ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

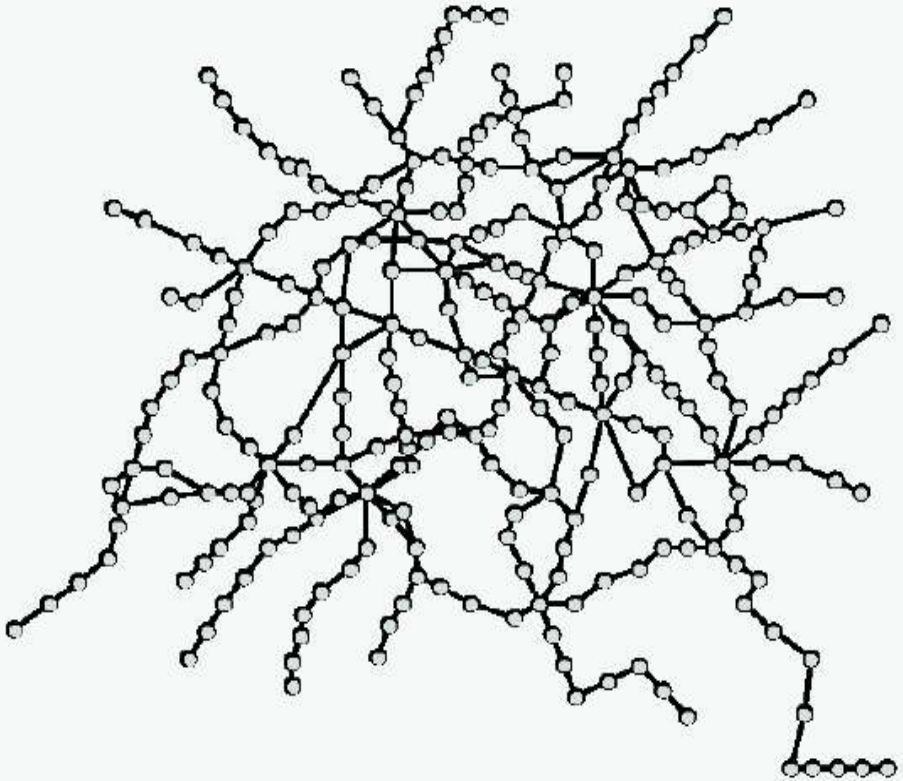
ГРАФОВИ
- вовед -

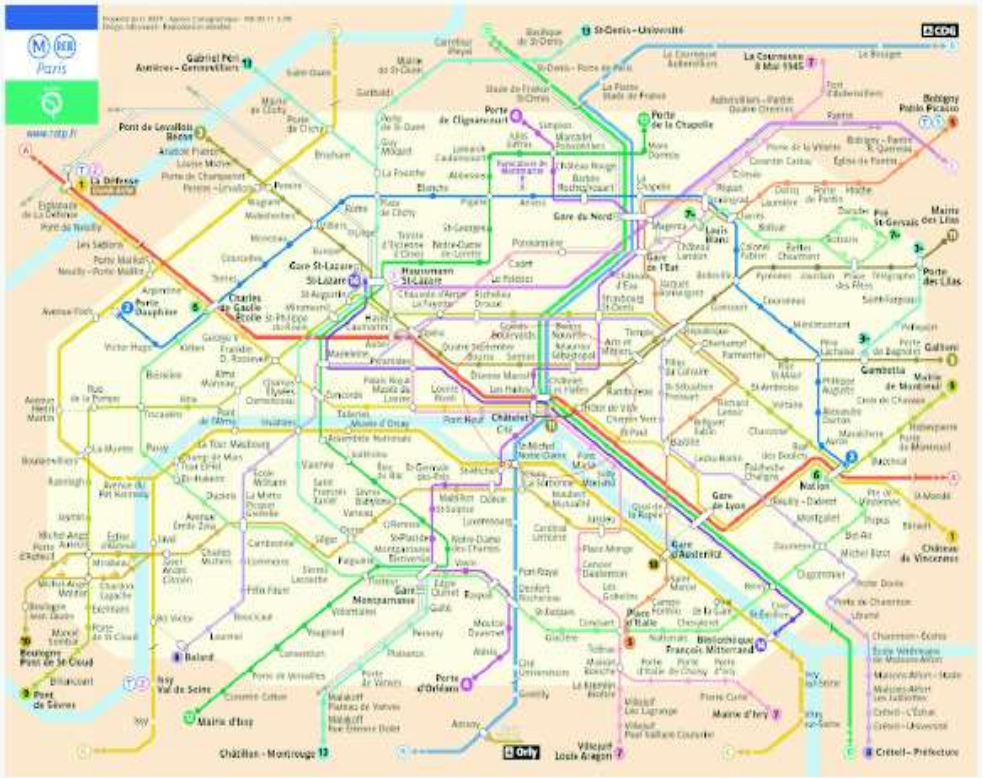
АЛГОРИТМИ И
ПОДАТОЧНИ СТРУКТУРИ

- предавања -

Што се тоа графови ?

Граф е множество на **темиња** меѓусебно поврзани со **ребра**.



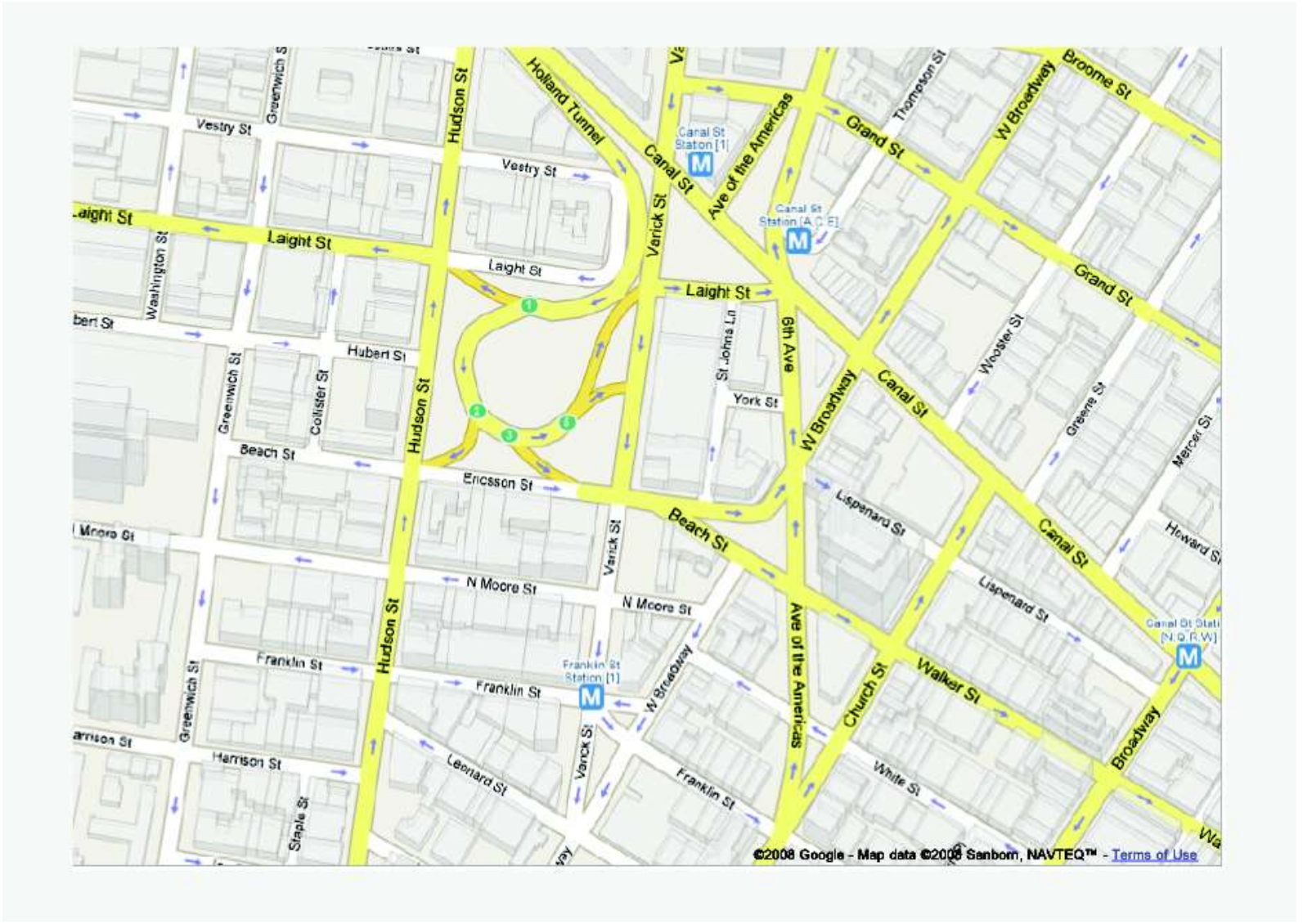


Зошто се изучуваат алгоритми за графови ?

- Интересна и многу корисна апстракција.
- Предизвикувачка област од компјутерските науки и дискретна математика.
- Стотици практични алгоритми се веќе развиени.
- Постојат илјадинци практични апликации.

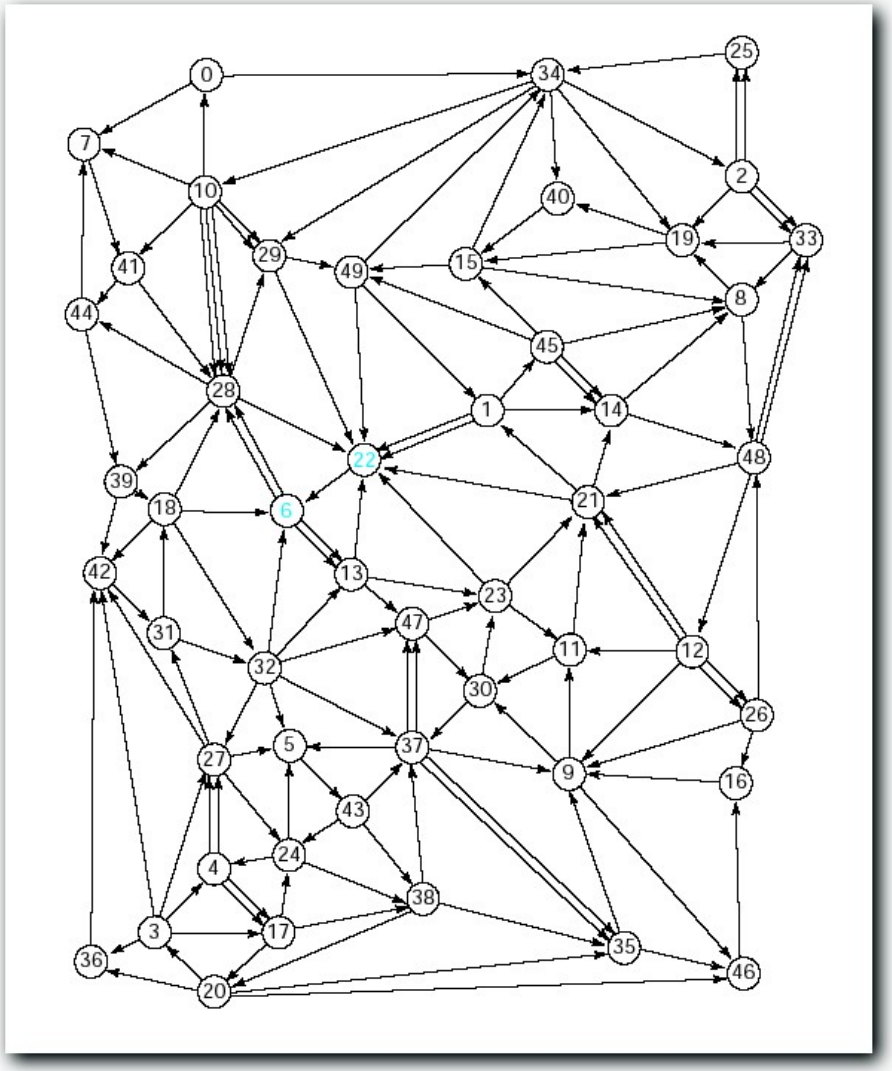
Ориентирани графови

Множество на **темиња** меѓусебно поврзани со **ориентирани ребра**.



Веб граф

Темиња: страници, **Ребра:** линкови.



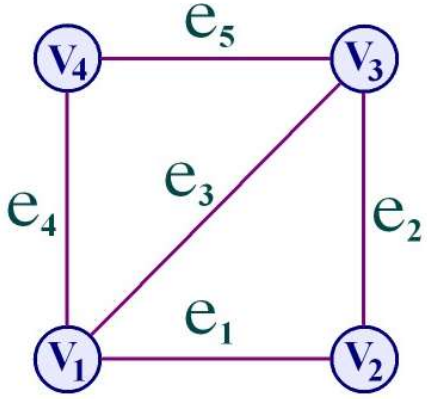
Апликации со графови

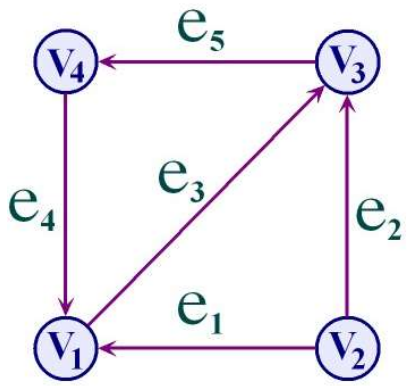
граф	теме	ребро
комуникација	телефон, компјутер	кабел
електрично коло	транзистор, кондензатор	жица
финансиски систем	валута	трансакции
игра	позиција на табла	потег
социјална мрежа	човек	пријателство

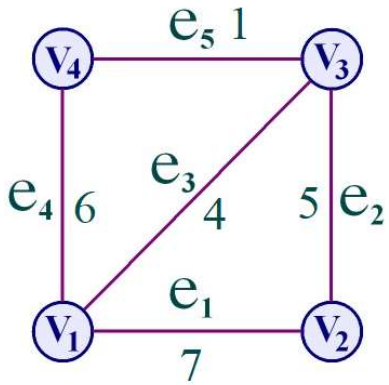
Дефиниција на графови

Граф G се дефинира како двојка (V, E), каде што V е множество темиња, а E е множество ребра при што на секое ребро е придружен пар темиња преку така наречената функција на соседство $E \rightarrow V \times V$. Темињата u и v во реброто $e = (u, v) \in E$, се нарекуваат краеви на реброто.

- Доколку двојките на темињата се разгледуваат како неподредени, т.е. кога (u, v) = (v, u), графот се нарекува **неориентиран**
- Кога ребрата на графот се разгледуваат како подредени двојки, т.е. кога (u, v) ≠ (v, u), графот се нарекува **ориентиран**
- Кога на ребрата на графот е придружен реален број (што може да претставува: тежина, растојание, цена, проточност, профит итн.) графови се нарекуваат **тежински**







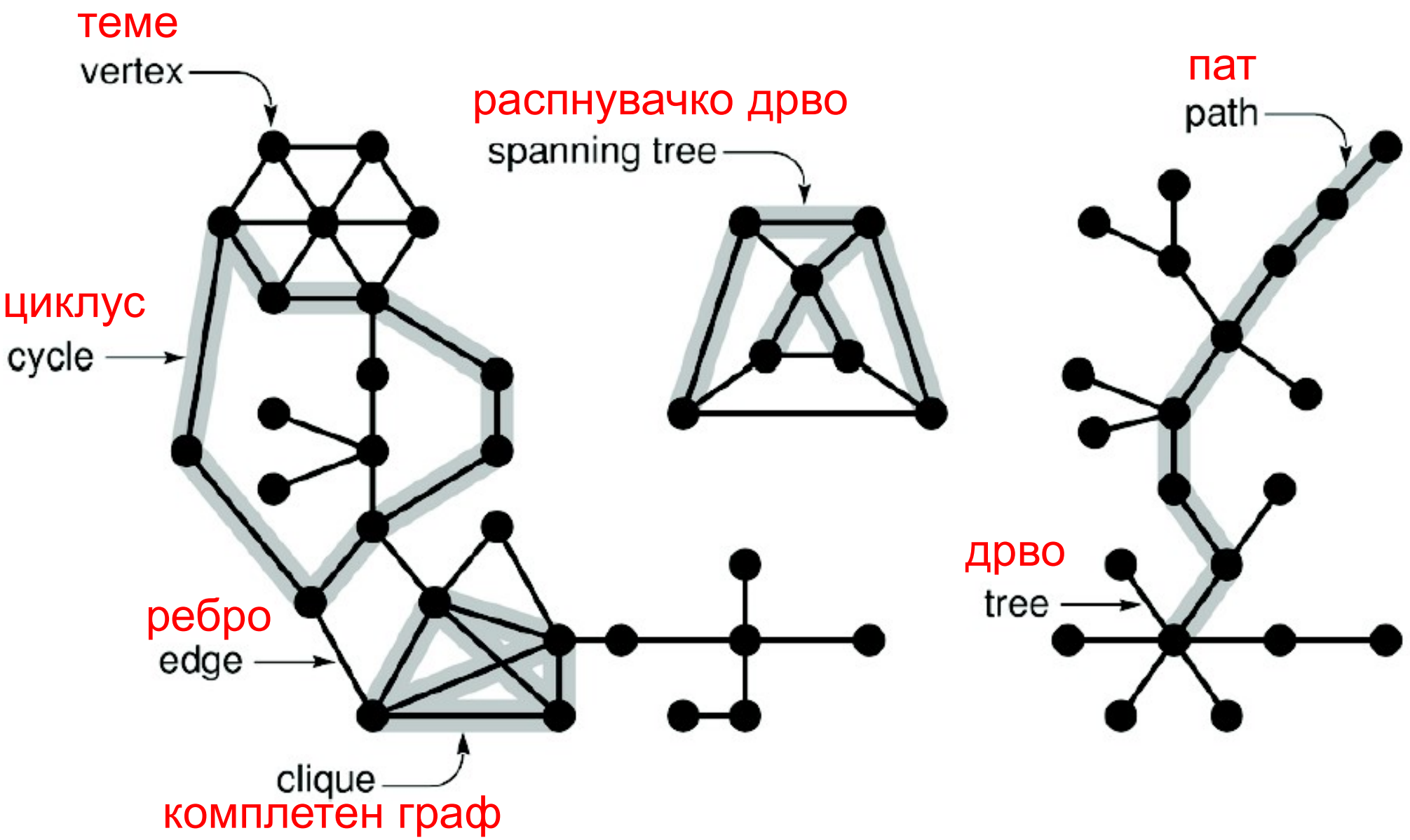
Терминологија за графови

- **Соседност на јазли**: за јазелот v се вели дека е соседен на јазелот u ако постои реброто (v, u)
- **Степен на јазел**: бројот на соседни јазли на даден јазел. Кај насоченият граф се разликуваат влезен и излезен степен на јазел, во зависност од насочеността на ребрата кон или од јазелот
- **Маршрута**: конечна низа од темиња така што две последователни темиња во маршрутата да се соседни
- **Верига**: маршрута во која ниту едно ребро не се повторува
- **Патека**: маршрута во која ниту едно теме не се повторува
- **Должина на патека**: бројот на ребра во патеката

Терминологија за графови

- **Циклус (јамка)**: проста патека во која што првото и последното теме се исти
- **Поврзан граф**: граф кај кој постои постои патека помеѓу секој пар на јазли. Потребен (но не и доволен) услов за еден граф да е поврзан е $|E| \geq |V| - 1$.
- **Дрво (стебло)**: поврзан граф што нема циклус
- **Распнувачко дрво**: поврзан граф во кој постои единствена патека помеѓу секои два јазли. Притоа важи: $|E| = |V|-1$ (**подмножество од графот**)
- **Комплетен граф**: секој пар темиња е поврзан со ребро
- **Регуларен граф**: ако степените на сите негови темиња се еднакви

Терминологија за графови



Неколку графовски проблеми

Пат. Дали постои пат помеѓу s и t ?
Најкраток пат. Кој е најкраткиот пат помеѓу s и t ?

Циклус. Дали постои циклус во даден граф?
Еулеров циклус. Дали постои циклус кој го користи секое ребро само еднаш ?
Хамилтонов пат. Дали постои циклус кој го користи секое теме само еднаш ?

Поврзаност. Дали сите темиња се поврзани ?
MST. Кој е најдобриот начин да се поврзат сите темиња?
Biconnectivity. Дали постои теме со чие отстранување ќе се прекине поврзаноста на графот?

Планарност. Дали може да се нацрта графот на рамнина притоа да нема вкрстени ребра?
Изоморфни графови. Дали две матрици на соседство претставуваат еден ист граф?

Проблеми за ориентирани графови

Пат. Дали постои ориентиран пат помеѓу s и t ?

Најкраток пат. Кој е најкраткиот ориентиран пат помеѓу s и t ?

Строга поврзаност. Дали сите темиња се взаемно достапни?

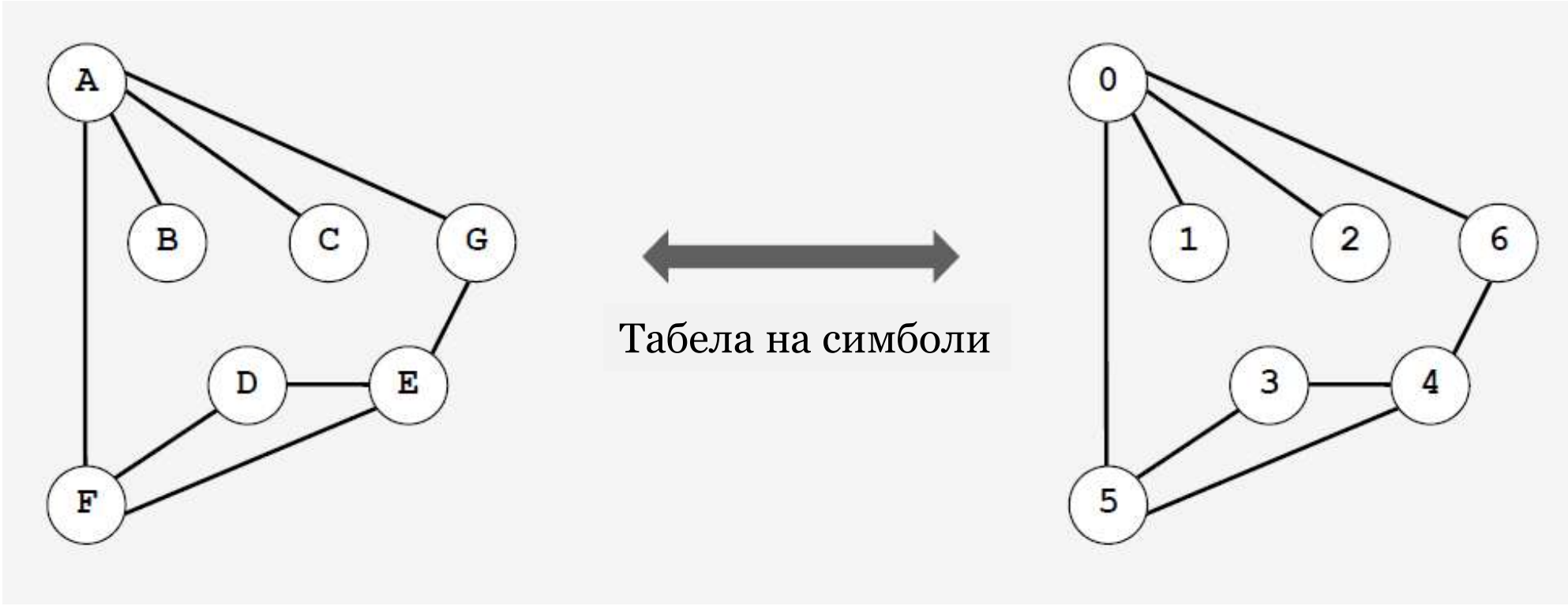
Транзитивно затворање. За кои темиња v и w постои пат помеѓу v и w ?

Тополошко сортирање. Дали може да се нацрта ориентираниот граф, така што сите ребра ќе покакажуваат од лево кон десно ?

Репрезентација на графови

Репрезентација на темиња

- Ќе се користат броевите 0..V-1, каде V е бројот на темиња во графот.
- Во пракса, преведете ги имињата на темињата во броеви со хеш табела.



Основни операции со граф

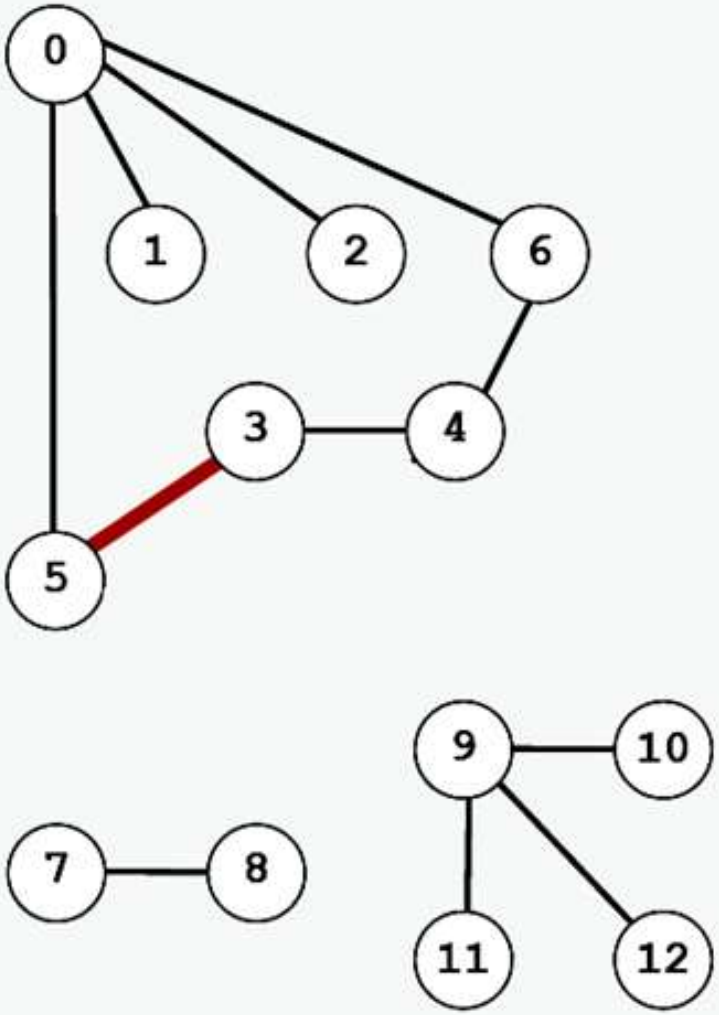
- Креирање на празен граф (или со дадени темиња)
- Додавање/отстранување на ребра во графот
- Добивање на соседи на дадено теме

Граф API

public class Graph	Податочен тип граф
Graph(int V)	Креирање на празен граф со V темиња
Graph(In in)	Креирање на граф од влезен поток
void addEdge(int v, int w)	Додавање ребро v-w
Iterable<Integer> adj(int v)	Враќа итератор на сите соседи на v
int V()	Го враќа бројот на темиња
String toString()	Враќа преставување во String

Репрезентација на ребра (листа)

Се нарекува и список на ребра (парови од темиња)



Неориентиран граф

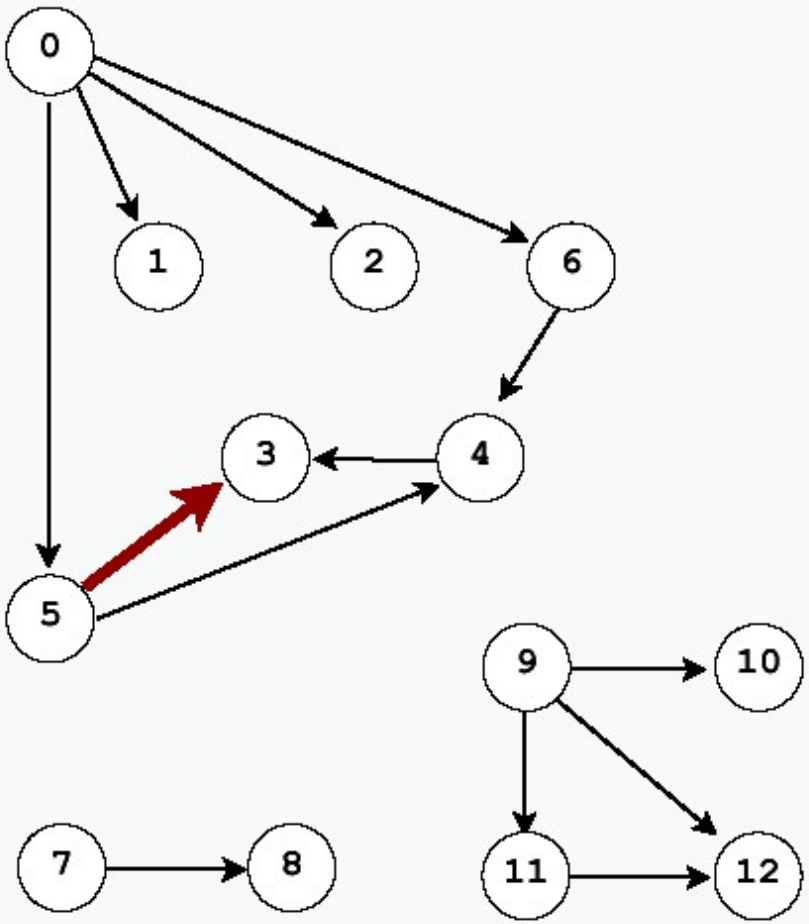
```
0 1
0 2
0 5
0 6
3 4
3 5
4 6
7 8
9 10
9 11
9 12
```

Можни имплементации -листа или низа од ребра (т.е. парови од темиња)
-подваријација: две низи од темиња (една за почетоците, друга за краевите на ребрата)

Пример:

```
p = [0 0 0 0 3 3 4 7  9  9  9]
q = [1 2 5 6 4 5 6 8 10 11 12]
```

Репрезентација на ребра (листа)

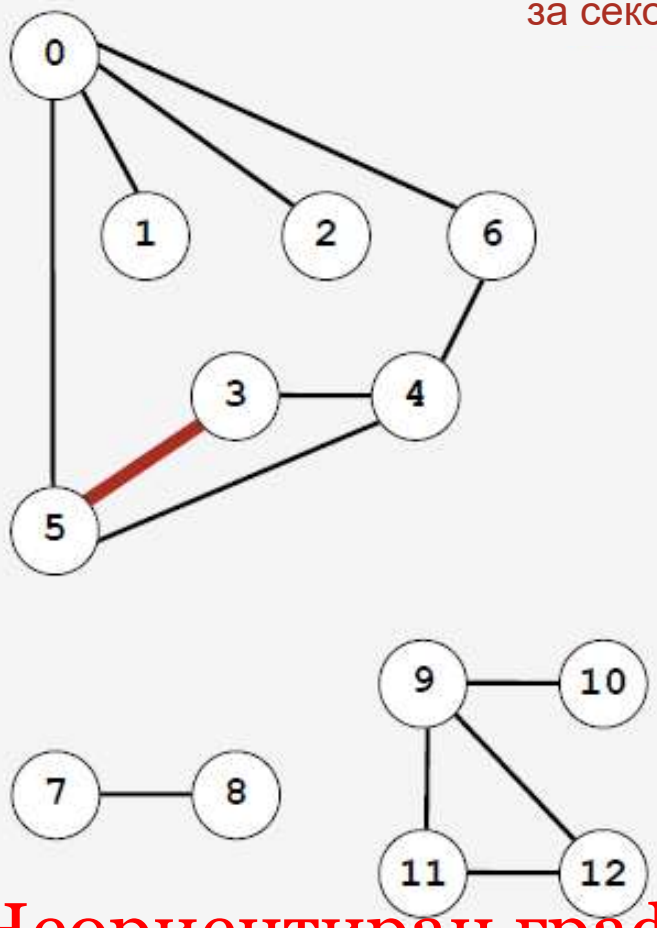


0	1
0	2
0	5
0	6
4	3
5	3
5	4
6	4
7	8
9	10
9	11
9	12
11	12
9	10
9	11
9	12

Ориентиран граф

Репрезентација на ребра (матрица на соседство)

Креираме матрица (adj) со димензии V x V.
За секоје ребро [v,w] поставуваме adj[v][w]=adj[w][v] = 1



Две вредности
за секое ребро

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	0	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Неориентиран граф

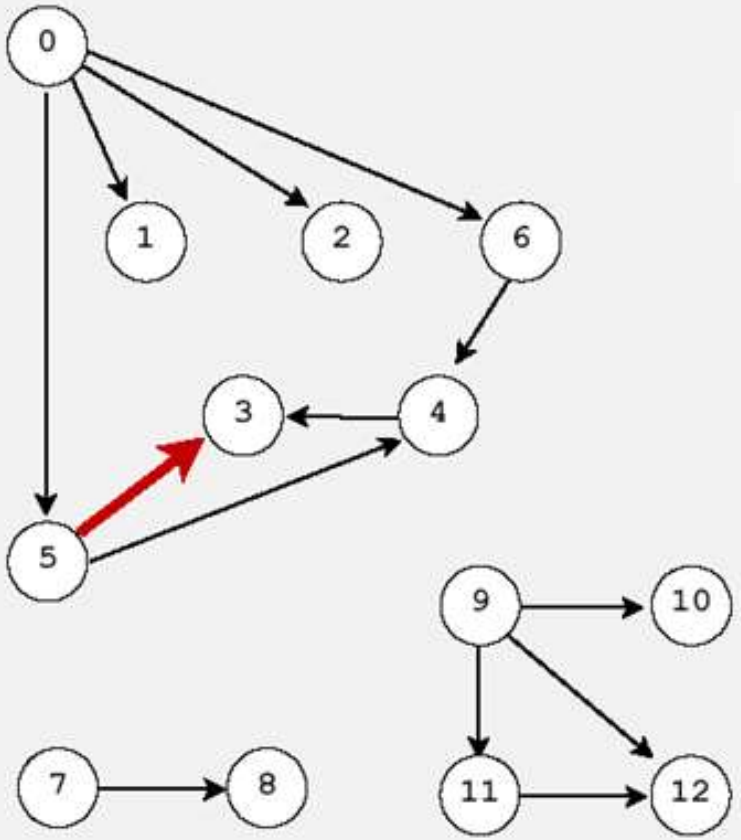
Имплементација (матрица на соседство)

```
int V; #broj na jazli
boolean adj[][]; #matrica na sosedstvo

procedure KreirajGraf(N) //vnesi rebro
begin
    V ← N
    for(i=0;i<V;i++)
      for(j=0;j<V;j++)
        adj[v][w] = 0
end

procedure DodadiRebro(v, w) //vnesi rebro
begin
    adj[v][w] ← 1
    adj[w][v] ← 1
end
```

Неориентиран граф

Репрезентација на ребра (матрица на соседство)

Креираме матрица со димензии V x V.
За секое ребро [v,w] поставуваме adj[v][w]=1



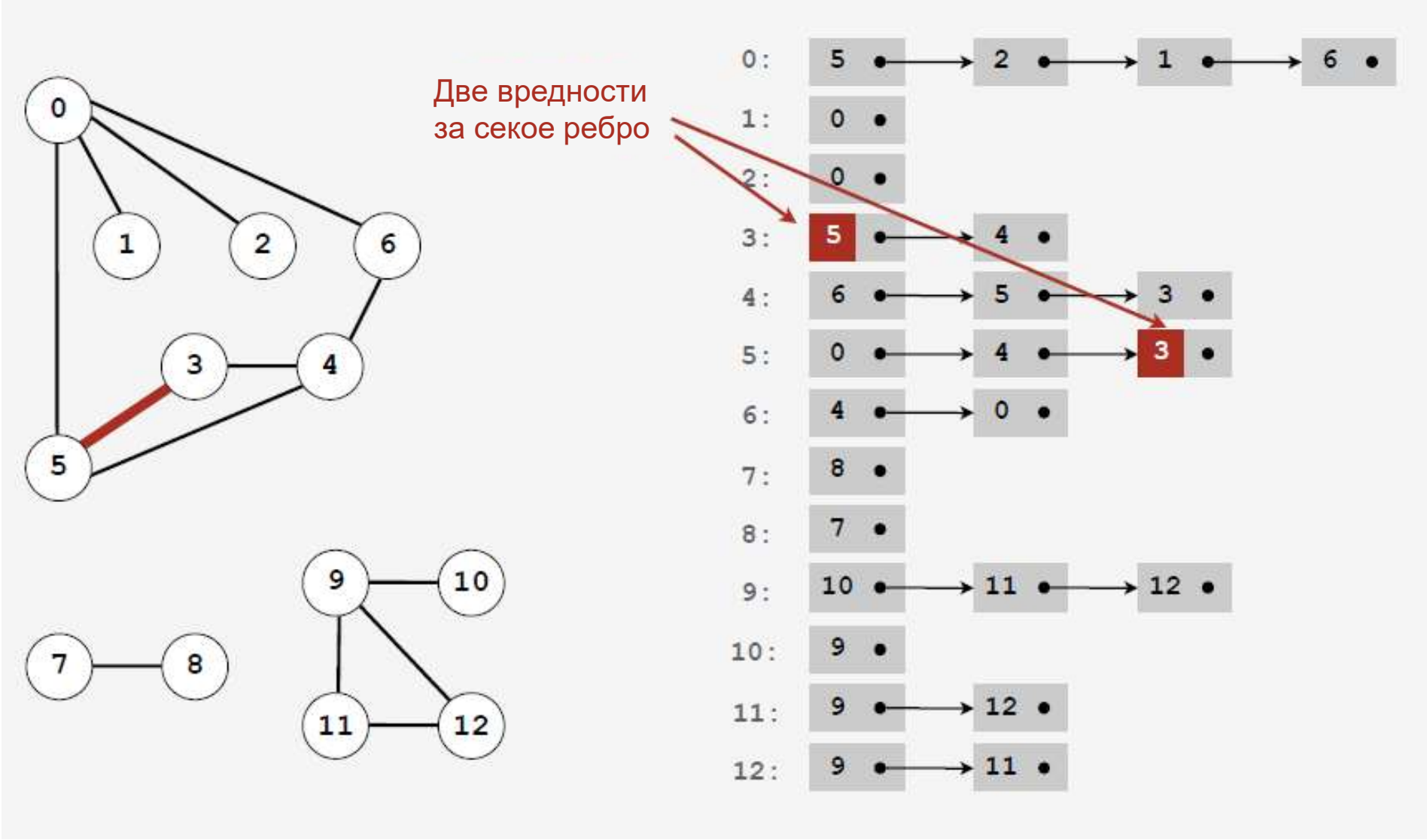
Од

До

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0

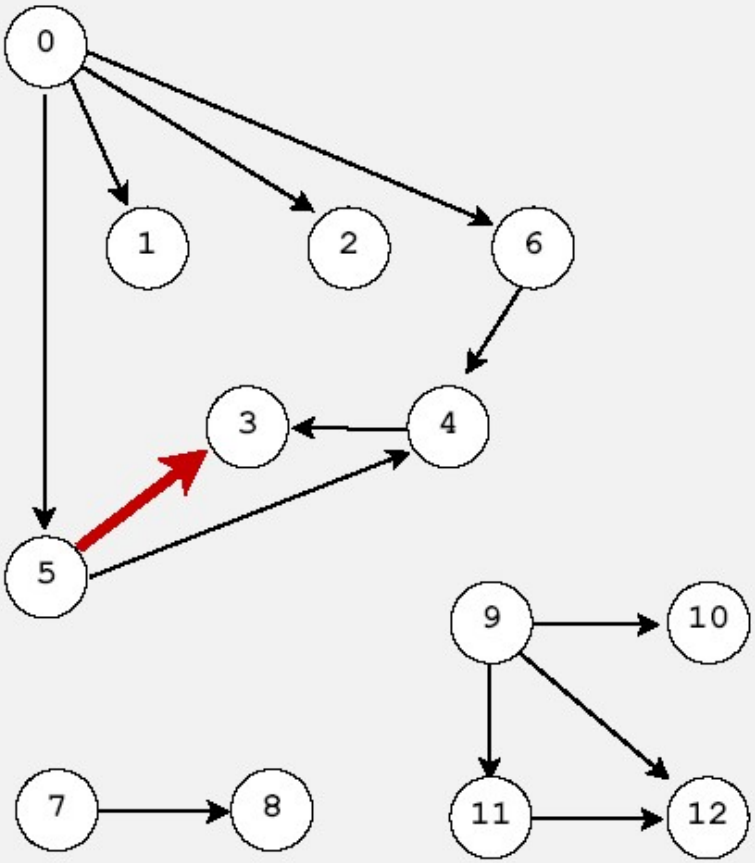
Ориентиран граф

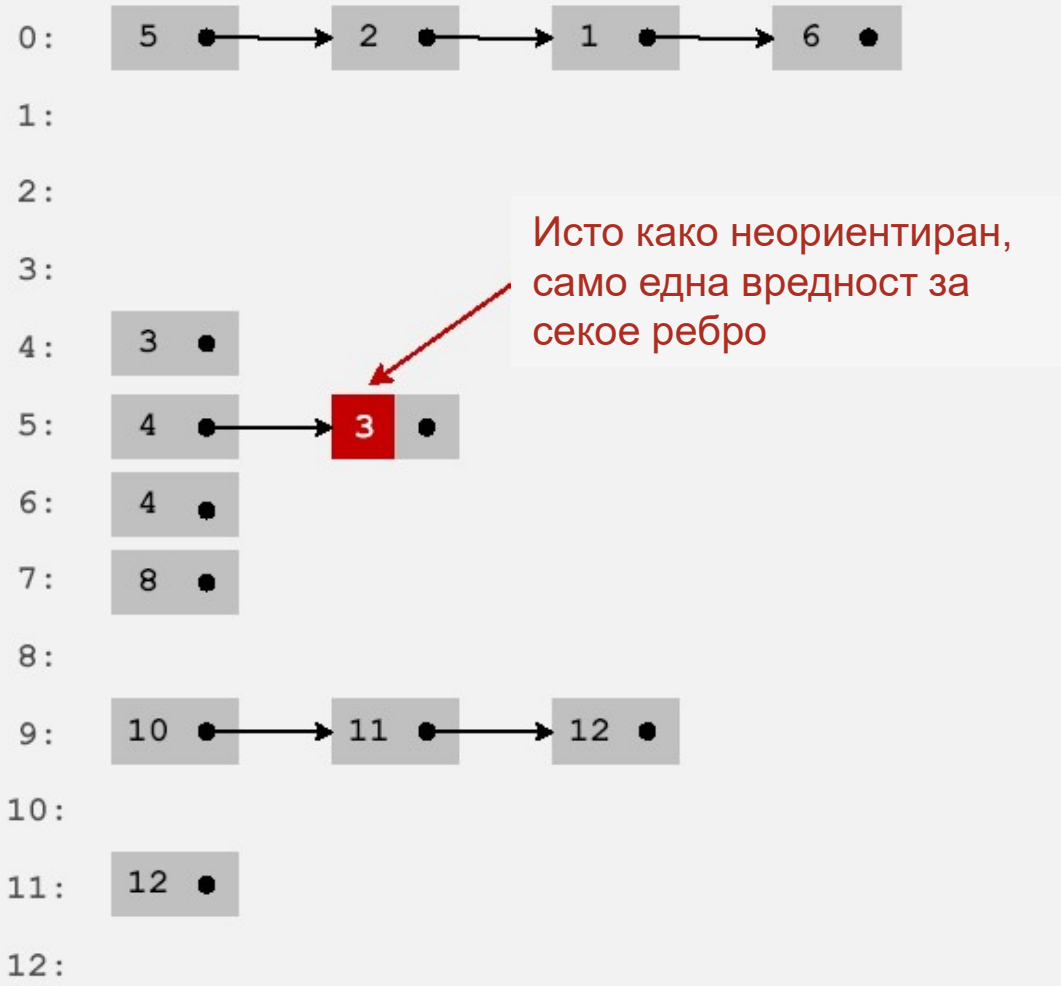
Репрезентација на ребра (листи на соседство)



Неориентиран граф

Репрезентација на ребра (листи на соседство)





Ориентиран граф

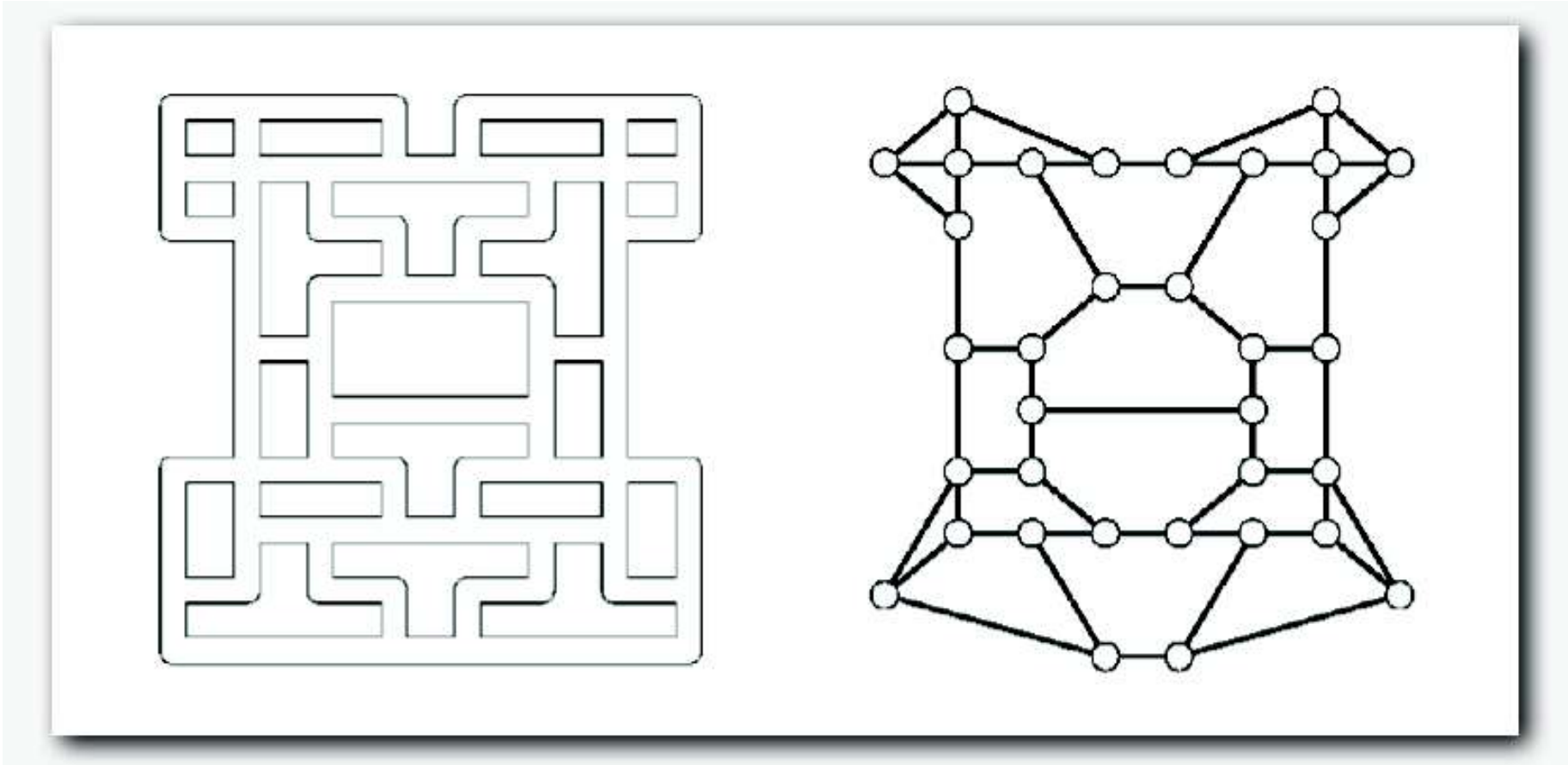
Репрезентација на графови

Во пракса, најчесто се користат листи на соседство, затоа што повеќето алгоритми се базираат на итеракција по ребрата закачени на дадено теме.
Реалните проблеми имаат тенденција да бидат 'ретки', т.е. да имаат голем број на темиња и мал (среден) број на 'соседни' темиња.

репрезентација	меморија	Има ребро помеѓу v и w ?	Итерирај помеѓу сите соседи на v
листа од ребра	E	E	E
матрица на соседство	V^2	1	V
листа на соседство	E + V	степен(v)	степен(v)

Прошетка низ лавиринт

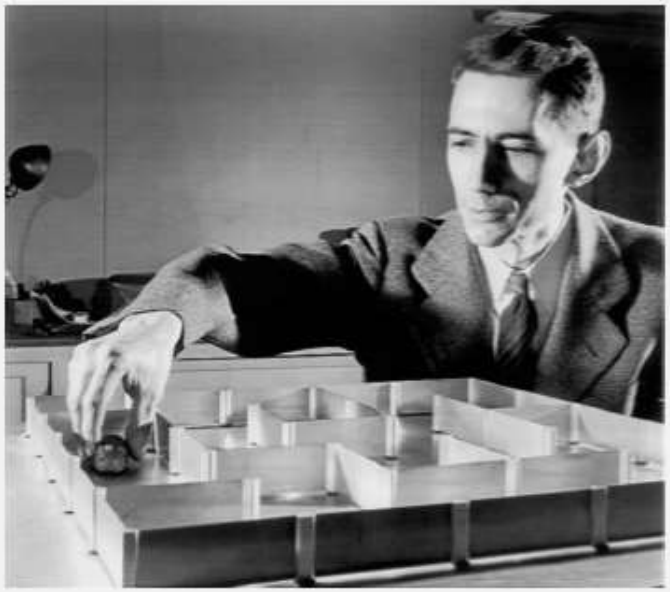
Теме: раскрсница
Ребро: ходник



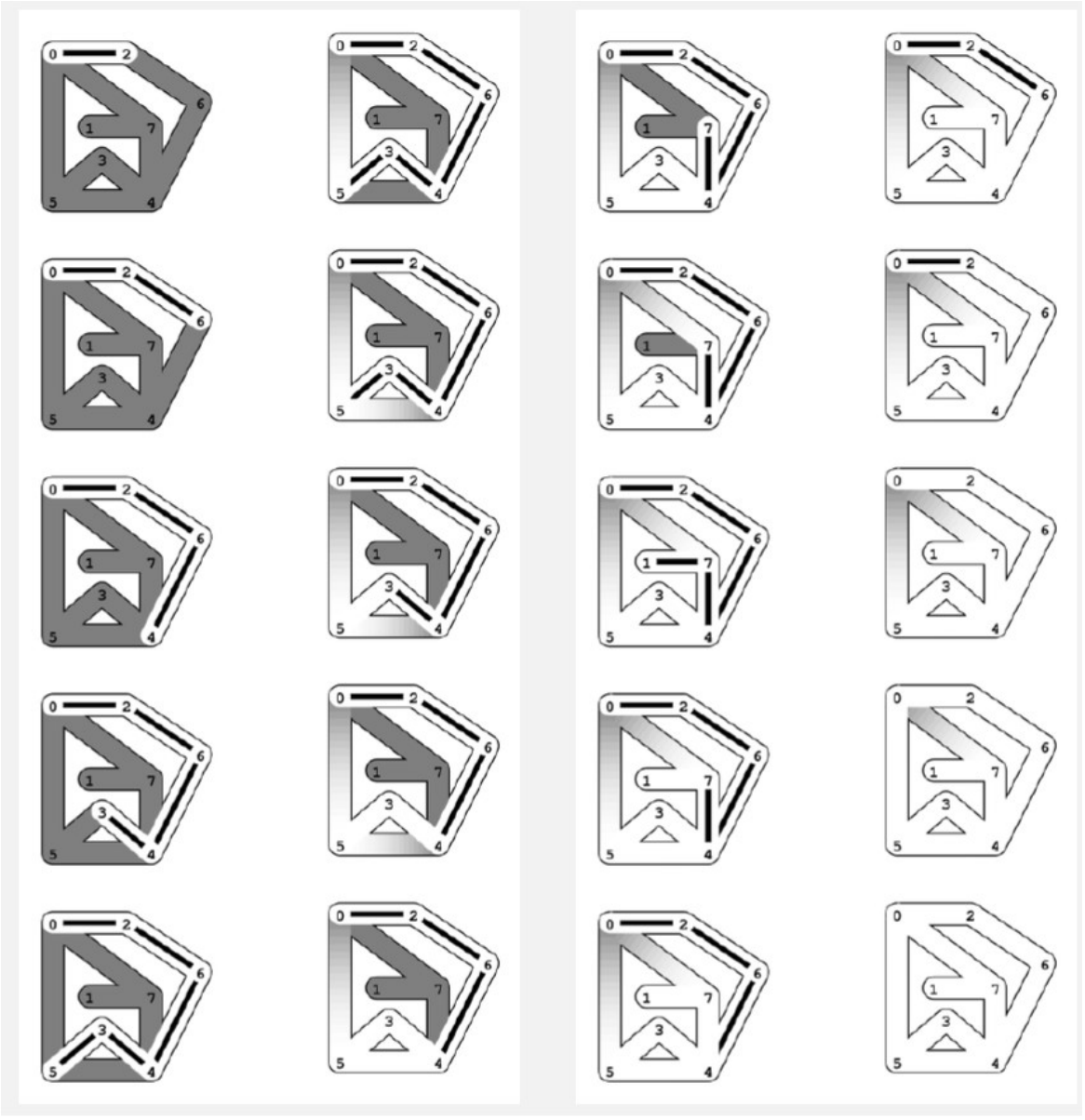
Цел: да се поминат сите ходници

Алгоритам

- Одвиткувај топче конец позади себе.
- Обележи ја секоја посетена раскрсница
- Обележи го секој посетен ходник



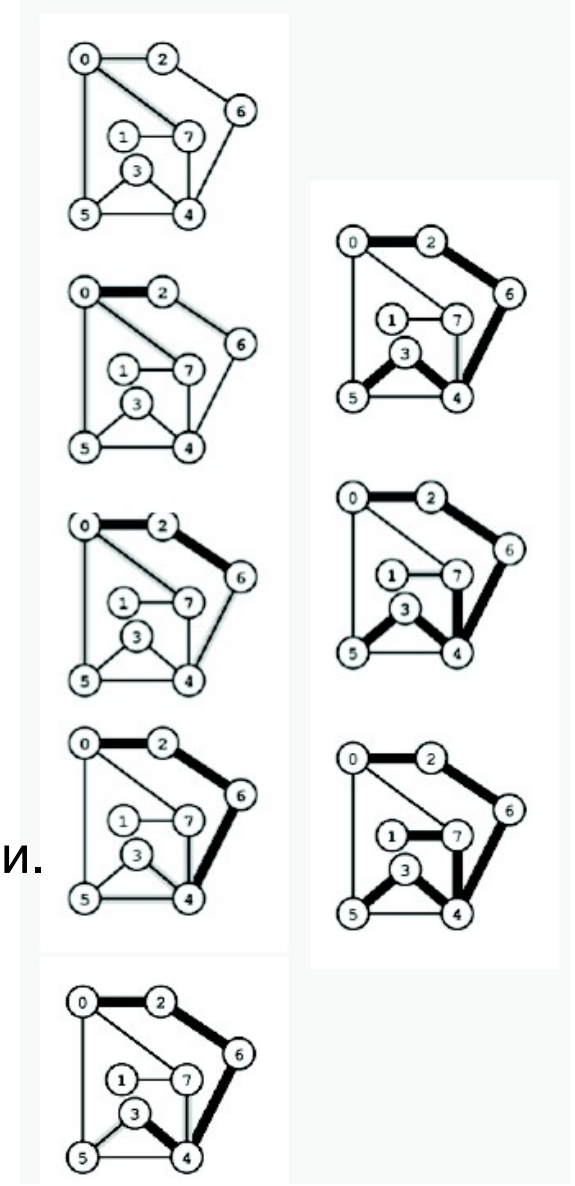
Изминување на лавиринт



Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.



Време на извршување:
O(E) бидејќи секое ребро е посетено најмногу два пати.
Најчесто помало од V за барање на пат во реални графови.

Типична примена:
Пронајди ги сите темиња поврзани со дадено теме s
Пронајди пат од s до t

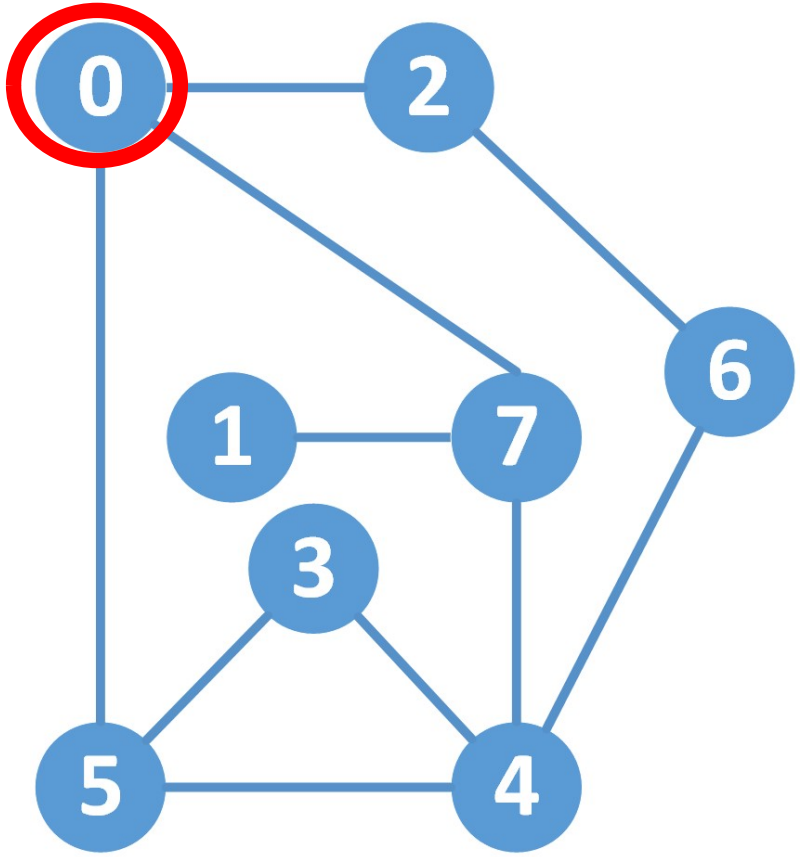
Имплементација на DFS

```
boolean marked[]; #obelezuvanje

procedure DFSSearch(s)
begin
    for(i=0;i<V;i++)
        marked[i] ← 0
    DFS(s)
end

procedure DFS(v)
begin
    marked[v] ← True
    for(w:Sosedi(v))     #sosedite se dobivaat vo zavisnost od repr.
      if(NOT marked[w]) then
        DFS(w)
end
```

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

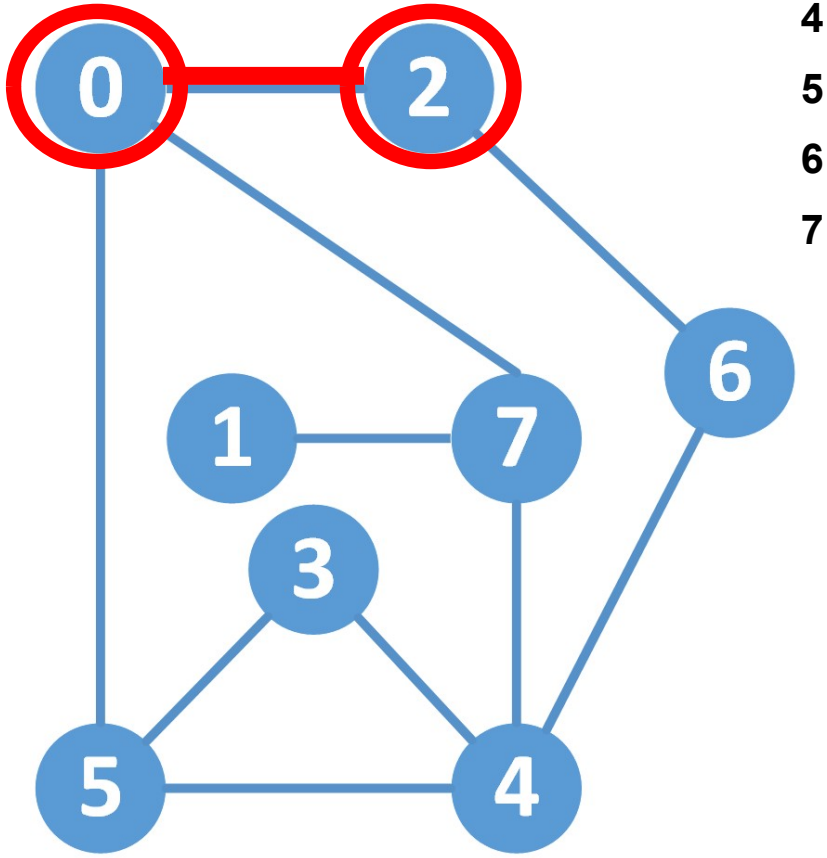


marked[]	
0	T
1	
2	
3	
4	
5	
6	
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

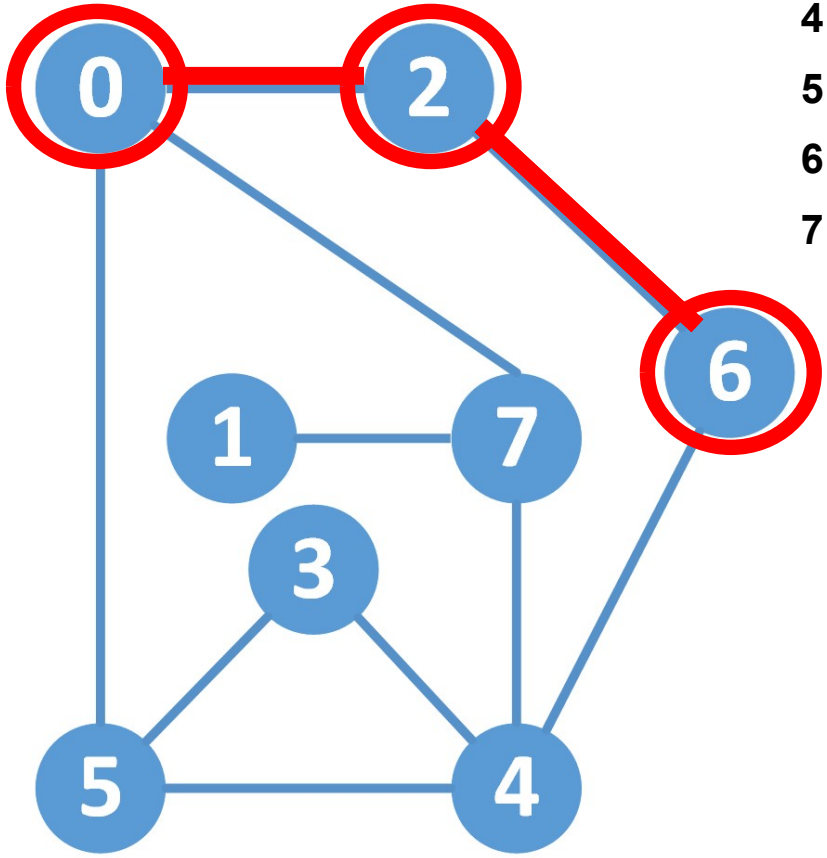


marked[]	
0	T
1	
2	T
3	
4	
5	
6	
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

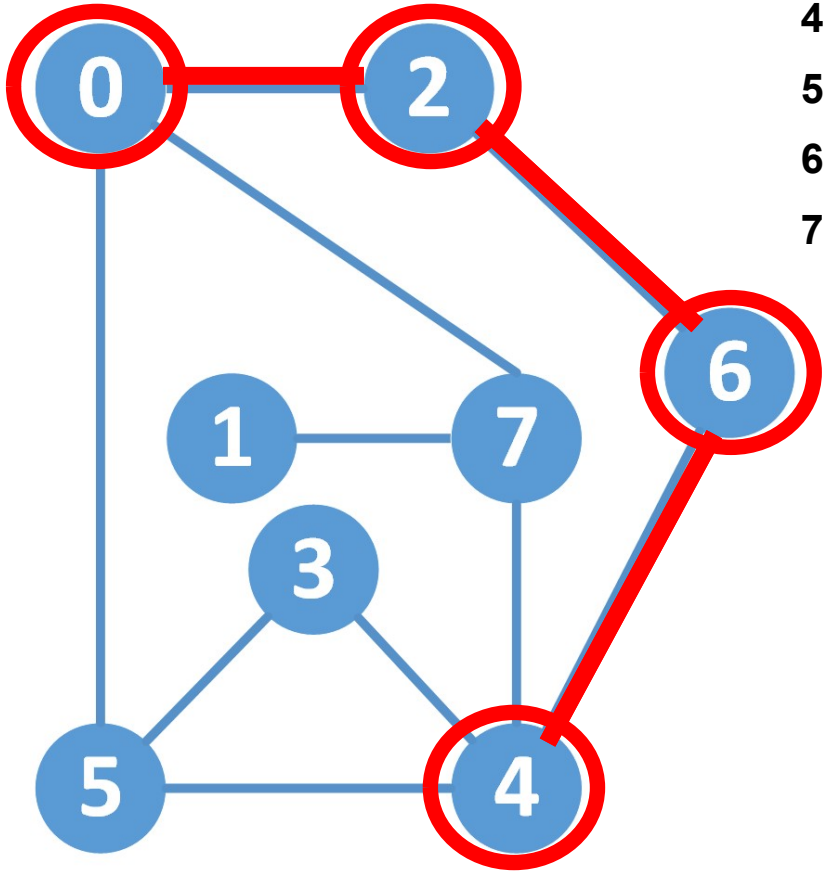


marked[]	
0	T
1	
2	T
3	
4	
5	
6	T
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

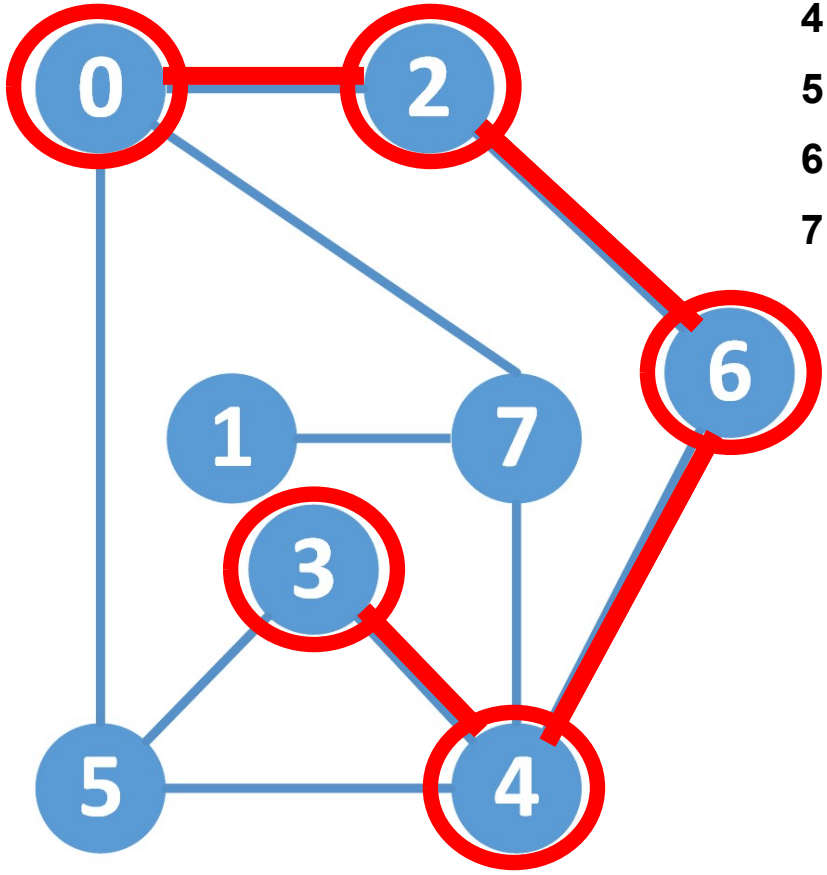


marked[]	
0	T
1	
2	T
3	
4	T
5	
6	T
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

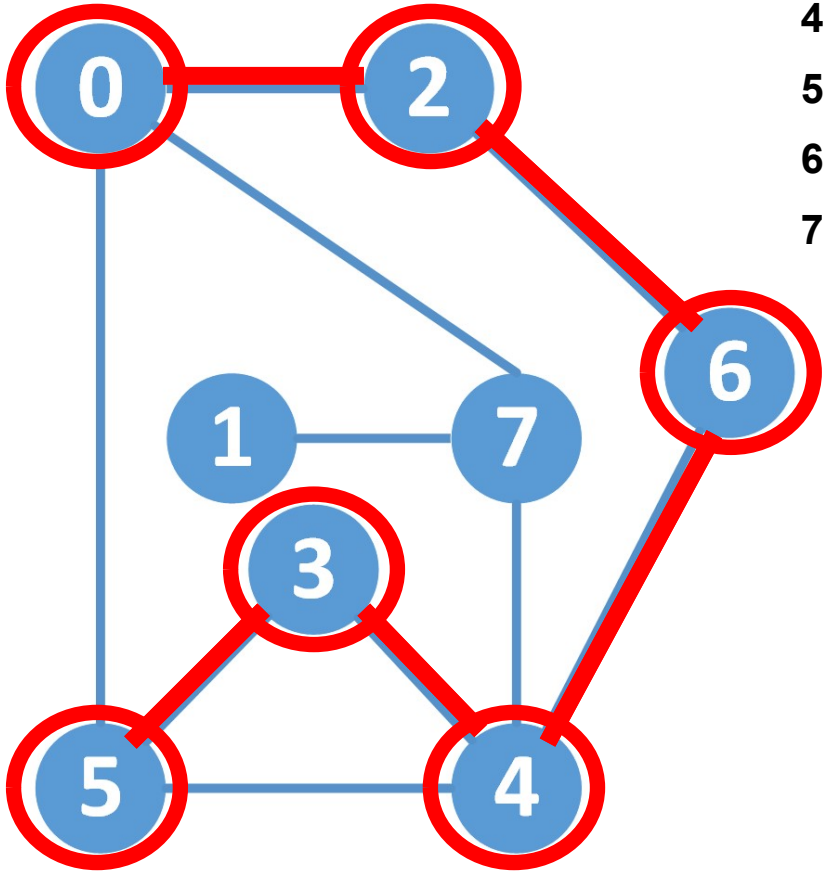


marked[]	
0	T
1	
2	T
3	T
4	T
5	
6	T
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.



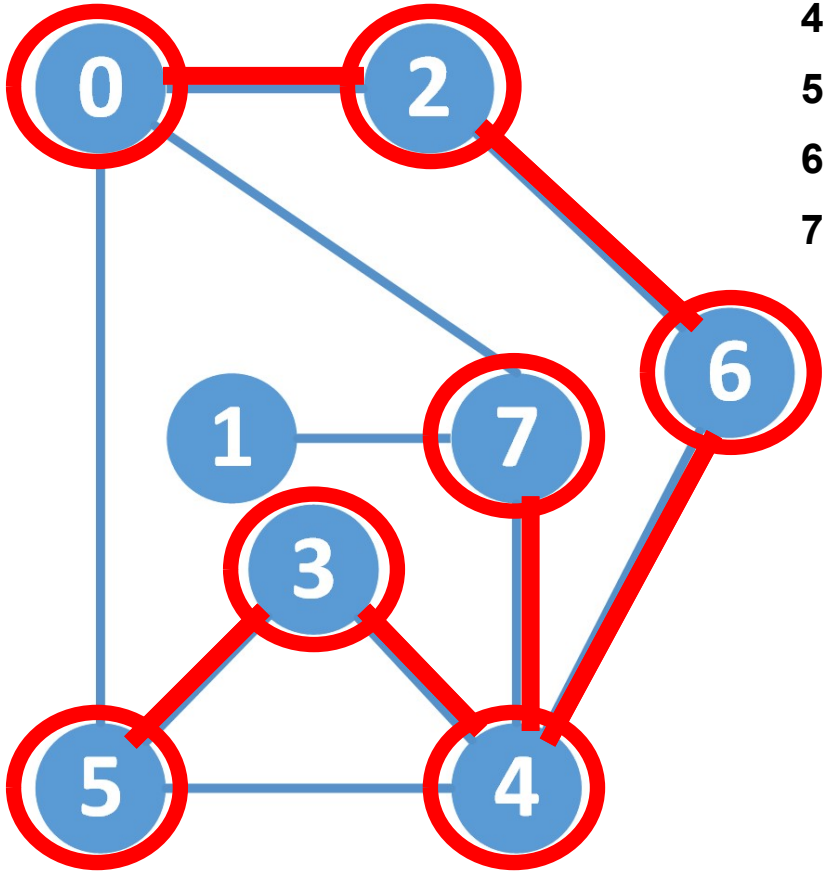
marked[]	
0	T
1	
2	T
3	T
4	T
5	T
6	T
7	

Пребарување на графови по длабочина (Depth First Search – DFS)

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

marked[]	
0	T
1	
2	T
3	T
4	T
5	T
6	T
7	T

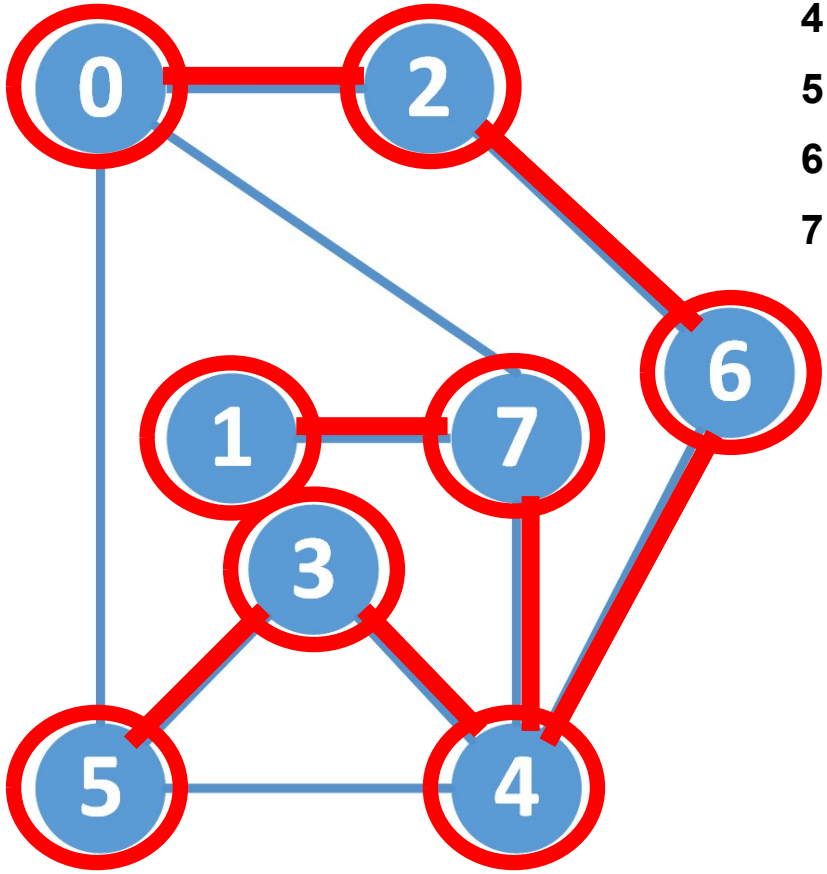
DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.



Пребарување на графови по длабочина (Depth First Search – DFS)

marked[]	
0	T
1	T
2	T
3	T
4	T
5	T
6	T
7	T

Цел: систематско изминување на граф
Идеја: изминување на лавиринт

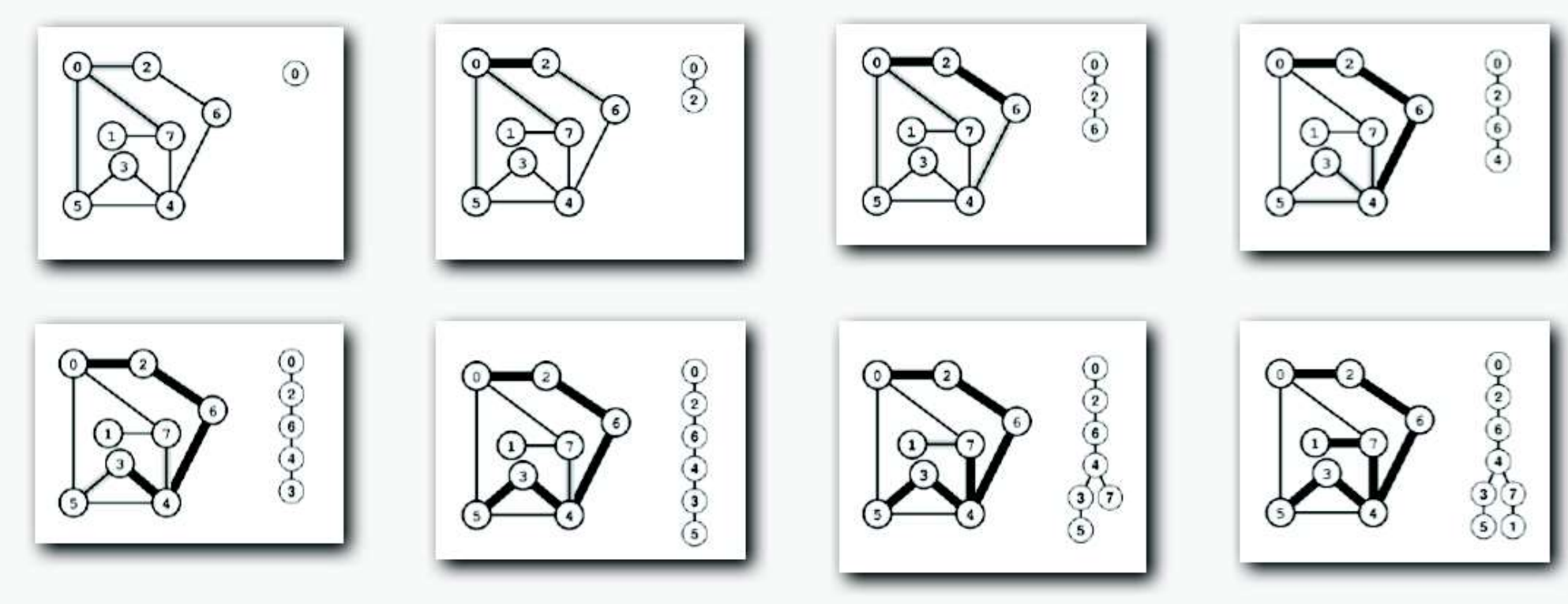


DFS (посета на темето s)
1. Обележи го s како посетен
2. Рекурзивно посети ги сите необележани темиња v поврзани со s.

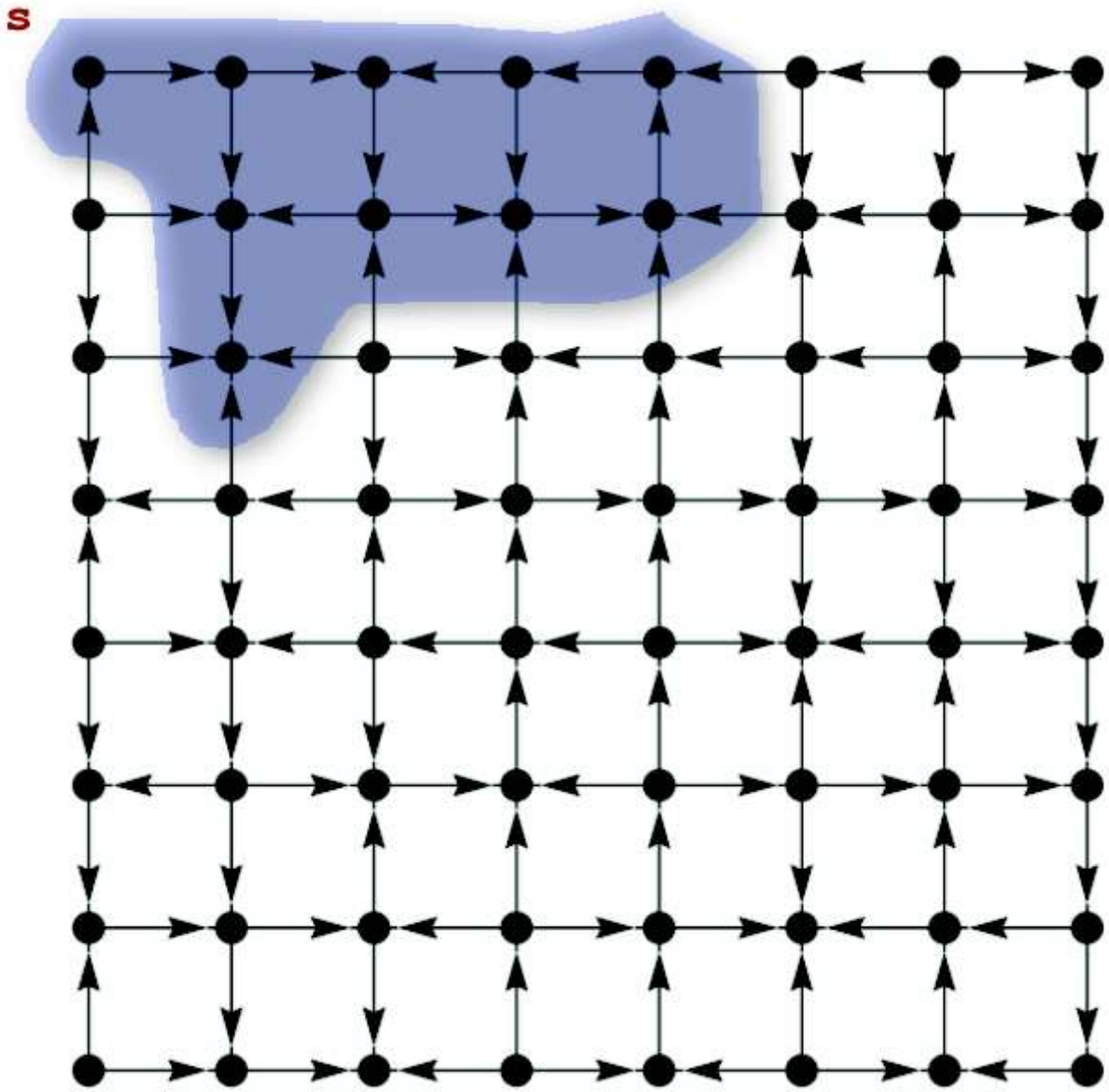
Алтернативна имплементација:
Наместо рекурзија секое непосетено теме да се става на **стек**

Конструкција на пат со DFS

Откако ќе се посети темето *v* за прв пат, запамети дека се дошло од *pred[v]*



Достапност



Достапност: примена во анализа на код

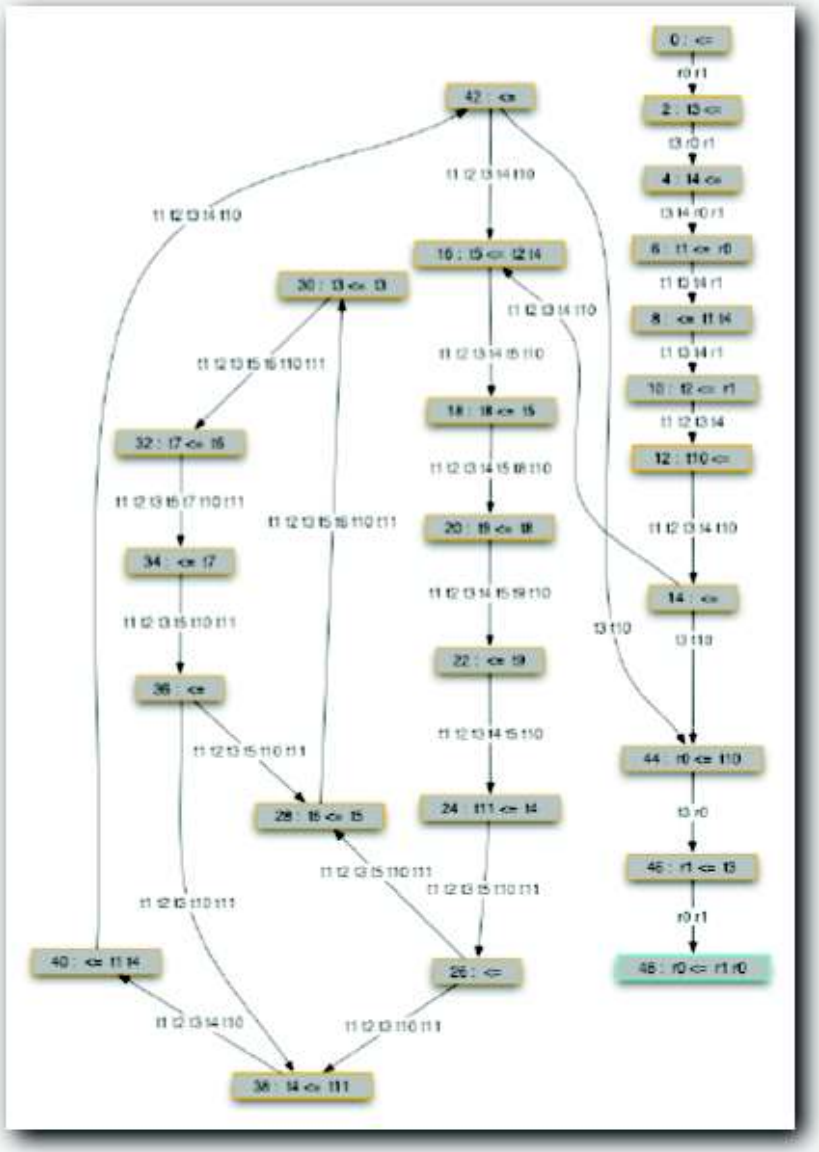
Секој програм е ориентиран граф

Теме: основен блок од инструкции (straight line program).

Ребро: скок (jump).

Елиминација на мртов код:
Пронајди го и отстрани го недостапниот код.

Детекција на бескраен циклус:
Одреди дали излезот е недостапен



Пребарување на графови по широчина (Breath First Search – BFS)

DFS: Стави ги непосетените темиња на стек.
BFS: Стави ги непосетените темиња во редица.

Најкраток пат: Најди пат од s до t кој е составен од најмал број на ребра.

Пребарување на графови по широчина (Breath First Search – BFS)

DFS: Стави ги непосетените темиња на стек.
BFS: Стави ги непосетените темиња во редица.

Најкраток пат: Најди пат од s до t кој е составен од најмал број на ребра.

BFS (почнувајќи од темето *s*)

1. Обележи го *s* како посетен и стави го во редица *Q*.
2. Повторувај се додека редицата *Q* не е празна
   - извади елемент *p* од редицата *Q*
   - сите необележани темиња поврзани со p
     внеси ги во редицата *Q* и обележи ги како посетени

Пребарување на графови по широчина (Breath First Search – BFS)

DFS: Стави ги непосетените темиња на стек.
BFS: Стави ги непосетените темиња во редица.

Најкраток пат: Најди пат од s до t кој е составен од најмал број на ребра.

BFS (почнувајќи од темето *s*)
1. Обележи го *s* како посетен и стави го во редица *Q*.
2. Повторувај се додека редицата *Q* не е празна
   - извади елемент *p* од редицата *Q*
   - сите необележани темиња поврзани со p
     внеси ги во редицата *Q* и обележи ги како посетени

Својство: BFS ги посетува темињата во растечки редослед според тоа колку се оддалечени од *s*

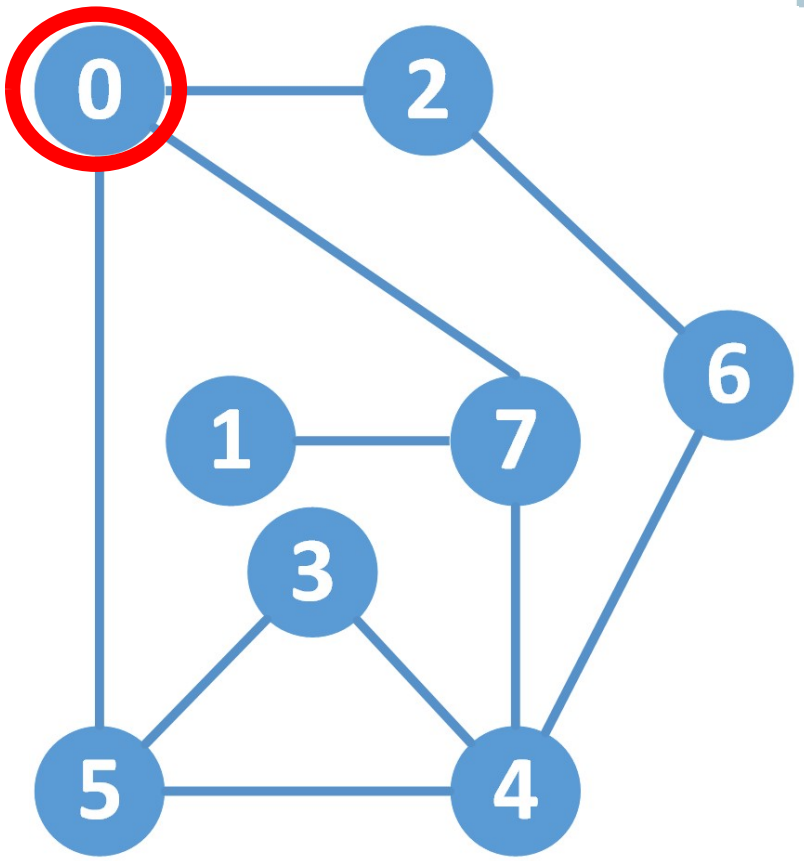
Имплементација на BFS

```
int distance[]
    #rastojanie od
    pocetnoto teme
procedure BFSSearch(s)
begin
    for(i=0;i<V;i++)
    distance[i] ← V + 1
    distance[s] ← 0
    BFS(s)
end
procedure BFS(v)
begin
    Queue q
    q.enqueue(v)
i   while(NOT
    q.empty())
    begin
      v = q.dequeue()
      for(w:Sosedi(v))
        if(distance[w] >
    V) then
        begin
          q.engueue(w)
          distance[w] =
    distance[v] + 1
        end
    end
end
```

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	8
2	8
3	8
4	8
5	8
6	8
7	8

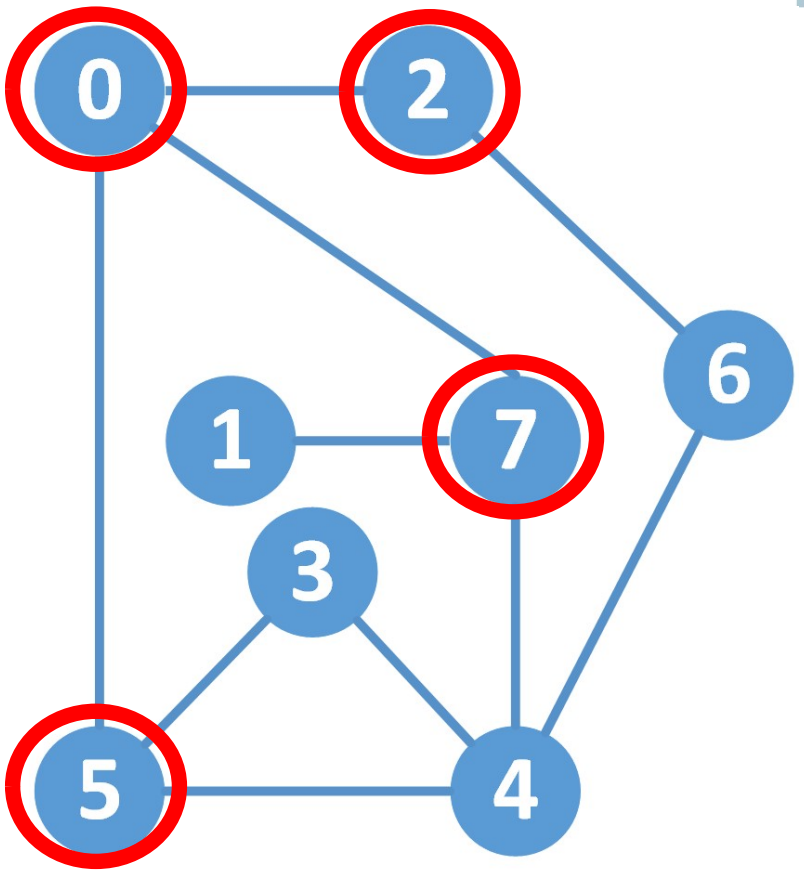
редица

0

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	8
2	1
3	8
4	8
5	1
6	8
7	1

редица

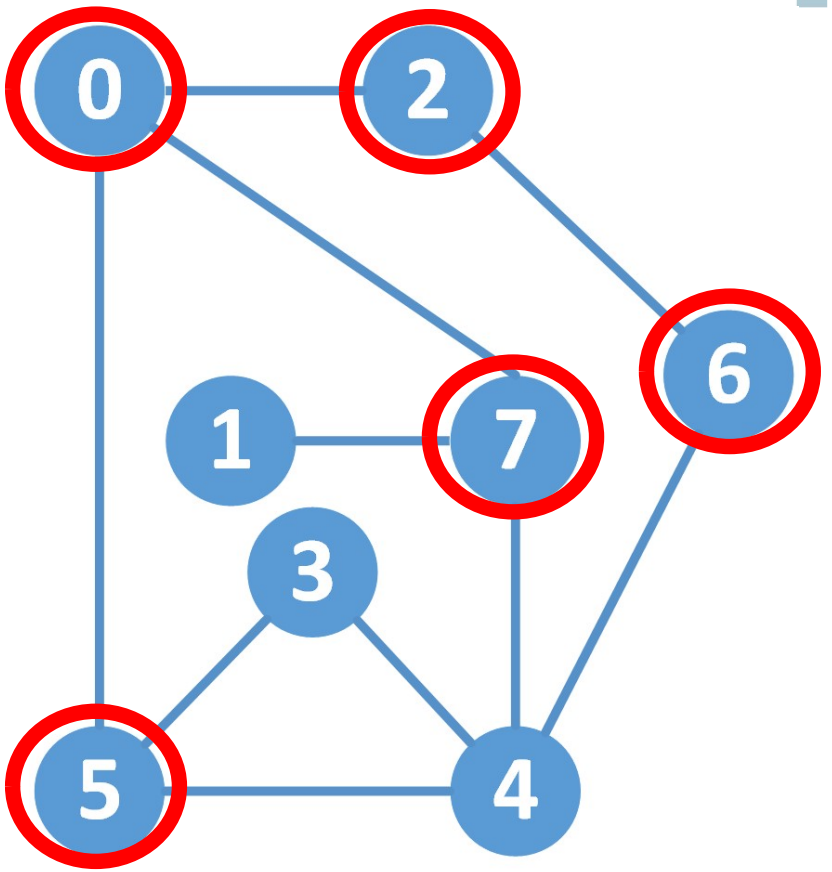
7 5 2

Излезе: 0
Соседи: 2, 5 и 7

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	8
2	1
3	8
4	8
5	1
6	2
7	1

редица

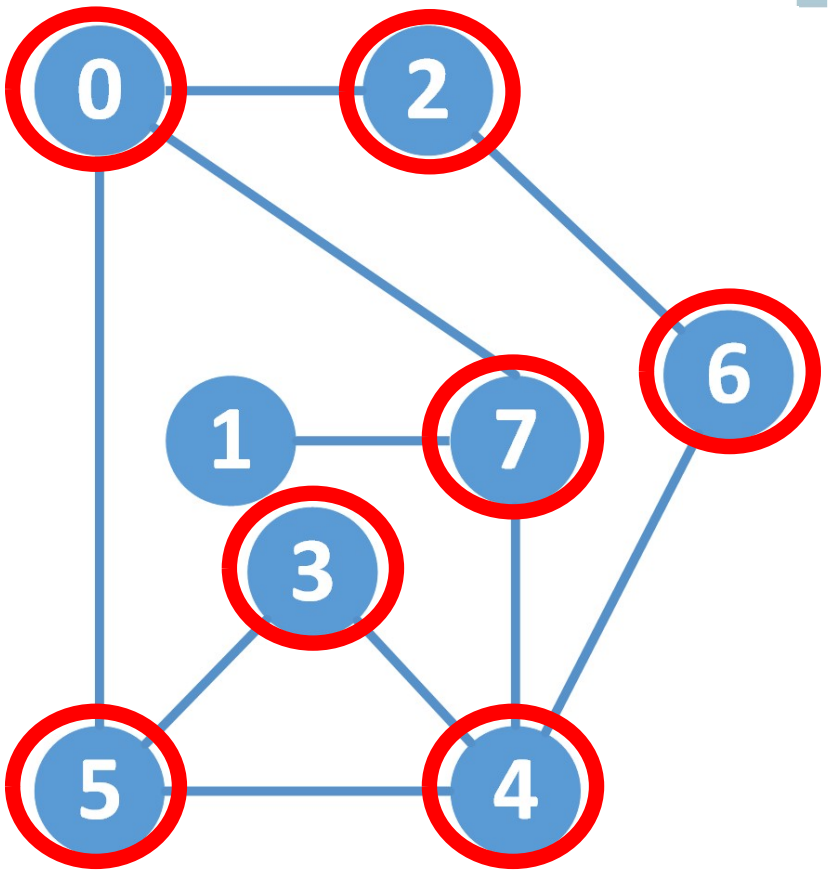
6 7 5

Излезе: 2
Соседи: 0 (посетен) и 6

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	8
2	1
3	2
4	2
5	1
6	2
7	1

редица

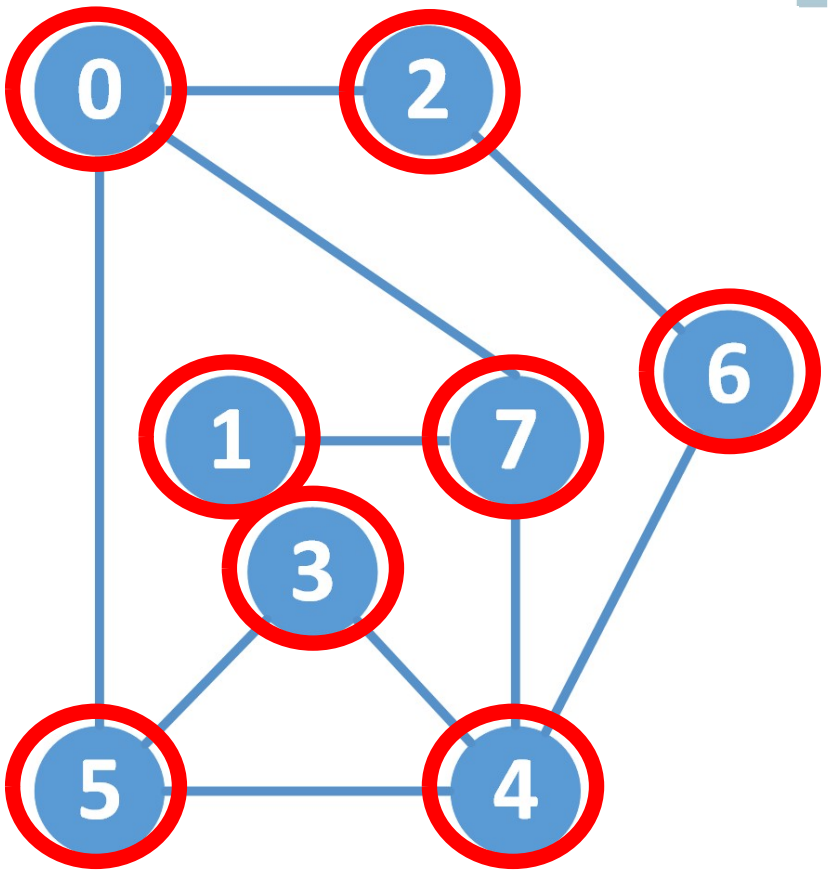
4 3 6 7

Излезе: 5
Соседи: 0 (посетен), 3 и 4

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	2
2	1
3	2
4	2
5	1
6	2
7	1

редица

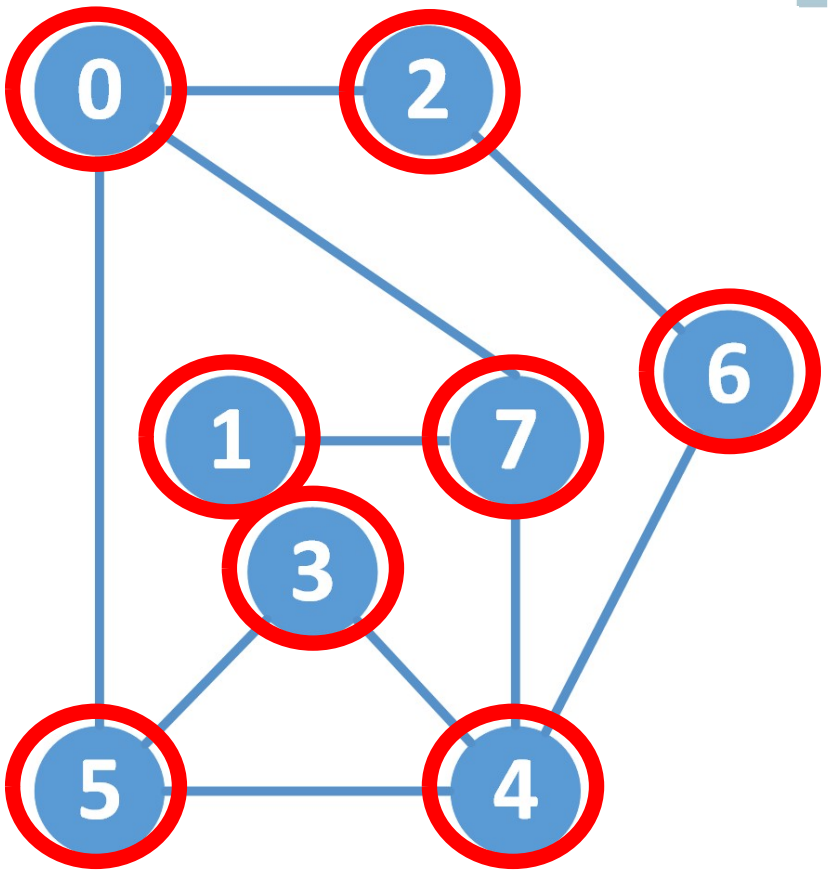
1 4 3 6

Излезе: 7
Соседи: 0 и 4 (посетени) и 1

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	2
2	1
3	2
4	2
5	1
6	2
7	1

редица

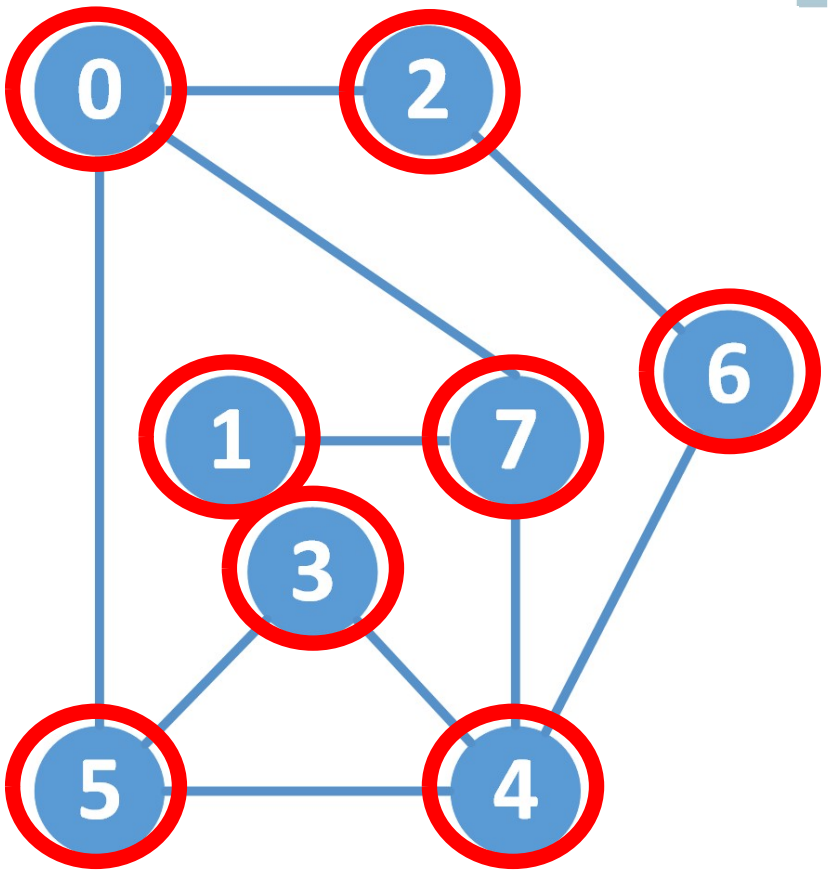
1 4 3

Излезе: 6
Соседи: сите посетени

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	2
2	1
3	2
4	2
5	1
6	2
7	1

редица

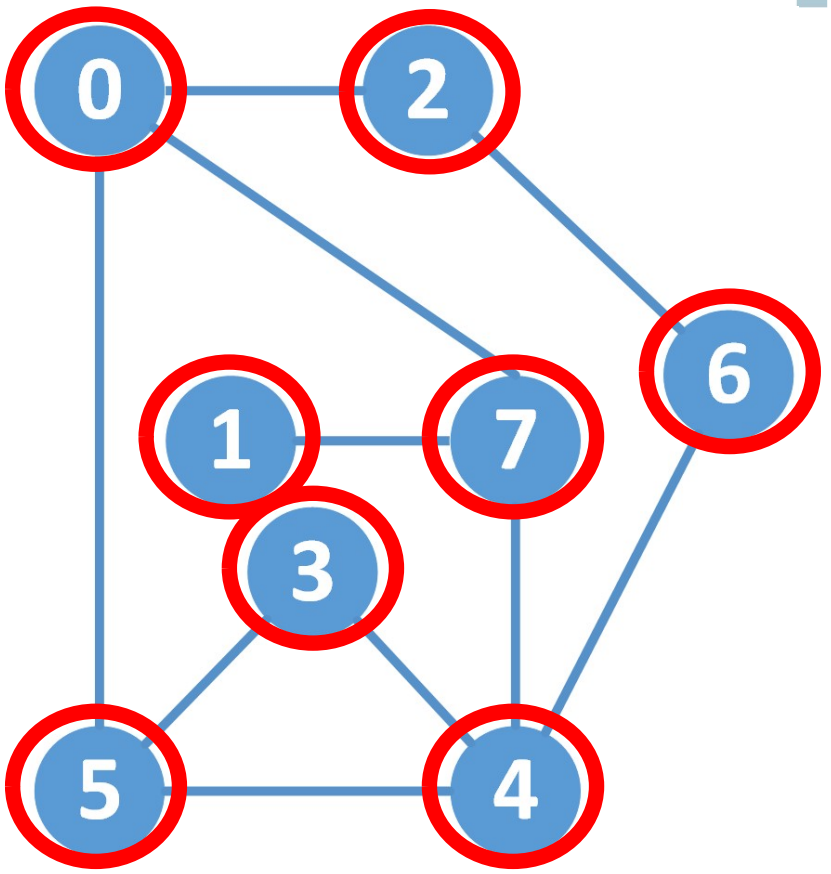
1 4

Излезе: 3
Соседи: сите посетени

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	2
2	1
3	2
4	2
5	1
6	2
7	1

редица

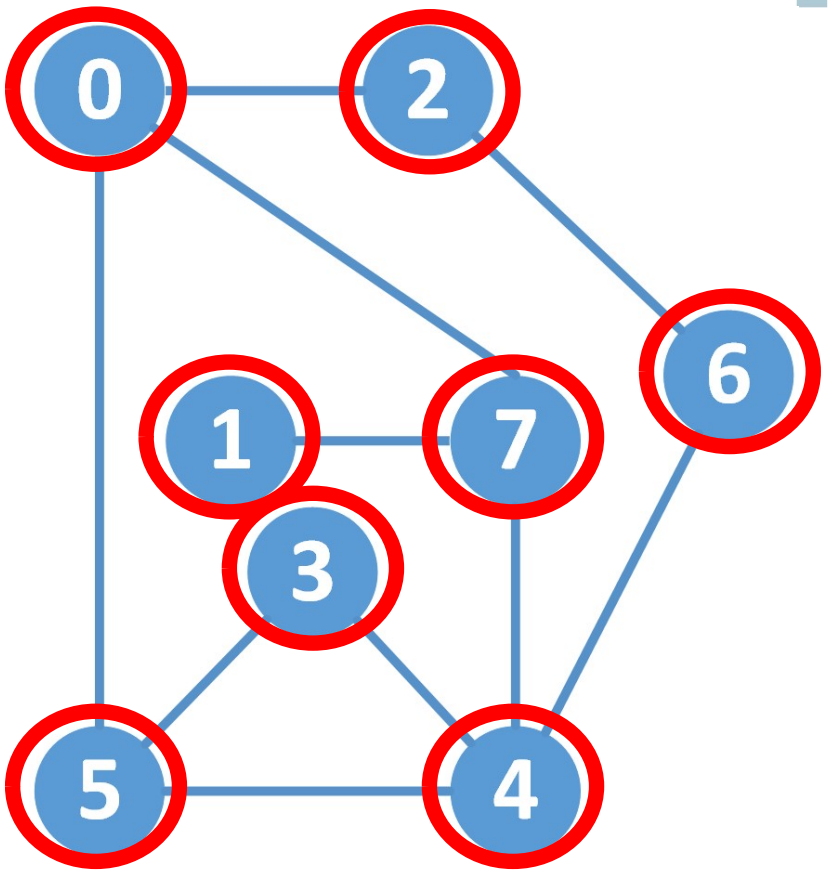
1

Излезе: 4
Соседи: сите посетени

Пребарување на графови по широчина (Breath First Search – BFS)

BFS (почнувајќи од темето s)

1. Обележи го s како посетен и стави го во редица Q
2. Повторувај се додека редицата Q не е празна:
   - 2.1. извади елемент p од редицата Q
   - 2.2. сите необележани темиња поврзани со p внеси ги во редицата Q и обележи ги како посетени



distance[]	
0	0
1	2
2	1
3	2
4	2
5	1
6	2
7	1

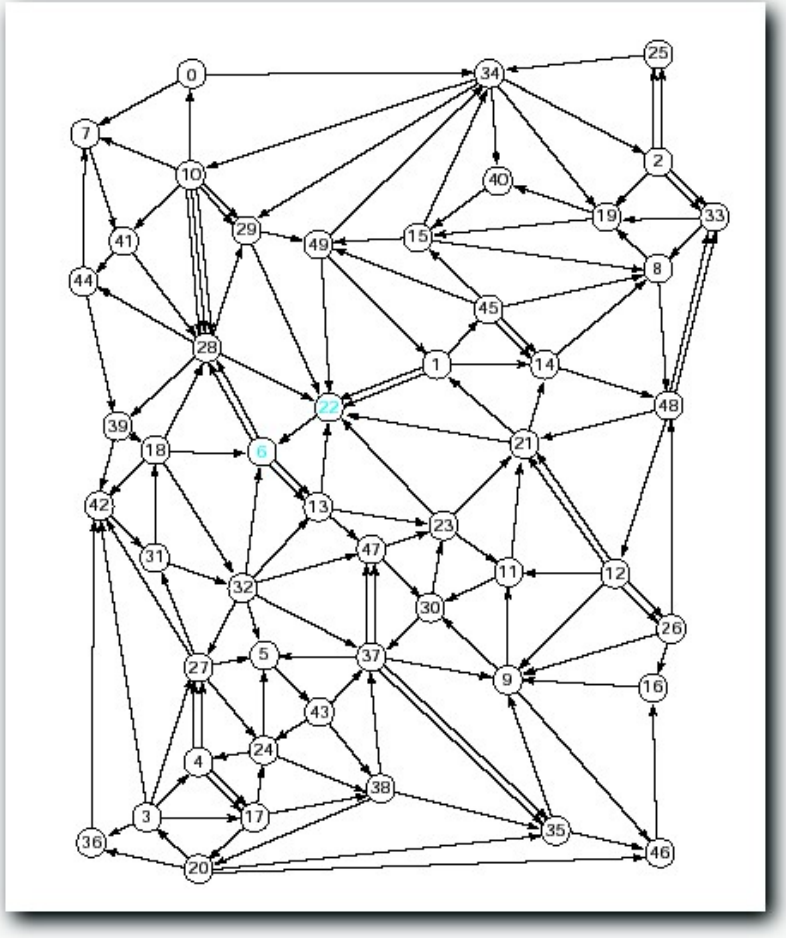
редица

Излезе: 1
Соседи: сите посетени

BFS апликација: Пајак (crawler)

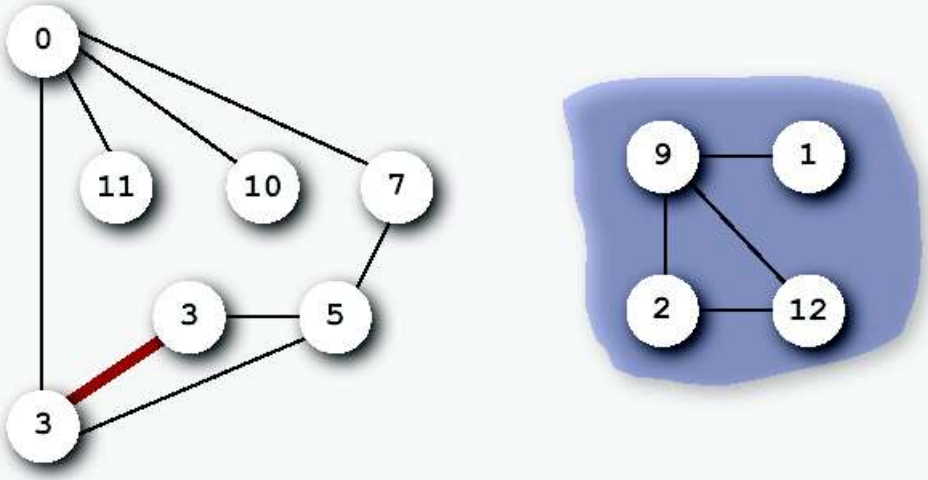
Проблем: Измини го целиот веб почнувајќи од почетна адреса X.
Решение: BFS

BFS:
- Почни од почетната страница X
- Одржувај редица Q од страници за посетување
- Одржувај множество S од пронајдени страници
- Извади страница p од Q и додади ги во Q и S сите страници кон кои покажува p. (доколку веќе ја нема во S).



Поврзани компоненти на граф

Деф: Темињата *v* и *w* се поврзани ако постои пат помеѓу тие две темиња.
Деф: Поврзана компонента на граф е максимално множество од поврзани темиња.
Цел: Препроцесирај го графот за одговори на прашањата:
Дали *v* е поврзан со *w* ? Со временска комплексност O(1)



Теме	Компонента
0	0
1	1
2	1
3	0
4	0
5	0
6	2
7	0
8	2
9	1
10	0
11	0
12	1

Поврзани компоненти на граф

Проблем: Раздели ги темињата во поврзани компоненти.

Поврзани компоненти

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.

Временска комплексност	Време на одговор	Дополнителна меморија
E + V	1	V

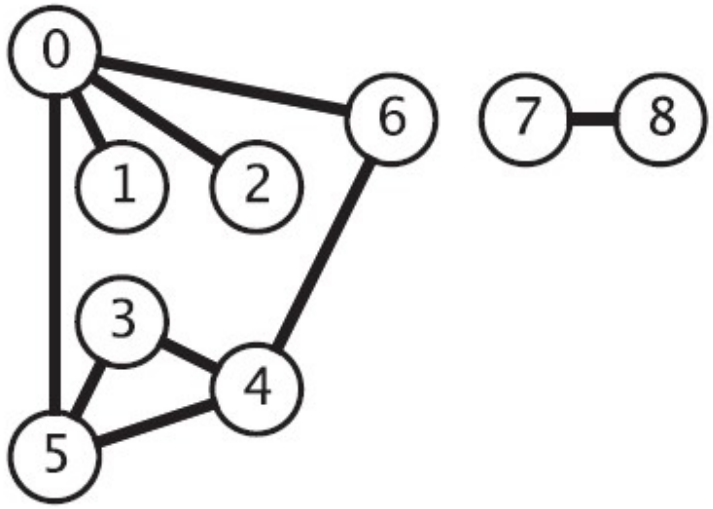
Имплементација на наоѓање на поврзани компоненти на граф

```
boolean marked[]; #obelezuvanje
int component[]; #na koja povrzana komponenta pripagja sekoe teme
int numComponents=0; #broj na komponenti

procedure CComponents(s)
begin
    for(i=0;i<V;i++) marked[i] ← 0
    for(i=0;i<V;i++)
          if(NOT marked[i]) then
               DFS(i)
               numComponents++;
end
procedure DFS(v)
begin
    marked[v] ← True
    component[v] = numComponents;
    for(w:Sosedi(v))     #sosedite se dobivaat vo zavisnost od repr.
      if(NOT marked[w]) then
        DFS(w)
end
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



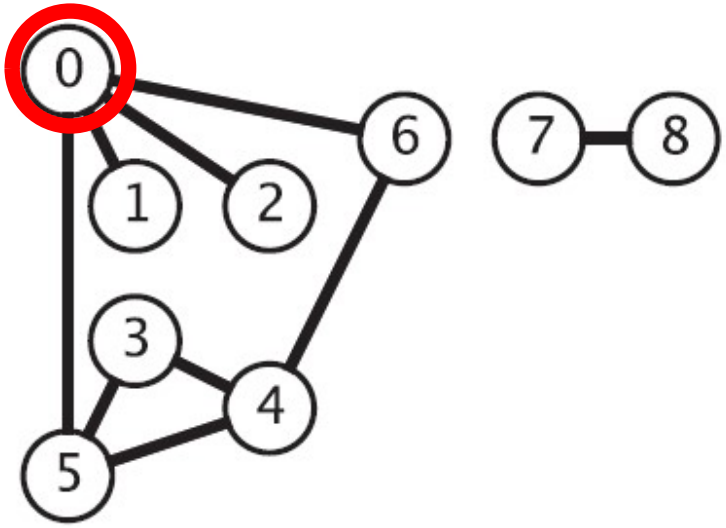
	marked[]	component[]
0		0
1		1
2		2
3		3
4		4
5		5
6		6
7		7
8		8

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



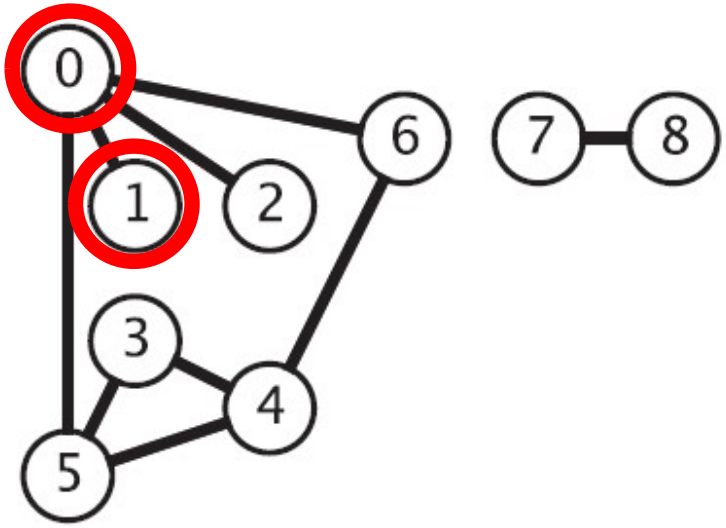
	marked[]		component[]
0	T	0	0
1		1	
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



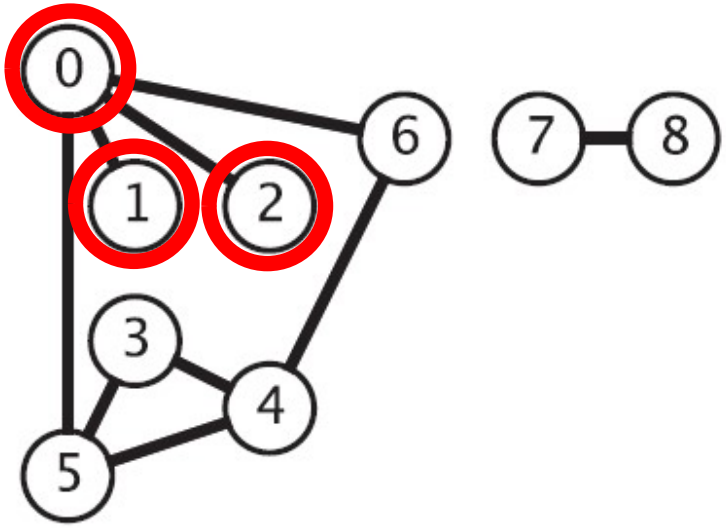
	marked[]		component[]
0	T	0	0
1	T	1	0
2		2	
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



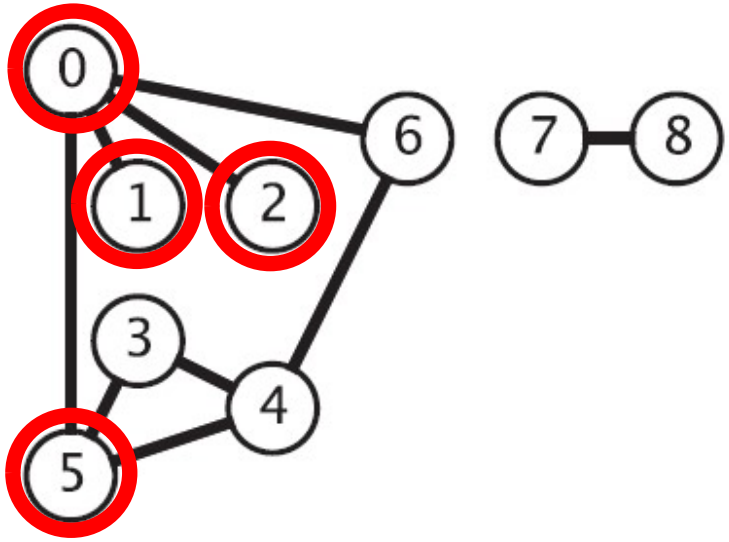
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3		3	
4		4	
5		5	
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



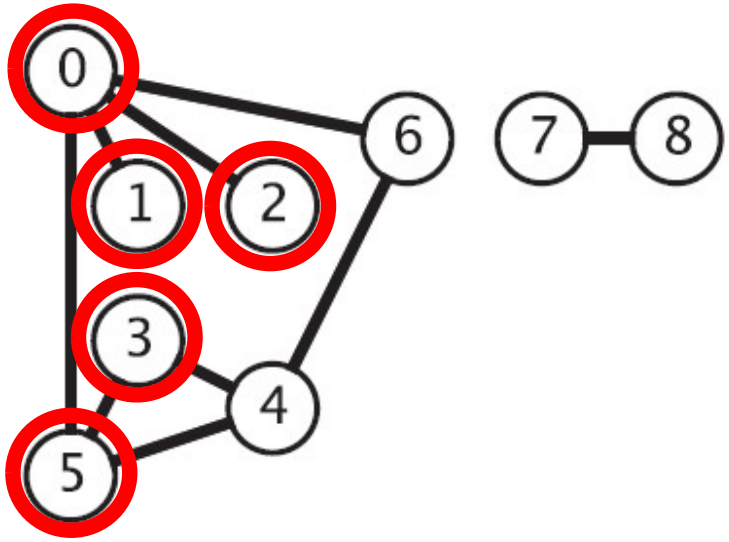
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3		3	
4		4	
5	T	5	0
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



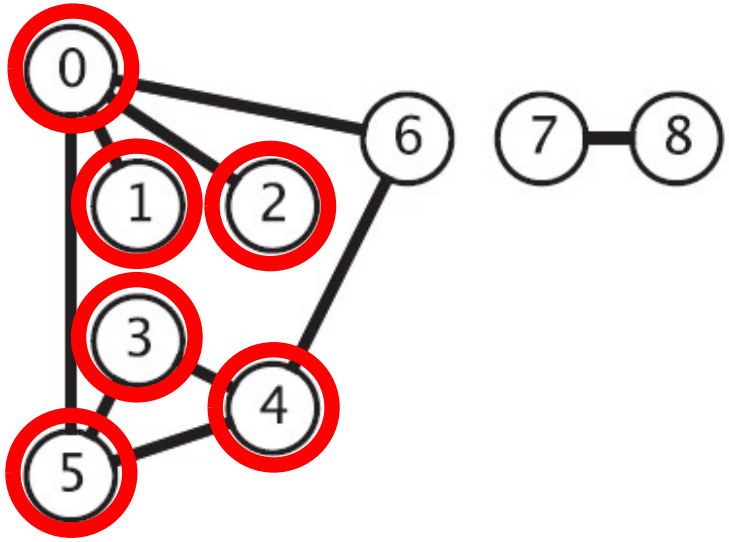
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3	T	3	0
4		4	
5	T	5	0
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



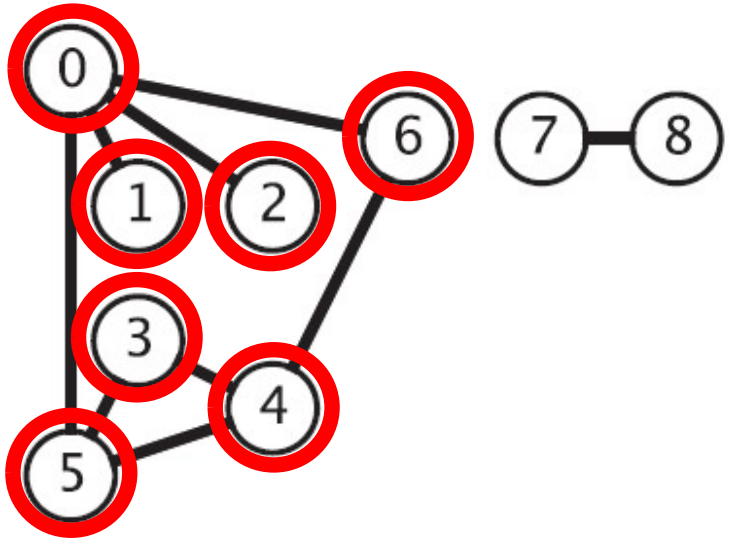
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3	T	3	0
4	T	4	0
5	T	5	0
6		6	
7		7	
8		8	

```
numComponents = 0
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



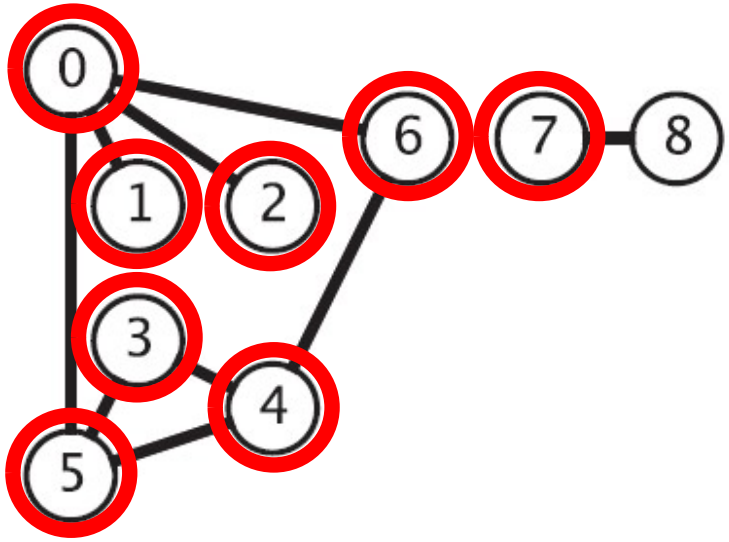
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3	T	3	0
4	T	4	0
5	T	5	0
6	T	6	0
7		7	
8		8	

```
numComponents = 1
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



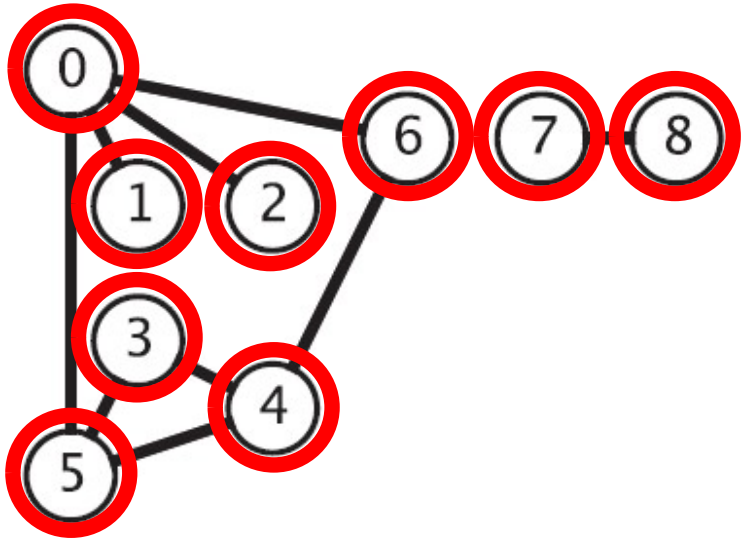
	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3	T	3	0
4	T	4	0
5	T	5	0
6	T	6	0
7	T	7	1
8		8	

```
numComponents = 1
```

Поврзани компоненти на граф

Иницијализирај ги сите темиња *v* како необележани

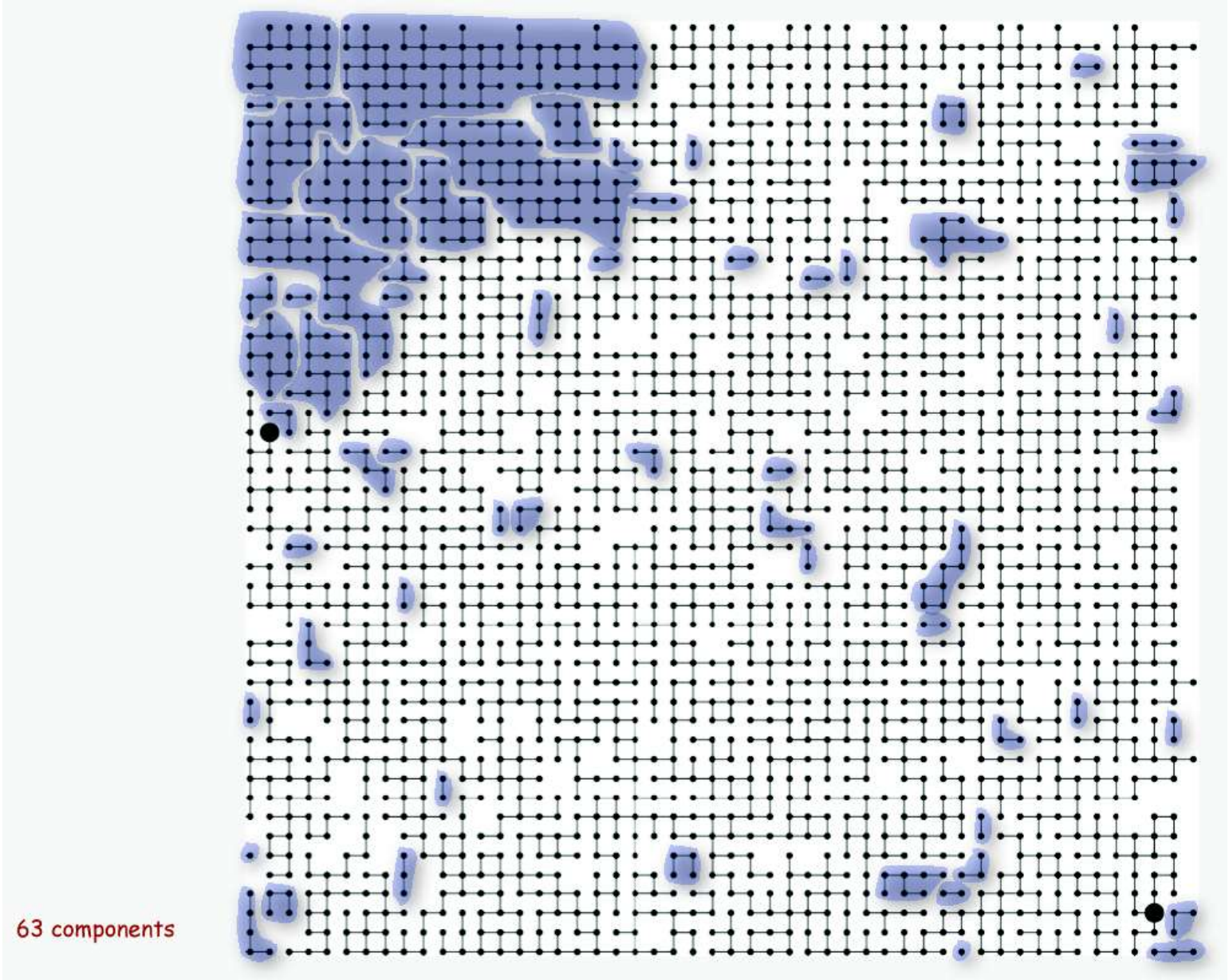
За секое необележано теме v изврши DFS почнувајќи од v, за да се идентификуваат сите темиња кои припаѓаат во компонентата на v.



	marked[]		component[]
0	T	0	0
1	T	1	0
2	T	2	0
3	T	3	0
4	T	4	0
5	T	5	0
6	T	6	0
7	T	7	1
8	T	8	1

```
numComponents = 2
```

Поврзани компоненти на граф



Поврзани компоненти, примена: Процесирање на слика

Проблем: Во 2-димензионална слика пронајди ги регионите од поврзани пиксели кои имаат иста боја.

Ефикасен алгоритам:

-Креирај граф во вид на матрица, пиксел = теме

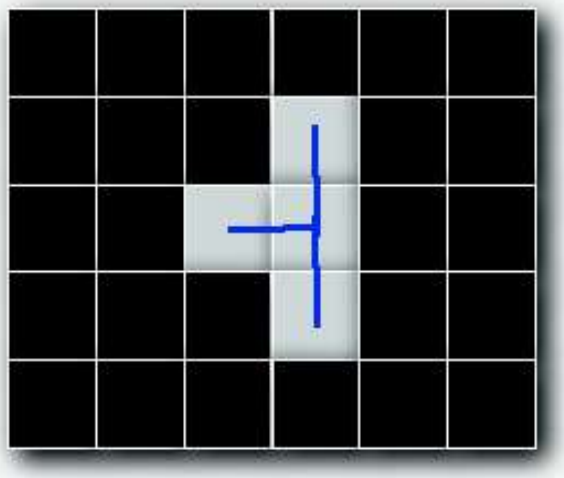
-Поврзи го секој пиксел со соседниот пиксел ако имаат иста боја

-Пронајди ги поврзаните компоненти на конструираниот граф

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

Поврзани компоненти, примена: Детекција на честички (ѕвезди)

- **Теме**: пиксел
- **Ребро**: помеѓу соседни пиксели доколку двата имаат grayscale вредност > 70
- **Честичка**: поврзана компонента со повеќе од 20-30 пиксели.

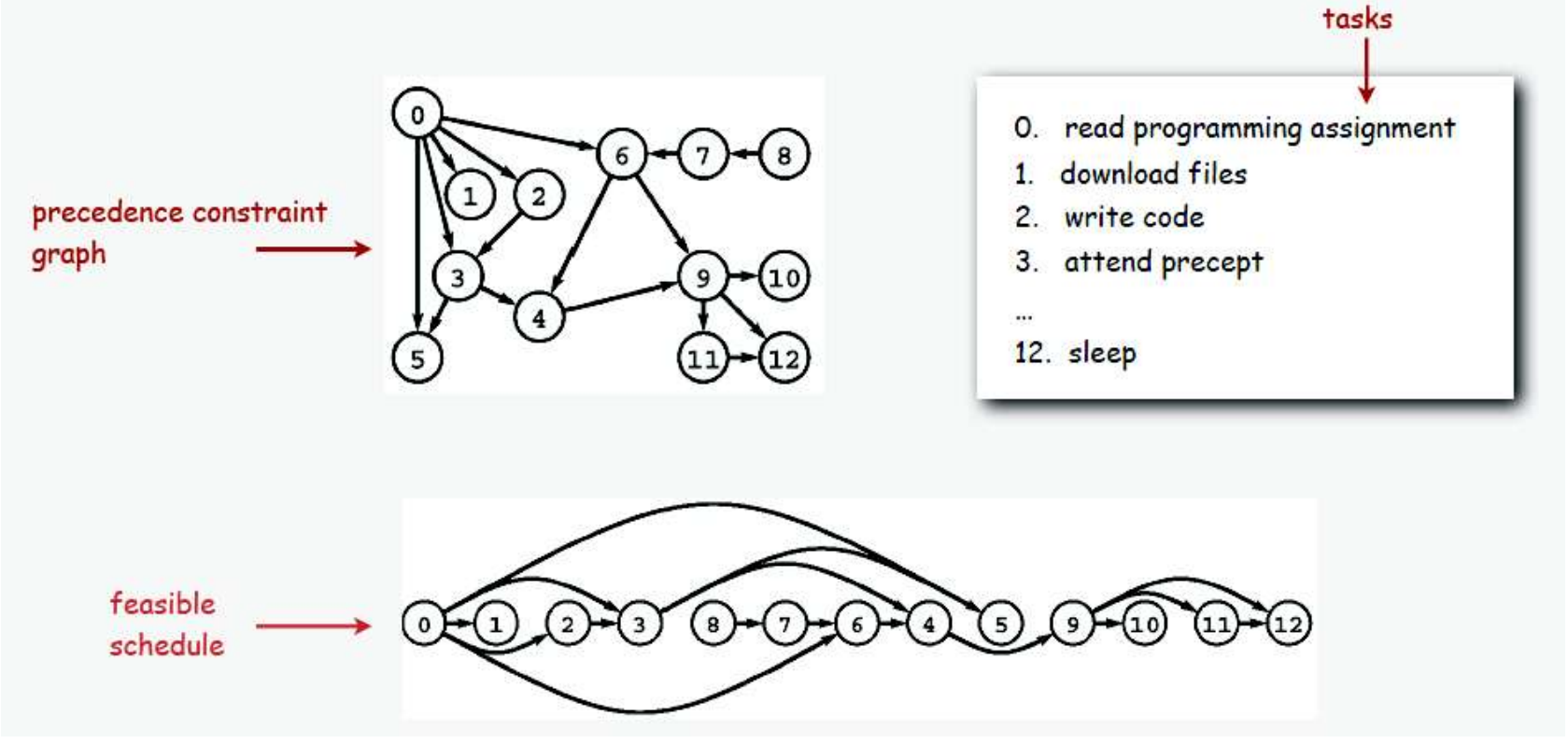


Временско планирање (Scheduling)

Проблем: За дадено множество задачи за кои постојат услови за предимство, по кој редослед треба да се извршуваат задачите?

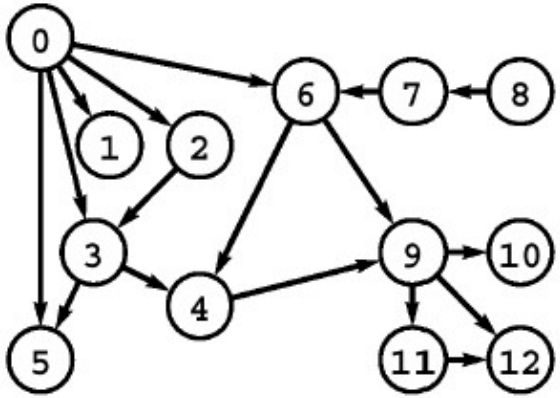
Моделирање со граф:

- Креирај теме v за секоја задача
- Креирај ребро v->w ако задачата v треба да се изврши пред задачата w

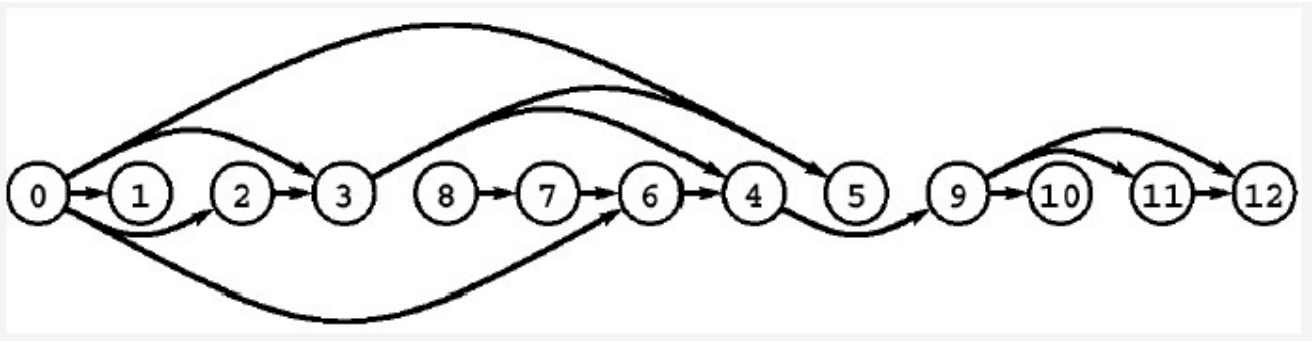


Тополошко сортирање

DAG: Directed acyclic graph (Ориентиран Ацикличен Граф)



Тополошко сортирање:
Нацртај DAG така што сите ребра ќе бидат ориентирани од лево кон десно.



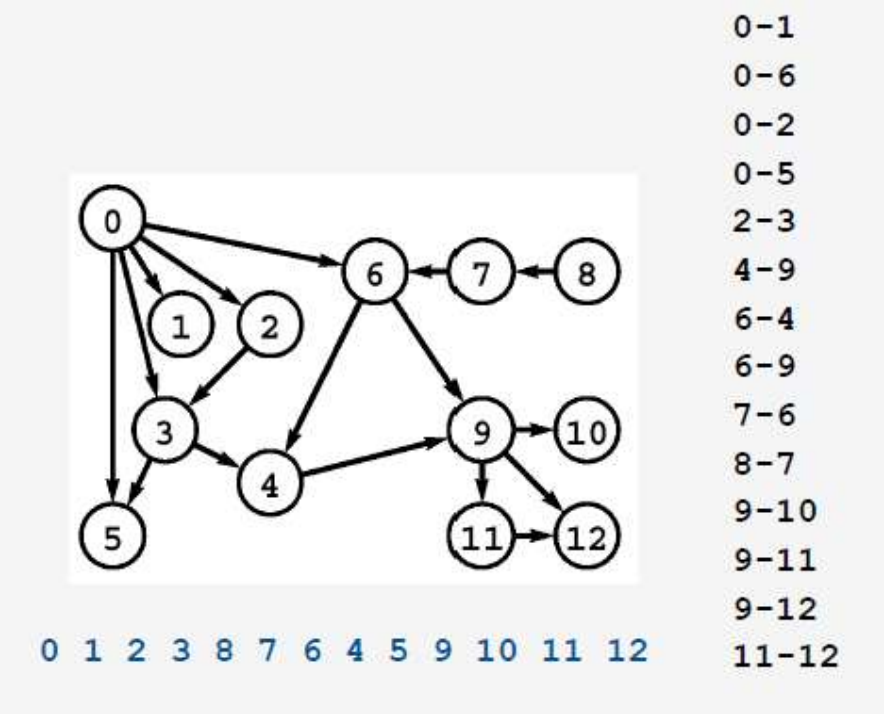
Факт: Ориентираниот граф G е DAG акко не содржи циклуси.

Тополошко сортирање

Проблем: Провери дали даден ориентиран граф G е DAG.
Ако ДА, пронајди тополошки редослед

Сложеност: Линеарно време O(V+E)

Решение:

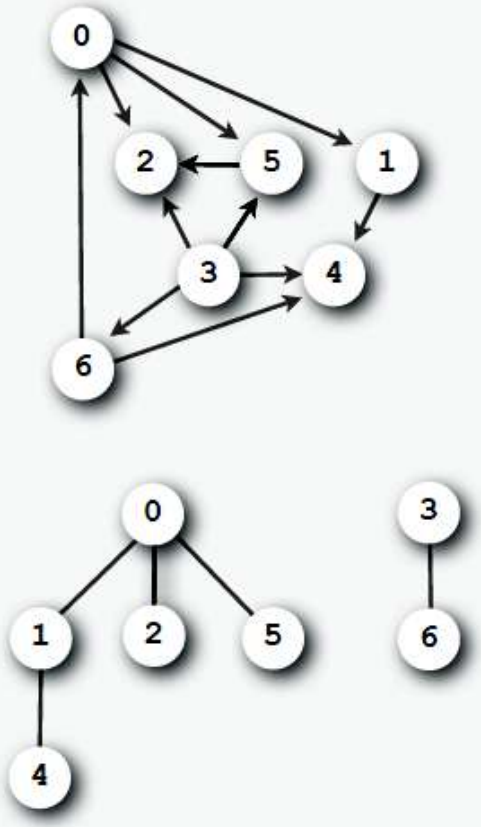


1. Изврши DFS од секое необележано теме
2. По обработка на секое теме, стави го на стек
3. На крај исчитај го стекот

Тополошко сортирање – имплементација - trace

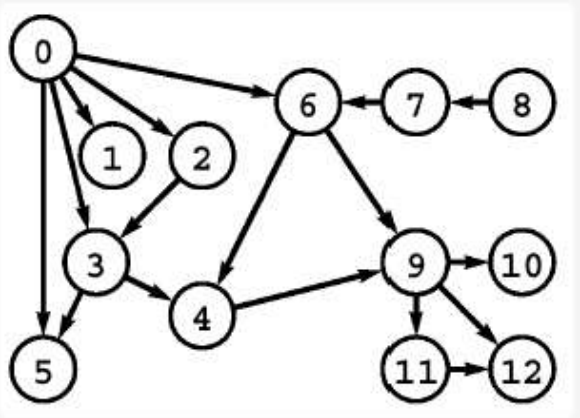
стек

	marked[]	sorted
visit 0:	1 0 0 0 0 0 0	-
visit 1:	1 1 0 0 0 0 0	-
visit 4:	1 1 0 0 1 0 0	-
leave 4:	1 1 0 0 1 0 0	4
leave 1:	1 1 0 0 1 0 0	4 1
visit 2:	1 1 1 0 1 0 0	4 1
leave 2:	1 1 1 0 1 0 0	4 1 2
visit 5:	1 1 1 0 1 1 0	4 1 2
check 2:	1 1 1 0 1 1 0	4 1 2
leave 5:	1 1 1 0 1 1 0	4 1 2 5
leave 0:	1 1 1 0 1 1 0	4 1 2 5 0
check 1:	1 1 1 0 1 1 0	4 1 2 5 0
check 2:	1 1 1 0 1 1 0	4 1 2 5 0
visit 3:	1 1 1 1 1 1 0	4 1 2 5 0
check 2:	1 1 1 1 1 1 0	4 1 2 5 0
check 4:	1 1 1 1 1 1 0	4 1 2 5 0
check 5:	1 1 1 1 1 1 0	4 1 2 5 0
visit 6:	1 1 1 1 1 1 1	4 1 2 5 0
leave 6:	1 1 1 1 1 1 1	4 1 2 5 0 6
leave 3:	1 1 1 1 1 1 1	4 1 2 5 0 6 3
check 4:	1 1 1 1 1 1 0	4 1 2 5 0 6 3
check 5:	1 1 1 1 1 1 0	4 1 2 5 0 6 3
check 6:	1 1 1 1 1 1 0	4 1 2 5 0 6 3



Ориентиран циклус

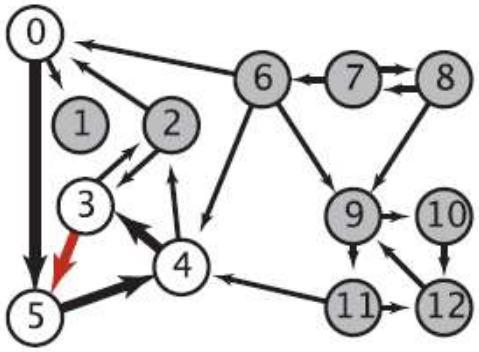
Проблем: Провери дали даден ориентиран граф G е содржи циклус.
Сложеност: Линеарно време O(V+E)




```
0-1
0-6
0-2
0-5
2-3
4-9
6-4
6-9
7-6
8-7
9-10
9-11
9-12
11-12
```

Ориентиран циклус

Проблем: Провери дали даден ориентиран граф G е содржи циклус.
Сложеност: Линеарно време O(V+E)



	marked[] 0 1 2 3 4 5 ...	onStack[] 0 1 2 3 4 5 ...
dfs(0)		
dfs(5)	1 0 0 0 0 0	1 0 0 0 0 0
dfs(4)	1 0 0 0 0 1	1 0 0 0 0 1
dfs(3)	1 0 0 0 1 1	1 0 0 0 1 1
check 5	1 0 0 1 1 1	1 0 0 1 1 1

Решение:

1. Изврши DFS од секое необележано теме
2. За секое теме означи дека е обележано и дел од моменталната патека (на стек)
   2.1. Доколку темето има сосед кој е обележан и е дел од моментална патека – циклус
3. Кога ќе се заврши со обработка на темето треба да се извади од моменталната патека

Детекција на циклус - JAVA

```
public class A extends B
{
    ...
}
```

```
public class B extends C
{
    ...
}
```

```
public class C extends A
{
    ...
}
```

```
% javac A.java
A.java:1: cyclic inheritance
involving A
public class A extends B { }
       ^
1 error
```