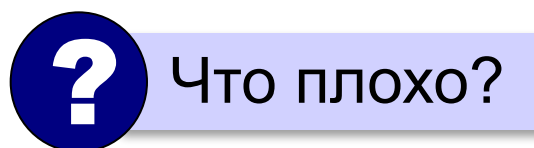
**Длинное число** – это число, которое не помещается в переменную одного из стандартных типов данных языка программирования.

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

# «Длинные» числа

**A = 12345678**

	0	1	2	3	4	5	6	7	8	9
<b>A</b>	1	2	3	4	5	6	7	8	0	0



- 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
<b>A</b>	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
<b>A</b>	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!

`longint`:

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

$$100! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot 99 \cdot 100$$



Как оценить количество цифр?

$$1 \cdot 2 \cdot 3 \cdot \dots \cdot 99 \cdot 100 <$$

201 цифра

основание 1000000

6 цифр в ячейке  $\Rightarrow$  34 ячейки

цел  $N = 33$

целтаб  $A[0:N]$

```
const N = 33;
```

```
var A: array[0..N]
```

```
of longint;
```

Основной алгоритм:

длинное  
число

```
[A] := 1
```

```
нц для k от 2 до 100
```

```
  [A] := [A] * k
```

```
кц
```

# Вычисление факториала

основание  $d = 1\ 000\ 000$

[A] = 12345678901734567

	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] := mod(s, d)
r := div(s, d)
```

```
s := A[0] * k;
A[0] := s mod d;
r := s div d;
```

? Что изменится для A[1]?

$s := A[1] * k + r$

# Вычисление факториала

## Умножение «длинного» числа на k:

```
r := 0
нц для i от 0 до N
  s := A[i]*k + r
  A[i] := mod(s, d)
  r := div(s, d)
кц
```

```
r := 0;
for i := 0 to N do begin
  s := A[i]*k + r;
  A[i] := s mod d;
  r := s div d
end;
```

## Вычисление 100!:

```
нц для k от 2 до 100
  ...
кц
```

```
for k := 2 to 100 do
  begin
    ...
  end;
```

# Вывод длинного числа

	3	2	1	0
A	0	1	2	3



Какое число?

[A] = 1000002000003

- найти старший ненулевой разряд

```
i := N
нц пока A[i] = 0
  i := i - 1
кц
```

```
i := N;
while A[i] = 0 do
  i := i - 1;
```

- вывести этот разряд

```
вывод A[i]
```

```
write( A[i] );
```

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

# Вывод длинного числа

## Вывод остальных разрядов:

нц для k от  $i-1$  до 0 шаг  $-1$

    Write6(A[k])

кц

со старшего

for k:=i-1 downto 0 do  
    Write6(A[k]);

## Write6:

x = 12345

x div 100000

012345

x mod 100000

x	M	x div M
12345	100000	0

# Вывод длинного числа

---

Вывод числа с лидирующими нулями:

```
алг Write6 (цел x)
нач
  цел M, xx
  xx := x
  M := 100000
  нц пока M > 0
    вывод div (xx, M)
    xx := mod (xx, M)
    M := div (M, 10)
  кц
кон
```

# Вывод длинного числа

---

Вывод числа с лидирующими нулями:

```
procedure Write6 (x: longint) ;
var M: longint;
begin
  M:= 100000 ;
  while M > 0 do begin
    write (x div M) ;
    x:= x mod M;
    M:= M div 10
  end
end;
```



# Алгоритмизация и программирование

## **§ 39. Структуры (записи)**

# Зачем нужны структуры?

## Книги в библиотеках:

- автор
- название
- количество экземпляров
- ...

символьные строки

целое число



Как хранить данные?

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

## Несколько массивов:

```
var authors: array[1..N] of string;  
    titles: array[1..N] of string;  
    count: array[1..N] of integer;  
    ...
```

**Задача:** объединить разнотипные данные в один блок.

# Структуры (записи)

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

НОВЫЙ ТИП ДАННЫХ

type

структура (запись)

**TBook** = record

author: string[40]; { автор, строка }

title: string[80]; { название, строка }

count: integer { количество, целое }

end;

# Объявление структур

```
const N = 100;  
var B: TBook; { одна структура }  
    Books: array[1..N] of TBook; { массив }
```



Сколько места занимает в памяти?

```
writeln(sizeof(TBook)); { 124 }  
writeln(sizeof(B)); { 124 }  
writeln(sizeof(Books)); { 12400 }
```

type

TBook = record

author: string[40];

title: string[80];

count: integer

end;

40 + 1 (размер) байт

80 + 1 (размер) байт

2 байта (*FreePascal*)

# Обращение к полям структур

---

## Точечная нотация:

```
B.author { поле author структуры B }
```

```
Books[5].count { поле count структуры  
Books[5] }
```

```
writeln(sizeof(B.author)); { 41 }  
writeln(sizeof(B.title)); { 81 }  
writeln(sizeof(B.count)); { 2 }
```

```
read(B.author); { ввод полей }  
read(B.title);  
read(B.count);
```

```
writeln(B.author, ' ', { вывод }  
B.title, ' ', B.count, ' шт. ');
```

# Обращение к полям структур

---

## Присваивание:

```
V.author := 'Пушкин А.С.';  
V.title := 'Полтава';  
V.count := 1;
```

## Использование:

```
p := Pos(' ', V.author);  
fam := Copy(V.author, 1, p-1); { фамилия }  
V.count := V.count - 1; { одну книгу взяли }  
if V.count = 0 then  
    writeln('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!

Типизированные файлы:

```
var F: file of TBook;
```

```
Assign(F, 'books.dat');
```

```
Rewrite(F);
```

```
  B.author := 'Тургенев И.С.';
```

```
  B.title := 'Муму';
```

```
  B.count := 2;
```

```
  write(F, B);
```

```
Close(F);
```

```
for i:=1 to N do  
  write(F, Books[i]);
```

ТОЛЬКО ДЛЯ TBook!

# Чтение структур из файла

```
Assign (F, 'books.dat') ;  
Reset (F) ;  
  Read (F, B) ;  
  writeln (B.author, ', ', B.title,  
          ', ', B.count) ;  
Close (F) ;
```

ТОЛЬКО для TBook!

```
for i:=1 to N do { известное количество }  
  read (F, Books[i]) ;
```

```
i:=0 ;  
while not Eof (F) do begin { до конца файла }  
  i:=i+1 ;  
  Read (F, Books[i])  
end ;
```

счётчик прочитанных структур



# Сортировка структур

Ключ – фамилия автора:



Какой метод?

```
for i:=1 to N-1 do
  for j:=N-1 downto i do
    if Books[j].author > Books[j+1].author
      then begin
        B:= Books[j];
        Books[j]:= Books[j+1];
        Books[j+1]:= B
      end;
end;
```

var B: TBook;



структуры перемещаются в памяти

# Указатели

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

```
type PBook = ^TBook;
```

*pointer*

указатель на  
переменную типа TBook

```
var P: PBook;
```

P

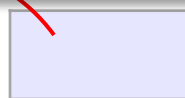


```
P := @B;
```

```
P^ .author
```

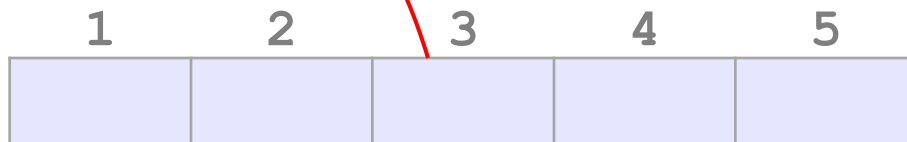
```
P := @Books[3];
```

B



```
B .author
```

Books

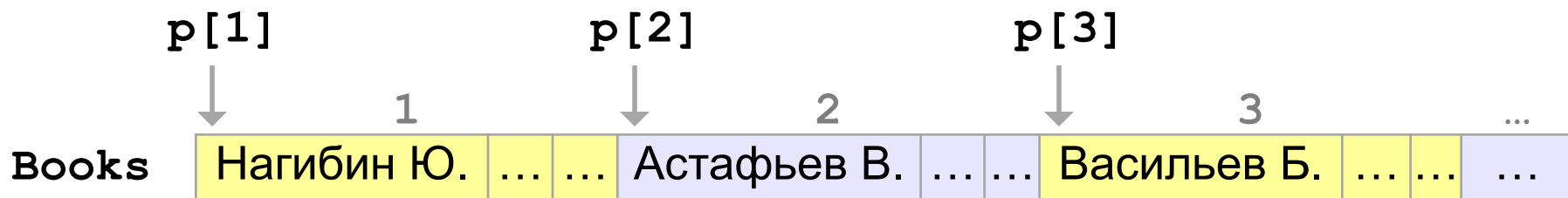


$P^{\wedge} .author \Leftrightarrow Books[3] .author$

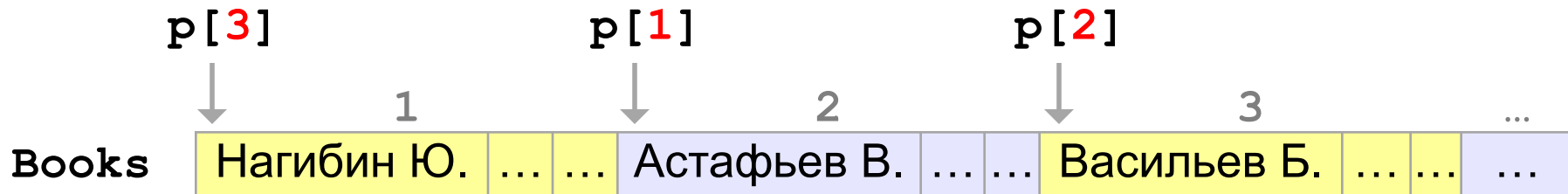
# Сортировка по указателям

```
var p: array[1..N] of PBook;
```

```
for i:=1 to N do  
  p[i] := @Books[i];
```



**Задача – переставить указатели:**



Сами структуры не перемещаются!

# Сортировка по указателям

```
for i:=1 to N-1 do
  for j:=N-1 downto i do
    if p[j]^author > p[j+1]^author
      then begin
        p1:=p[j]; p[j]:=p[j+1];
        p[j+1]:=p1
      end;
end;
```

обращение к полям  
через указатели

переставляем  
указатели!

```
var p1: PBook;
```

## Вывод результата:

```
for i:=1 to N do
  writeln(p[i]^author, '; ',
          p[i]^title, '; ',
          p[i]^count);
```

# Алгоритмизация и программирование

2-е издание

## § 40. Множества

# Зачем нужны множества?

*Задача.* Определить количество знаков препинания в символьной строке.

```
count := 0;
for i := 1 to Length(s) do
  if (s[i] = '.' ) or (s[i] = ',' )
    or (s[i] = ';' ) or (s[i] = ':' )
    or (s[i] = '!' ) or (s[i] = '?' ) then
    count := count + 1;
```



Некрасиво!

**Использование множества:**

ВХОДИТ ВО

МНОЖЕСТВО

```
if s[i] in ['.', ',', ';', ':', '!', '?'] then
  count := count + 1;
```

# Использование множеств

```
if s[i] in ['0'.. '9'] then  
    count := count + 1;
```

диапазон

```
if s[i] in ['a'.. 'z', '0'.. '9',  
           '.', '-', '_'] then  
    { символ правильный }
```

Переменная типа «множество»:

```
var digits: set of '0'.. '9';
```

множество цифр

# Использование множеств

*Задача.* Вывести все различные цифры, присутствующие в символьной строке *s*.

```
var s: string;  
    i: integer;  
    c: char;  
    digits: set of '0'..'9';  
begin  
  readln(s);  
  digits:=[]; { пустое множество }  
  for i:=1 to Length(s) do  
    if s[i] in ['0'..'9']  
    then digits:=digits + [s[i]];  
  for c:='0' to '9' do  
    if c in digits then writeln(c)  
end.
```

СЛОЖИТЬ ДВА  
МНОЖЕСТВА

Вывод через перебор



# Использование множеств

```
var cs: set of char;  
    bs: set of byte;
```

до 255  
элементов

Имена для элементов множества:

стили шрифта

жирный

курсив

```
type TFontStyles = (fsBold, fsItalic,  
                   fsUnderline);
```

подчёркивание

```
var fs: set of TFontStyles;
```

Операции с множеством:

```
fs := [fsBold, fsItalic]; { присвоить значение }  
fs := fs + [fsUnderline]; { добавить элементы }  
fs := fs - [fsItalic];    { исключить элементы }
```

# Вывод множества на экран

первый

```
type TFontStyles = (fsBold, fsItalic,  
                    fsUnderline);
```

```
var style: TFontStyles;
```

последний

```
fs := [fsBold, fsItalic];
```

```
for style:=fsBold to fsUnderline do
```

```
  if style in fs then
```

```
    write(1)
```

```
  else write(0);
```



Что будет выведено?

fsBold

110

fsUnderline

fsItalic

# Сравнение множеств

```
var A, B, C: set of 0..9;  
...  
A := [1, 2, 3, 4]; B := [2, 3]; C := [2, 3]
```

## Равенство/неравенство:

```
if A = B then write('A = B'); { нет }  
if B = C then write('B = C'); { да }
```

```
if A <> B then write('A <> B'); { да }  
if C <> B then write('C <> B'); { нет }
```

## Включение одного в другое:

```
if A >= B then write('A >= B'); { да }  
if C <= B then write('C <= B'); { да }
```

```
if A > B then write('A > B'); { да }  
if C < B then write('C < B'); { нет }
```

# Множества в памяти

```
var digits: set of 0..9;
```



Как хранить в памяти?

Пустое множество:

```
digits := [];
```

0123456789

0000000000

Непустое множество:

```
digits := [1, 3, 5, 7];
```

0123456789

0101010100

Пересечение множеств:

```
digits := [1, 3, 5, 7] *  
          [2, 3, 4];
```

0123456789

0101010100

0011100000

0001000000

ЛОГИЧЕСКОЕ  
УМНОЖЕНИЕ

AND

[3]



Аппаратно!

# Множества в памяти

## Объединение множеств:

```
digits := [1, 3, 5, 7] +
          [2, 3, 4];
```

логическое  
сложение

0123456789

0101010100

0011100000

0111110100

OR

[1, 2, 3, 4, 5, 7]

## Вычитание множеств:

```
digits := [1, 3, 5, 7] -
          [2, 3, 4];
```

0123456789

0101010100

0011100000

0100010100

[1, 5, 7]



Как сделать с помощью битовых операций?

# Алгоритмизация и программирование

## § 40. Динамические массивы

# Чем плох обычный массив?

```
const N = 100 ;  
var A: array[1..N] of integer ;
```

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 (*FreePascal*)

## Объявление:

```
var A: array of integer;
```

размер  
не указан

## Выделение памяти:

```
read(N);  
SetLength(A, N);
```



Какие индексы?

[0..N-1]

установить  
длину

## Чтение данных:

```
for i:=0 to N-1 do read(A[i]);
```

```
for i:=0 to High(A) do read(A[i]);
```

наибольший  
индекс



Массив знает свой размер!

# Динамические массивы (*FreePascal*)

---

Определение длины:

```
writeln( Length(A) );
```

Освобождение памяти:

```
SetLength(A, 0);
```

Динамические матрицы:

```
var A: array of array of integer;
```

```
SetLength(A, 4, 3);
```

```
writeln( High(A) );
```

```
writeln( High(A[0]) );
```

A

3

2


# Динамические массивы (*FreePascal*)

---

В подпрограммах:

```
procedure printArray (X: array of integer);  
begin  
  for i:= 0 to High (X) do  
    write (X[i], ' ' )  
end;
```

# 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?

Расширение по 1 элементу:

```
read(x) ;  
while x <> 0 do begin  
    SetLength(A, Length(A) + 1) ;  
    A[High(A)] := x ;  
    read(x)  
end ;
```



Что плохо?

# Расширение массива

## Расширение по 10 элементов:

```
N := 0; { счётчик элементов }
read(x);
while x <> 0 do begin
  if N > High(A) then
    SetLength(A, Length(A) + 10);
  A[N] := x;
  N := N + 1;
  read(x)
end;
```



Зачем нужен счётчик?

# Как это работает?

## Эксперимент:

```
SetLength (A, 100) ;
write (sizeof (A) ) ;
write (100* sizeof (integer) ) ;
```

4

200



A – это указатель!



размер массива

```
type TArray = record
    data: array of integer;
    size: integer
end;
```



Как записать в файл?

# Как это работает?

```
var A: array of array of integer;
```

**!** Динамическая матрица – это массив указателей!

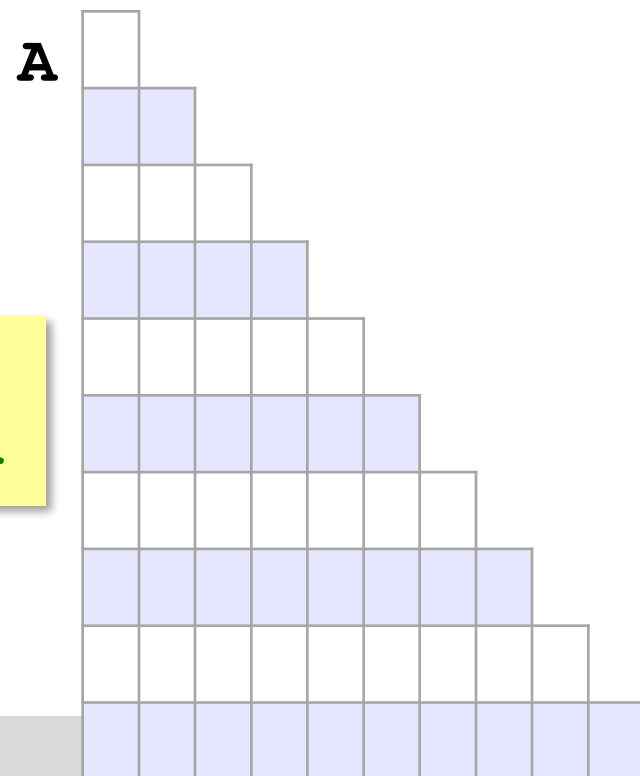
```
SetLength(A, 10);
```

ВЫДЕЛИТЬ МАССИВ  
указателей

Строки разной длины:

```
for i:=0 to High(A) do  
  SetLength(A[i], i+1);
```

```
writeln(Length(A[0])); { 1 }  
writeln(Length(A[9])); { 10 }
```



# Алгоритмизация и программирование

## § 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)

---

```
нц 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 счётчик слова
все
кц
```

# Хранение данных

## Пара «слово-счётчик»:

```
type
  TPair = record
    word: string;      { слово }
    count: integer     { счётчик }
  end;
```

## Список таких пар:

```
type
  TWordList = record
    data: array of TPair;
    size: integer
  end;
```

ДИНАМИЧЕСКИЙ  
МАССИВ

КОЛИЧЕСТВО СЛОВ  
В СПИСКЕ

# Хранение данных

## Переменная-список:

```
var L: TWordList;
```

## Начальные значения:

```
SetLength (L.data, 0);  
L.size := 0;
```

## Вывод результата:

```
Assign (F, 'output.dat');  
Rewrite (F);  
for p:=0 to L.size-1 do  
    writeln (F, L.data [p].word, ': ',  
            L.data [p].count);  
Close (F);
```

```
автомат: 1  
ананас: 12  
...
```



Как объявить p?

# ОСНОВНОЙ ЦИКЛ

```
while not Eof(F)      { пока не конец файла }
do begin
  readln(F, s);      { читаем слово }
  p:= Find(L, s);    { ищем его в словаре}
  if p >= 0 then     { если нашли... }
    Inc(L.data[p].count)
                        { ...увеличить счётчик }
  else begin        { иначе... }
    p:= FindPlace(L, s); { найти место }
    InsertWord(L, p, s); { вставить в список }
  end
end;
end;
```



Как объявить s?

# Поиск слова

```
function Find(L: TWordList;  
             word: string): integer;  
var i: integer;  
begin  
    Find := -1;  
    for i := 0 to L.size-1 do  
        if L.data[i].word = word then begin  
            Find := i;  
            break  
        end  
    end  
end;
```

вернуть -1, если  
нет в списке

вернуть номер  
элемента в списке

ВЫЙТИ ИЗ ЦИКЛА

# Поиск места вставки

```
function FindPlace (L: TWordList;  
                    word: string): integer;  
var i, p: integer;  
begin  
    p := -1;           -1: не найдено  
    for i := 0 to L.size-1 do  
        if L.data[i].word > word then begin  
            p := i;   запомнить номер  
            break  
        end;  
    if p < 0 then p := L.size;  
    FindPlace := p  
end;
```

первое слово  
«больше» заданного

если не найдено,  
вставить в конец

# Вставка слова

1	автомат	1	автомат	1	1
2	ананас	12	ананас	12	2
	...		...		
k-1	дар	3	дар	3	k-1
дереву	дом	15	дереву	1	k
k+1	дорога	5	дом	15	k+1
	...		дорога	5	
	ящерица	1	...		
			ящерица	1	L.size-1

Сдвиг вниз: с последнего

```
for i:=L.size-1 downto k+1 do
  L.data[i] := L.data[i-1];
```



# Вставка слова

СПИСОК МЕНЯЕТСЯ

```
procedure InsertWord(var L: TWordList;  
                    k: integer;  
                    word: string);  
  
var i: integer;  
begin  
    IncSize(L);  
    for i:=L.size-1 downto k+1 do  
        L.data[i] := L.data[i-1];  
    L.data[k].word := word; { записать слово }  
    L.data[k].count := 1   { встретилось 1 раз }  
end;
```

увеличить размер,  
если нужно

СДВИГ ВНИЗ

# Расширение списка

СПИСОК МЕНЯЕТСЯ

```
procedure IncSize (var L: TWordList) ;  
begin  
    Inc(L.size) ;  
    if L.size > Length(L.data) then  
        SetLength(L.data, Length(L.data) + 10)  
end;
```

если новый размер больше  
размера массива

добавить 10  
элементов

# Вся программа

```
program AlphaList;  
  { объявления типов TPair и TWordList }  
var F: text;  
    w: string;  
    L: TWordList;  
    p: integer;  
  { процедуры и функции }  
begin  
  SetLength(L.data, 0);  
  L.size := 0;  
  Assign(F, 'input.dat');  
  Reset(F);  
  { основной цикл: составление списка слов }  
  Close(F);  
  { вывод результата в файл output.dat }  
end.
```

# Модули

```
program AlphaList;  
  { процедура 1 }  
  { процедура 2 }  
  { процедура 3 }  
  { процедура 4 }  
  { ... }  
  { процедура N-1 }  
  { процедура N }  
begin  
  { программа }  
end.
```

```
program AlphaList;  
uses WordList;  
begin  
  { программа }  
end.
```

```
unit WordList;  
  { процедура 1 }  
  { процедура 2 }  
  { процедура 3 }  
  { процедура 4 }  
  { ... }  
  { процедура N-1 }  
  { процедура N }  
end.
```



- проще разбираться («разделяй и властвуй»)
- модуль пишет другой программист

# Структура модуля

```
unit WordList;  
interface  
    ...  
implementation  
    ...  
end.
```

«**интерфейс**» –  
общедоступная информация:  
• объявление типов данных  
• объявления процедур и  
функций

«**реализация**» – внутренняя  
информация модуля:  
• код процедур и функций  
• внутренние данные

# Модуль WordList

```
unit WordList;  
interface  
  { объявления типов TPair и TWordList }  
function Find(L: TWordList;  
             word: string): integer;  
function FindPlace(L: TWordList;  
                  word: string): integer;  
procedure InsertWord(var L: TWordList;  
                     k: integer; word: string);  
  
implementation  
  { код процедур и функций }  
end.
```



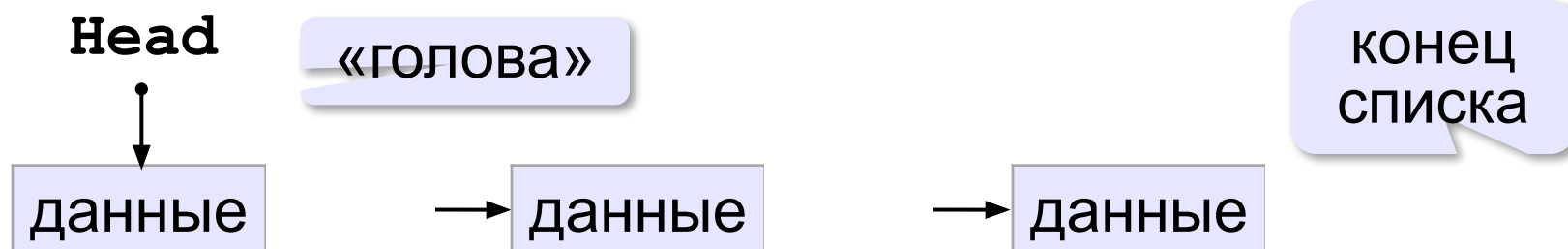
# Подключение модуля

```
program AlphaList;  
uses WordList;    { подключение модуля }  
var F: text;  
    s: string;  
    L: TWordList;  
    p: integer;  
begin  
  { тело основной программы }  
end.
```

ТИП ИЗВЕСТЕН ИЗ  
ИНТЕРФЕЙСА МОДУЛЯ

МОЖНО ИСПОЛЬЗОВАТЬ  
ВСЕ ПРОЦЕДУРЫ,  
ОБЪЯВЛЕННЫЕ В  
ИНТЕРФЕЙСЕ МОДУЛЯ

# СВЯЗНЫЕ СПИСКИ



 узлы могут размещаться в разных местах в памяти

 только последовательный доступ

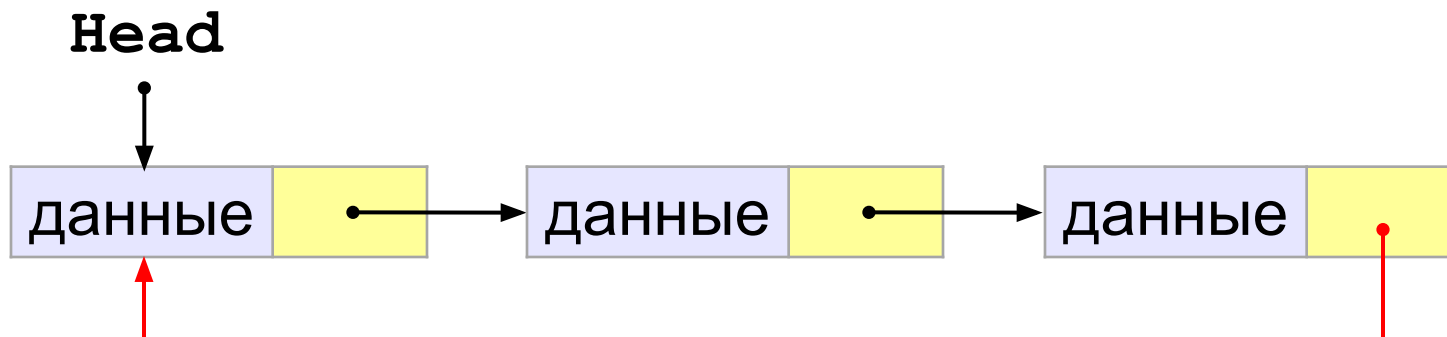
## Рекурсивное определение:

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





# СВЯЗНЫЕ СПИСКИ

## Циклический список:



## Двусвязный список:



-  обход в двух направлениях
-  сложнее вставка и удаление

# Связные списки: структуры данных

## Односвязный список:

```
type
  PNode = ^TNode;
  TNode = record
    data: integer;
    next: PNode
  end;
```

указатель

ссылка на  
следующий узел

## Двусвязный список:

```
type
  PNode = ^TNode;
  TNode = record
    data: integer;
    prev, next: PNode
  end;
```

ссылки на  
предыдущий  
(*previous*) и  
следующий узлы

# Алгоритмизация и программирование

## § 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*.



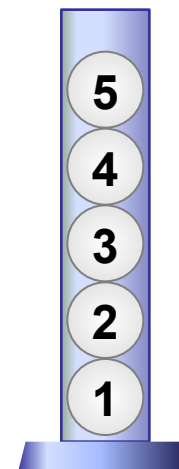
## Системный стек:

- 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.

```
нц пока файл не пуст  
  прочитать x  
  добавить x в стек  
кц
```



```
нц пока стек не пуст  
  вытолкнуть число из стека в x  
  записать x в файл  
кц
```

# Использование динамического массива

```
type TStack = record
    data: array of integer;
    size: integer
end;
```

«Втолкнуть» **x** в стек:

ИЗМЕНЯЕТСЯ

```
procedure Push(var S: TStack; x: integer);
begin
    if S.size > High(S.data) then
        SetLength(S.data, Length(S.data) + 10);
    S.data[S.size] := x;
    S.size := S.size + 1
end;
```

# Использование динамического массива

---

«Вытолкнуть» из стека в **x** :

ИЗМЕНЯЕТСЯ

```
function Pop (var S:TStack) : integer;  
begin  
  S.size := S.size - 1;  
  Pop := S.data[S.size]  
end;
```

Инициализация стека :

```
procedure InitStack (var S: TStack);  
begin  
  SetLength (S.data, 0);  
  S.size := 0  
end;
```

# Использование динамического массива

---

## Заполнение стека:

```
var F: text;
```

```
InitStack (S) ;  
while not Eof (F) do begin  
    read (F, x) ;  
    Push (S, x)  
end;
```

## Вывод результата в файл:

```
for i:=0 to S.size-1 do begin  
    x:= Pop (S) ;  
    writeln (F, x)  
end;
```



# 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



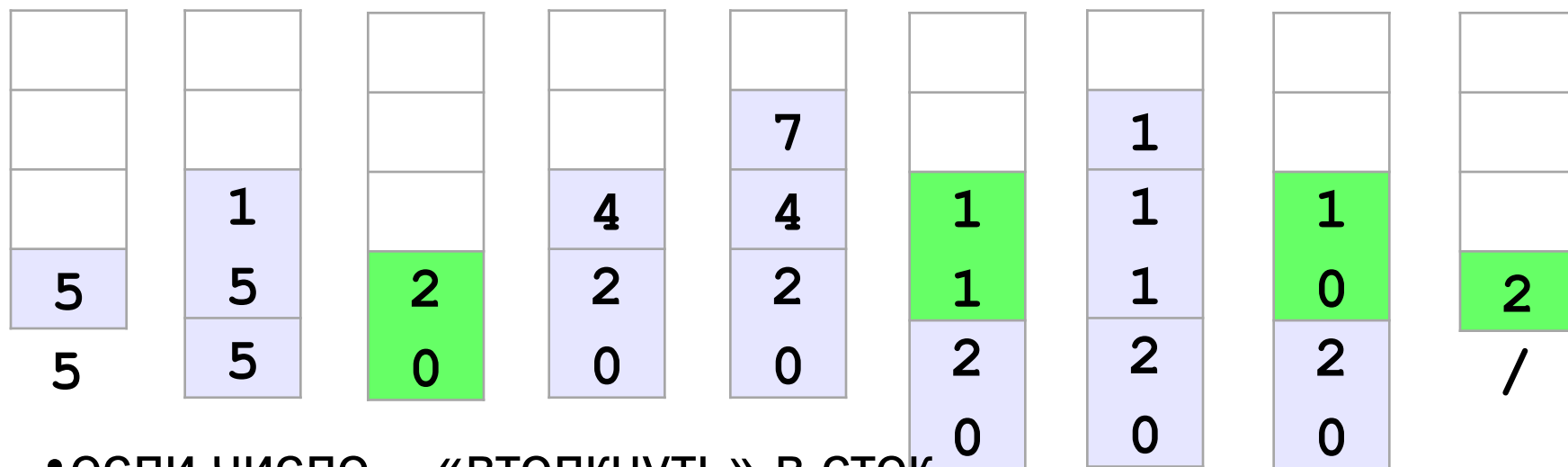
- не нужны скобки
- вычисляем с начала



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, правильно ли расставлены скобки.

$() [ \{ () [] \} ]$  ✓     $[ () ]$  ✗     $[ () ] \}$  ✗     $) ($  ✗     $( [ ] ]$  ✗

**Для одного типа скобок:**

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



Когда выражение правильное?

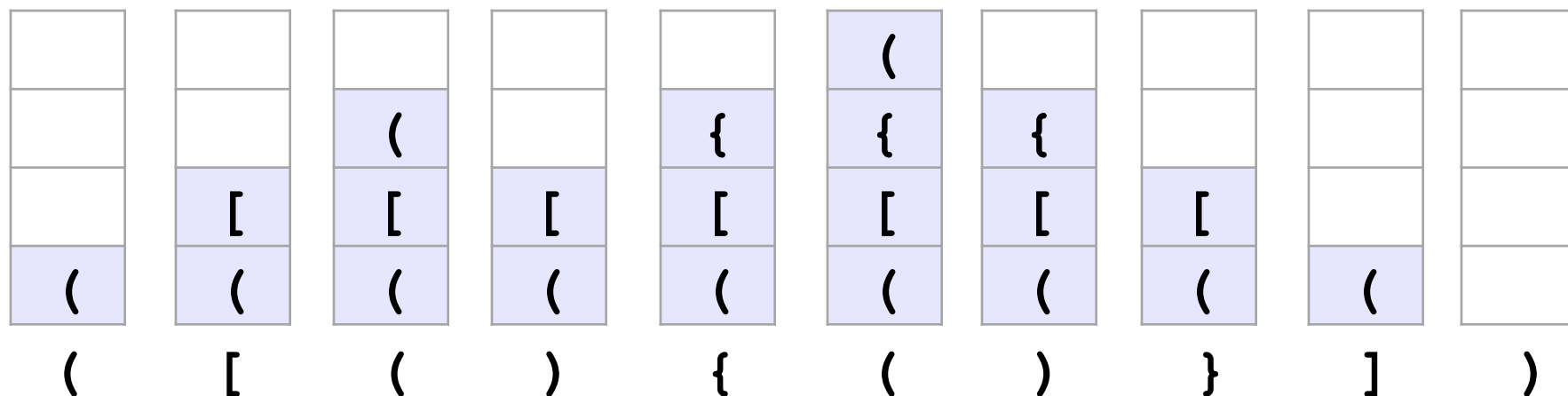
- счётчик всегда  $\geq 0$
- в конце счётчик = 0

$( \{ [ ] \} ]$



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)

---

## Модель стека:

```
type TStack = record
    data: array of char;
    size: integer
end;
```

## Стек пуст:

```
function isEmpty (S: TStack): boolean;
begin
    isEmpty := (S.size = 0)
end;
```

# Скобочные выражения (стек)

## Константы и переменные:

```
const L = '([{'; { открывающие скобки }  
      R = ')]}'; { закрывающие скобки }  
var S: TStack;  
    p, i: integer;  
    str: string; { рабочая строка }  
    err: boolean; { признак ошибки }  
    c: char;
```

## Вывод результата:

```
if not err then  
    writeln('Выражение правильное.')else writeln('Выражение неправильное.');
```

# Скобочные выражения (стек)

```
for i:=1 to Length(str) do begin
```

```
  p:=Pos(str[i], L);
```

открывающую  
скобку в стек

```
  if p > 0 then Push(S, str[i]);
```

```
  p:=Pos(str[i], R);
```

если закрывающая  
скобка...

```
  if p > 0 then begin
```

```
    if isEmpty(S) then err:=True
```

```
  else begin
```

```
    c:=Pop(S);
```

если не та скобка...

```
    if p <> Pos(c, L) then err:=True
```

```
  end;
```

```
  if err then break
```

```
end
```

```
end;
```



# 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  $(x_0, y_0)$ .

	1	2	3	4	5
1	0	1	0	1	1
2	1	1	1	2	2
3	0	1	0	2	2
4	3	3	1	2	2
5	0	1	1	0	0

$(2, 1)$   
→

	1	2	3	4	5
1	0	2	0	1	1
2	2	2	2	2	2
3	0	2	0	2	2
4	3	3	1	2	2
5	0	1	1	0	0

## Заливка: использование очереди

```
добавить в очередь точку  $(x_0, y_0)$   
запомнить цвет начальной точки  
нц пока очередь не пуста  
  взять из очереди точку  $(x, y)$   
  если  $A[y, x] = \text{цвету начальной точки}$  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  $(x, y+1)$   
все  
кц
```

# Очередь (динамический массив)

---

```
type TPoint = record
    x, y: integer
end;
TQueue = record
    data: array of TPoint;
    size: integer
end;
```

## Построение структуры «точка»:

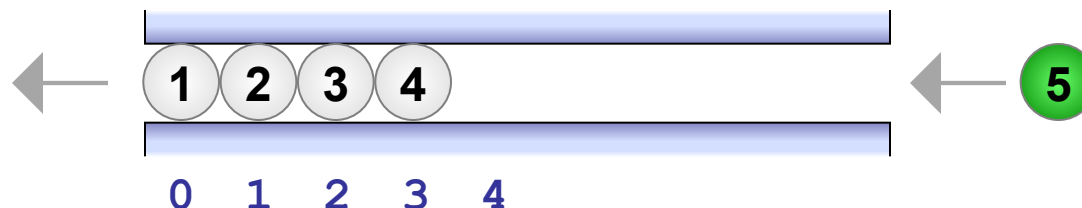
```
function Point(x, y: integer): TPoint;
begin
    Point.x := x;
    Point.y := y
end;
```

# Очередь (динамический массив)

Добавить точку в очередь:

```
procedure Put (var Q: TQueue; pt: TPoint);  
begin  
  if Q.size > High(Q.data) then  
    SetLength(Q.data,  
              Length(Q.data) + 10);  
  Q.data[Q.size] := pt;  
  Q.size := Q.size + 1;  
end;
```

расширить,  
если нужно



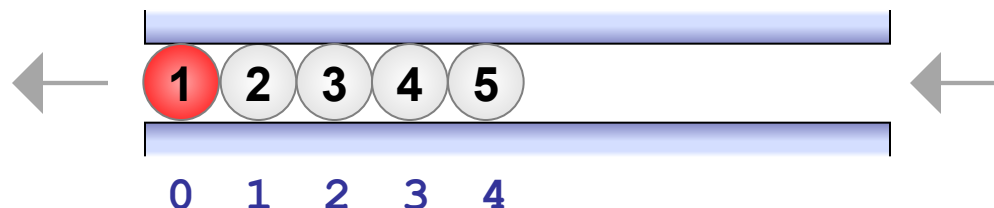
# Очередь (динамический массив)

Получить первую точку в очереди:

```
function Get(var Q:TQueue): TPoint;  
var i: integer;  
begin  
  Get:= Q.data[0];  
  Q.size:= Q.size - 1;  
  for i:=0 to Q.Size - 1 do  
    Q.data[i]:= Q.data[i+1];  
end;
```

уменьшить  
размер

продвинуть  
оставшиеся  
элементы



? Что плохо?

# Заливка

---

## Константы и переменные:

```
const XMAX = 5; YMAX = 5;  
      NEW_COLOR = 2;  
var Q: TQueue;  
     x0, y0, color: integer;  
     A: array[1..YMAX,1..XMAX] of integer;  
     pt: TPoint;
```

## Начало программы:

```
{ заполнить матрицу A }  
x0 := 2; y0 := 1;      { начать заливку отсюда }  
color := A[y0, x0];   { цвет начальной точки }  
Put(Q, Point(x0, y0));
```

# Заливка (основной цикл)

пока очередь не пуста

```
while not isEmpty(Q) do begin
  pt := Get(Q); { взять точку из очереди }
  if A[pt.y, pt.x] = color then begin
    A[pt.y, pt.x] := NEW_COLOR;
    if pt.x > 1 then
      Put(Q, Point(pt.x-1, pt.y));
    if pt.x < XMAX then
      Put(Q, Point(pt.x+1, pt.y));
    if pt.y > 1 then
      Put(Q, Point(pt.x, pt.y-1));
    if pt.y < YMAX then
      Put(Q, Point(pt.x, pt.y+1))
  end
end;
```



Что можно улучшить?



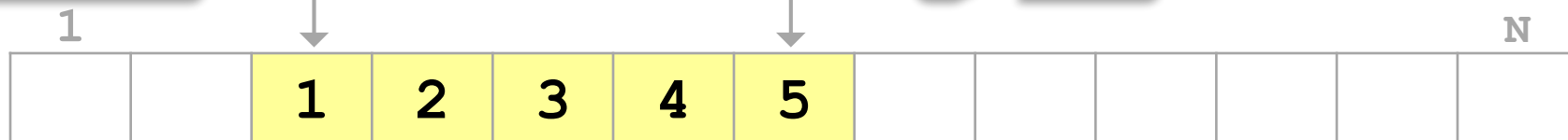
# Очередь: статический массив

ГОЛОВА

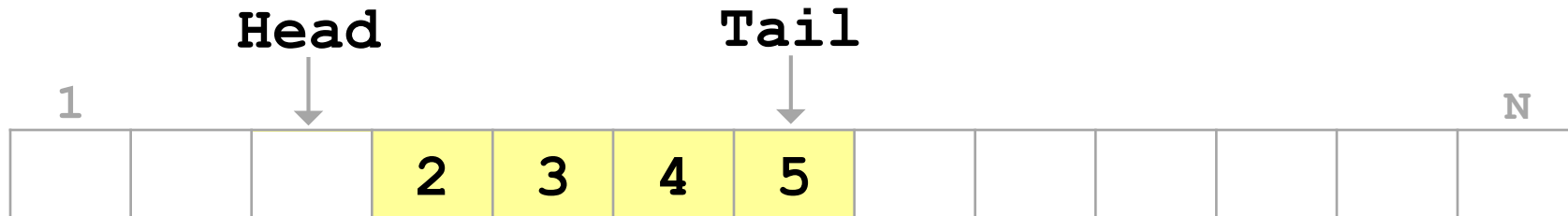
Head

Tail

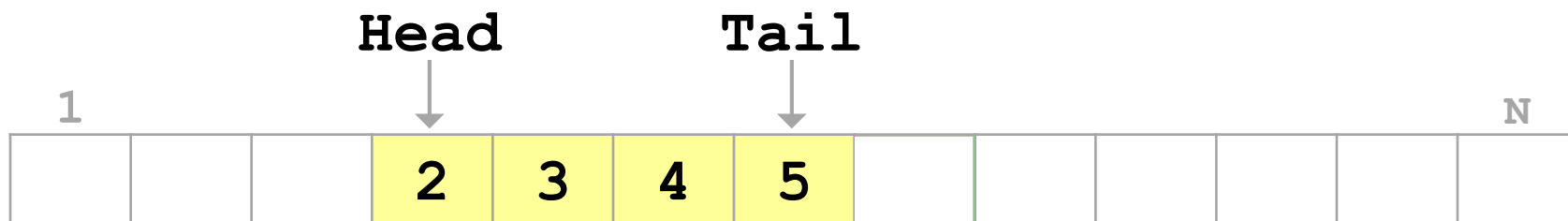
ХВОСТ



## Удаление элемента:



## Добавление элемента:



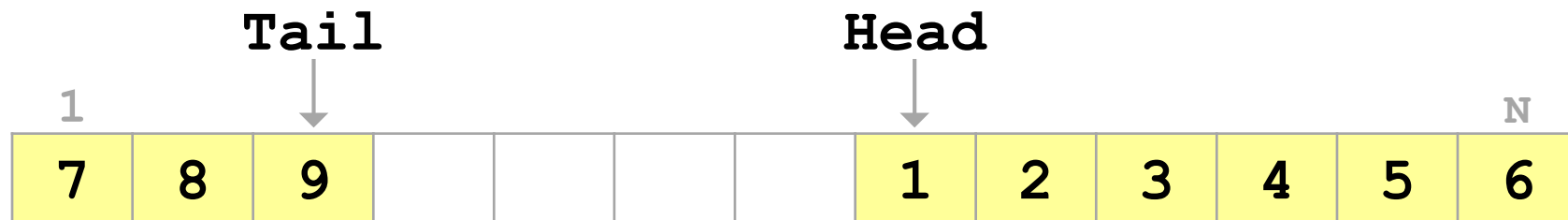
■ не двигаем элементы



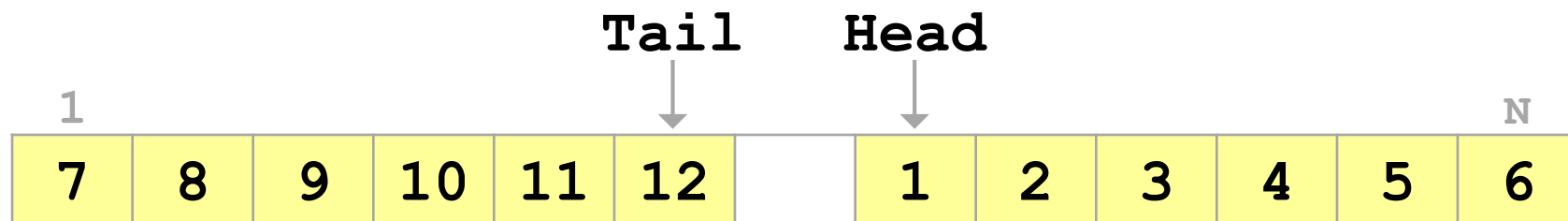
■ нужно знать размер

# Очередь: статический массив

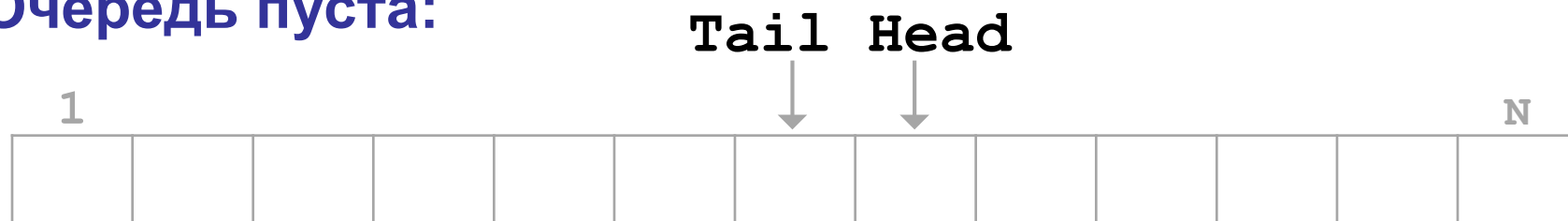
## Замыкание в кольцо:



## Очередь заполнена:



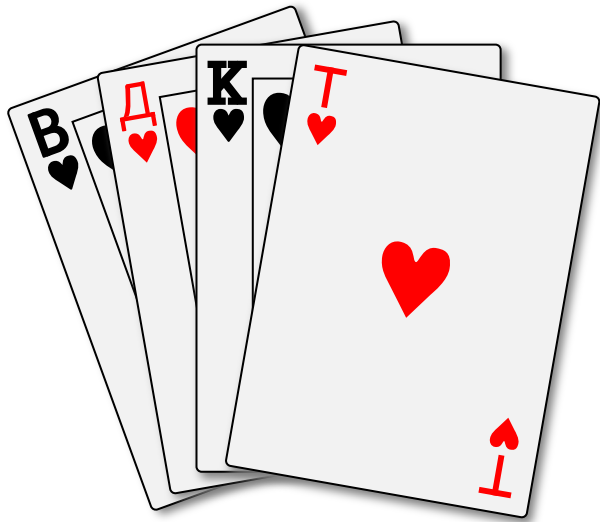
## Очередь пуста:



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:

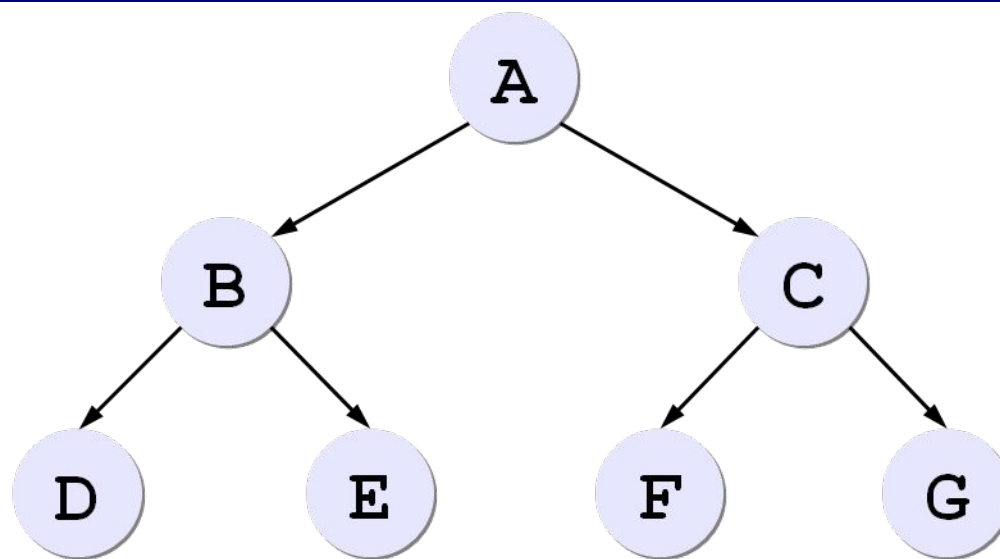
- статический массив (кольцо)
- динамический массив
- СВЯЗНЫЙ СПИСОК

# Алгоритмизация и программирование

## § 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:

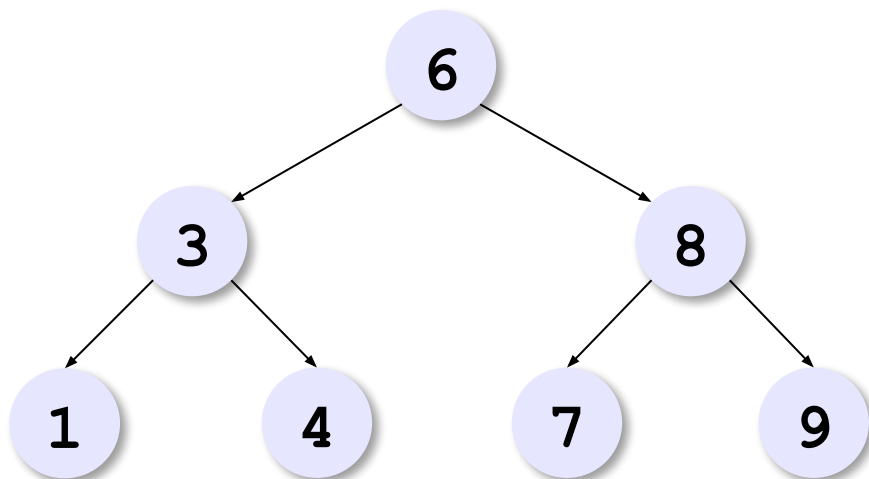
- 1) пустая структура – это **двоичное дерево**
- 2) двоичное дерево – это корень и **два** связанных с ним отдельных двоичных дерева («левое» и «правое» поддеревья)

## 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  $\Leftrightarrow$  «посетить» все узлы по одному разу.

$\Rightarrow$  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

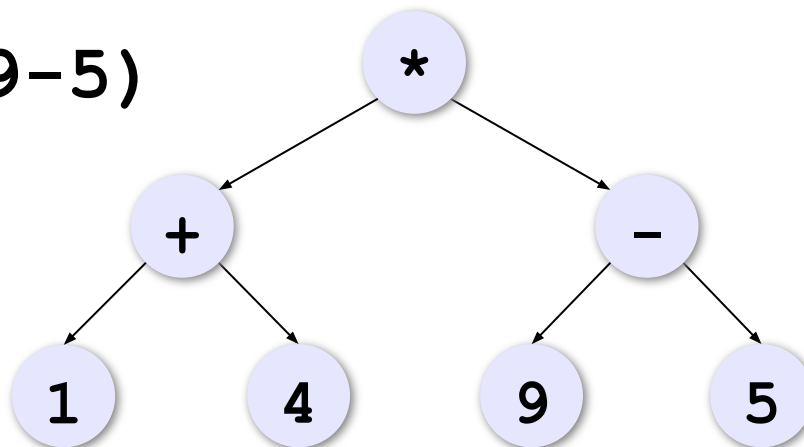
**ЛПК** – «**левый-правый-корень**» (в обратном порядке):

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

нц пока стек не пуст

$V :=$  выбрать узел с вершины стека

посетить узел  $V$

если у узла  $V$  есть правый сын то

    добавить в стек правого сына  $V$

все

если у узла  $V$  есть левый сын то

    добавить в стек левого сына  $V$

все

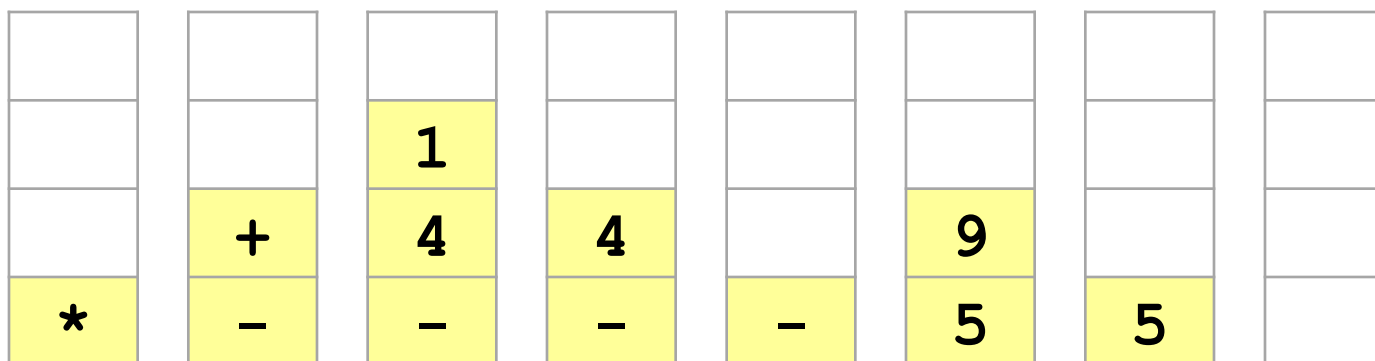
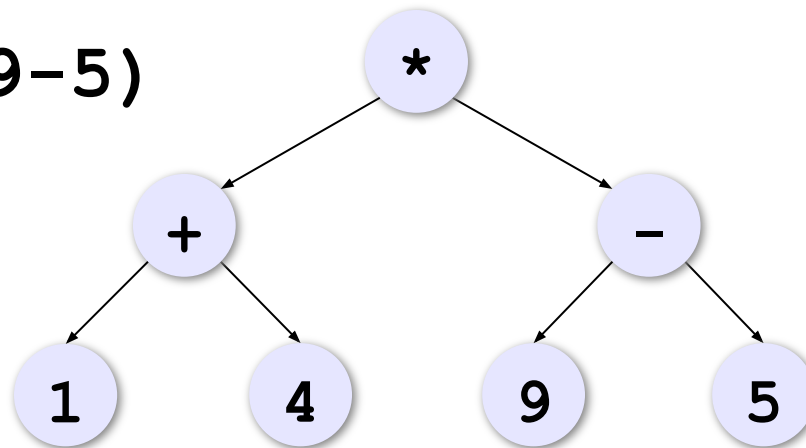
кц



Почему сначала добавить правого сына?

# Обход КЛП – обход «в глубину»

$(1+4) * (9-5)$



**\* + 1 4 - 9 5**

## Обход «в ширину»

записать в очередь корень дерева

нц пока очередь не пуста

$V :=$  выбрать узел из очереди

    посетить узел  $V$

    если у узла  $V$  есть левый сын то

        добавить в очередь левого сына  $V$

    все

    если у узла  $V$  есть правый сын то

        добавить в очередь правого сына  $V$

    все

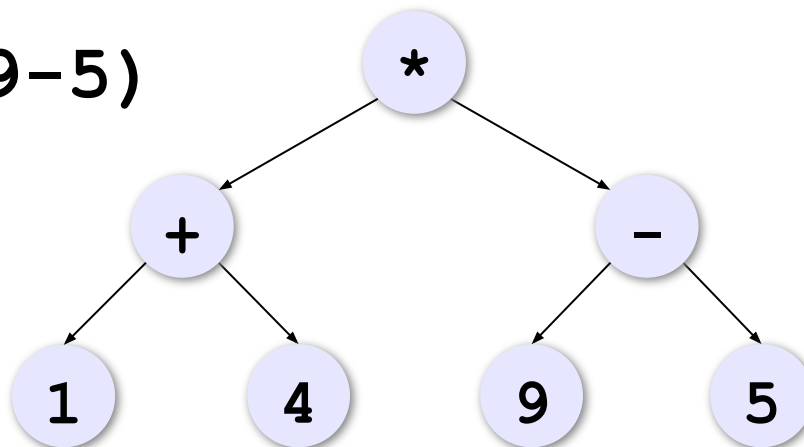
кц



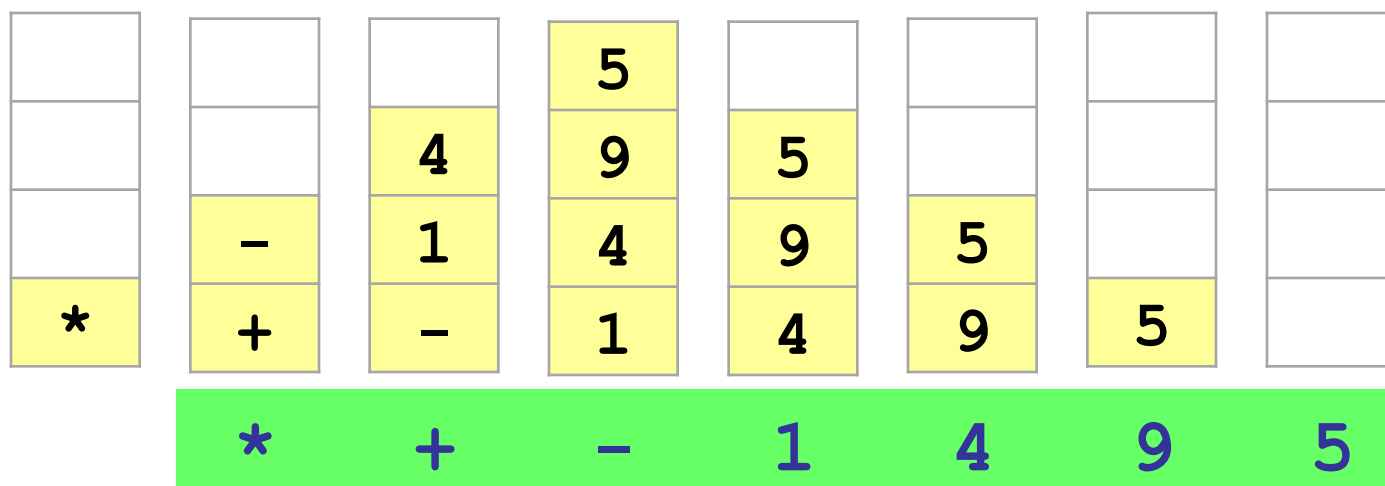
Почему сначала добавить левого сына?

# Обход «в ширину»

$$(1+4) * (9-5)$$



голова  
очереди



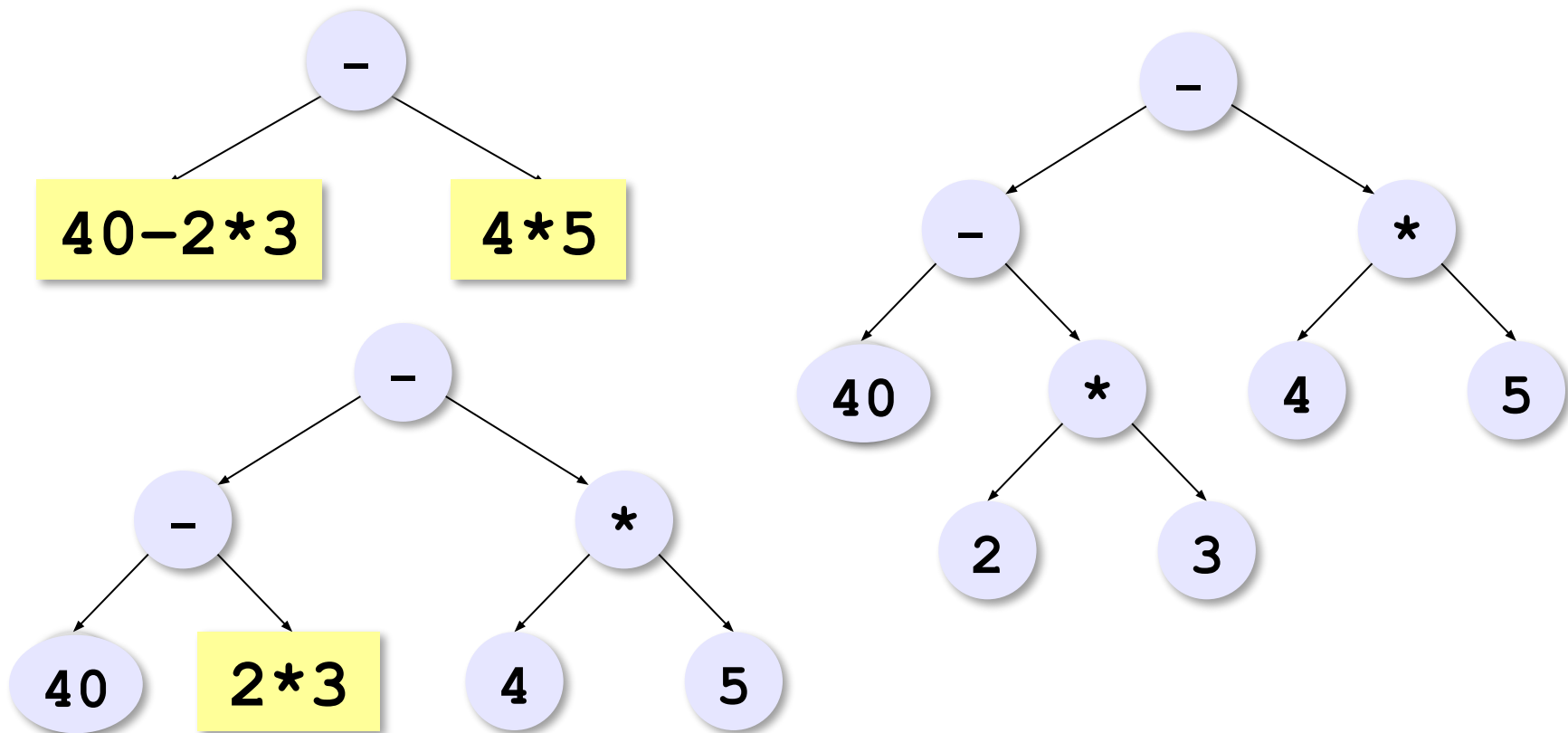
# Вычисление арифметических выражений

$$40 - 2 * 3 - 4 * 5$$



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**

**все**

**поместить операцию в корень дерева**

**построить левое поддерев**

**построить правое поддерев**



**Рекурсия!**

# Вычисление арифметических выражений

## Вычисление по дереву:

$n1 :=$  значение левого поддерева

$n2 :=$  значение правого поддерева

**результат := операция** ( $n1$ ,  $n2$ )

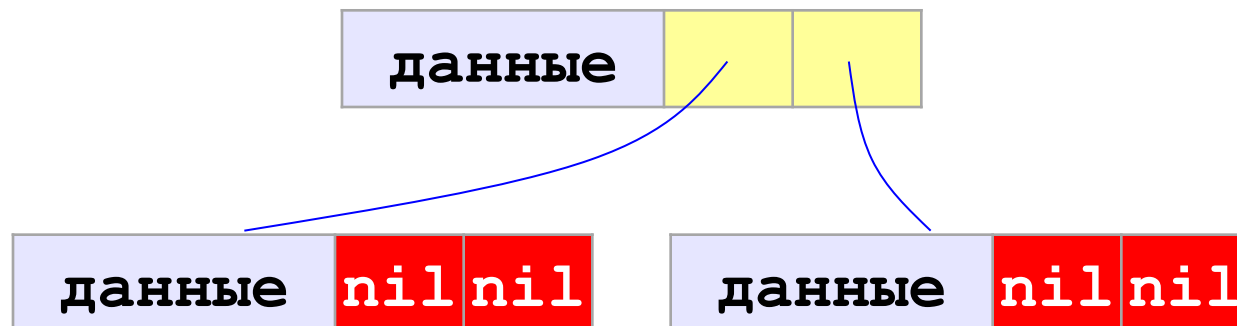


Рекурсия!



# Использование связанных структур

Дерево – **нелинейная** структура  $\Rightarrow$  динамический массив неудобен!



ссылка вперёд

type

```
PNode = ^TNode; { указатель на узел }
```

```
TNode = record { узел дерева }
```

```
  data: string[20];
```

```
  left, right: PNode { ссылки на сыновей }
```

```
end;
```

# Работа с памятью

---

```
var p: PNode; { указатель на узел }
```

Выделить память для узла:

```
New (p) ;
```

Обращение к новому узлу (по указателю):

```
p^.data := s;  
p^.left := nil;  
p^.right := nil;
```

Освобождение памяти:

```
Dispose (p) ;
```

# Основная программа

```
var T: PNode; { ссылка на дерево }
    s: string; { строка с выражением }
{ процедуры и функции }
begin
  readln(s);
  T:= Tree(s); { построить дерево }
  writeln('Результат: ',
         Calc(T)); { вычисление }
end.
```



Нужно построить **Tree** и **Calc**!

# Построение дерева

```
function Tree(s: string): PNode;  
var k: integer;  
begin  
  New(Tree);           { выделить память }  
  k:=LastOp(s); { вернет 0, если нет операции }  
  if k = 0 then begin { создать лист }  
    Tree^.data:=s;  
    Tree^.left:=nil;  
    Tree^.right:=nil  
  end  
  else begin { создать узел-операцию }  
    Tree^.data:=s[k];  
    Tree^.left:= Tree(Copy(s, 1, k-1));  
    Tree^.right:= Tree(Copy(s, k+1, Length(s)-k))  
  end  
end;  
end;
```

вернёт адрес  
нового дерева



Рекурсия!

# Вычисление по дереву

```
function Calc(Tree: PNode) : integer;  
var n1, n2, res: integer;  
begin  
  if Tree^.left = nil then  
    Val(Tree^.data, Calc, res)  
  else begin  
    n1 := Calc(Tree^.left);  
    n2 := Calc(Tree^.right);  
    case Tree^.data[1] of  
      '+' : Calc := n1 + n2;  
      '-' : Calc := n1 - n2;  
      '*' : Calc := n1 * n2;  
      '/' : Calc := n1 div n2;  
    else Calc := MaxInt  
    end  
  end  
end;  
end;
```



Рекурсия!

# Приоритет операции

```
function Priority(op: char) : integer;  
begin  
  case op of  
    '+', '-': Priority := 1;  
    '*', '/': Priority := 2  
    else     Priority := 100  
  end  
end;  
end;
```

# Последняя выполняемая операция

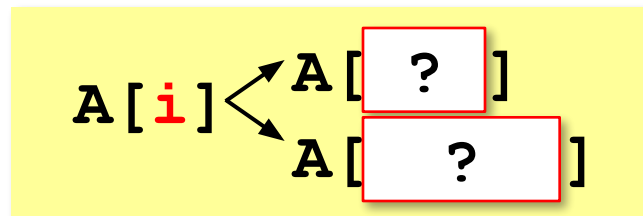
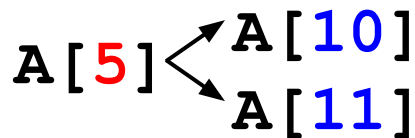
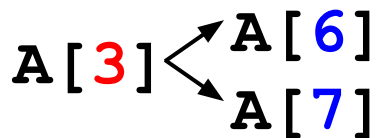
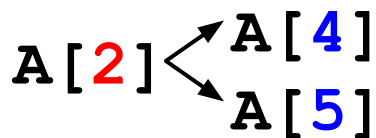
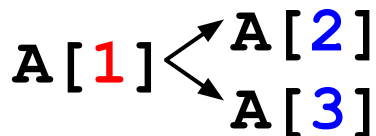
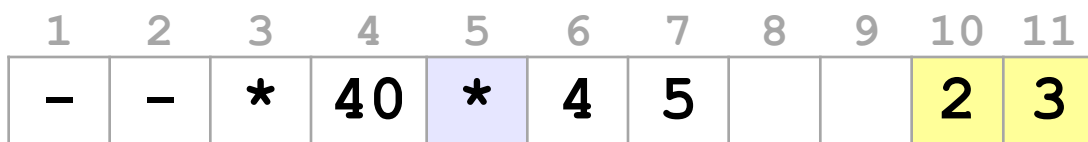
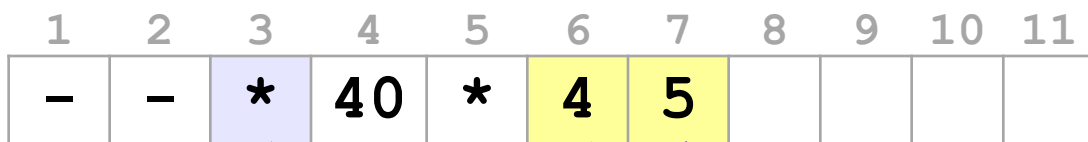
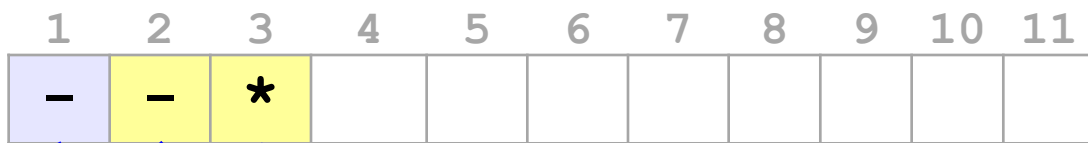
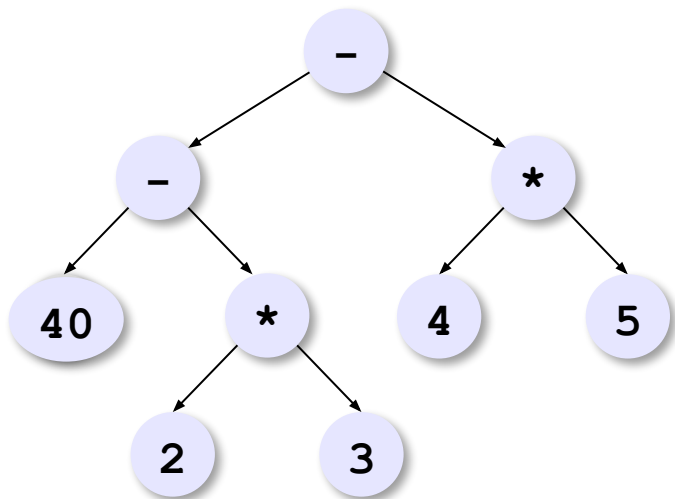
```
function LastOp (s: string): integer;  
var i, minPrt: integer;  
begin  
  minPrt := 50; { любое между 2 и 100 }  
  LastOp := 0; { если нет операции }  
  for i := 1 to Length(s) do  
    if Priority(s[i]) <= minPrt then begin  
      minPrt := Priority(s[i]);  
      LastOp := i  
    end  
  end;  
end;
```

вернёт номер  
СИМВОЛА



Почему <=?

# Двоичное дерево в массиве



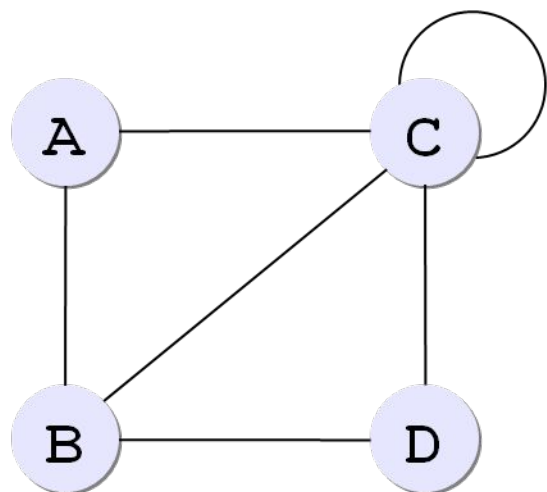


# Алгоритмизация и программирование

## § 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

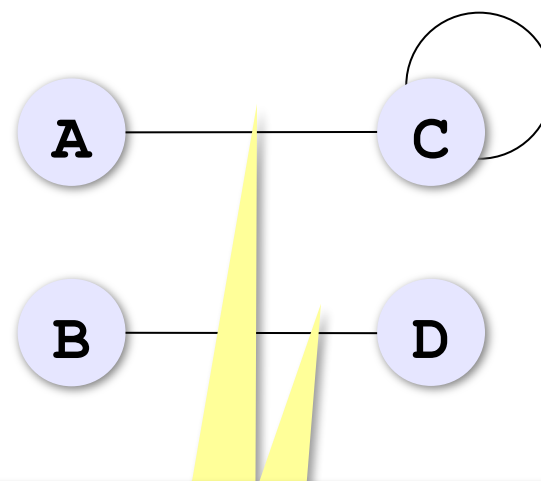
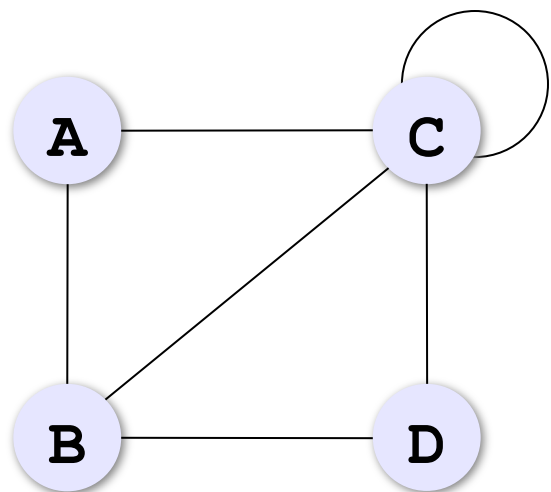
## Список смежности:

( **A** (B, C) ,  
**B** (A, C, D) ,  
**C** (A, B, C, D) ,  
**D** (B, C) )

петля

# Связность графа

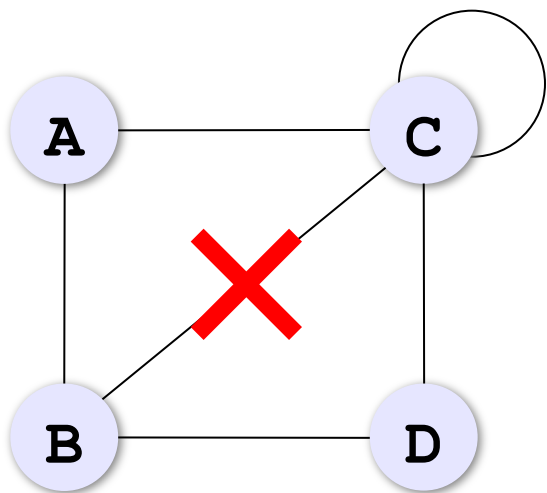
**Связный граф** – это граф, между любыми вершинами которого существует путь.



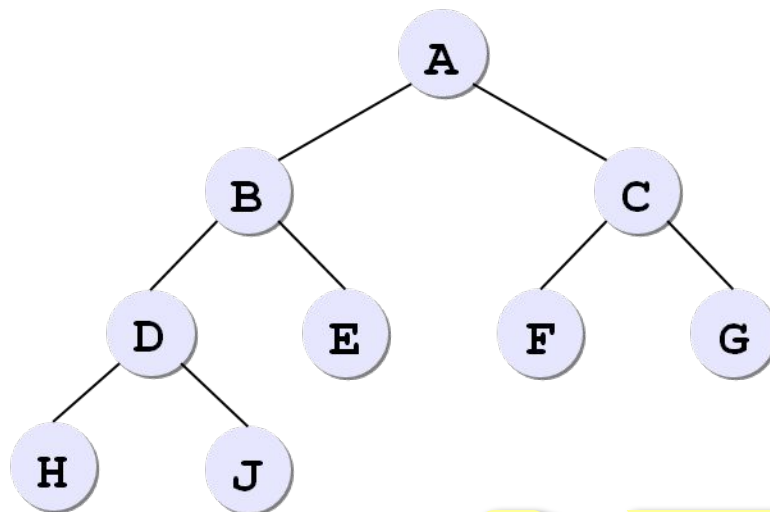
**КОМПОНЕНТЫ СВЯЗНОСТИ**

# Дерево – это граф?

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

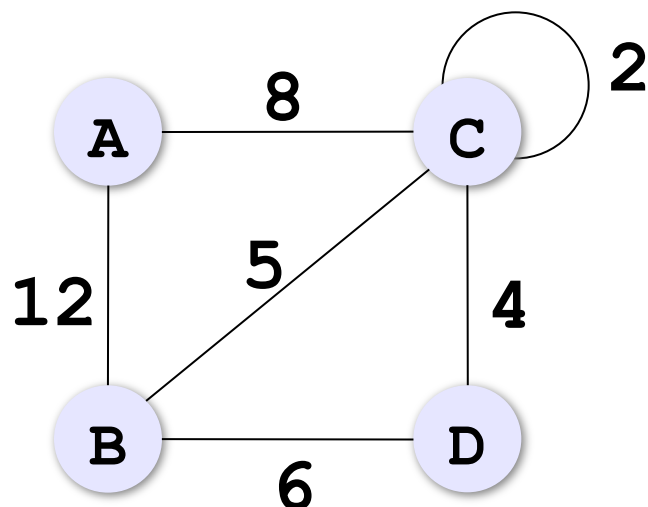


ABC ABDC  
BCD CCC...



дерево

# Взвешенные графы



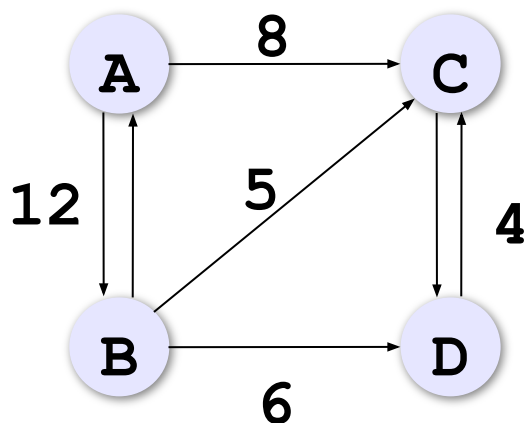
вес ребра

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	



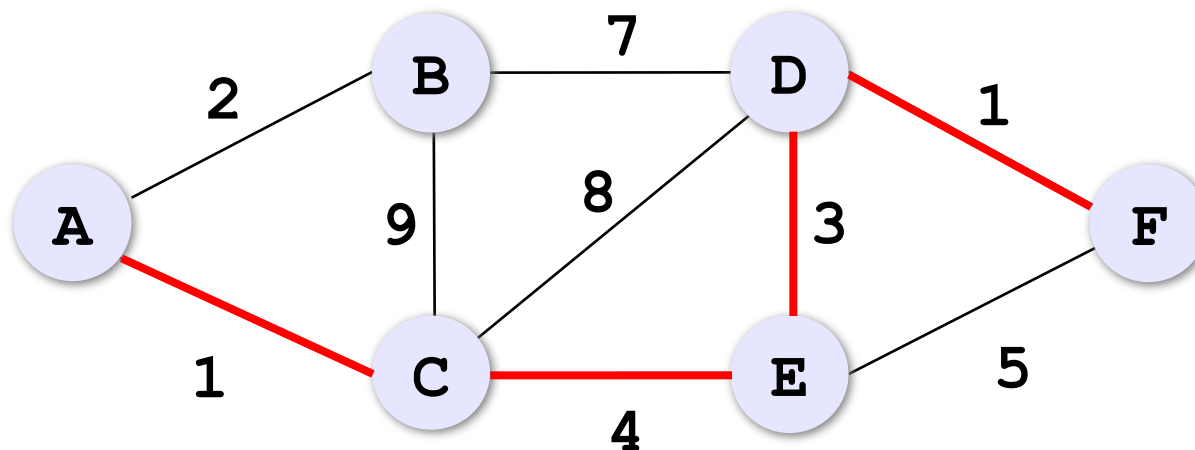
Весовая матрица может быть несимметрична!

# Жадные алгоритмы



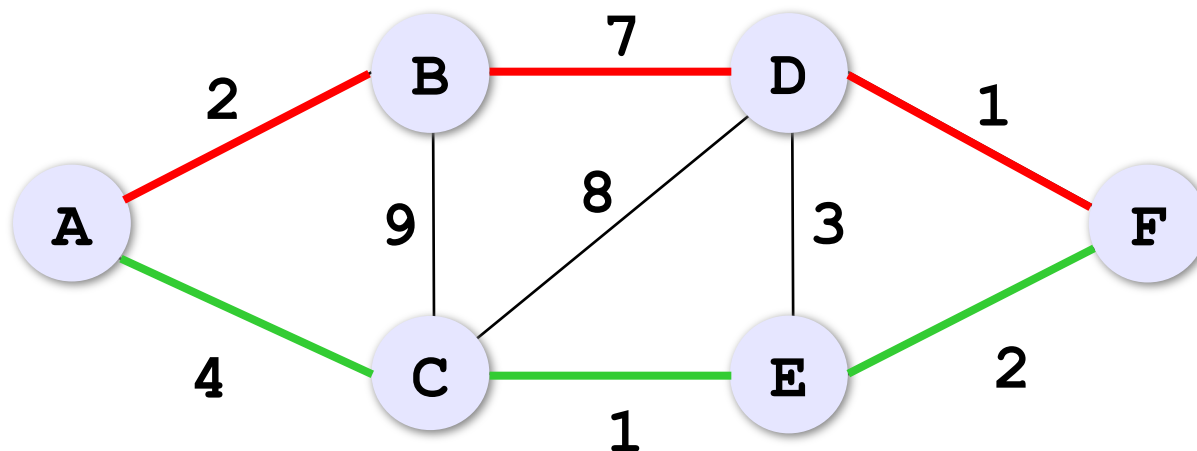
**Жадный алгоритм** – это многошаговый алгоритм, в котором на каждом шаге принимается решение, лучшее в данный момент.

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



# Жадные алгоритмы

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



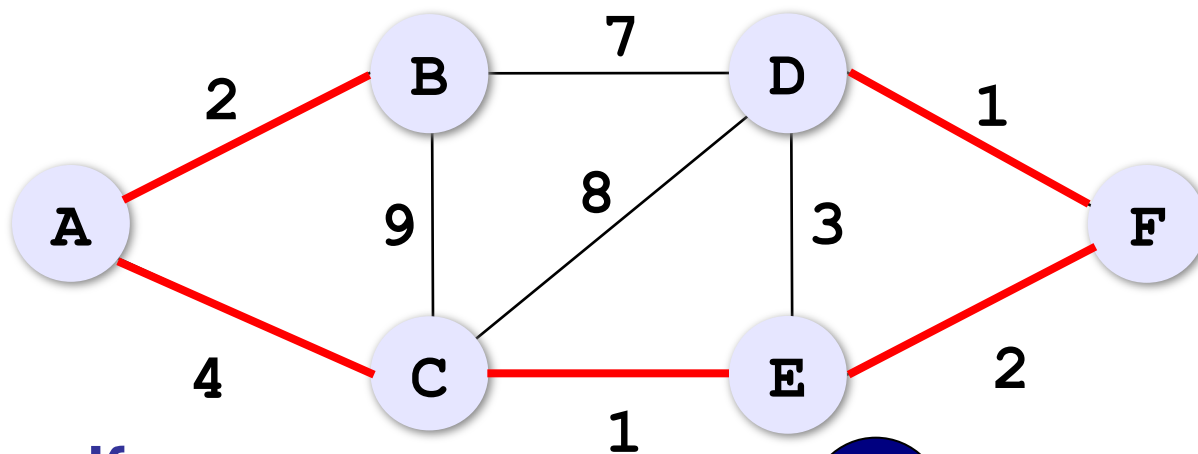
**?** Это лучший маршрут?

**!** 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**)



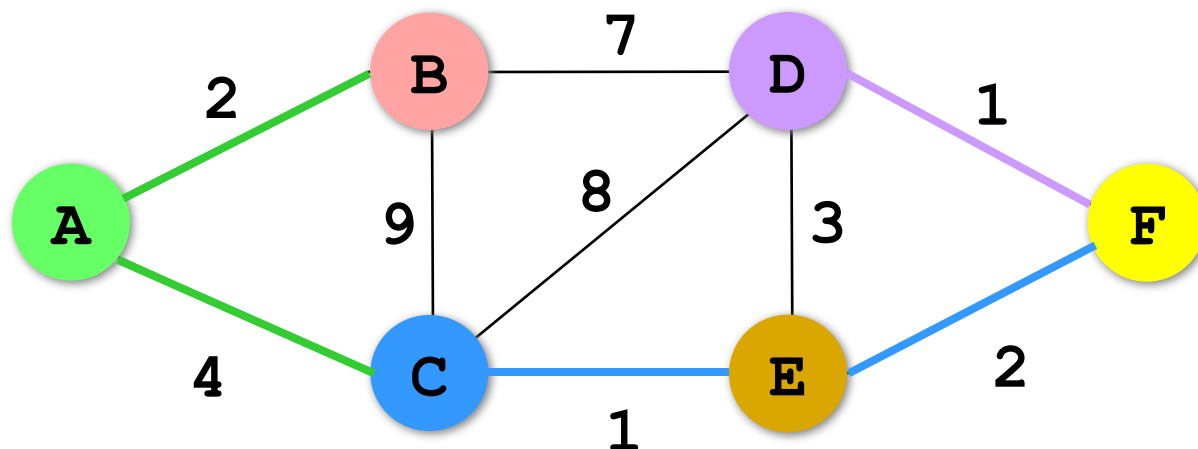
## Алгоритм Крускала:

- 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:=1 to N do 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 N = 6;  
var      { весовая матрица }  
    W: array[1..N,1..N] of integer;  
    { цвета вершин }  
    col: array[1..N] of integer;  
    { номера вершин для выбранных ребер }  
    ostov: array[1..N-1,1..2] of integer;  
    i, j, k, iMin, jMin, min, c: integer;
```

## Вывод результата:

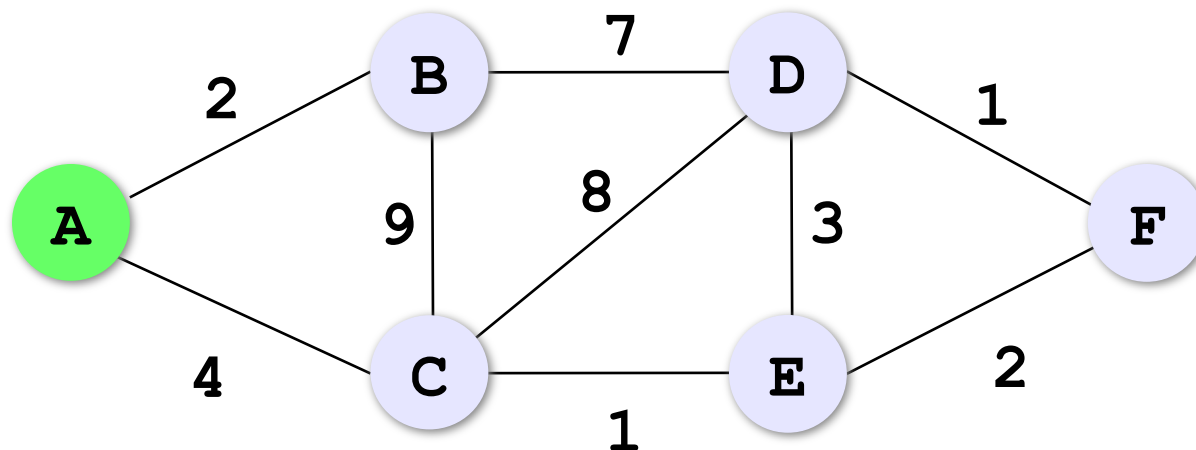
```
for i:=1 to N-1 do  
    writeln('(', ostov[i,1], ', ',  
           ostov[i,2], ')');
```

# Раскраска вершин

```
for k:=1 to N-1 do begin
  { поиск ребра с минимальным весом }
  min:=MaxInt;
  for i:=1 to N do
    for j:=1 to N do
      if (col[i] <> col[j]) and
        (W[i,j] < min) then begin
        iMin:=i; jMin:=j; min:=W[i,j]
      end;
  ostov[k,1]:=iMin; { добавление ребра }
  ostov[k,2]:=jMin;
  c:=col[jMin];
  for i:=1 to N do { перекрашивание вершин }
    if col[i]=c then
      col[i]:=col[iMin]
end;
```

# Кратчайший маршрут

## Алгоритм Дейкстры (1960):



Э.В.  
Дейкстра

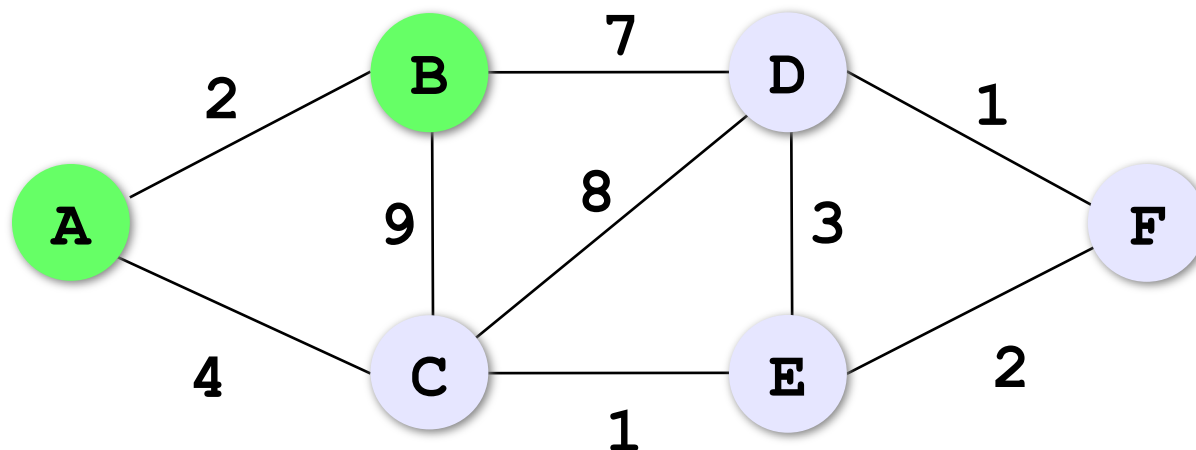
	A	B	C	D	E	F
R	0	2	4	$\infty$	$\infty$	$\infty$
P	x	A	A	A	A	A

кратчайшее расстояние  
откуда ехать

ближайшая от A  
невыбранная вершина

# Кратчайший маршрут

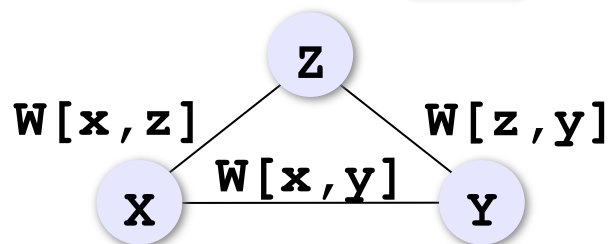
## Алгоритм Дейкстры (1960):



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

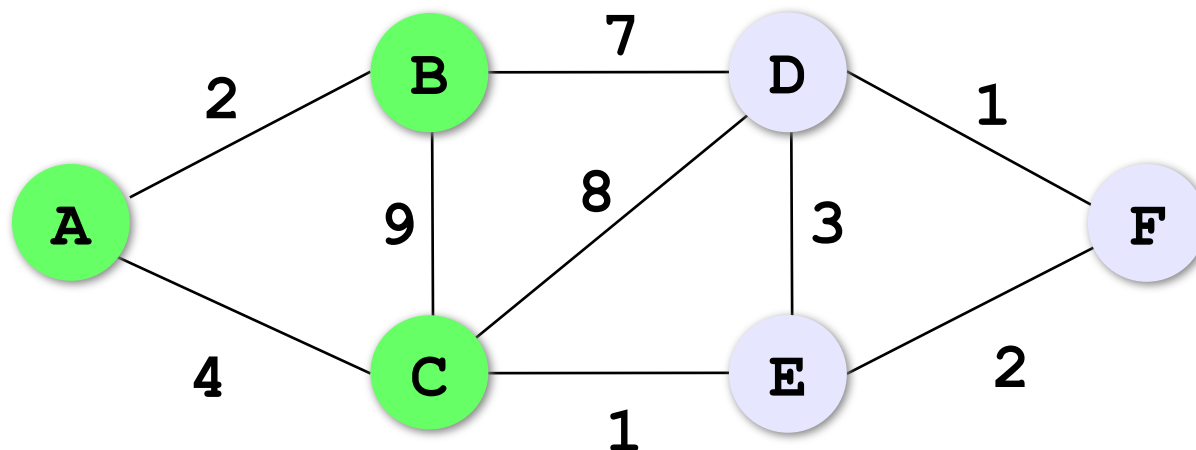


может быть так, что

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

# Кратчайший маршрут

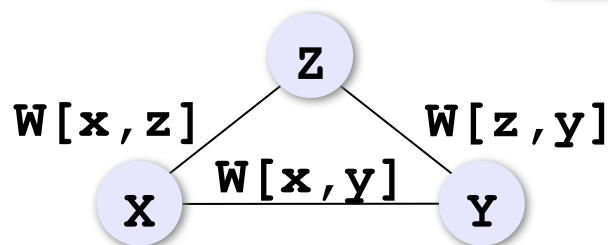
## Алгоритм Дейкстры (1960):



Э.В. Дейкстра

	A	B	C	D	E	F
R	0	2	4	9	5	$\infty$
P	x	A	A	B	C	A

кратчайшее расстояние  
откуда ехать

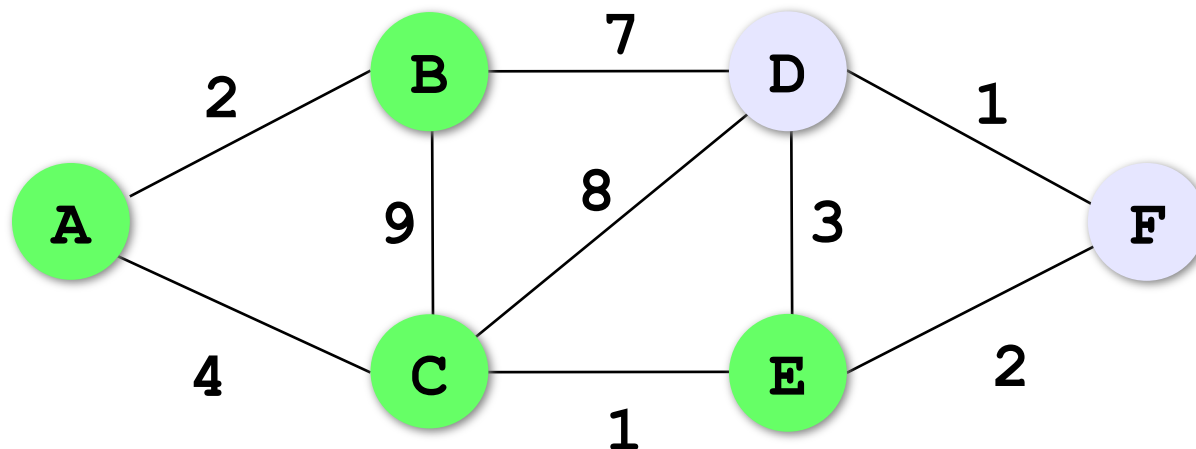


может быть так, что

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

# Кратчайший маршрут

## Алгоритм Дейкстры (1960):



Э.В.  
Дейкстра

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

кратчайшее расстояние  
откуда ехать



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



# Кратчайший маршрут

	А	В	С	Д	Е	Ф
Р	0	2	4	8	5	7
Р	х	А	А	Е	С	Е

длины кратчайших маршрутов из А в другие вершины

?

Как найти сам маршрут?

	А	В	С	Д	Е	Ф
Р	0	2	4	8	5	7
Р	х	А	А	Е	С	Е

А → С → Е → Ф

# Алгоритм Дейкстры

## Данные:

```
const N = 6;  
var W: array[1..N,1..N] of integer;  
    active: array [1..N] of boolean;  
    R, P: array [1..N] of integer;  
    i, j, min, kMin: integer;
```

## Начальные значения (выбор начальной вершины):

```
for i:=1 to N do begin  
    active[i] := True; { вершины не выбраны }  
    R[i] := W[1,i]; { только маршруты из A }  
    P[i] := 1      { вершина A }  
end;  
active[1] := False; { вершина A выбрана }  
P[1] := 0;         { это начало маршрута }
```

# Алгоритм Дейкстры

## Основной цикл:

```
for i := 1 to N-1 do begin
  min := MaxInt;
  for j := 1 to N do
    if active[j] and (R[j] < min) then begin
      min := R[kMin];
      kMin := j
    end;
  active[kMin] := False;
  for j := 1 to N do
    if R[kMin] + W[kMin, j] < R[j] then begin
      R[j] := R[kMin] + W[kMin, j];
      P[j] := kMin
    end
  end;
end;
```

выбор следующей  
вершины,  
ближайшей к A

проверка  
маршрутов через  
вершину kMin

# Алгоритм Дейкстры

---

Вывод результата (маршрут  $1 \rightarrow N$ ):

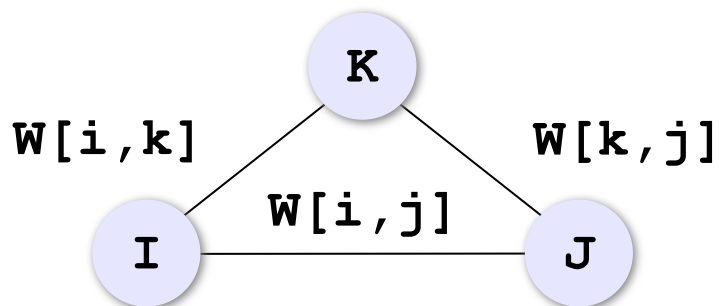
```
i := N; { конечная вершина }  
while i <> 0 do begin  
  write(i:5);  
  i := P[i] { к следующей вершине }  
end;
```

для начальной  
вершины  $P[i]=0$

# Алгоритм Флойда

Все кратчайшие пути (из любой вершины в любую):

```
for k:= 1 to N do
  for i:= 1 to N do
    for j:= 1 to N do
      if  $W[i,k] + W[k,j] < W[i,j]$  then
         $W[i,j] := W[i,k] + W[k,j]$  ;
```



Как найти сам маршрут?

# Алгоритм Флойда + маршруты

---

## Дополнительная матрица:

```
for i:=1 to N do begin
  for j:=1 to N do
    P[i,j] := i;
  P[i,i] := 0;
end;
```

## Кратчайшие длины путей и маршруты:

```
for k:=1 to N do
  for i:=1 to N do
    for j:=1 to N do
      if W[i,k]+W[k,j]<W[i,j] then begin
        W[i,j] := W[i,k]+W[k,j];
        P[i][j] := P[k][j]
      end;
```

# Задача коммивояжера

Коммивояжер (бродячий торговец) должен выйти из города 1 и, посетив по разу в неизвестном порядке города  $2, 3, \dots, N$ , 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 ( $i=1, \dots, N$ ). 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

## **§ 44. Динамическое программирование**

# Что такое динамическое программирование?

## Числа Фибоначчи:

$$F_1 = F_2 = 1$$

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

```
function Fib(N: integer) :
    integer;
```

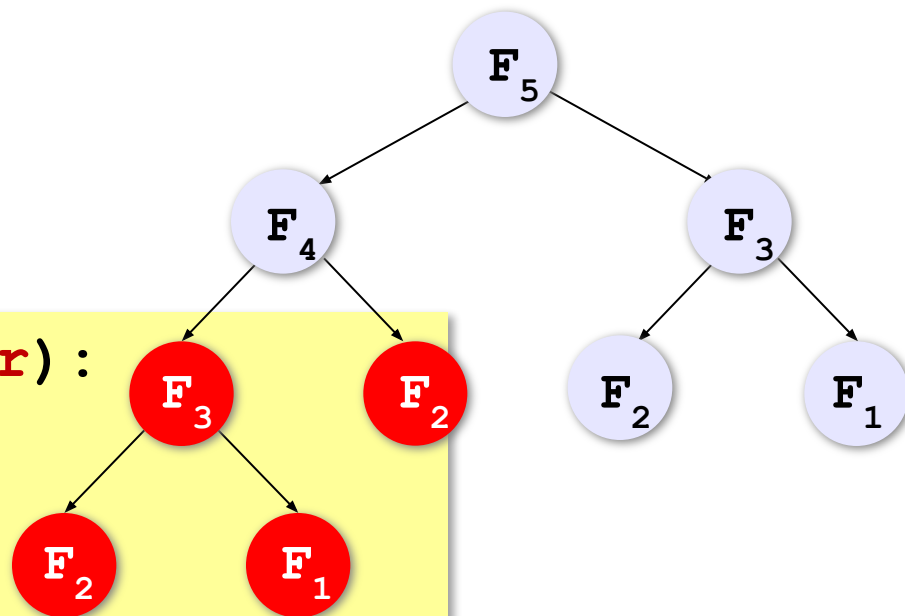
```
begin
```

```
  if N < 3 then
```

```
    Fib := 1
```

```
  else Fib := Fib(N-1) + Fib(N-2)
```

```
end;
```



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 N = 10;
var F: array[1..N] of integer;
```

## Заполнение массива:

```
F[1] := 1; F[2] := 1;
for i := 3 to N do
  F[i] := F[i-1] + F[i-2];
```

$F_{45}$ : рекурсия: **8 с**

дин. программирование: **< 0,01 с**

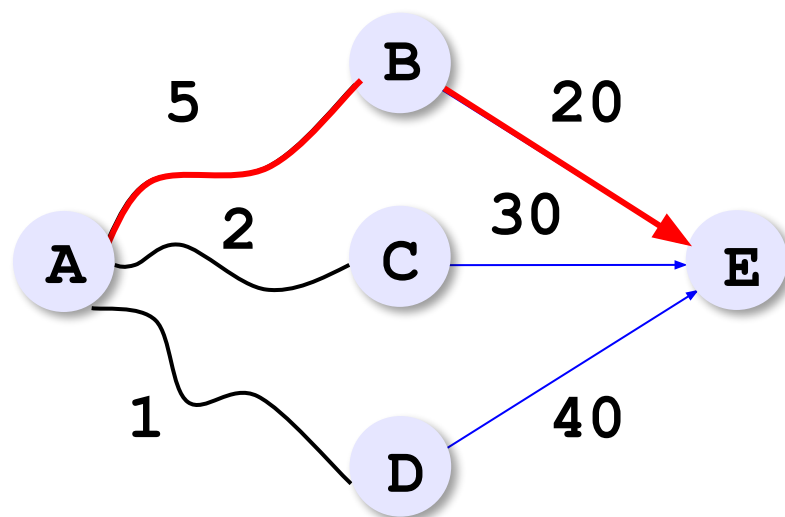


Можно ли обойтись без массива?

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.

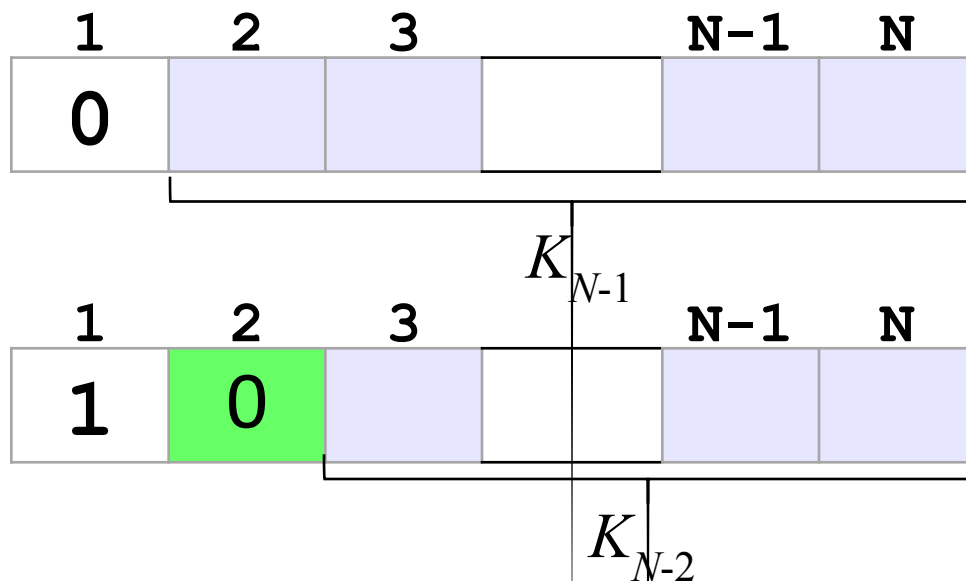
**Простые случаи:**

$$N = 1: \quad \mathbf{0} \quad \mathbf{1} \quad K_1 = 2$$

$$N = 2: \quad \mathbf{00} \quad \mathbf{01} \quad \mathbf{10} \quad K_2 = 3$$

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

**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»?

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

$$10 = 5 + 5 \quad K = 2$$



Не даёт оптимального решения!

# Оптимальное решение

$K_N$  – минимальное число бидонов для  $N$  литров

Сначала выбрали бидон...

$$\left. \begin{array}{l} 1 \text{ л: } K_N = 1 + K_{N-1} \\ 5 \text{ л: } K_N = 1 + K_{N-5} \\ 6 \text{ л: } K_N = 1 + K_{N-6} \end{array} \right\} \text{min}$$

Рекуррентная формула:

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

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

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



# 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  $p_i$  ( $i=1, \dots, N$ ) 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  $p_i$  ( $i=1, \dots, N$ ) 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	$w$
1	2	0	0	2	2	2	2	2	2	2	
2	4	0									
3	5	0									
4	7	0									

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  $w \geq 4$ :

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  $w \geq 5$ :

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  $w \geq 7$ :

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!

max

для  $w \geq p_i$ :

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

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

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

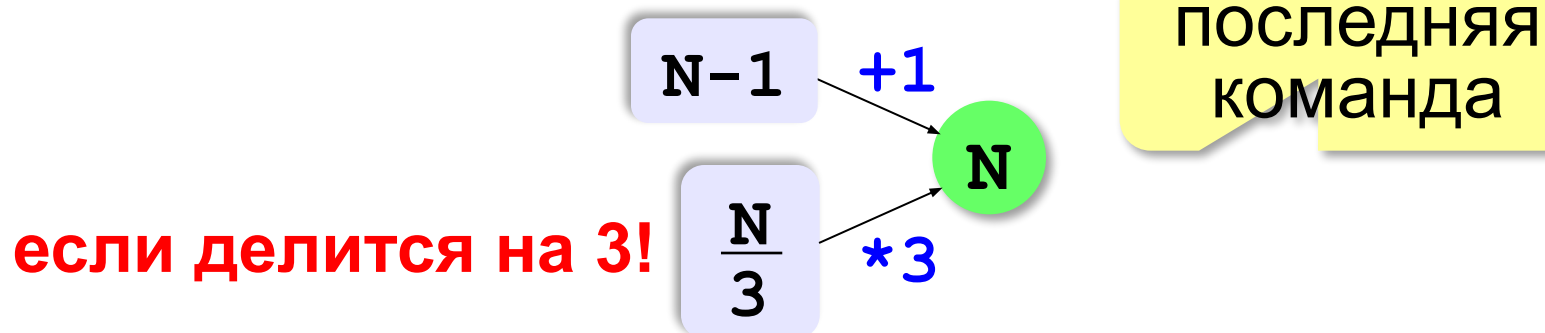
Сколько есть разных программ, с помощью которых можно из числа **1** получить число **20**?



Как решать, не выписывая все программы?

# Количество программ

Как получить число  $N$ :



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:

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

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

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

Только точки изменения:

20

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 to N do begin  
  K[i] := K[i-1];  
  if i mod 3 = 0 then  
    K[i] := K[i] + K[i div 3]  
end;
```



Где ответ?

# Размен монет

---

**Задача.** Сколькими различными способами можно выдать сдачу размером  $W$  рублей, если есть монеты достоинством  $p_i$  ( $i=1, \dots, N$ )? 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  $w < p_i$ :  $T[i, w] = T[i-1, w]$

без этой монеты

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

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](mailto:kpolyakov@mail.ru)

**ЕРЕМИН Евгений Александрович**

к.ф.-м.н., доцент кафедры мультимедийной

дидактики и ИТО ПГГПУ, г. Пермь

[eremin@pspu.ac.ru](mailto:eremin@pspu.ac.ru)

# Источники иллюстраций

---

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