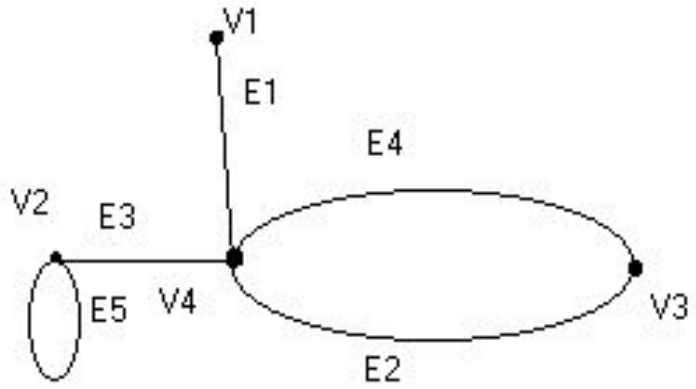
Графы

Оглавление:

Некоторые определения

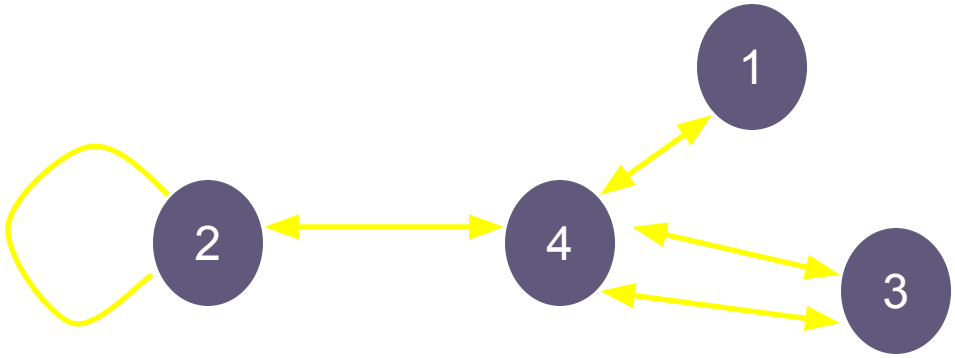


Способы задания графов

E	$V_{нач}$ $V_{кон}$
e_1	V_2 V_4
e_2	V_4 V_3
e_3	V_4 V_2
e_4	V_4 V_3
e_5	V_2

	1	2	3	4	5
1	0	0	0	1	0
2	0	½	0	1	0
3	0	0	0	2	0
4	1	1	2	0	0
5	0	0	0	0	0

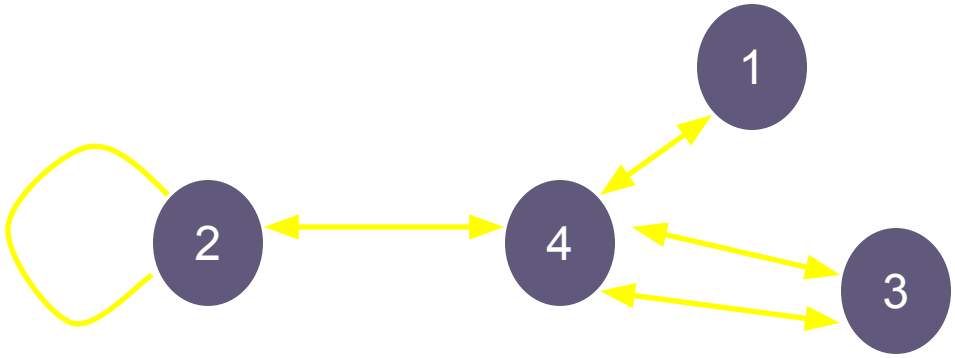
<1,4>
<2,2> петля <2,4>
<3,4> <3,4>
<4,1> <4,2> <4,3> <4,3>



Способы задания графов

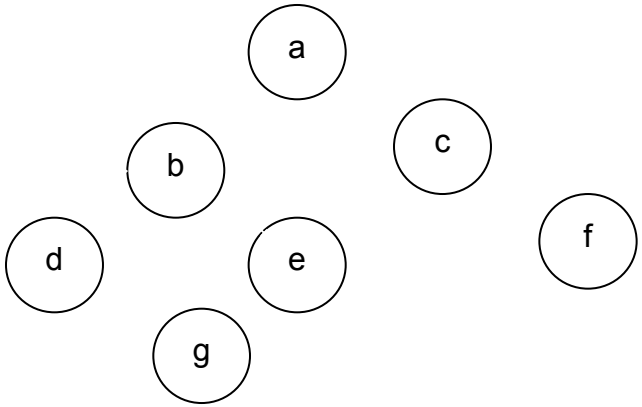
	0	1	2	3	4
1	1	4			
2	2	2	4		
3	2	4	4		
4	4	2	1	3	3
5	0				

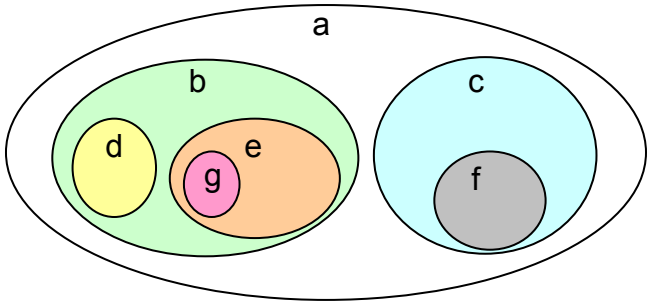
<1,4>
<2,2> <2,4>
<3,4> <3,4>
<4,2> <4,1> <4,3> <4,3>



Деревья

Способы задания деревьев





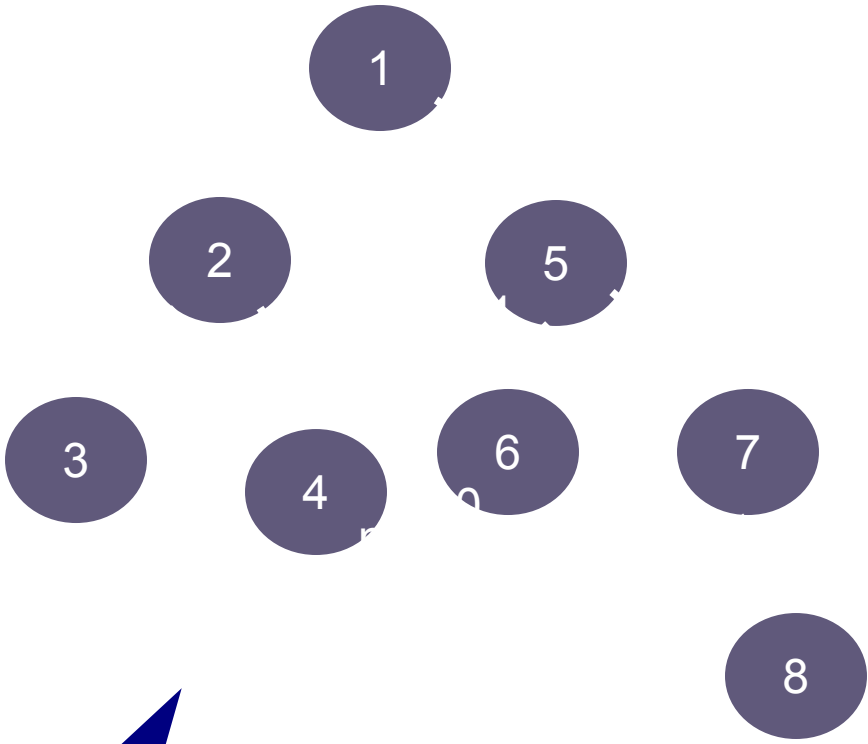
Некоторые определения

- **Степенью узла**
- **Степенью дерева**
- **Высотой узла**
- **Высотой дерева**
- **Бинарным деревом**
- **Лист-**

Сформировать бинарное дерево из N узлов

```
Type     Ref=^Node;
Node=Record
            Left, Right:Ref;
            Key:тип_элемента
          End;
Var root:Ref;{корень дерева}
```

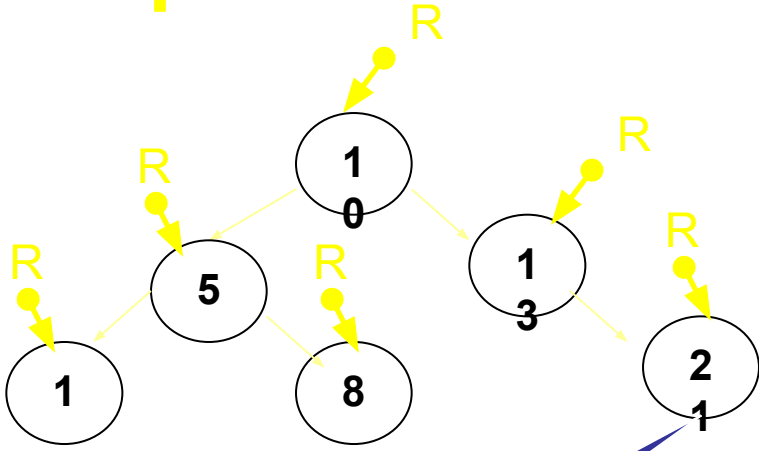
```
Function BuildT(n:integer):ref;
Var R:Ref;
Begin
 If n=0 then BuildT:=nil
 Else begin
      New(R);
      {Создать корень поддерева}
      Write('введите значение звена');
      readln(R^.key);
      {сосчитать узлы слева и справа}
      nl:=(n-1)div 2;nr:=n-1-nl;
      R^.Left:=BuildT(nl);
     {построить левое поддерево}
     R^.Right:=BuildT(nr);
     {построить правое поддерево}
     BuildT:=R;
{прицепить дерево к имени функции}
  End;
End;
```

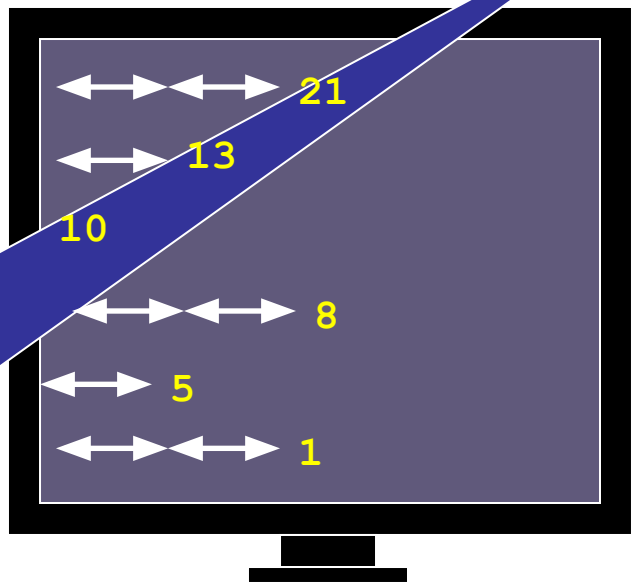


Построим бинарное дерево из 8 узлов. N=8

Вывод дерева на экран

```
Procedure PrintT(R:ref;h:integer);
Var i:integer;
Begin
  If r<>nil
  then begin
         PrintT(R^.Right, h+1);
         for i:=1 to h do
            write('    ');
         writeln(R^.Key);
         PrintT(R^.Left, h+1);
       End;
End;
```





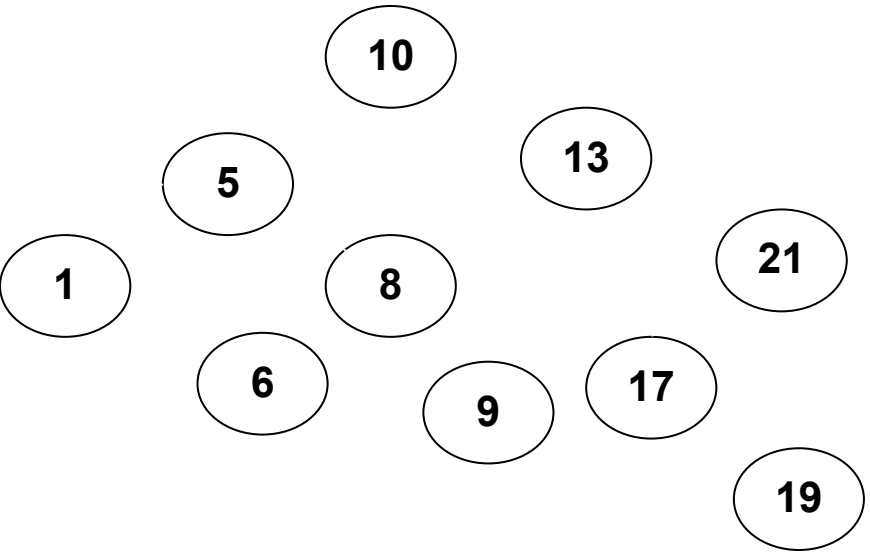
Для вывода де а экран, поверне
90 град тив часовой

Следую ивный вызов подпрограммы

Теперь пойдем в левое поддерево.

2

Дерево поиска

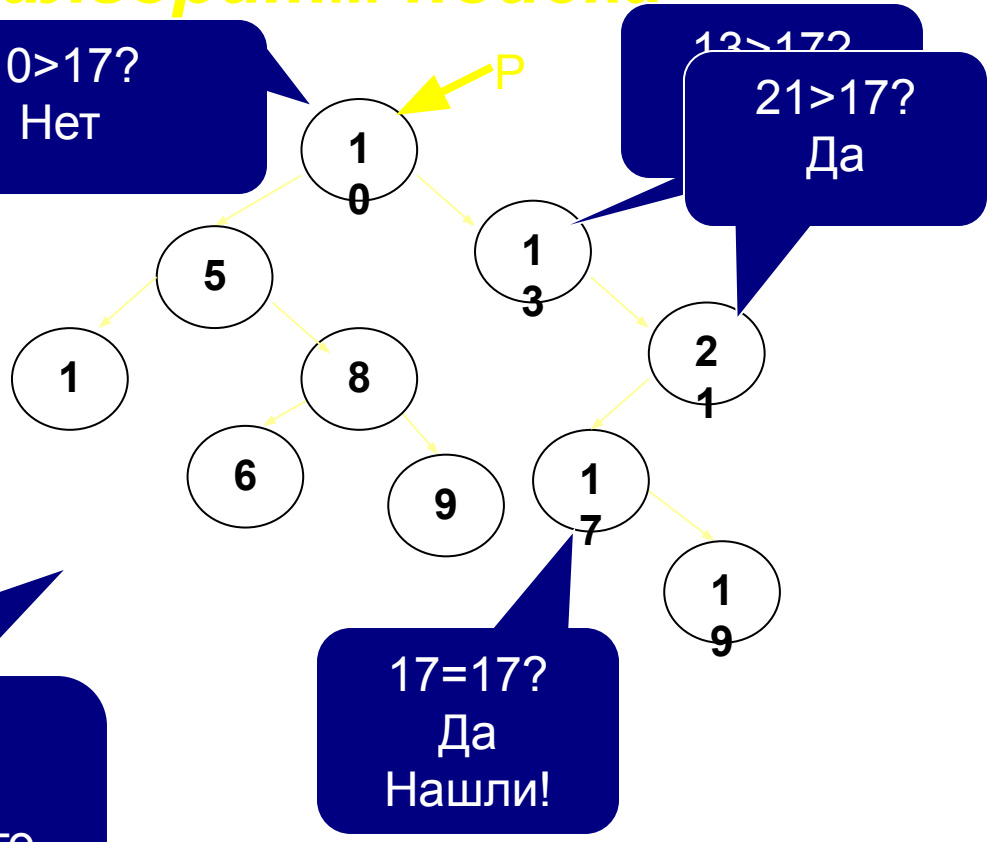


Итерационный алгоритм поиска

ввели искомое значение

встали на корень

пока не дошли до конца и не нашли

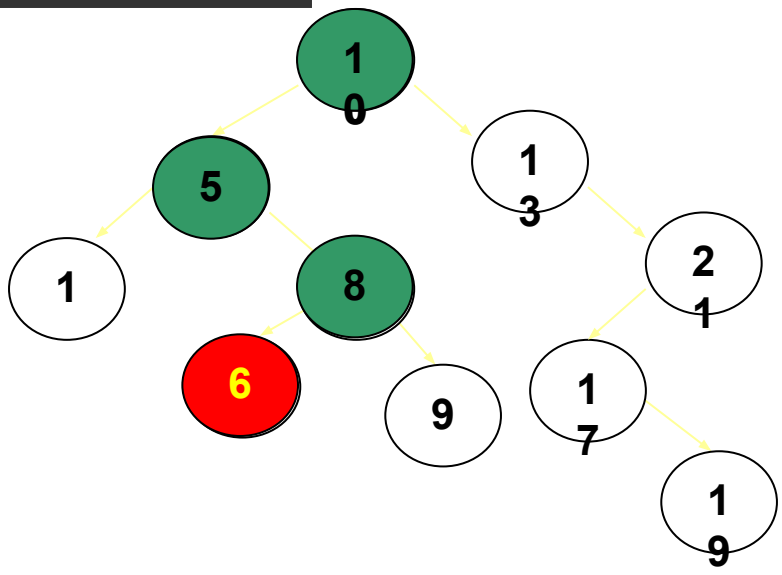


Пусть мы ищем число 17. Встаем на корень дерева и сравниваем значение текущего звена с ключом поиска

Рекурсивный алгоритм поиска

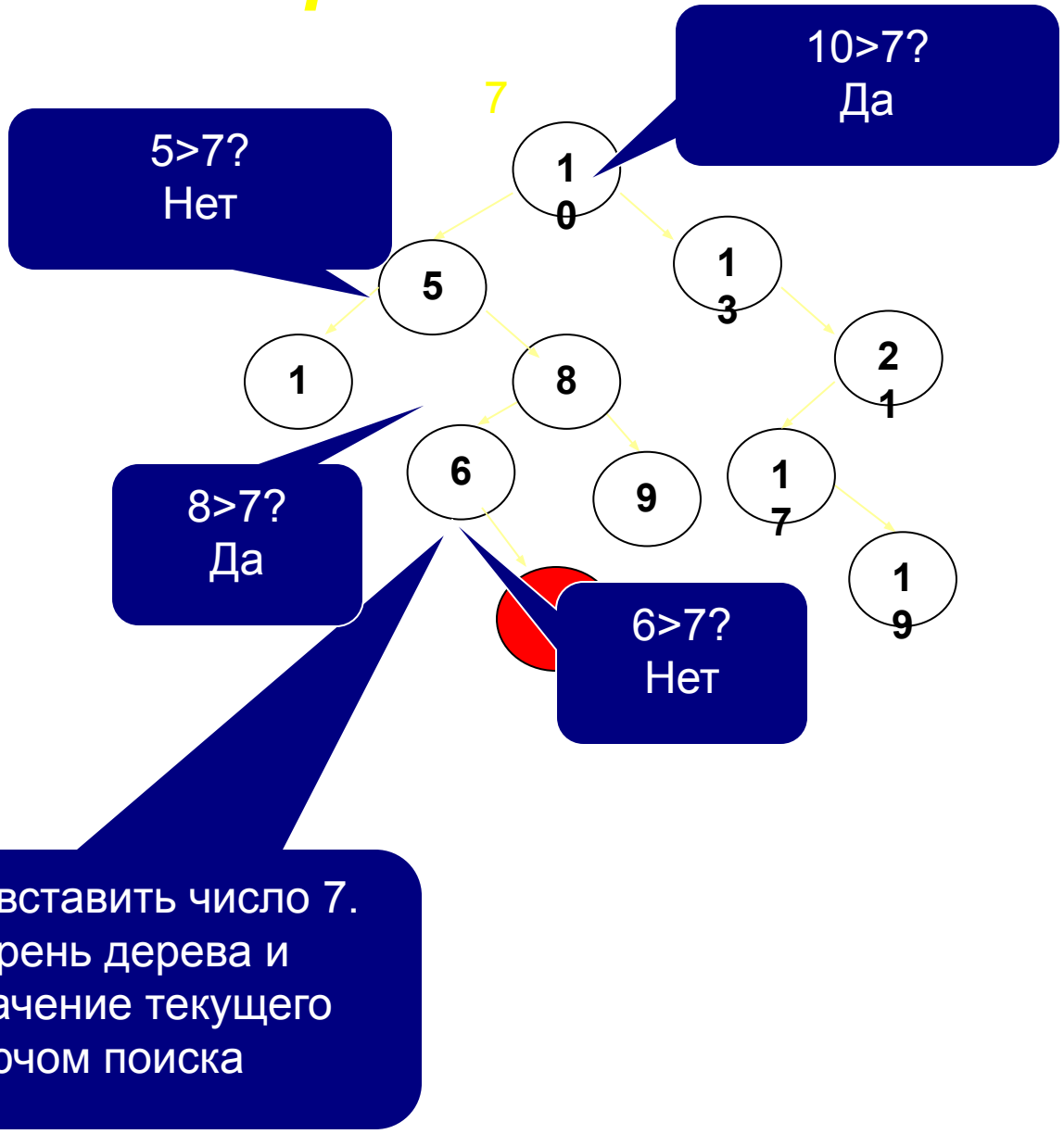
```
Function Seek(p:ref;x:integer):Ref;
Begin
If  (p=nil) or (p^.key=x)
Then Seek:=p
Else If p^.key>x
     Then Seek:=Seek(p^.left,x)
   Else Seek:=Seek(p^.right,x)
End;
```

6

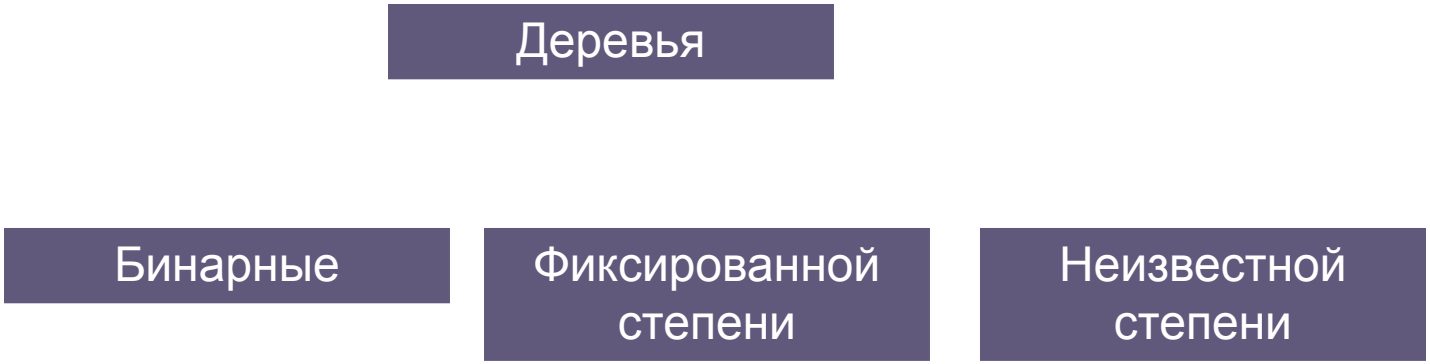


Рекурсивный алгоритм вставки

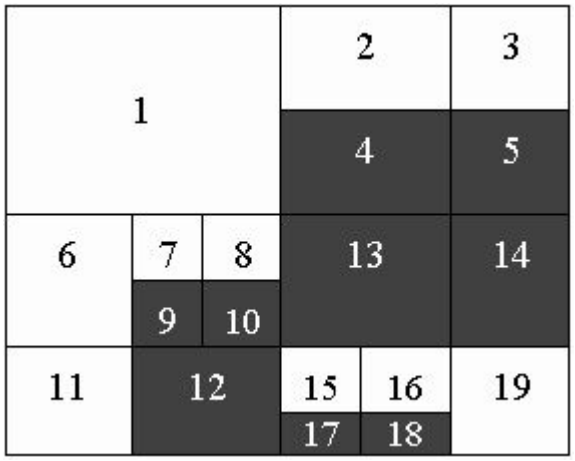
Var

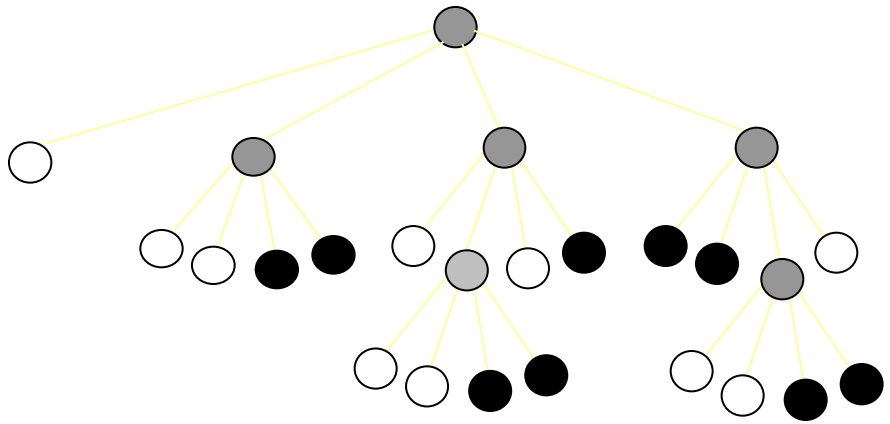


Сильноветвящиеся деревья

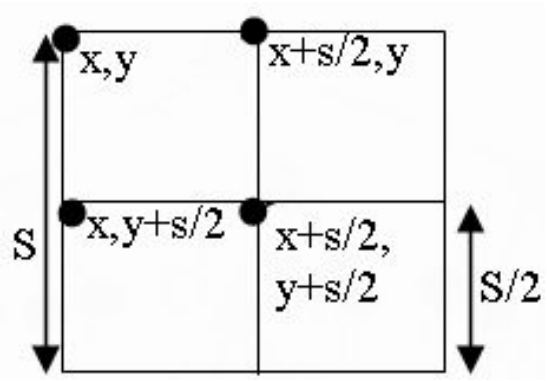


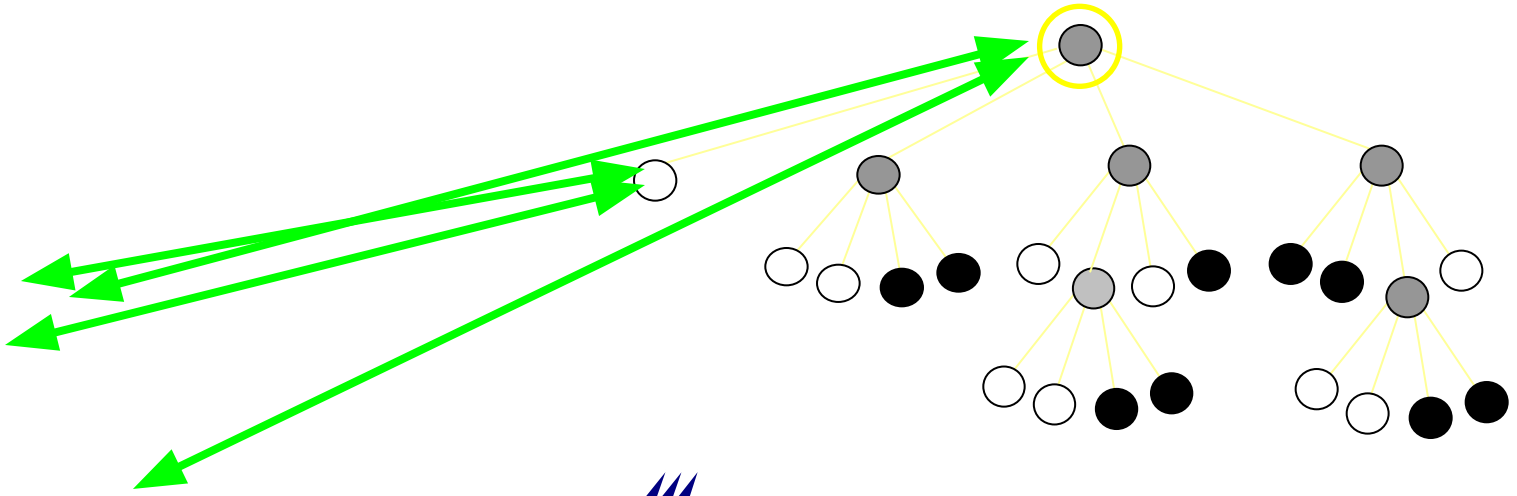
Черно-белые деревья

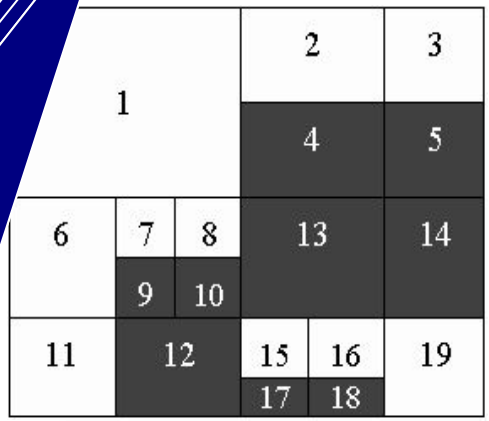


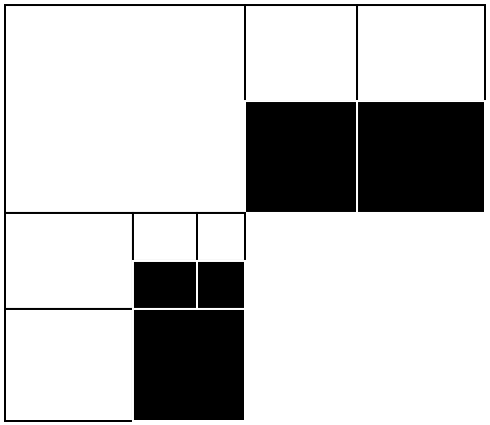


4 ссылки на квадраты
цвет текущего квадрата





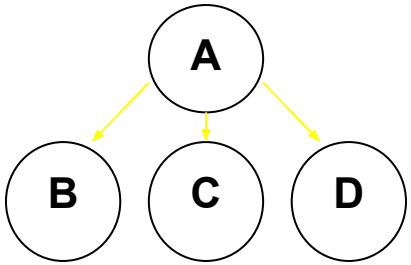




Далее аналогично

итерация пойдет по левой ветке и нарисует 1-ый белый квадрат

Генеалогическое древо



Указатель на звено дерева

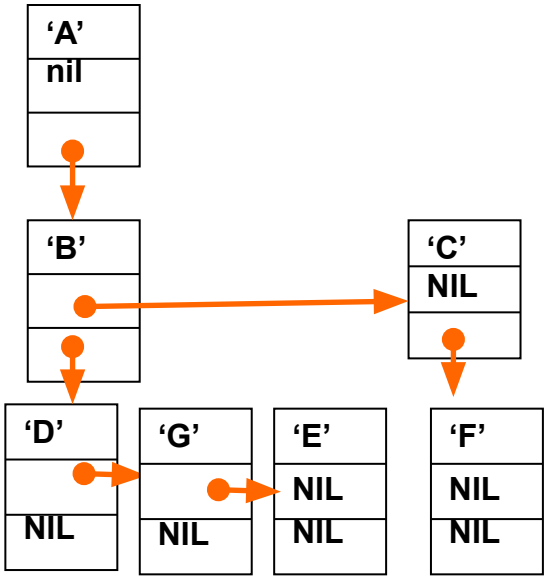
Звено дерева

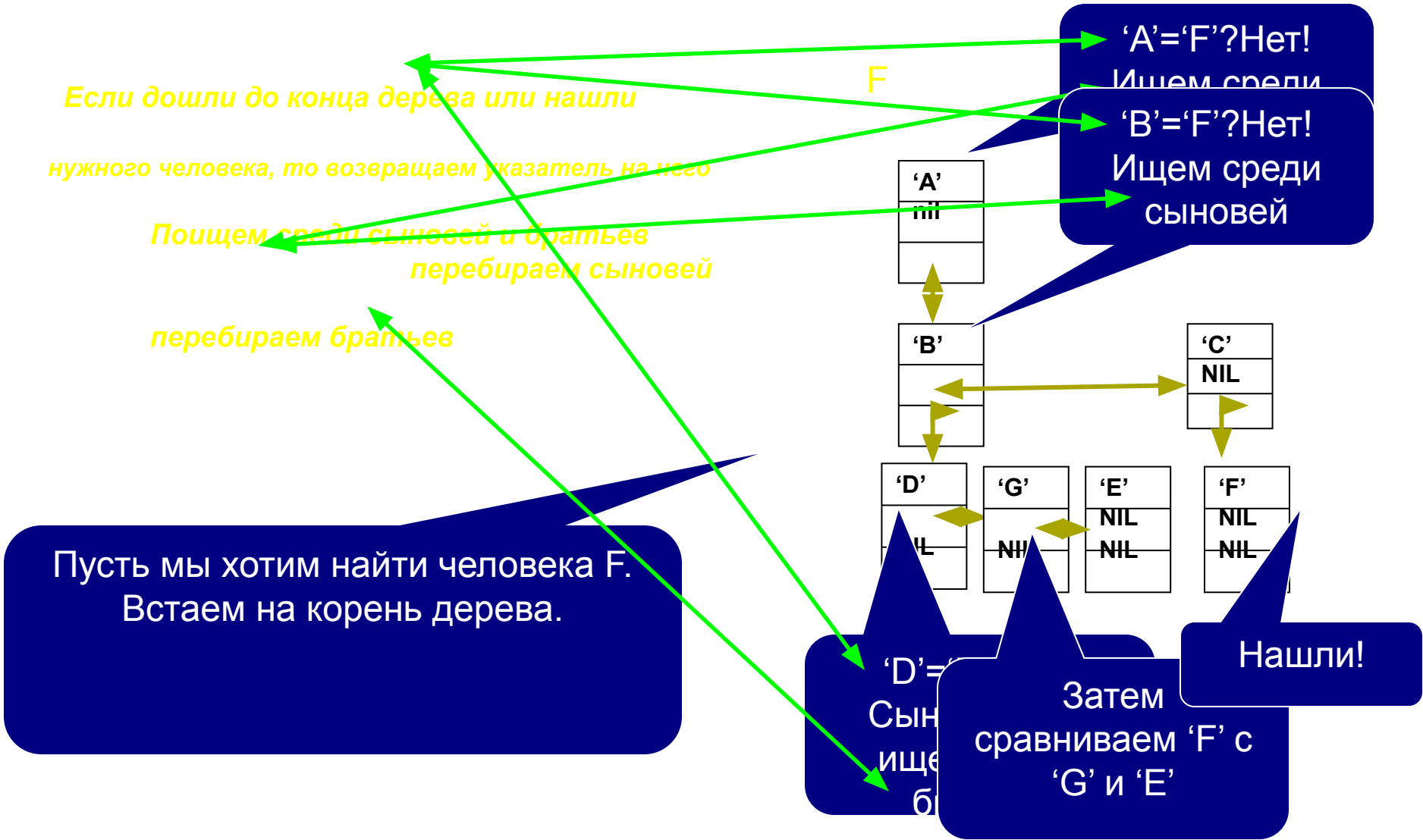
Имя человека

Указатель на звено потомка

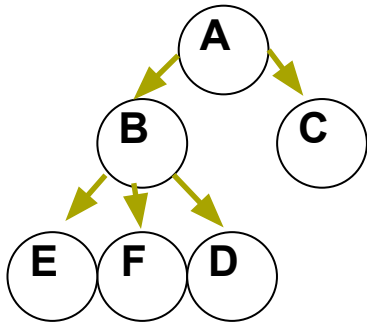
Указатель на следующее звено этого уровня-брата

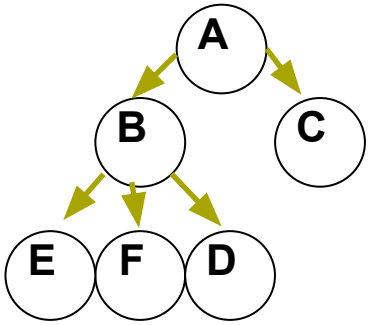
Указатель на предка




Если дошли до конца дерева или нашли
нужного человека, то возвращаем указатель на него
Поищем среди сыновей и братьев
перебираем сыновей
перебираем братьев
F
‘A’=‘F’?Нет!
Ищем среди
‘B’=‘F’?Нет!
Ищем среди
сыновей
‘A’
nil
‘B’
‘C’
NIL
‘D’
‘G’
‘E’
NIL
NIL
‘F’
NIL
NIL
Пусть мы хотим найти человека F.
Встаем на корень дерева.
‘D’=
Сын
ище
б
Затем
сравниваем ‘F’ с
‘G’ и ‘E’
Нашли!

Преобразование дерева в строку

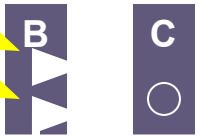




если поддерево пустое
то добавим пустые скобки
если поддерево не пустое
вначале добавим ‘(’
встаем на сына

R

T
R



бежим по «братьям»пока они есть
каждого «брата» преобразуем
в строковый вид и переходим
к следующему
не забудем поставить ‘,’

а также скобку

S=A(B(E,F,D),C())

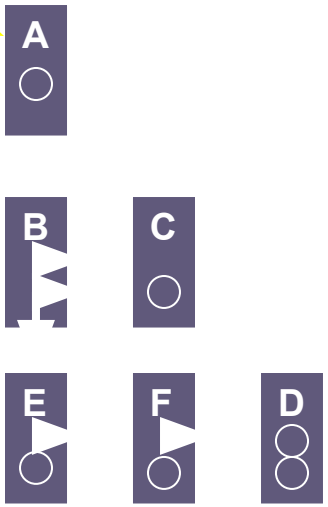
Рекурсивный вызов!

Рекурсивный возврат!

S='A(B(E,F,D),C())'

если строка еще не закончилась

R



создаем новый узел - корень

запоминаем его значение

смещаемся на 2: название и скобку

если скобки пустые, то поддерева нет

сыновей нет

поддерево все же есть

разбираем его на части

встаем на первого «сына»

{пока «сыновья не закончились»

Рекурсивный вызов!

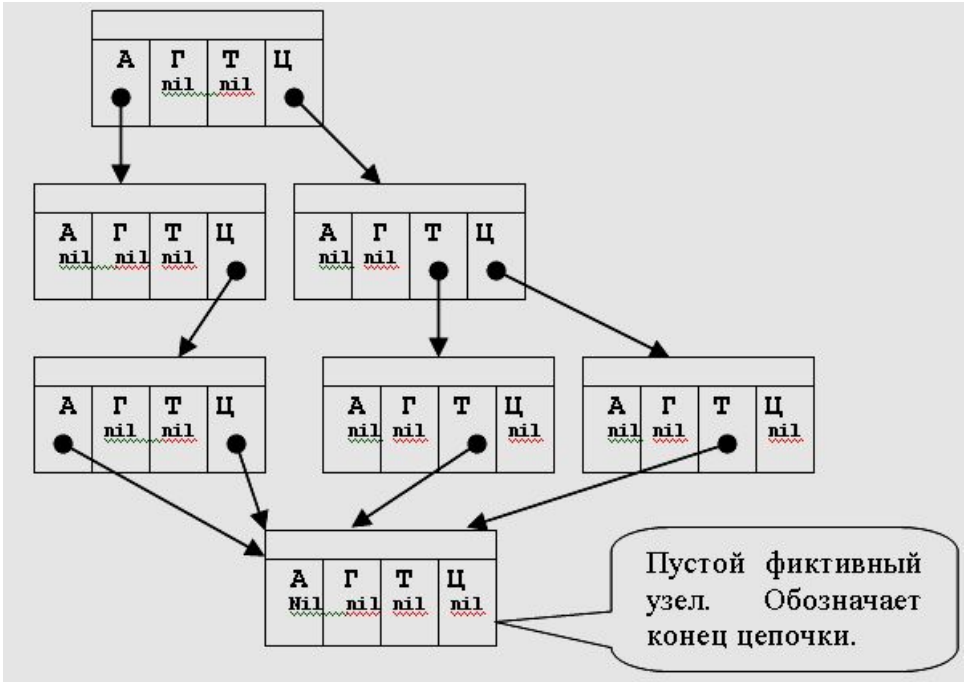
вырезаем их из строки и прицепляем к дереву

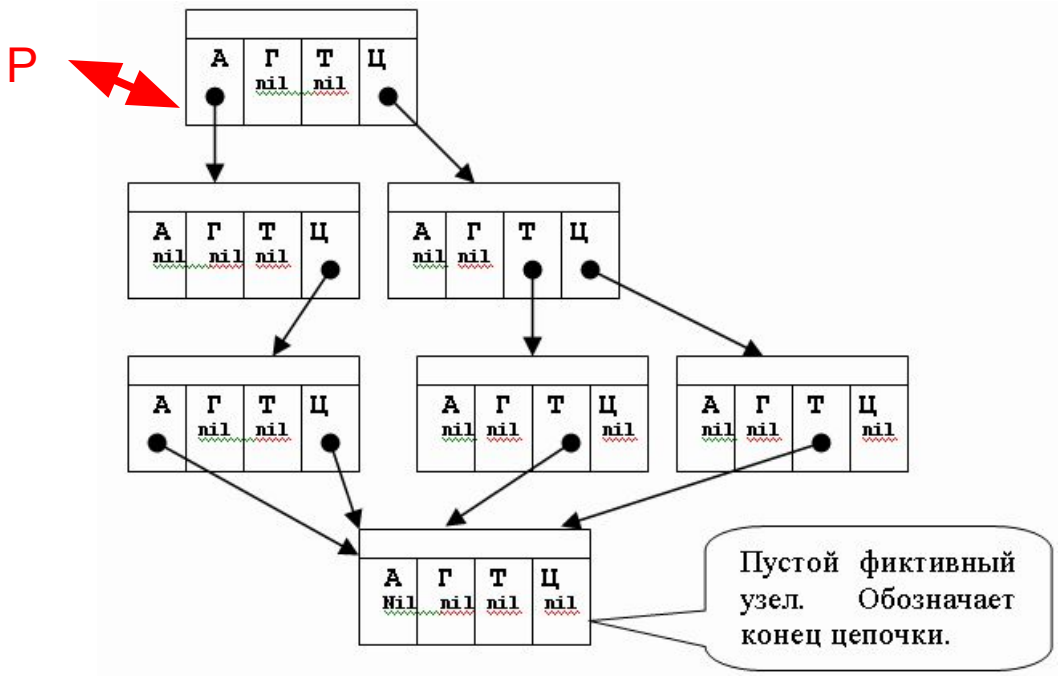
Рекурсивный возврат!

Цепочки ДНК

Задание:

- **разработайте структуру данных для эффективного хранения цепочек ДНК;**
- **опишите эффективный по времени алгоритм, позволяющий определить, есть ли в базе данная цепочка ДНК.**


А Г Т Ц
nil nil
А Г Т Ц
nil nil nil
А Г Т Ц
nil nil
А Г Т Ц
nil nil nil
А Г Т Ц
nil nil
А Г Т Ц
nil nil nil
А Г Т Ц
Nil nil nil nil
Пустой фиктивный узел. Обозначает конец цепочки.



встаем на начало дерева и строки

S=‘ЦТТ’

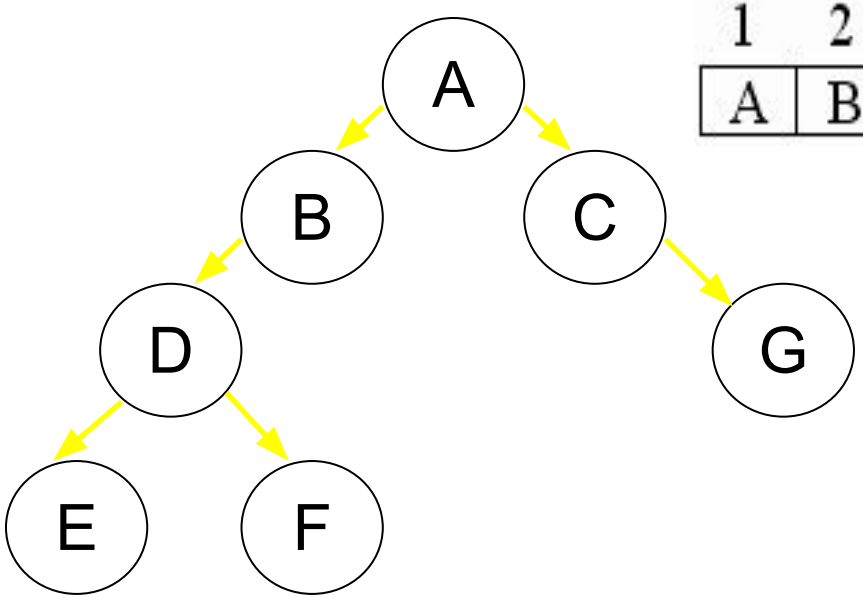
двигаемся по дереву, используя коды ДНК,как индексы массива

сдвигаемся к следующему звену и следующему коду ДНК

Если строка S закончилась и мы не столкнулись с пустой ссылкой nil, то цепочка принадлежит базе.

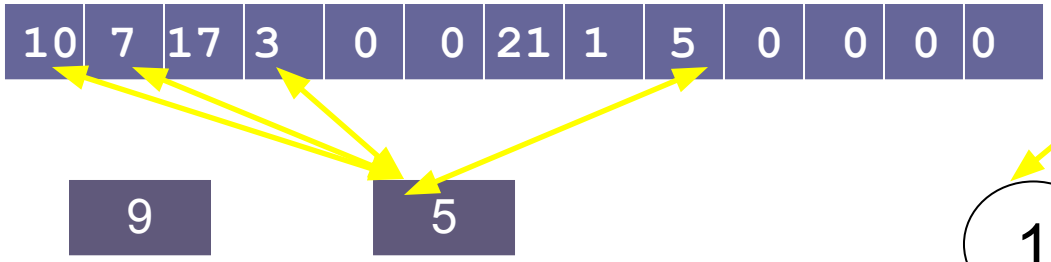
звена нет, то цепочка ДНК не принадлежит базе

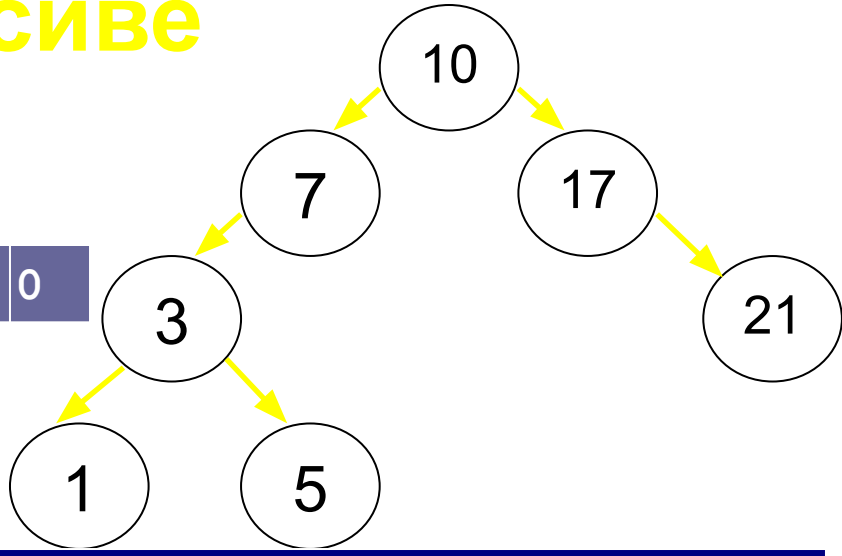
Хранение деревьев в массиве



1	2	3	4	5	6	7	8	9	10	11
A	B	C	D	0	0	G	E	F	0	0

Дерево поиска в массиве



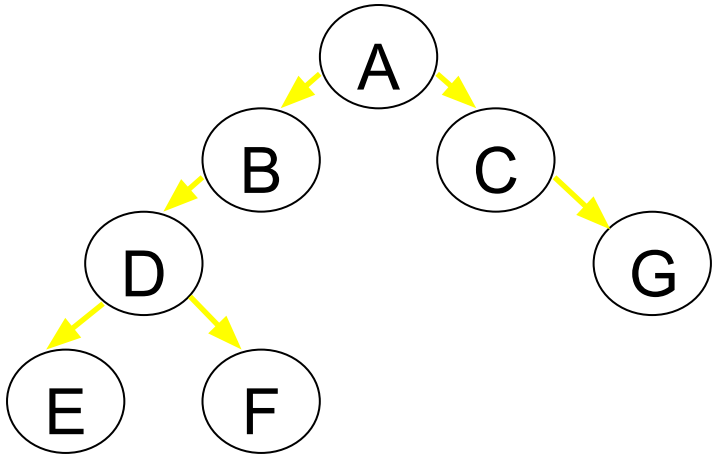


Недостатки:
Неэффективно по памяти для несбалансированных деревьев;
Нельзя хранить в дереве число 0

```
Function Seek(r:ref;key:integer):ref;
Begin
 if (r=nil)or(r^.key=key)
 then Seek:=r
 else If key<r^.key
      then Seek:=Seek(r^.Left,key)
     else Seek:=Seek(r^.Left,key)
End;
```

```
Function Seek( i, key:integer):integer;
Begin
  if (val[i]=0)or(Val[i]=key)
  then Seek:=i
  else If key<Val[i]
       then Seek:=Seek(i*2,key)
       else Seek:=Seek(i*2+1,key)
End;
```

Хранение дерева в массиве с указанием индексов вершин

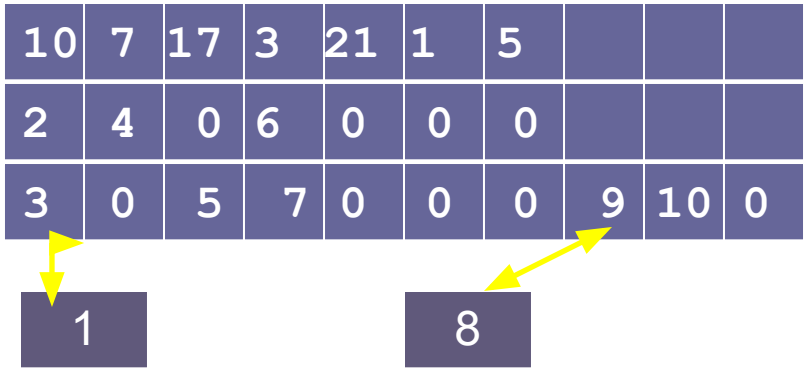


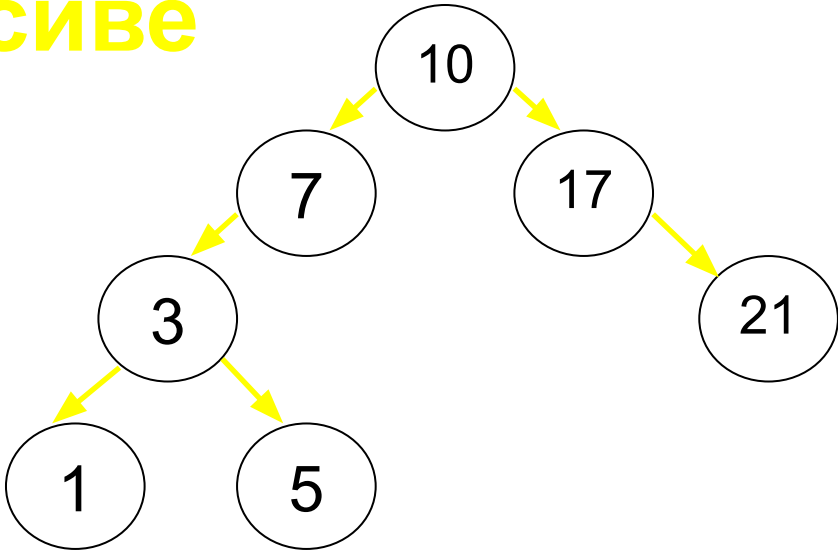
	1	2	3	4	5	6	7	8	9	10	11
Left	2	4	0	5	0	0	0				
Right	3	0	7	6	0	0	0	9	10	11	0
Val	A	B	C	D	E	F	G				

Хранение дерева в массиве с указанием индексов вершин

	1	2	3	4	5	6	7	8	9	10	11
Left	2	4	0	5	0	0	0				
Right	3	0	7	6	0	0	0	9	10	11	0
Val	A	B	C	D	E	F	G				

Дерево поиска в массиве

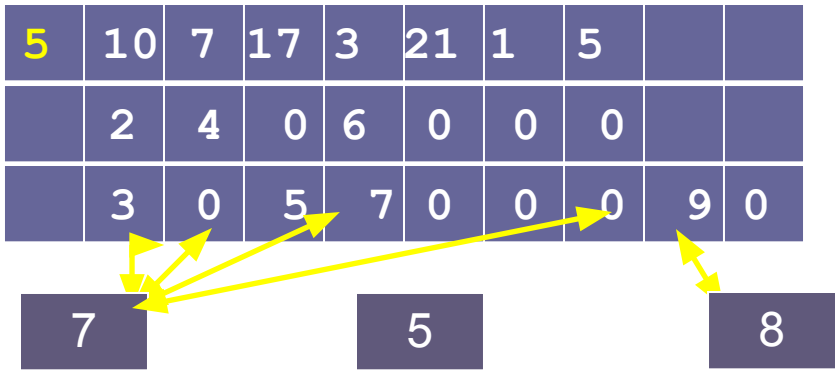


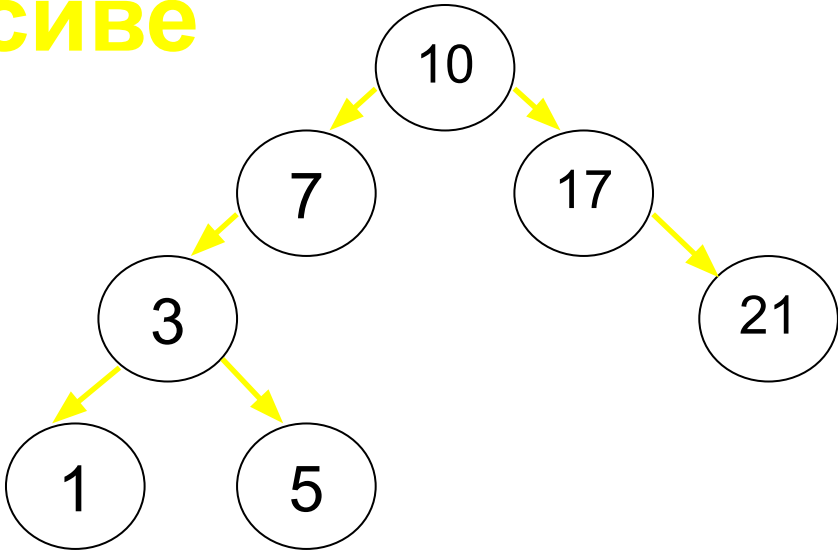



```
Function Seek( root, key:integer):integer;
begin
 If root= Null
 then Seek:=Null
 else Begin
       x:= root; f:=false;
        repeat
           If key = m[x].Val then f:=true
           else If key<m[x].Val
                then x:=m[x].Left
                else x:=m[x].Right
        until  f  or  (x=null)
        Seek:=x
      End;
End;
```

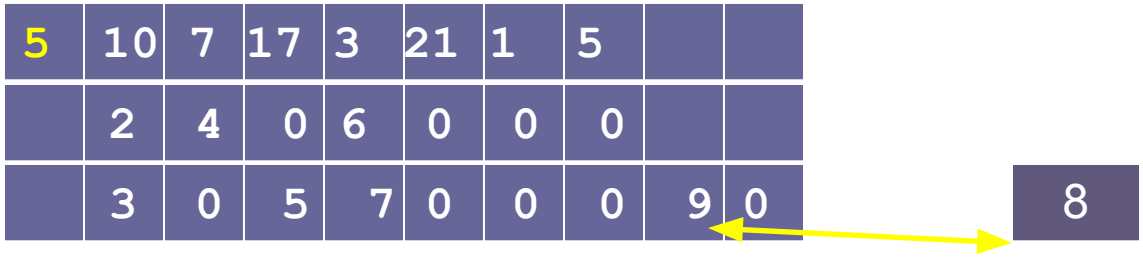
Можно ли ускорить решение?

Дерево поиска в массиве





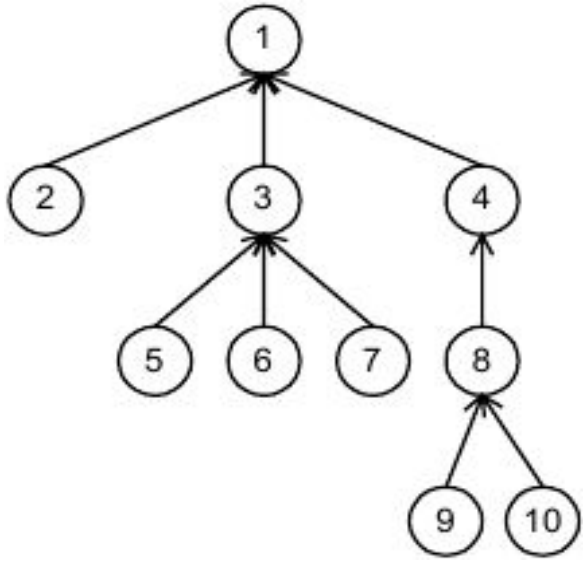

```
Function Seek( root, key:integer):integer;
begin
  m[null].val := key;
  While key <> m[root].val do
    If key < m[root]. val then root:=m[root].left
    else x:=m[root].right;
  if (root <> null) then Seek:=root else Seek:=Null
End;
```




```
function get_free : integer;
```

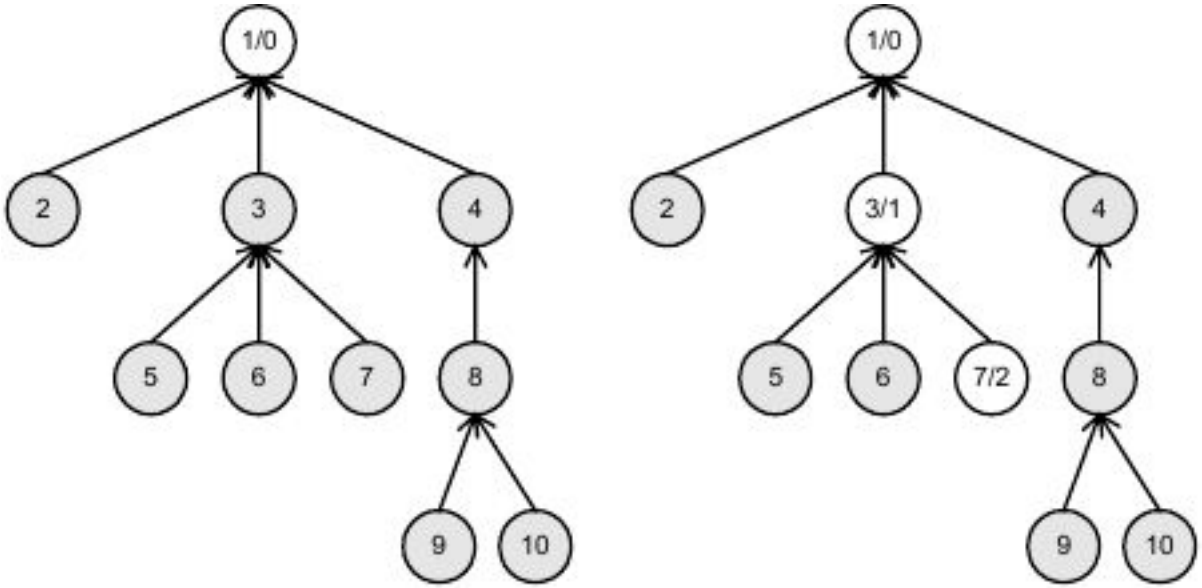
```
Procedure Add(var root:integer; t:тип);
```

Корневые деревья (root tree)

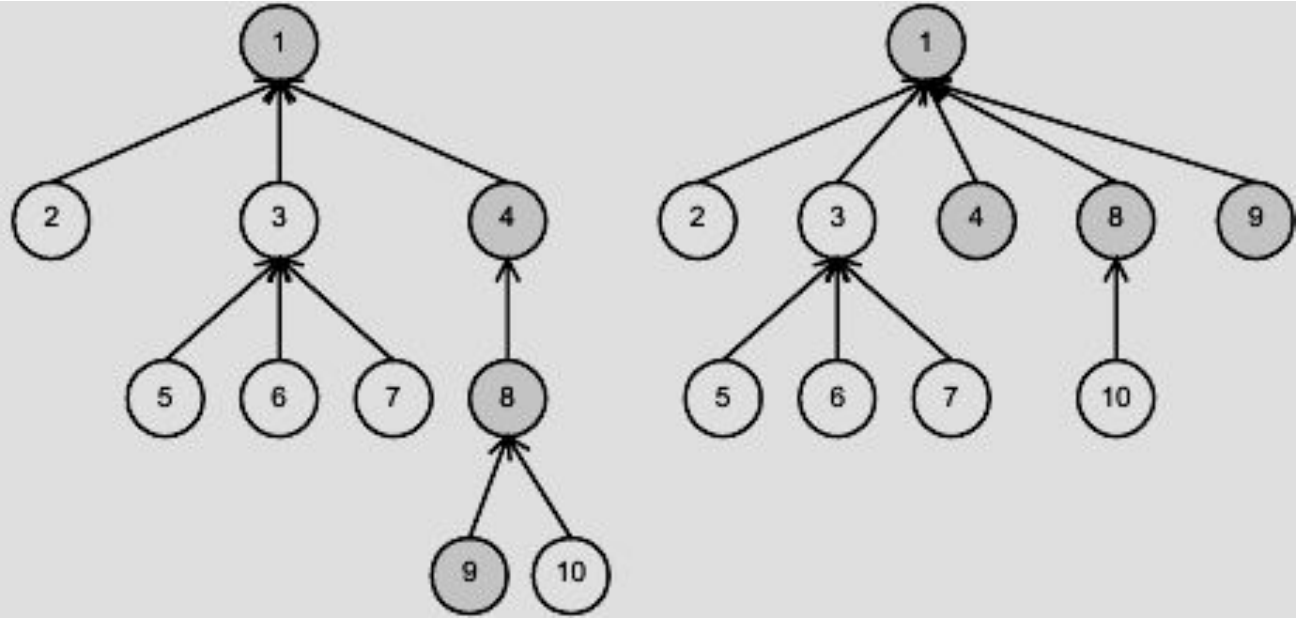


Свойства	Описание
RTree.Root	Номер корневой вершины
Items[]	Значения, хранимые в узлах (внешний)
Count	Количество узлов (внешний)
Parent[]	Номер вершины – предка для данной (внешний)
Операции	**Описание**
RTree.Init	Инициализация дерева
RTree.AddChild(P, Item)	Добавление наследника для узла P

Расчет уровней вершин

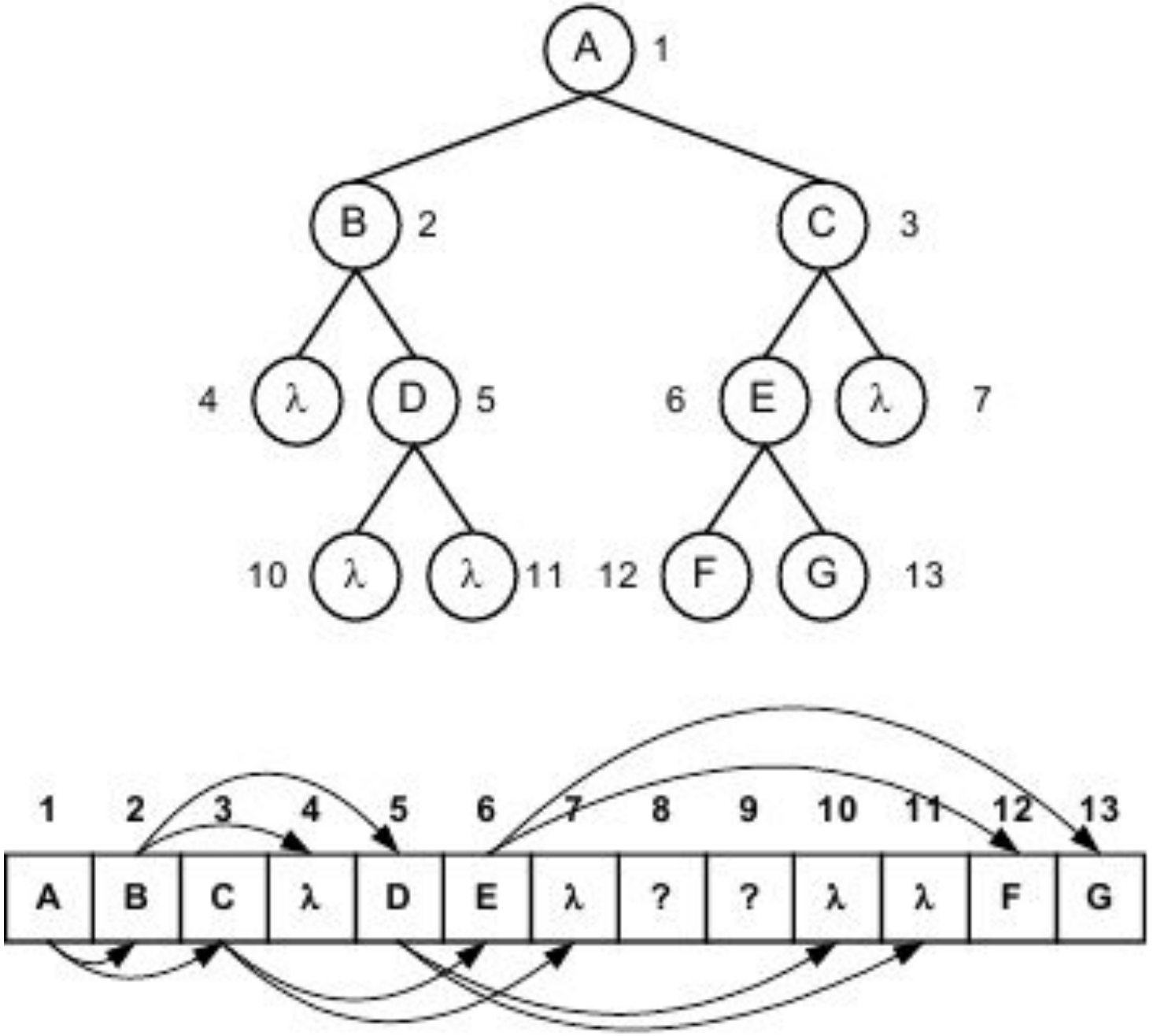


Нахождение корня и сжатие путей


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

Упаковка двоичного дерева в массив

Индекс ячейки	Индексы ячеек-потомков
1	2, 3
2	4, 5
3	6, 7
4	8, 9
5	10, 11
…	…
i	$2i$, $2i+1$


A 1
B 2
C 3
4 λ D 5
6 E λ 7
10 λ λ 11
12 F G 13
1 2 3 4 5 6 7 8 9 10 11 12 13
A B C λ D E λ ? ? λ λ F G

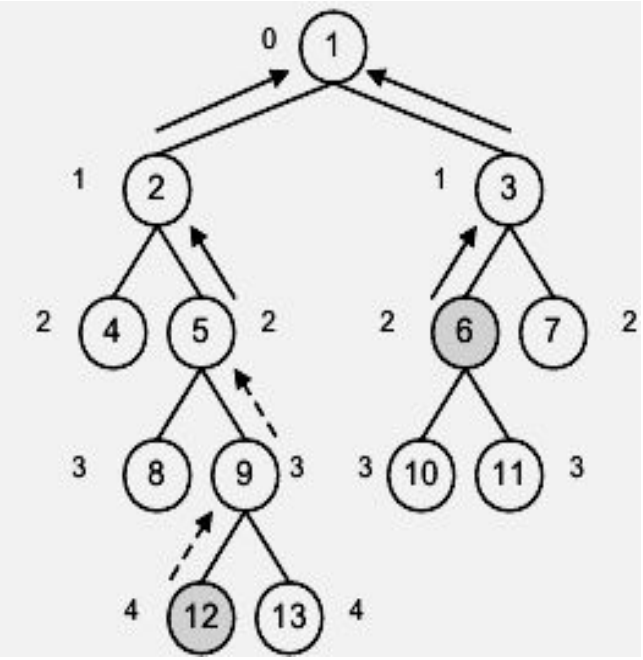
Вычисление уровня (расстояния до корня)

Поиск элемента в дереве

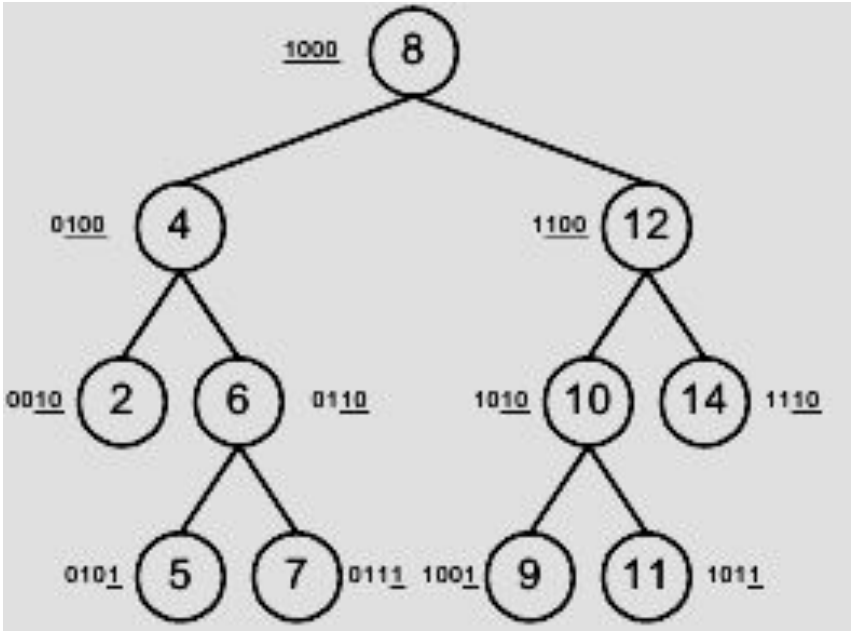
Высота дерева

Наибольший общий предок

Простой алгоритм

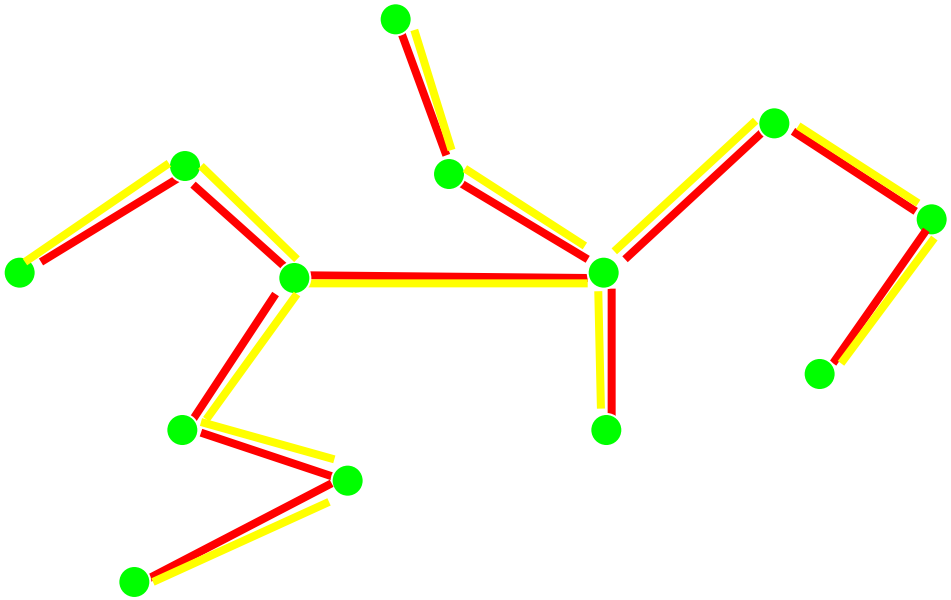


Алгоритм для двоичных деревьев



Обход вершин графа

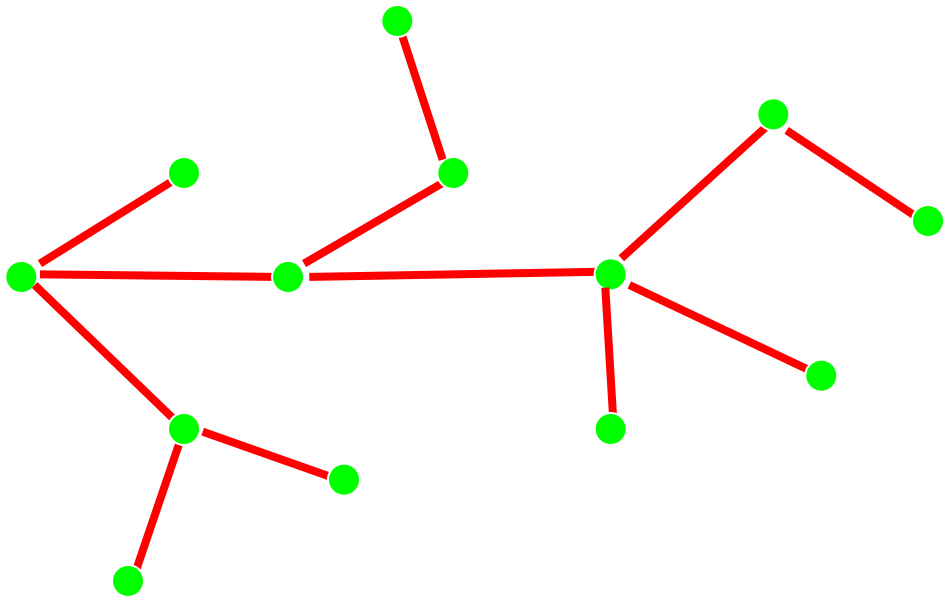
- 1) Поиск в глубину



Main

данный цикл необходим, если граф не связный

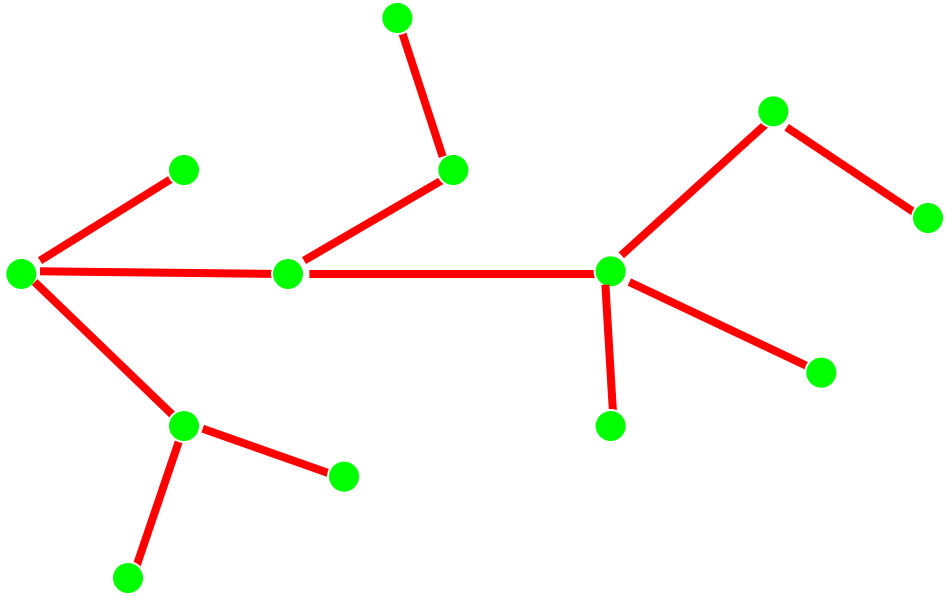
2) Поиск в ширину.



Поиск в ширину дает минимальный путь, если стоимость каждой дуги одинакова

7

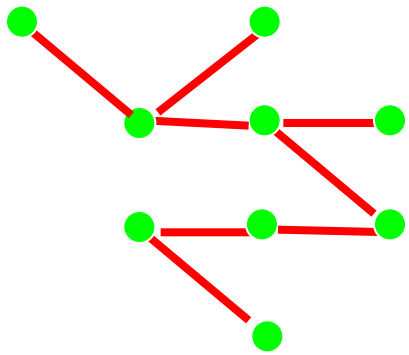
10 11 5 9 13 3



```
procedure PS(p:integer);
Begin
 Очередь:=0; Очередь<=p; New[V]:=False;
 While Очередь не пуста do
   Begin
    P<=Очередь;
    Обработать p;
    For i:=1 to Point[P,0]do
     Begin
      u:=Point[p,i];
      If New[u]
      then  Begin
              Очередь<=u;New[u]:=false;
            End
     End
  End
End;
```

3	2	4	12		
2	1	4			
1	7				
4	2	6	7	12	
3	6	8	9		
5	4	5	7	9	13
3	3	4	6		
2	5	9			
3	5	6	8		
2	11	12			
2	10	11			
4	1	4	10	11	
1	6				

Построение каркаса или остова

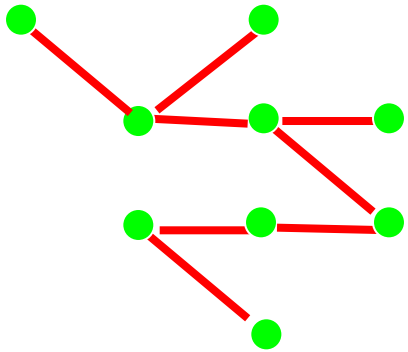


добавить в список T новую ветвь {v,u}

r-произвольная вершина

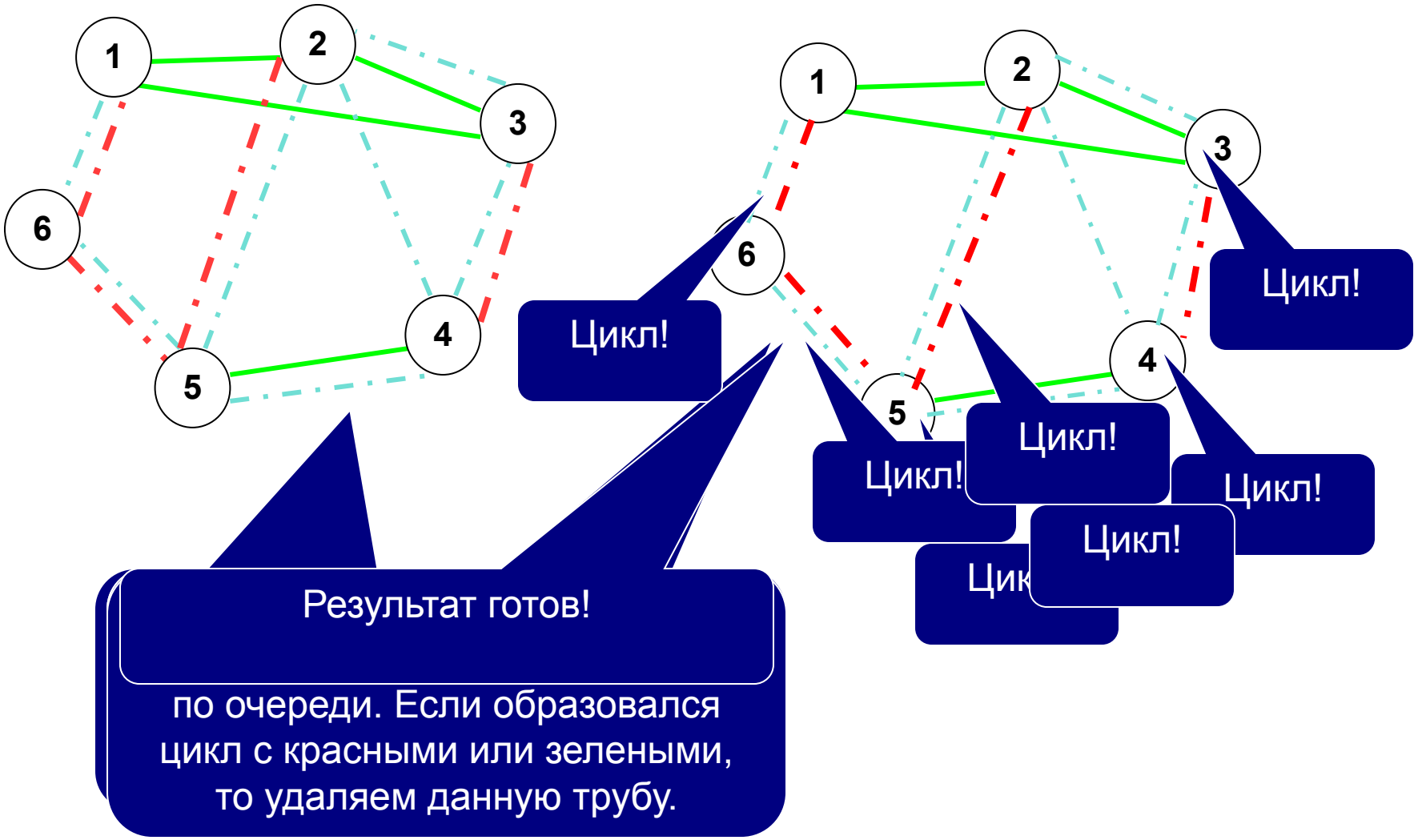
Построение каркаса или остова

```
procedure WGD(v:integer);
Var i, u: integer;
Begin
 New[v]:=false;
 For i:=1 to Point[v,0] do   Begin
   u:=Point[v,i];
   If New[u]
   then Begin
        inc(T[v,0]);T[v, T[v,0]]:=u;
        inc(T[u,0]);T[u, T[u,0]]:=v;
        WGD(u)
     End
 End {for}
End;
```



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

Производство

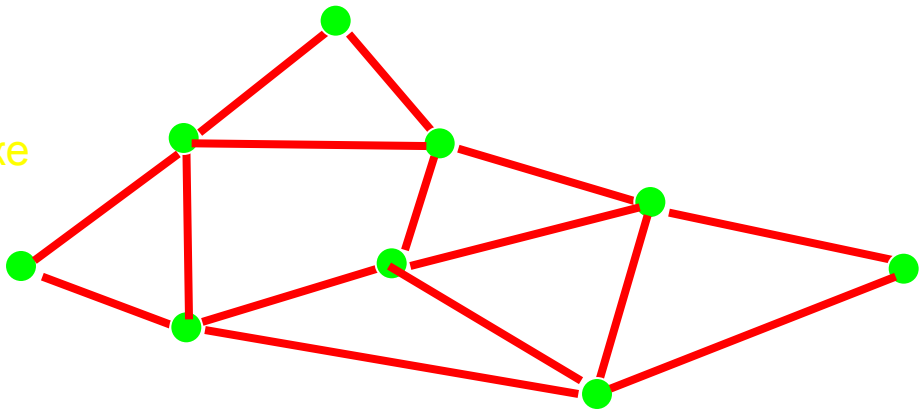

1
2
3
4
5
6
Цикл!
Цикл!
Цикл!
Цикл!
Цикл!
Цикл!
Цикл!
Результат готов!
по очереди. Если образовался цикл с красными или зелеными, то удаляем данную трубу.

Эйлеровы пути

ТЕОРЕМА ***Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более чем две вершины нечетной степени.***

элемент остается в стеке
есть ребра в вершине V

1-ая вершина в списке
удалить U и V из списка

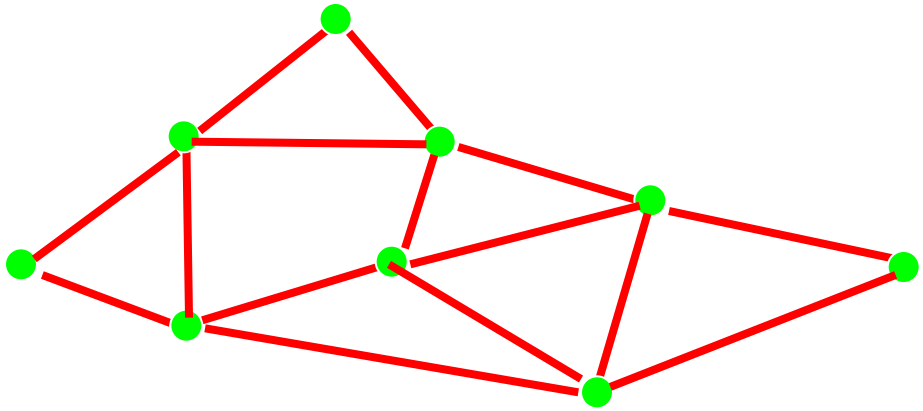


поместить V в стек CE

5
4
3
2

5

5



```
стек:=0;CE:=0;
V:= произвольная вершина графа нечетной
степени, если она есть;
Push(V);
While стек не пуст do    Begin
   V:=стек[top];{элемент остается в стеке}
   If Point[V,0]<>0 {есть ребра в вершине V}
   then Begin
          U:=Point[V,1];{1-ая вершина в списке}
          Push(U); {удалить U и V из списка}
          Point[V]:=Point[V]/{U};
          Point[U]:=Point[U]/{V};
          V:=U
        End
  else Begin
         V:=pop;  CE<=V {поместить V в стек CE}
       End
 End;
```

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

Гамильтонов цикл

Задача о коммивояжере

v-номер текущего города, Count - счетчик количества пройденных городов, Cost- стоимость текущего решения

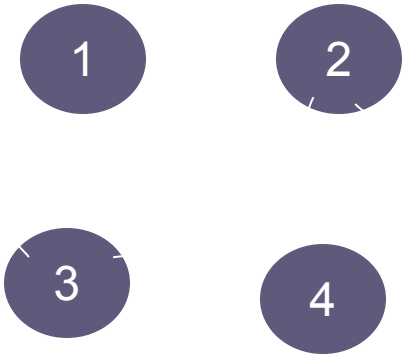
Стоимость текущего решения не больше лучшего

если город последний, то…

Последний город пути
Если путь дешевле, то запомнить его

Город v пройден, запомним его номер
пометить и запомнить город
перебираем все соседние непросмотренные города

возвращаем город v в число непройденных



	1	2	3	4
1		4	0	9
2	5		0	5
3	6	4		1
4	0	2	6	

V=1,Cost=0

V=2,Cost=4

V=3,Cost=0

V=4,Cost=9

Третье отсечение

V=3,Cost=4

V=2,Cost=4

V=4,Cost=1

V=4,Cost=9

V=2,Cost=3

Первое отсечение

V=4,Cost=5

V=4,Cost=9

V=1,Cost=8

Первая оценка

V=1,Cost=5

Второе отсечение

Вторая оценка

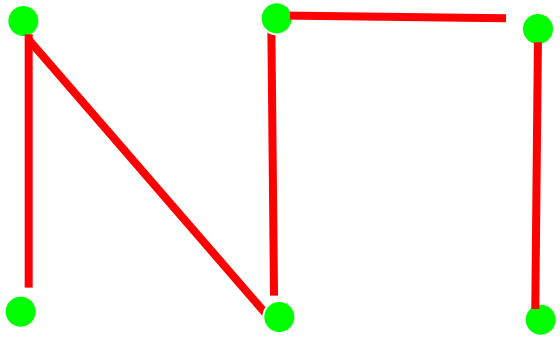
```
procedure Rec(v,Count:byte;Cost:longint
var i:integer;
Begin
  If Cost<=BestC
       If Count=
       then Begin
            Cost:=Cost+A[v,1];Way[N]:=v;
            If Cost<BestCost then Begin BestCost:=Cost; BestWay:=Way End
          End
      else Begin
            nNew[v]:=false;Way[Count]:=v
            For i:=1 to N do If nNew[i]  then Rec(i,Count+1,Cost+A[v,i]);
            nNew[v]:=true
          End
     End
End;
```

Нахождение кратчайших путей в графе

Нахождение кратчайших путей от фиксированной вершины. Алгоритм Дейкстры

```
Begin {main}
 For i:=1 to N do
    D[i]:=A[s,i];
 D[s]:=0;
 T:=[1..N]-[s];
 While T<>[ ] do
  Begin
   Min:=32000;
   For i:=1 to N do
     If (i in T) and (D[i]<Min)
     Then begin
             u:=i; Min:=D[i]
          end;
   T:=T-[u]
   For v:=1 to N do
     If v in T then
       D[v]:=min(D[v],D[u]+A[u,v])
    End; {While}
 End.
```

Далее действуем аналогично, пока все нерассмотренные города не закончатся

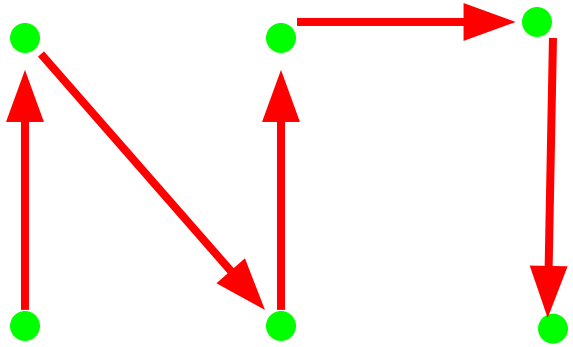


0	1	4	3	6	5

0	1	∞	∞	∞	∞
∞	0	5	2	∞	7
∞	1	0	∞	∞	1
2	∞	1	0	4	∞
∞	∞	∞	3	0	∞
∞	∞	∞	∞	1	0

Алгоритм восстановления кратчайшего пути по известным кратчайшим расстояниям

```
Begin {main}
 top:=0;   Push(t);   v:=t;
 While v<>s do
  Begin
   for i:=1 to N do
    if D[v]=D[i]+A[i,v]
    then begin
           u:=I; Push(u);
           v:=u; break
         end
   End;
End.
```



0	1	4	3	6	5

0	1	∞	∞	∞	∞
∞	0	5	2	∞	7
∞	1	0	∞	∞	1
2	∞	1	0	4	∞
∞	∞	∞	3	0	∞
∞	∞	∞	∞	1	0

Далее действуем аналогично

D[u]+A[u,v]=D[v]
Это вершина 3
D[3]=4, 4+1=5=D[6]

прямого пути между s и v.

1

1

1

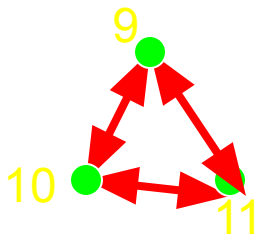
5 6 3 4 2 1

Кратчайшие пути между всеми парами вершин. Алгоритм Флойда

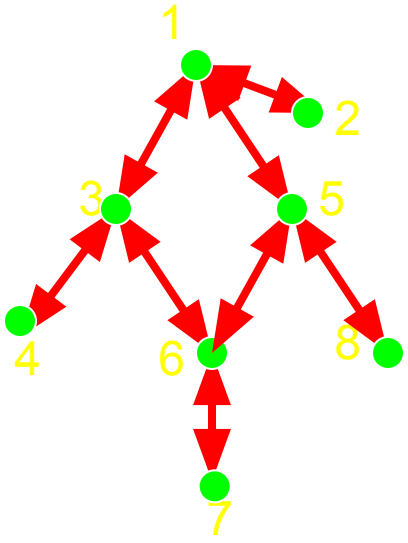
Переберем все пары вершин (i,j), для каждой пары выберем такую вершину m, путь через которую дешевле, чем прямой путь между I и j без захода в m. В отличие от Дейкстры, алгоритм Флойда позволяет использовать отрицательную стоимость дуг, но должен отсутствовать цикл отрицательной суммарной стоимости

Топологическая сортировка

вершина i еще не напечатана



идем до конца ветки, т.е. до листа
все вершины, в которые из i ведут стрелки,
уже напечатаны - так что можно печатать i,
не нарушая корректности

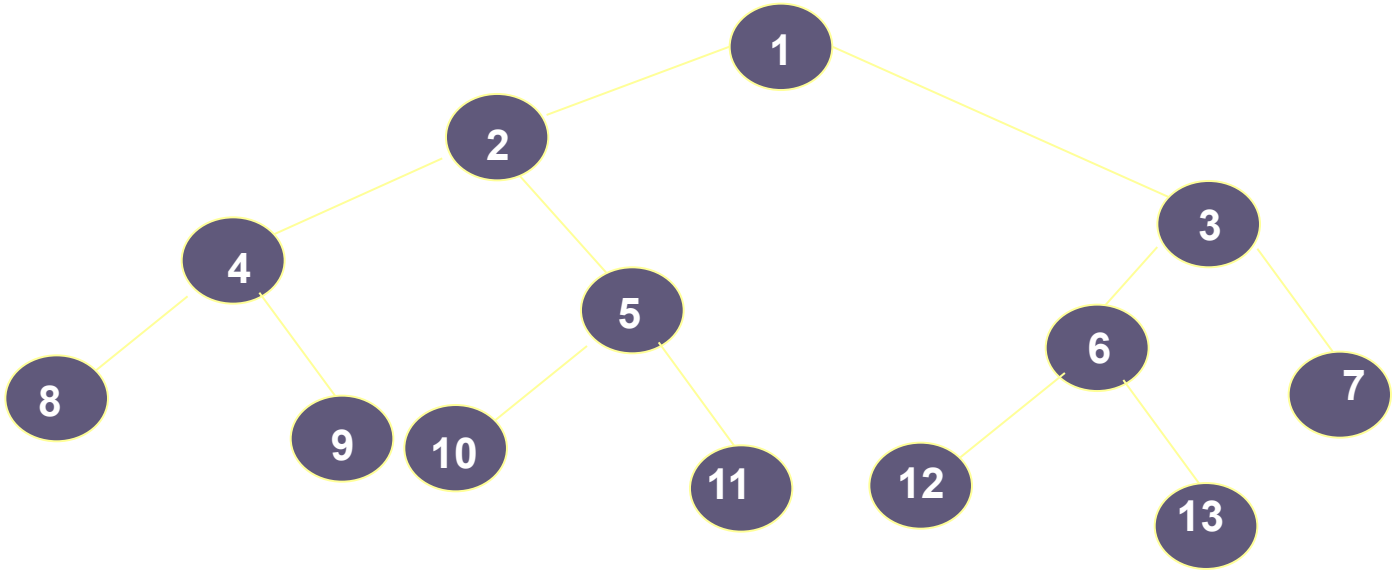


Основная программа:

7 6 4 3 8 5 2 1 11 10 9

Второй цикл необходим только в том случае, если существуют изолированные вершины или граф несвязный

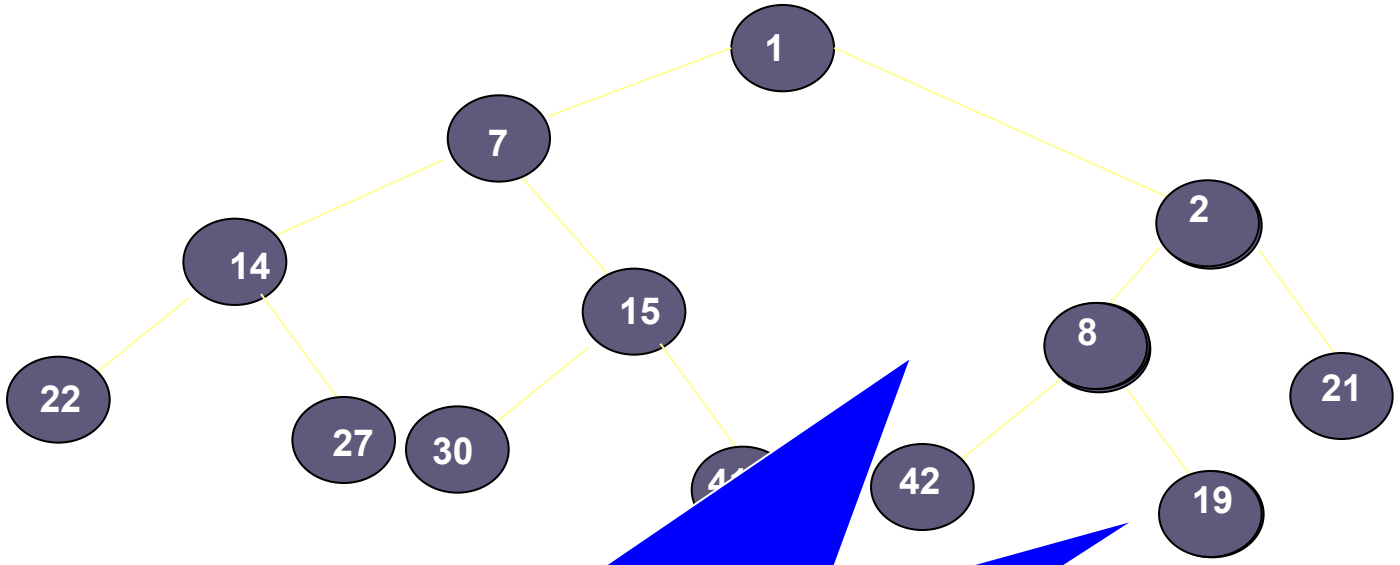
Очередь с приоритетом



- Свойство 1
- Свойство 2

- Свойство 3

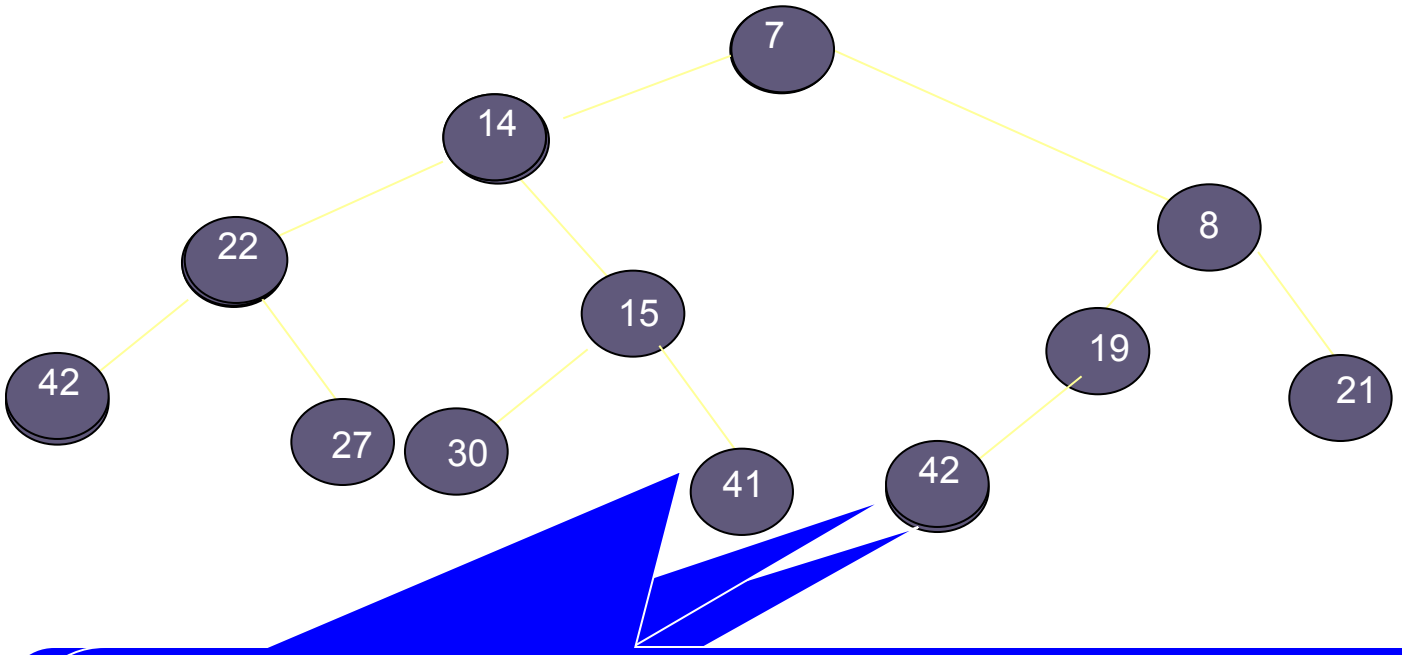
Добавление нового элемента в кучу



Это «всплытие» продолжается до тех пор, пока ключ объекта не станет больше (или равен) ключа его отца или пока объект не «всплывет» до самого корня дерева. Время работы операции INSERT прямо пропорционально высоте дерева - O(log N).

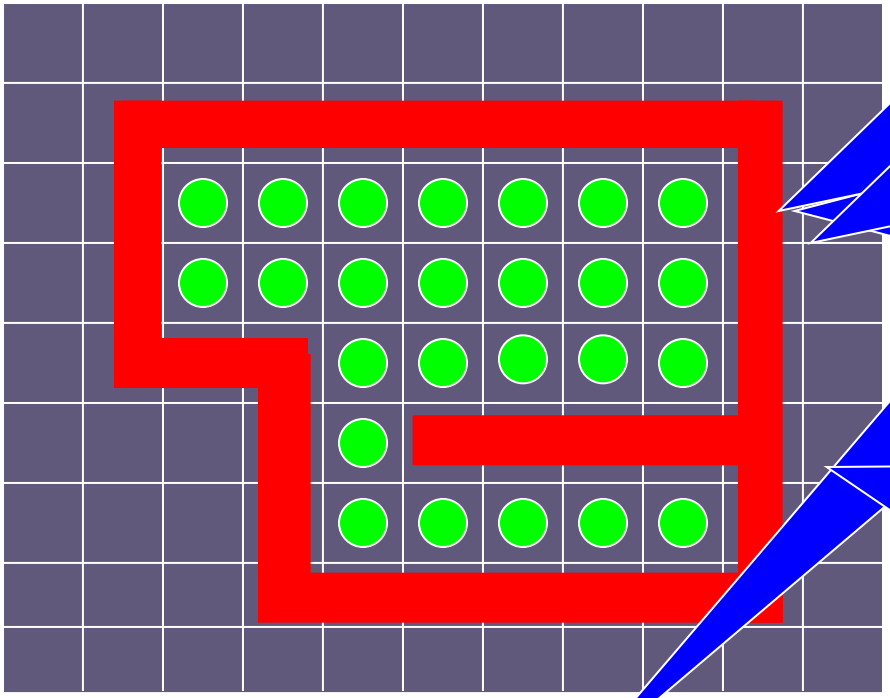
Что при этом может произойти?

Удаление минимального элемента из кучи



Если теперь окажется, что ключ объекта в корне меньше (или равен) ключей объектов в его сыновьях (что очень маловероятно), то свойство кучи нигде не нарушено и удаление было проведено корректно. В противном случае, выберем сына корня с минимальным значением ключа и поменяем объект в корне с объектом в этом сыне. В результате объект, находившийся в корне, «спускается» на одну позицию вниз.

Волновой алгоритм. Закраска замкнутых областей



Пока очередь не пуста:

Процесс продолжается аналогично для всех точек, находящихся в очереди. «Наткнувшись» на границу или уже покрашенную область, цветная волна останавливается, так как координаты точки не попадут в очередь

очередь:

4 7	4 8	5 7	4 6	3 7	4 9	5 8	3 8				

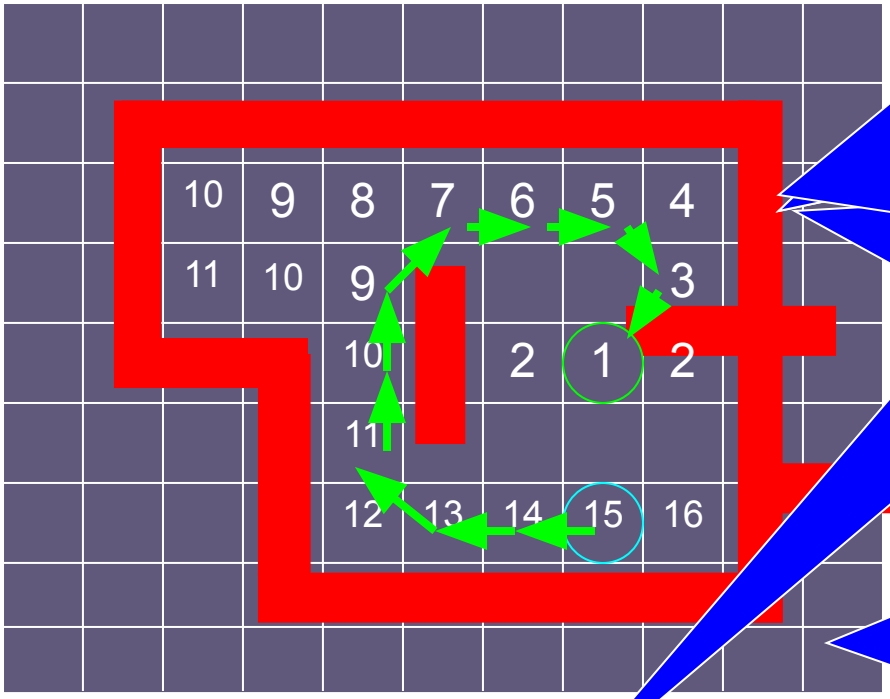
закрасить точку поместим ее в очередь
пока очередь не пуста
извлекаем из очереди очередную точку
если сосед слева существует
и не покрашен, то
покрасить его
поместить его координаты в очередь

если сосед справа существует
и не покрашен, то
покрасить его
поместить его координаты в очередь

если сосед сверху существует
и не покрашен, то
покрасить его
поместить его координаты в очередь

если сосед снизу существует
и не покрашен, то
покрасить его
поместить его координаты в очередь

Волновой алгоритм. Поиск пути в лабиринте



После заполнения матрицы A, путь восстанавливается элементарно: встаем в точку с координатами старта Si, Sj (голубая окружность). Рассматриваем соседние точки и ищем минимум. (Красные стены имеют код 255). Минимум=14, смещаемся в эту клетку и повторяем процесс пока не дойдем до точки финиша.

помещаем ее координаты в очередь:

5	5	5	4	3	3	3					
8	9	7	9	9	8	7					

пометим точку поместим ее в очередь
пока очередь не пуста
извлекаем из очереди очередную точку
если сосед слева существует
и не помечен, то
пометим его
поместить его координаты в очередь

если сосед справа существует
и не помечен, то
пометим его
поместить его координаты в очередь

если сосед сверху существует
и не помечен, то
пометим его
поместить его координаты в очередь

если сосед снизу существует
и не помечен, то
пометим его
поместить его координаты в очередь

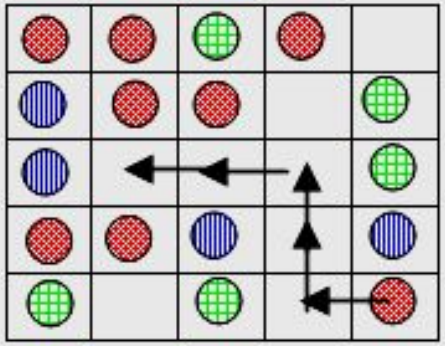
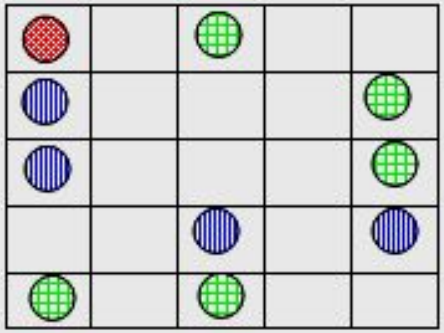
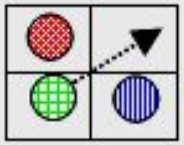
Lines (20 баллов)

- Задание: напишите программу, которая по заданному состоянию игрового поля, начальным и конечным координатам шарика, определит состояние игрового поля после перемещения шарика и возможного удаления образовавшихся одноцветных «линий».

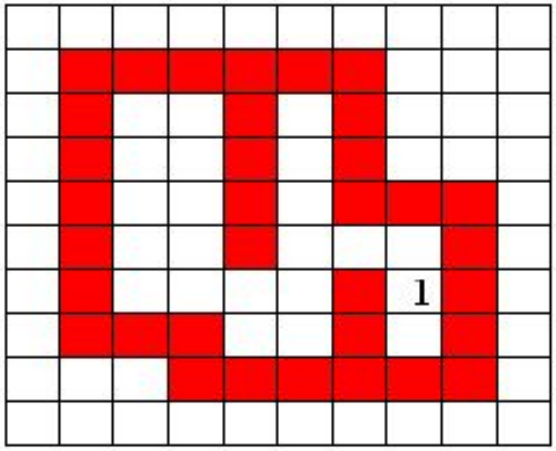
- **Формат входного файла input.txt:**

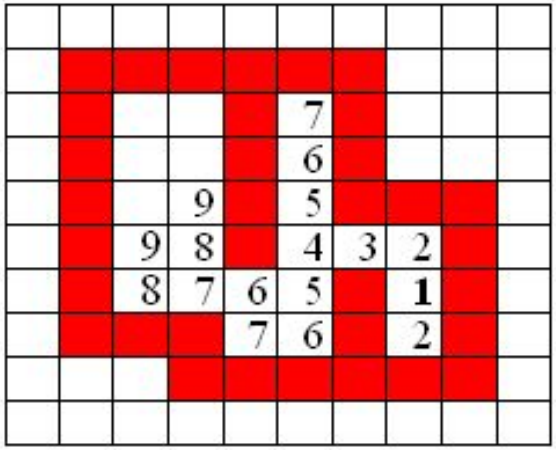
- **Формат выходного файла output.txt:**

- **Замечания**

Input.txt	Output.txt
5 4 11210 31102 30002 11303 20201 5 5 3 2 	10200 30002 30002 00303 20200 
2 2 10 23 2 1 1 2 	**NO SOLUTION**

- Решение


1


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

```
    максимальный размер поля

    игровое поле
     матрица для поиска пути

                                  ввод данных

                                заполним препятствиями


{считаем цвет шарика и заполним матрицы

число 65535, а не 255, используется из-за больших размеров поля
```

создадим файл

{если путь перемещения существует, то выведем результат

вектор направлений движения шарика

очередь

поместить в очередь

извлечь из очереди

проверка на попадание в поле

поиск пути шариком

инициализация очереди

{пока очередь не пуста

извлечем из очереди координаты очередной клетки

{переберем 4-х соседей, для каждого
проверим что он существует и не содержит шарика

{в этом случае пометим его и запомним в очереди

построим путь

если добрались до конечной точки

процедура удаляет одноцветные линии
ti,tj-*начальное положение,* dj,di-*вектор смещения*

считает длину одноцветной линии

встали в начало
{пока цвет совпадает – бежим «вправо»
{«вправо» - условно, т.к. все зависит от значений di, dj

{теперь – «влево»}

пощем путь

```
{если нашли, то
    {переберем 4 возможные линии
```

{если линия существует

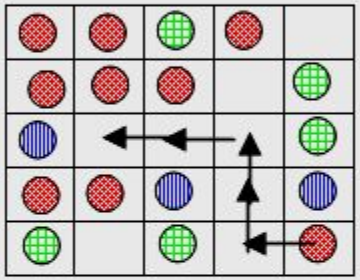
```
то удалим ее
```

удалим шарик, если линии были

Lines-2 (30 баллов)

Формат входного файла input.txt:

Формат выходного файла output.txt

Input.txt	Output.txt
5 3 5 5 11210 11102 30002 11303 20201 	3 2 6 {при перемещении шарика в данную клетку образуются четыре фигуры: вертикальная линия (4 очка), диагональная (4). Диагональный крест (min(2,4)=2*2=4) и обычный крест (min(4,3)=3*2=6). Выбираем максимум} Позиция (5,2) недоступна, а (2,4) дает 4 очка
4 3 1 1 1020 0020 2220 1110	NO SOLUTION

проверяет допустимость хода

ход возможен, если целевая клетка пуста, а рядом с ней есть шарики «нашего» цвета

поместить координаты в очередь

извлечь координаты из очереди

координаты в игровом поле

{ищет достижимые клетки от активного шарика

{пока очередь не пуста

извлечем из очереди очередную клетку
{переберем ее соседей

{если сосед на поле и там нет шарика, то

пометим его как рассмотренного и запомним в очереди

ищет все допустимые для хода клетки,
использует волновой алгоритм

вектор возможного перемещения

possible

удаляет одноцветные линии в заданном направлении

считает количество одноцветных шариков
в заданной линии

лучший вариант}

находит минимум из 2-х}

поиск обычного креста

поиск креста из диагоналей

MaxCalc

Solve

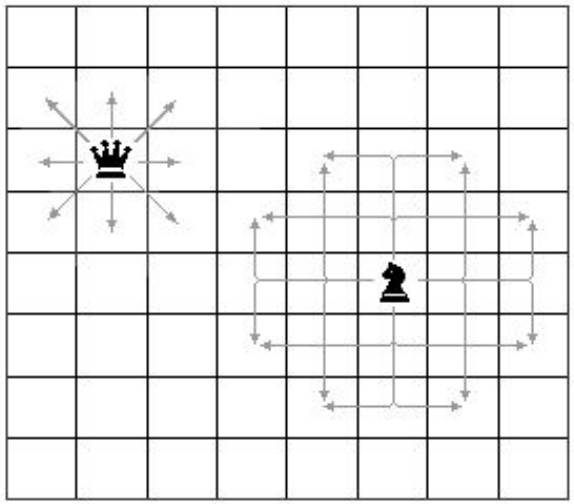
уберем активный шарик со стартовой позиции
если можем попасть в клетку с координатами (i,j)

вспомним состояние поля – его будем портить
если имеет смысл туда прыгать, то

перемещаем туда шарик
считаем «прибыль»
{*если лучше, чем раньше, то запомним это*

восстановим поле

Задача C. «Камелот»



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

Числа в матрицах показывают за сколько ходов (+1) фигура сможет добраться до данной клетки. Найдем сумму элементов матрицы, а среди этих сумм – минимум. Эта клетка и будет точкой встречи.

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

Идея – ищем путь от короля до всех клеток, то есть, считаем, за сколько ходов король сможет добраться до этой клетки, затем делаем это же для коня. Складываем две матрицы ходов и ищем наименьшее значение – это и будет точка встречи.

ходы коня

ходы короля

пометить и поместить в очередь стартовую точку
пока очередь не пуста

извлечь из очереди очередную точку
проверяем 8 соседей

если ход в пределах доски и клетка не помечена
помечаем большим числом
помещаем ее координаты в очередь

ищем минимум среди суммы матриц

-2, так как начинали заполнять матрицы с 1

ввод из файла условий

считаем для коня

считаем для короля

находим расстояние

проверяем случай, когда конь не ходит - конь и король стоят рядом

Задача B. «Гонки в лабиринте»

Описание входного файла input.txt

Описание выходного файла output.txt

Замечания

Input.txt

```
6 5   {размер поля}
1 1   {координаты входа в лабиринт}
6 2   {координаты выхода}
0   1   255 3   255
255 0   0   0   2
255 255 255 255 255
0   0   0   255 0
3   255 1   255 2
0   0   0   255 0
```

Output.txt

4
RTDL

0 →	1	255	3	255
255	0	0	0	2
255	255	255	255	255
0	0	0	255	0
3	255	1	255	2
0	0 ←	0	255	0

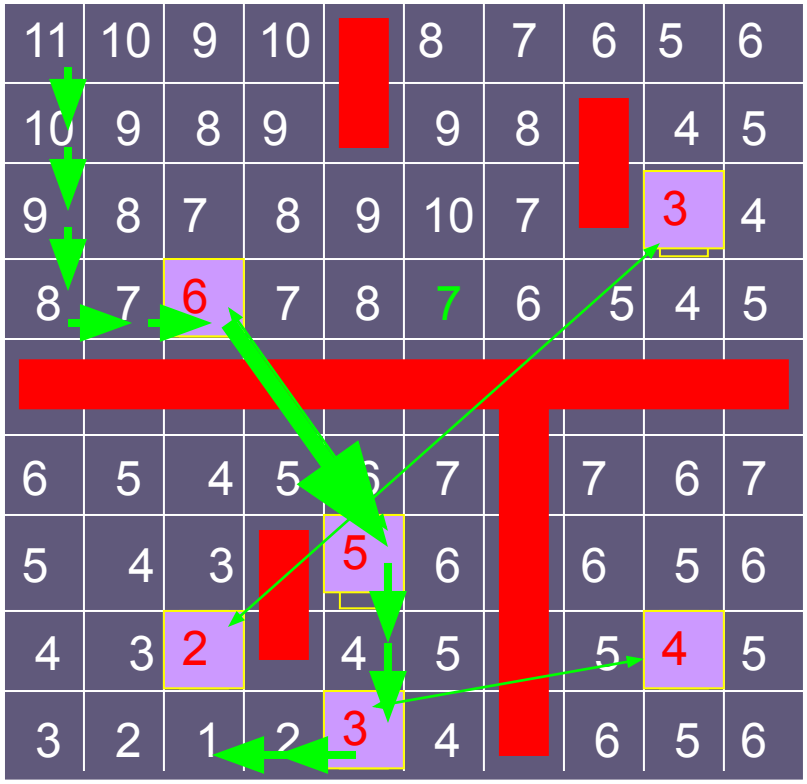
Input.txt

```
5 5
1 1
5 5
0 255 1 255 0
0 255 0 255 0
0 255 2 255 0
0 255 0 255 0
1 255 2 255 0
```

Output.txt

NO SOLUTION

0	255	1	255	0
0	255	0	255	0
0	255	2	255	0
0	255	0	255	0
1	255	2	255	0



Дошли до точки старта. Ответ:
Res='DDDRRTDDLL'

(4,3)	(3,9)	(8,9)	
(7,5)	(8,3)	(9,5)	

Структура данных

максимальное количество телепортов

игровое поле

{описание одного входа в телепорт

описание телепортов

очередь

массив для результата

он еще и динамический

игровое поле

результирующий путь

варианты смещений Алисы

Волновой алгоритм поиска

стоимости кратчайшего пути

инициализируем очередь
поместим точку куда надо попасть в матрицу и очередь

пока очередь не пуста

извлечем координаты очередной точки
перебираем 4 направления движения

если там не были

пометим, что можем туда попасть
за w ходов, поместим в очередь
{переберем телепорты
если нашли, то попробуем

воспользуемся телепортом

дальше нет смысла искать

Загрузка матрицы поля

заполним массив a числами 65535
откроем файл считаем размер поля
считаем координаты точки старта
считаем координаты точки финиша
считаем поле}

если не пусто и не препятствие
то - телепорт
сохраним его
0 строка -вход
1 *строка - выход*

если препятствие, то пометим его 65535

2{*количество телепортов*

Восстановим путь

восстанавливает путь в матрице

если до точки старта «волна» не дошла, то}
решения нет

пока не дошли до финиша

ищем вокруг себя наименьшее число

если меньшего нет, значит был использован телепорт

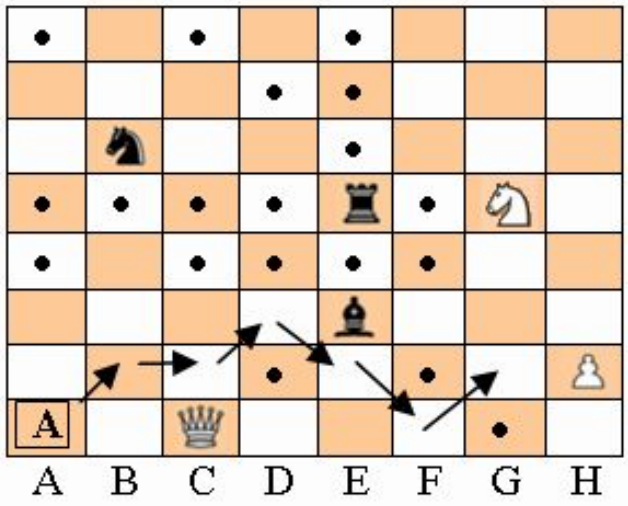
найдем его номер

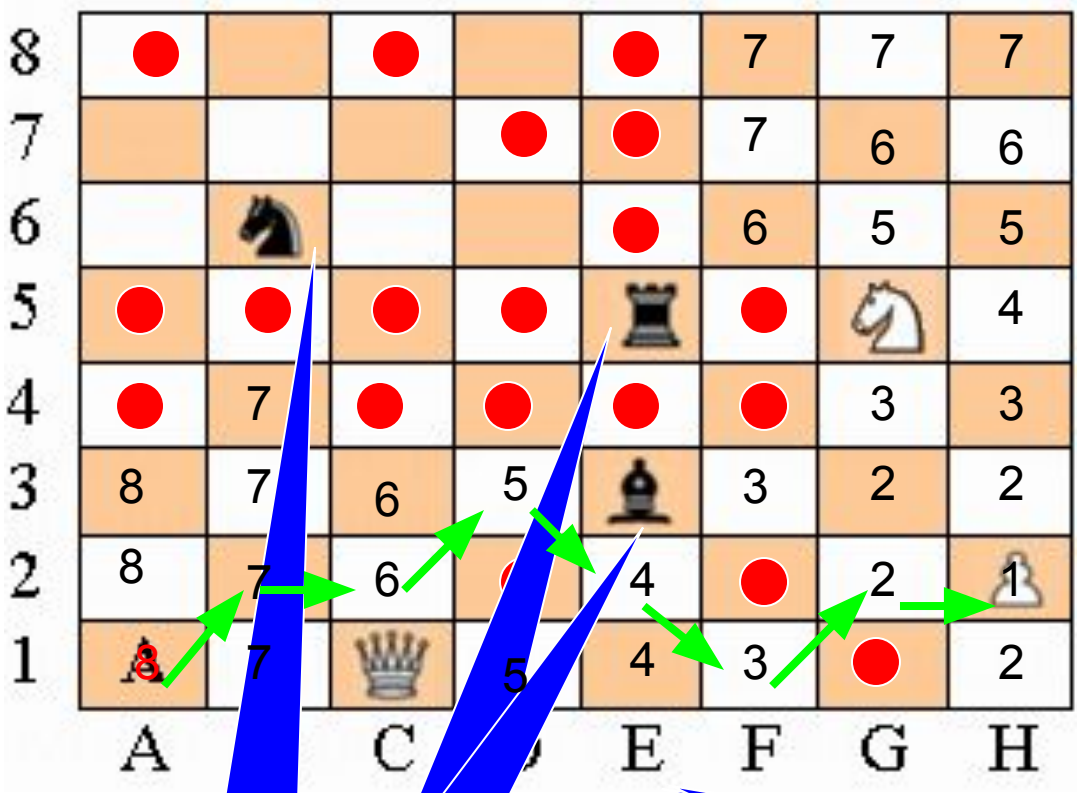
теперь координаты выхода

увеличим длину пути на 1
запомним ход *сместимся в эту клетку матрицы*
если это финиш, то выход

Задача C. Шахматная партия Алисы

Input.txt	Output.txt
A1 H2 5 BLD E5 BSN E3 WFZ C1 WKN G5 BKN B6	6 B2 C2 D3 E2 F1 G2 6






```
Const BKN=240;
BSN=241; BLD=242;
BFZ=243; WKN=244;
WSN=245; WLD=246;
WFZ=247; WP =248;
```

```
tFig=record{фигура}
        typ:byte;     {тип}
        i:byte;{координаты}
        j:char;
      end;
tFigs=ar[1..100] of tFig;
```

Черный слон

Если после волнового заполнения матрицы клетка с Алисой осталась с 0, то решения нет (Алисе не добраться до пешки). Если 8-ая параллель в столбце белой пешки осталась равной 0, то решения нет (пешке не пройти). В противном случае восстанавливаем путь Алисы, двигаясь в минимальном направлении

загрузим данные

откроем файл

считаем координаты Алисы

считаем координаты пешки

считаем количество фигур

загрузим очередную фигуру

определим, какая она

запомним ее координаты

Пометим биты клетки ладьи

помечает «битые клетки» ладьи

{*«бежим» вверх до препятствия*

{*«бежим» вниз до препятствия*

{*«бежим» вправо до препятствия*

{*«бежим» влево до препятствия*

Пометим биты клетки слона

помечает «битые клетки» слона

Пометим биты клетки коня

помечает «битые клетки» коня

Пометим биты клетки ферзя

помечает «битые клетки» ферзя

Волновой поиск пути

поиск минимальной стоимости пути от Алисы до пешки

инициализация очереди

помечаем точку финиша 1

{пока очередь не пуста

извлекаем координаты точки из очереди

{перебираем 8 направлений движения

{если ход возможен и там небыли

то помечаем клетку числом +1 и помещаем ее коорд.в очередь

Восстановим путь

восстановим путь Алисы

пока не дошли до пешки

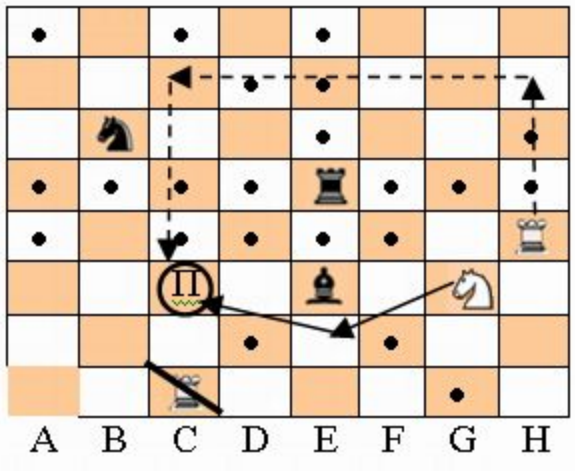
{переберем 8 вариантов направления
{если пришли от туда, то

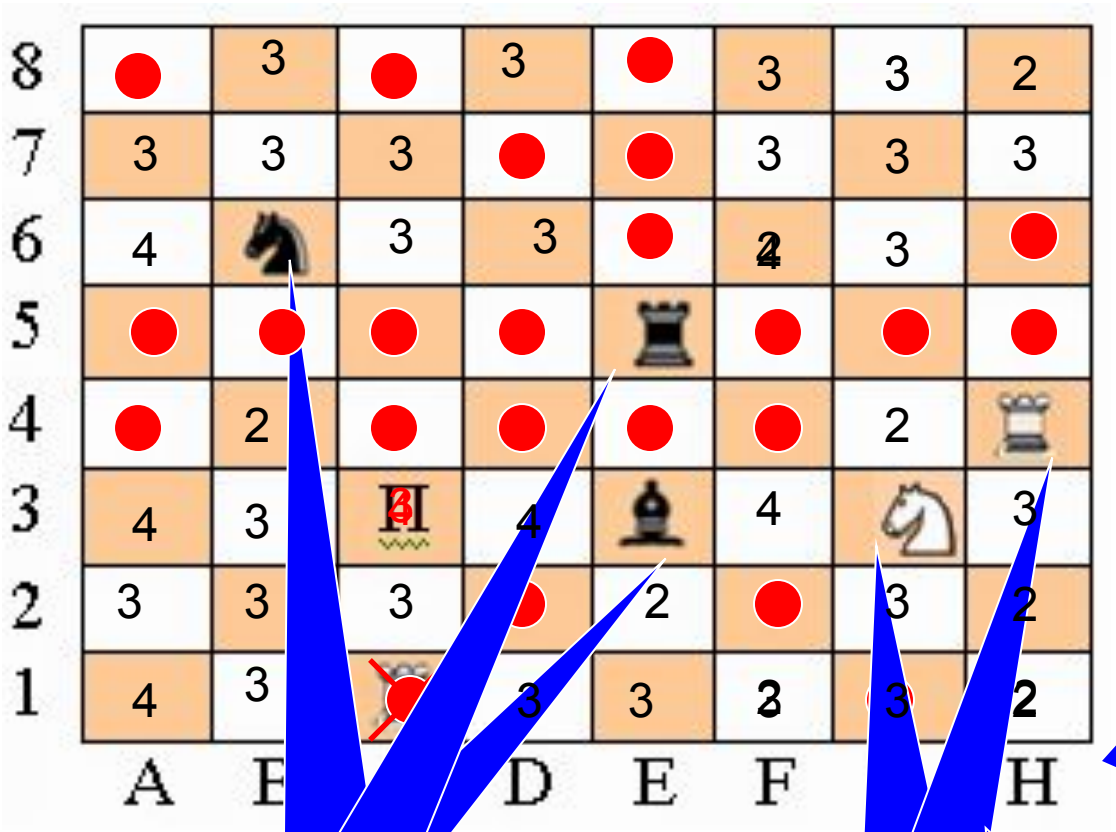
смещаемся туда

приклеим результат к строке

удалим последние координаты

Задача B. Верхом на шахматной фигуре

Input.txt	Output.txt
C3 6 BLD E5 BSN E3 WLD H4 WKN G3 BKN B6 WLD C1 	2 G3



Этап первый: загрузка матрицы поля, выделение черных и белых фигур

Этап второй: трассировка битых полей черных фигур, исключение битых белых фигур

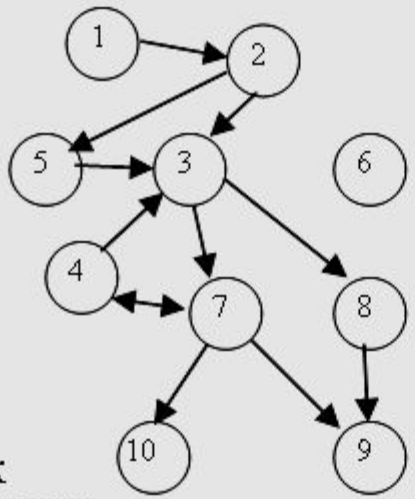
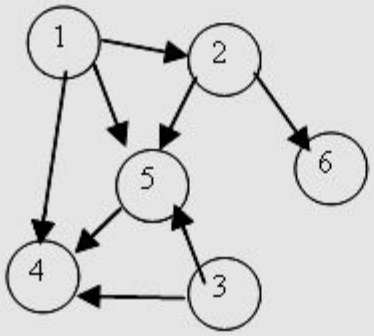
Этап третий: поиск модифицированным волновым алгоритмом кратчайшего расстояния от каждой «небитой» белой фигуры до пещеры

Черная ладья

Белая ладья

Для каждой фигуры запоминаем количество ходов, за которые она может добраться до пещеры и ищем среди них минимум

Спам

Input.txt	Output.txt
10 кол-во компьютеров 1 2 ПК №1 связан с ПК №2 2 3 5 ПК №2 связан с №3,5 2 8 7 2 3 7 1 3 0 3 10 9 4 1 9 0 0 1 2 3 4 5 6 7 8 9 10 порядок выключения ПК 	1 9 9 9 9 6 9 9 9 10
6 3 2 5 4 2 5 6 2 4 5 0 1 4 0 5 2 3 1 4 6 	4 6 4 4 5 6

Структура данных

Ввод данных

Считаем количество ПК-вершин в графе

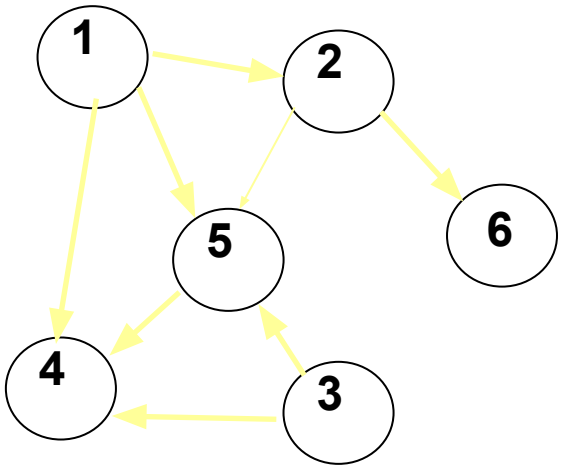
Для каждой вершины графа

Считаем количество вершин, непосредственно связанных ней

Повторяем нужное количество раз

Считаем номер очередного ПК

упорядочим номера ПК по убыванию



3	5	4	2	
2	6	5		
2	5	4		
0				
1	4			
0				

для каждого выключаемого ПК

считать номер выключаемого ПК

запомним состояние сообщений

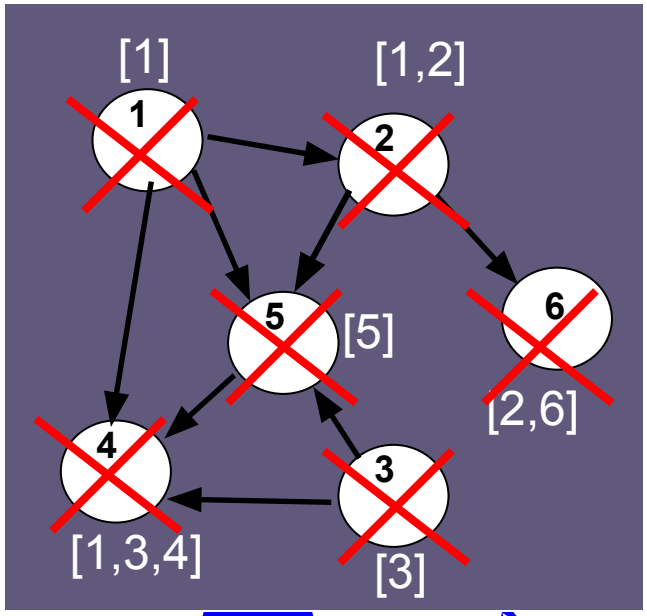
выключим ПК – теперь он не рассылает сообщения

переберем все ПК

если ПК еще не выключен

найдем номер ПК, кому посылать сообщения

если нашли ПК, то «сливаем» ему все, что у нас есть



3	5	4	2	
2	6	5		
2	5	4		
0				
1	4			
0				

5 2 3 1 4 6

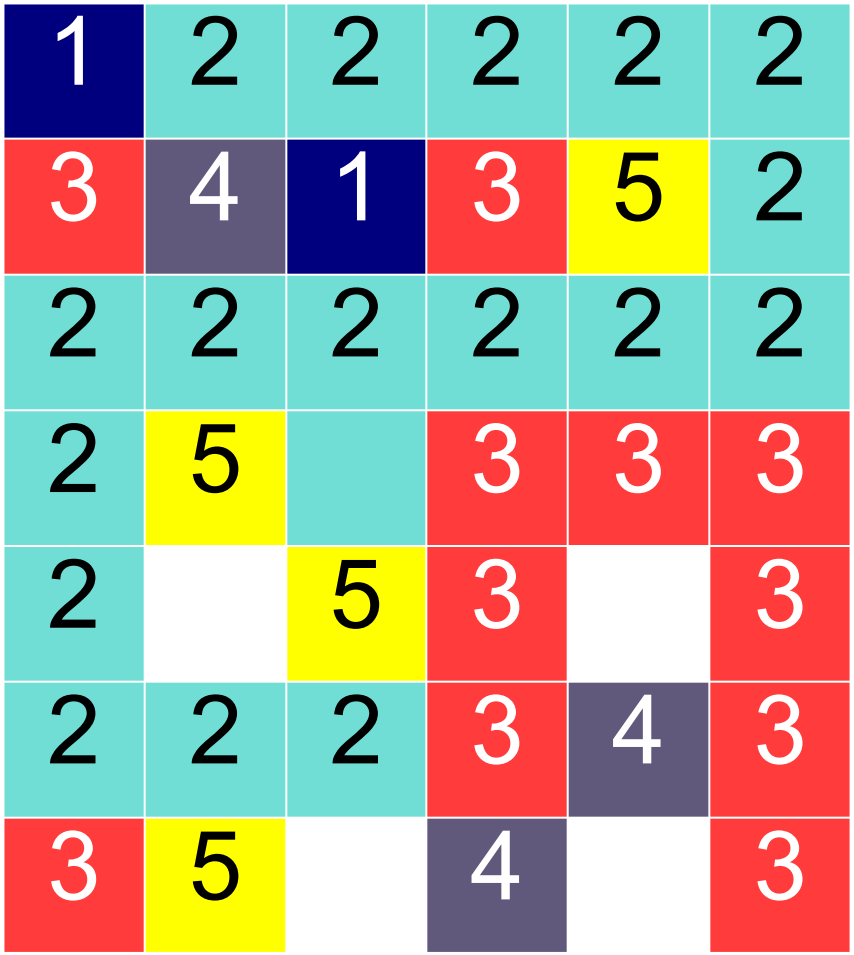
Теперь надо восстановить ответ. Для этого перебираем ПК в обратном порядке из выключения: 6, 4, 1, 3, 2, 5. и ПК в порядке возрастания их номеров. Тот ПК, который содержит сообщение и имеет минимальный номер и является ответом для спам сообщения

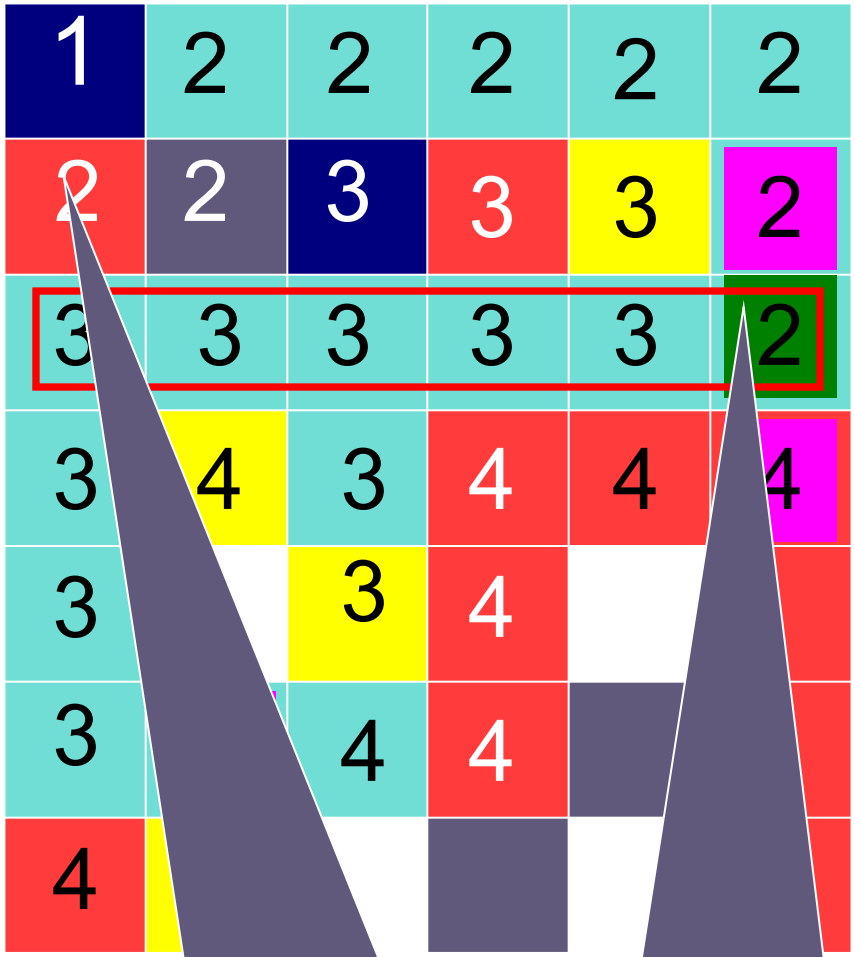
Задача B. Путешествие на хамелеоне

- **Формат входного файла input.txt**

- **Формат выходного файла output.txt**

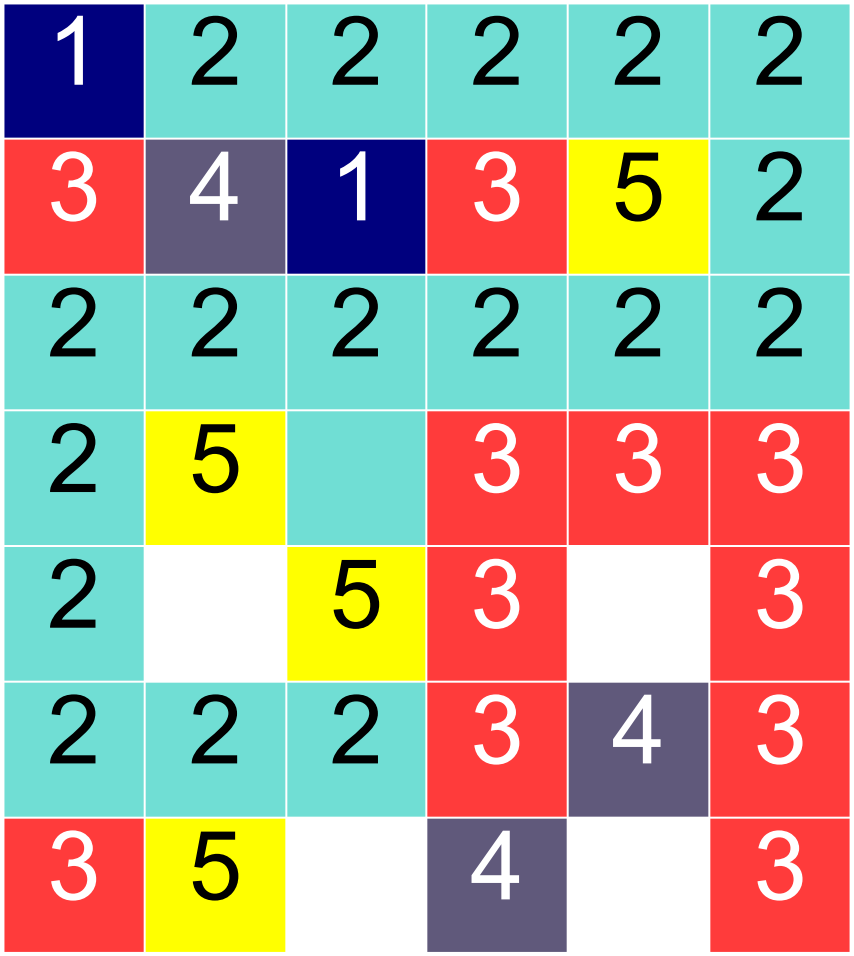
Input.txt	Output.txt
5 4 1 1 1 2 2 1 2 3 3 1 2 3 1 1 3 3 1 1 1 3	1
5 4 1 2 3 4 2 2 3 4 3 3 3 4 4 4 4 4 5 5 5 5	4

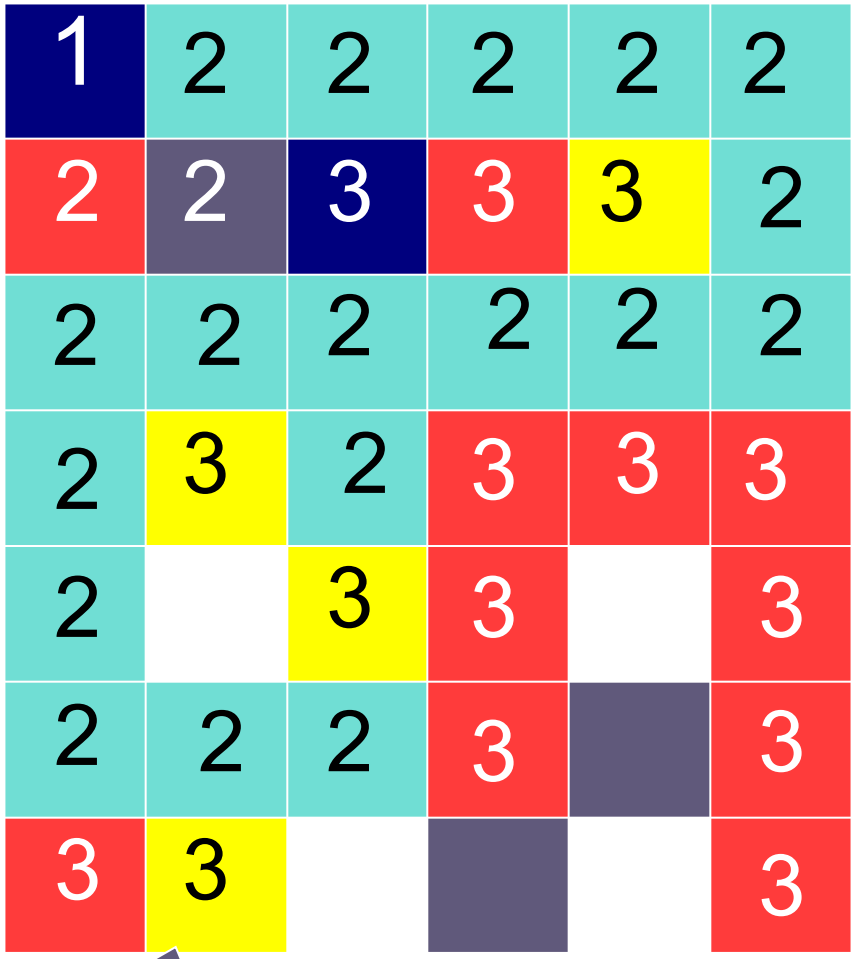




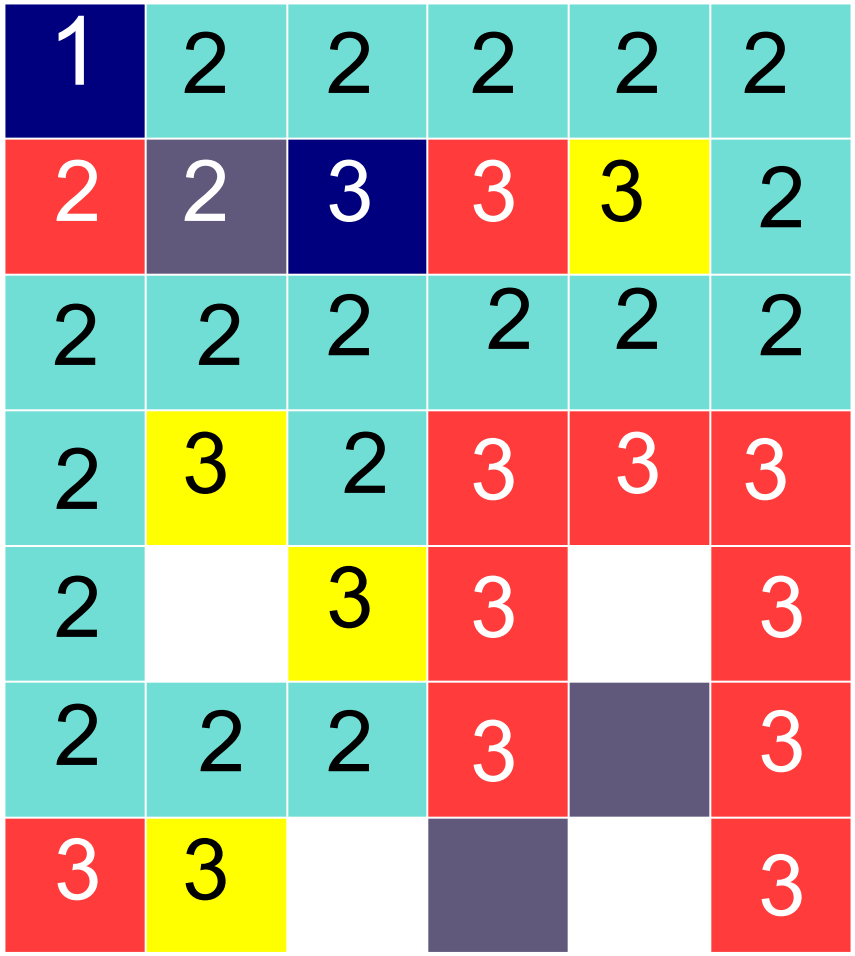
Почему не обход в ширину?

Конфликт! Все поле «левее» необходимо пересчитать, но этих ячеек нет в очереди!





Из каждой помеченной клетки пытаемся рекурсивно перекрасить все одноцветные клетки, помечая их числом, равным текущему


1 2 2 2 2 2
2 2 3 3 3 2
2 2 2 2 2 2
2 3 2 3 3 3
2 3 3 3
2 2 2 3 3
3 3 3

Задача D. 38 попугаев

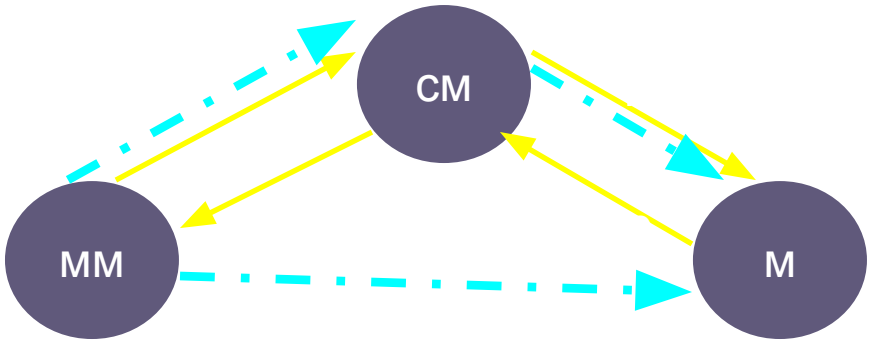
- **Формат входного файла input.txt**

- **Формат выходного файла output.txt**

Input.txt	Output.txt
4 4 3 Кролик 10 см Алиса 2 м Болванщик 38 попугай ЧК 100 мм 1 см=10 мм 1 м=100 см 1 м=1000 мм 1 мартышка=10 попугай Кролик Алиса Кролик ЧК Болванщик ЧК	< = ?
6 5 5 Krolik 10 sm Alisa 20 sm CK 100 mm BK 1 sm Barmoglot 1000000 sm Bolv 200 mm 1 sm=10 mm 1 m=100 kuku 1 m=1000 mm 1 km=1000 m 1 mart=10 pop Barmoglot BK Krolik Alisa Krolik CK Bolv CK Alisa Bolv	> < = > =

Input.txt

```
4 4 3
Кролик 10 см
Алиса 2 м
Болванщик 38 попугай
ЧК 100 мм
1 см=10 мм
1 м=100 см
1 м=1000 мм
1 мартышка=10 попугай
Кролик Алиса
Кролик ЧК
Болванщик ЧК
```



мм	см	м	мартышка	попугай
1	10	0	0	0
1/10	1	1/100	0	0
0	1/100	1	0	0
0	0	0	1	10
0	0	0	1/10	1

Можем представить ситуацию графом, где вершины – это единицы, а дуги – соотношение между ними. Тогда, если мы сможем добраться из вершины ‘мм’ в ‘м’, то мы сможем установить соотношение между ними

{добавляет имя в список имен}

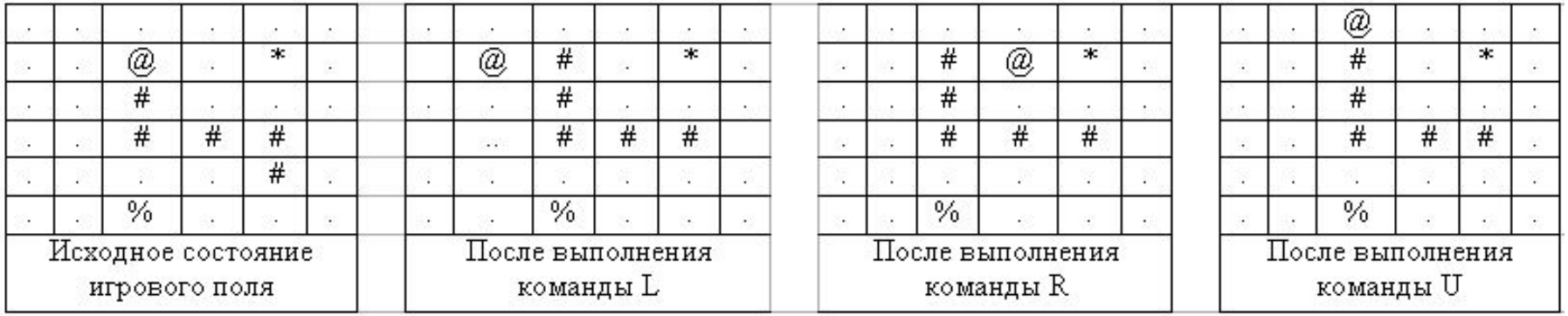
{ищет имя и возвращает его индекс}

{строит таблицу соотношений}
{между различными единицами}

Задача установления пути между всеми парами вершин решается алгоритмом Флойда, однако на практике он не работает, так как ищет не путь, а наименьший путь и из-за падения точности все значения обнуляются

if W^[i, j]=0 then

Задача A. Волшебная Змейка


. . . . . .
. . @ . * .
. . # . . .
. . # # # .
. . . . # .
. . % . . .
Исходное состояние игрового поля
. . . . . .
. @ # . * .
. . # . . .
. .. # # # .
. . . . . .
. . % . . .
После выполнения команды L
. . . . . .
. . # @ * .
. . # . . .
. . # # # .
. . . . . .
. . % . . .
После выполнения команды R
. . @ . . .
. . # . * .
. . # . . .
. . # # # .
. . . . . .
. . % . . .
После выполнения команды U

Input.txt	Output.txt
1 {хвост будет расти} 5 5 {размер поля} ***** {игровое поле} *...* *...* *...* ****@ 15 {количество команд в программе} LLLLUUUURRRRDDD {программа}	15
1 7 10 ....%..... .******... .******... .*.***.... ....*..... .*...*.... .*.@...... 34 LLUUUUURDLURRRRRDLLLDRRRUUULLLLLLU	7
0 7 10 ....%..... .******... .******... .*.***.... ....*..... .*...*.... .*.@...... 34 LLUUUUURDLURRRRRDLLLDRRRUUULLLLLLU	18

{считаем размеры поля}

{считаем само поле}
{если это голова Змеки, то запомним ее координаты}

{считаем направление движения Змейки}

{инициализируем очередь}

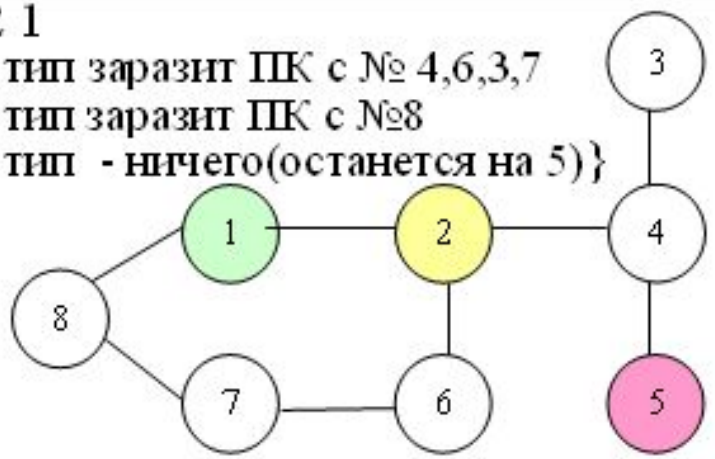
{зададим вектор направления движения Змейки}

{посмотрим, что впереди Змейки}
{Если камень или собственное тело, то - выход}
{Если пусто, то вместо головы – тело}
{новое «звено» - в очередь}
{голову - вперед}
{удалим «конец» хвоста}

{если яблоко, то:}
{увеличим количество яблок}
{если режим 0, то тело не удлиняется}

{сдвинем координаты головы}
{поместим голову в матрицу}

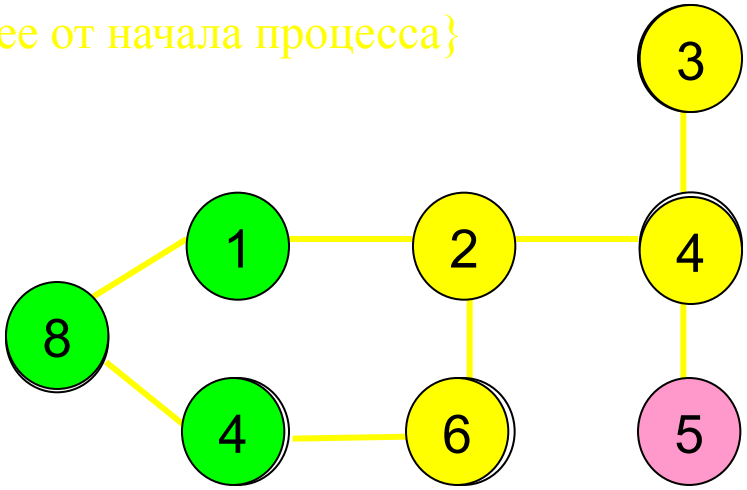
Задача B. Компьютерный вирус

Input.txt	Output.txt
8 3 {8 ПК и 3 типа вируса} 2 1 5 {1 тип вируса -2 ПК,2 тип – 1 ПК} 2 2 8 {1 ПК связан с двумя ПК: №2 и 8} 3 1 6 4 {связи не всегда действуют } 1 4 { в 2-х направлениях: если 1 связан } 3 3 2 5{со 2,то 2 не обязательно связан с 1} 1 4 2 2 7 2 6 8 2 7 1	5 2 1 {1 тип заразит ПК с № 4,6,3,7 2 тип заразит ПК с №8 3 тип - ничего(останется на 5)} 

{для каждого типа вирусов будет своя очередь}

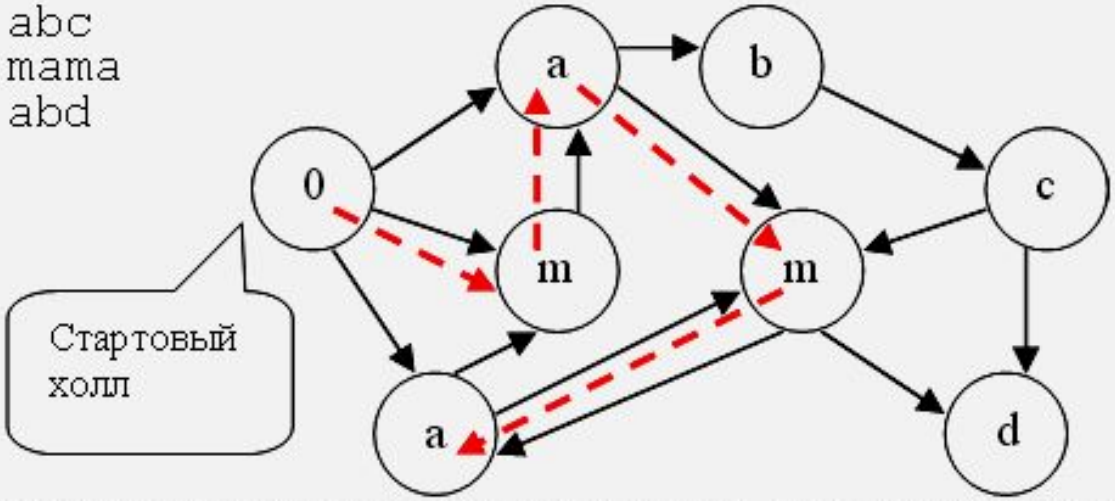
{время, прошедшее от начала процесса}

{переберем все вирусы}



Задача D. Темный лабиринт (30 баллов)

input.txt	output.txt
7 3 {количество залов и слов} amambdc {буквы, соответствующие залам} 3 1 2 3 {стартовый холл связан с тремя залами 1, 2, 3} 2 5 4 {1 зал связан с двумя: с 4 и 5} 1 1 {2 зал связан с 1} 2 2 4 2 3 6 2 4 7 0 {из 6 зала нет выхода в другие залы} 2 6 4 abc mama abd	yes {abc} yes {mama} no {abd}



Пунктирными стрелками показан путь составления слова ‘mama’

```
Procedure Rec(v,L,len:byte;var
   Res:boolean);
var i,k:byte;
begin
   if L>len then res:=true
   else begin
          for i:=   to a[v,0] do
            begin
              k:=  [v,i];
              if   it[k]=st[L])and NewF[k]
              the  begin
                     NewF[k]:=false;
                     Rec(k,L+1,len,res);
                     NewF[k]:=true;
                     d
            end
        end;
end;
```

L>len, слово найдено