Algoritmi in podatkovne strukture 1

Visokošolski strokovni študij Računalništvo in informatika

Algoritmi na grafih

Jurij Mihelič, UniLj, FRI

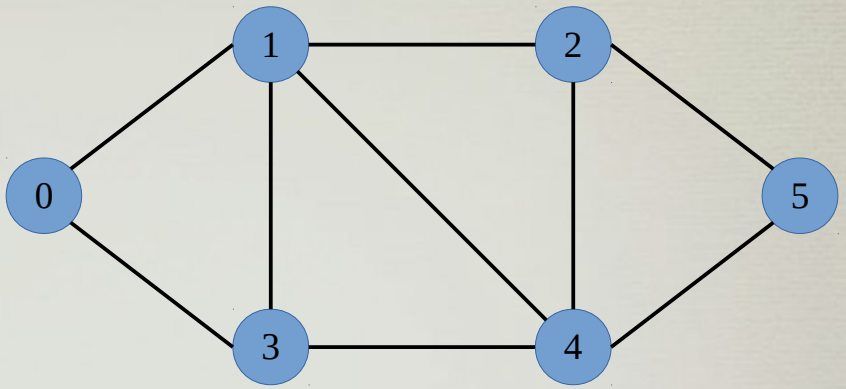
Poti v grafih

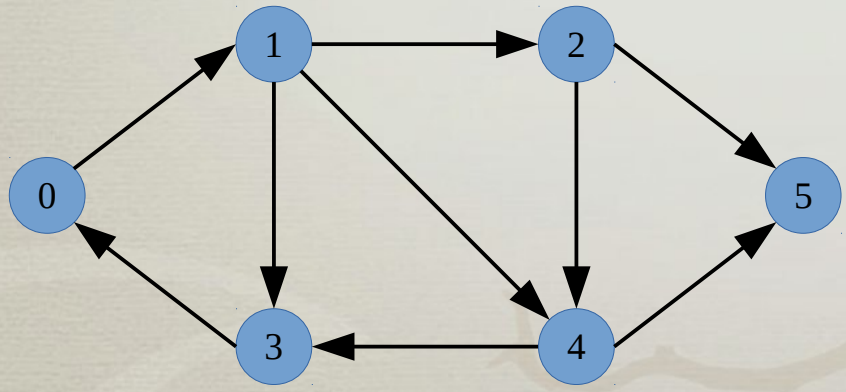
- Sprehod (*walk*)
  - zaporedje povezav oz. zaporedje vozlišč

    $(v_1,v_2), (v_2,v_3), (v_3,v_4), \ldots, (v_{l-1},v_l)$ oz. $v_1, v_2, v_3, v_4, \ldots, v_l$
  - začetek naslednje povezave je enak koncu prejšnje
  - odprt in zaprt sprehod
- Steza (*trail*)
  - vsaka povezava nastopa le enkrat
- Pot (*path*)
  - vsako vozlišče nastopa le enkrat
- Cikel
  - zaprta pot

Število sprehodov

- Množenje matrike sosednosti
  - $A, A^2, A^3, \ldots$
  - A^l … št. sprehodov dolžine natanko l
- Časovna zahtevnost
  - $l \cdot O(\mathrm{MM})$
  - MM … matrično množenje

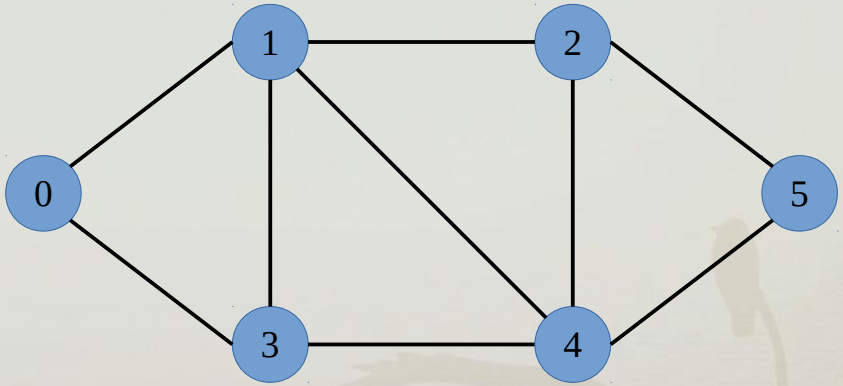




	0	1	2	3	4	5
0	-	1	-	1	-	-
1	1	-	1	1	1	-
2	-	1	-	-	1	1
3	1	1	-	-	1	-
4	-	1	1	1	-	1
5	-	-	1	-	1	-

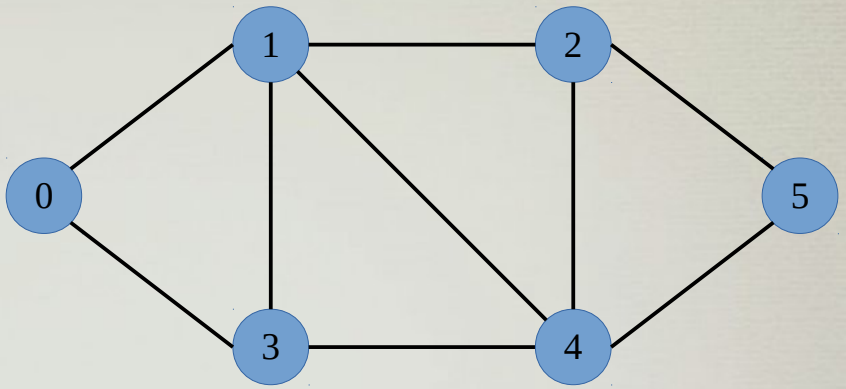
Dosegljivost

- Dosegljivost vozlišč
  - Ali obstaja pot iz u v v?
  - Kako dolga je lahko najdaljša pot?
  - Pregledamo matrike A, A^2, A^3, …, A^{n-1}
  - $O(n \cdot MM)$



Število trikotnikov

- Kaj je trikotnik (v grafu)?
- Kje v A^3 najdemo št. trikotnikov?
- Št. trikotnikov
  - $\mathrm{tr}(A^3) / 6$



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

Obhodi grafov

- Iskanje v globino (*depth-first search*)
  - načelo **poguma**
  - gremo naprej, če le lahko
  - lahko pridemo zelo daleč od začetka

- Iskanje v širino (*breadth-first search*)
  - načelo **previdnosti**
  - najprej raziščemo vso svojo okolico

Iskanje v globino

- Ideja algoritma
  - začnemo v
    poljubnem ne-obiskanem vozlišču
  - izpišemo vozlišče
  - rekurzivno obiščemo
    poljubnega ne-obiskanega soseda
  - če ni nobenega ne-obiskanega soseda,
    se **vrnemo** en korak nazaj
  - kdaj se rekurzija ustavi?
  - na kateri podatkovni strukturi temelji algoritem?

Iskanje v globino

- Gozd iskanja v globino
  - sestoji iz **dreves iskanja v globino**
  - kdaj je rezultat drevo in kdaj gozd?

- Izrek
  - $\mathrm{dfs}(v)$ obišče vsa iz v dosegljiva vozlišča

- Vrstni red obiskovanja
  - vstop – ko vozlišče prvič obiščemo
  - izstop – ko vozlišče zadnjič obiščemo

Iskanje v širino

- Ideja algoritma
  - začnemo v **poljubnem ne-obiskanem** vozlišču
  - izpišemo vozlišče
  - vse **sosede** dodamo **v vrsto** za obiskovanje
  - naslednjega obiščemo **prvega** iz vrste

Iskanje v širino

- Gozd iskanja v širino
  - sestoji iz **dreves iskanja v širino**
  - kdaj je rezultat drevo in kdaj gozd?

- Izrek
  - $\mathrm{bfs}(v)$ obišče vsa iz v dosegljiva vozlišča

- Vrstni red obiskovanja
  - ob dodajanju v vrsto

DFS vs BFS

- pogum
- globina
- sklad (implicitno)
- gozd iskanja v globino
- dva vrstna reda
  - vstopni in izstopni

- previdnost
- širina
- vrsta
- gozd iskanja v širino
- en vrstni red

- obišče vsa dosegljiva vozlišča
- $O(n+m)$ – seznam sosedov
- $O(n^2)$ – matrika sosednosti

Psevdokoda DFS in BFS

- DFS

```
fun dfs(v) is
    // oznamičmo vozlišče v
    time += 1; mark[v] = time
    forall u in N(v) do
        if mark[u] == 0 then dfs(u)
```

- Skupno

```
fun dfs/bfs_init() is
    forall v in V do mark[v] = 0
    time = 0

fun dfs/bfs_full() is
    dfs/bfs_init()
    forall v in V do
        if mark[v] == 0 then dfs/bfs(v)
```

- BFS

```
fun bfs(v) is
    q = Queue()
    // označimo začetno vozlišče v
    time += 1; mark[v] = time
    q.enqueue(v)
    while not q.empty() do
        v = q.dequeue()
        forall u in N(v) do
            if mark[u] == 0 then
                // označimo vozlišče u
                time += 1; mark[v] = time
                q.enqueue(u)
```

Uporaba DFS / BFS

- Dosegljivost vozlišč
  - neusmerjeni ali usmerjeni graf
  - je iz vozlišča u vozlišče v dosegljivo?

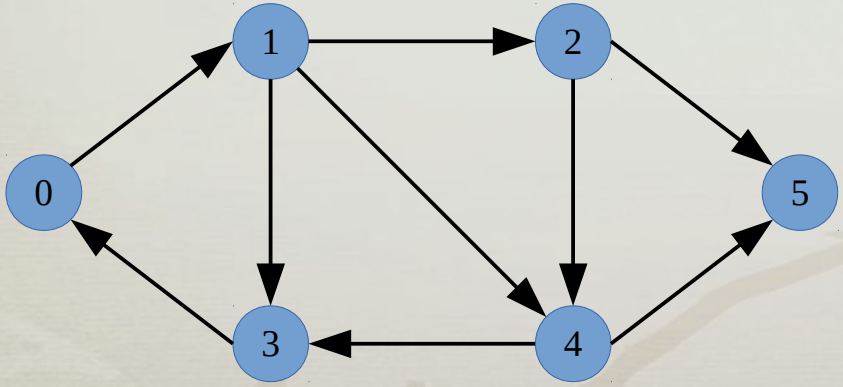
- Cikličnost grafa
  - neusmerjeni ali usmerjeni graf
  - ali je graf cikličen / acikličen?

Uporaba DFS / BFS

- Najkrajša pot
  - dolžina poti je enaka številu povezav na poti
  - ali DFS najde najkrajšo pot?
  - ali BFS najde najkrajšo pot?

Topološko urejanje

- Problem
  - topološko razvrsti vozlišča usmerjenega grafa
- Topološko
  - če $uv \in E$, potem je u pred v
- Primer
  - kje je težava?

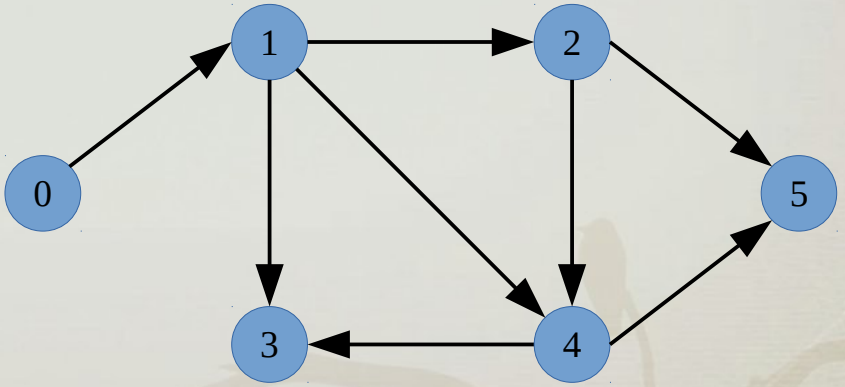


DAG – directed acyclic graph

Topološko urejanje

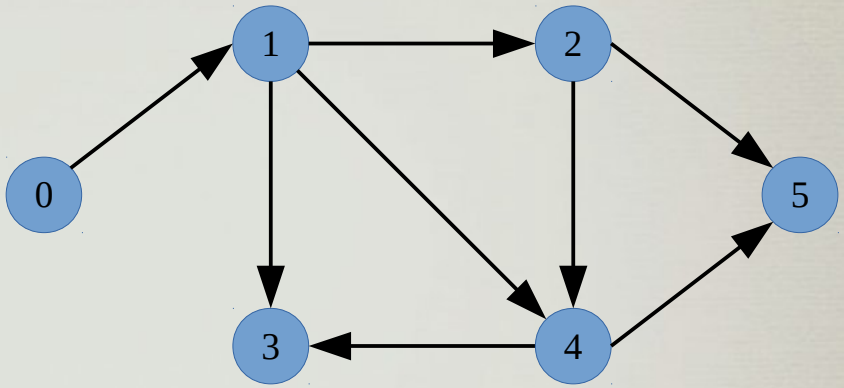
- Ideja algoritma (preko DFS)
  - izvedemo DFS na celotnem grafu
    - povezava do že obiskanega vozlišča ⇒ cikel
  - **izstopni** vrstni red obiskovanja
  - v **obratnem** vrstnem redu

- Pravilnost
- Detekcija ciklov



Topološko urejanje

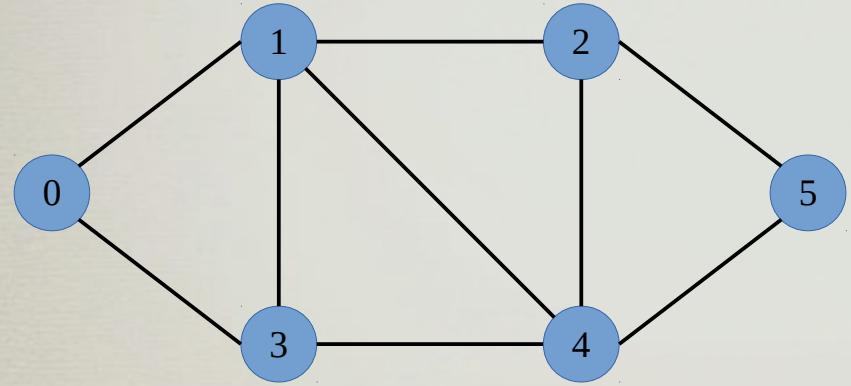
- Ideja algoritma (odstranjevanje vozlišč)
  - odstranimo vsa vozlišča z vhodno stopnjo 0
  - jih dodamo v seznam
  - ponavljamo postopek

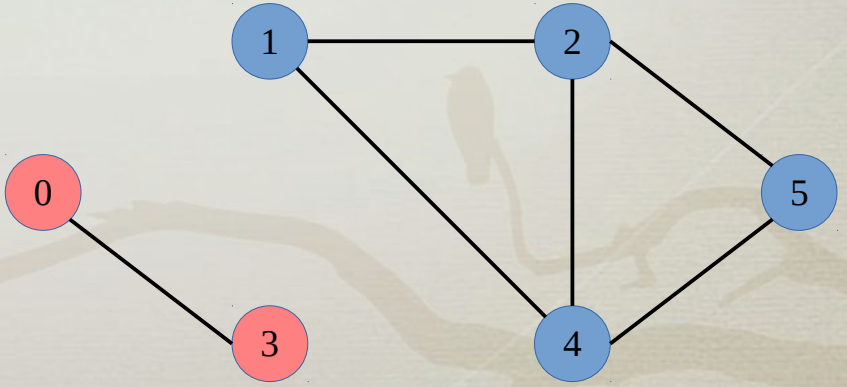


- Izrek
  - vsak DAG ima vsaj eno vozlišče z vhodno stopnjo 0

Povezanost neusmerjenega grafa

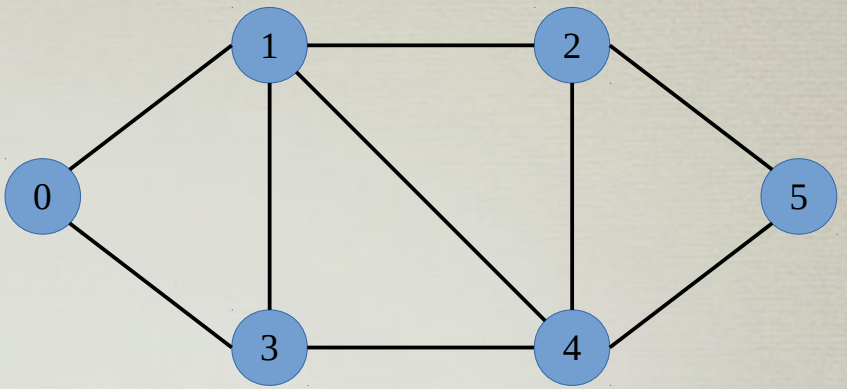
- Neusmerjeni graf
  - je **povezan**, če
    med vsakim parom vozlišč
    obstaja pot



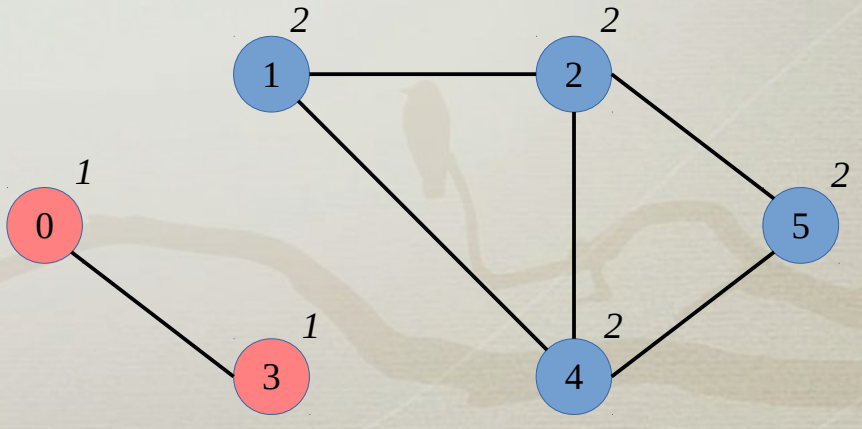


Povezanost neusmerjenega grafa

- Preverjanje povezanosti
  - uporabimo dfs(*v*)
  - če so **vsa** vozlišča označena, je graf povezan



- Povezane komponente
  - razdelitev grafa na največje povezane podgrafe
  - vozlišča v isti komponenti enako označimo
  - vsaka uporaba dfs(*v*) drugače označuje vozlišča

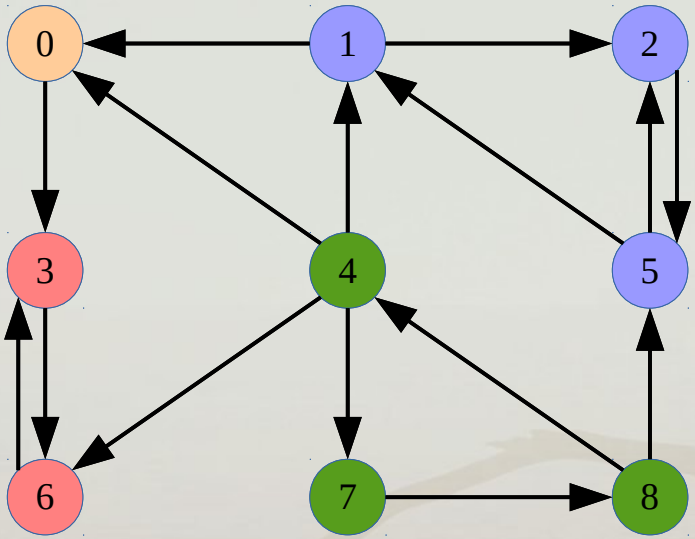


Povezanost usmerjenega grafa

- Usmerjeni graf
  - je **šibko povezan**, če je ustrezen neusmerjeni graf povezan
    - vse usmerjene povezave spremenimo v neusmerjene
  - je **povezan**, če za vsak par vozlišč u in v obstaja pot iz u v v ali v v u
  - je **krepko povezan**, če za vsak par vozlišč u in v obstaja pot iz u v v in v v u

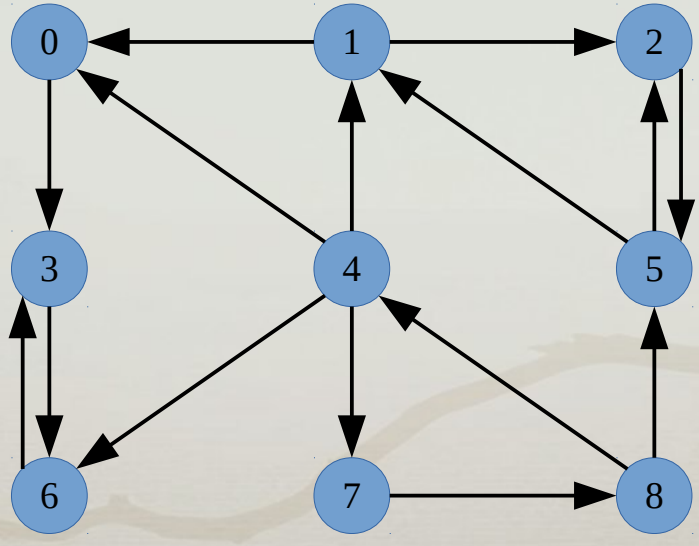
Krepko povezane komponente

- Definicija
  - SCC – *strongly connected components*
  - razdelitev grafa na največje krepko povezane podgrafe
- Primer



Krepko povezane komponente

- Kosaraju-ov algoritem
  - izračunaj **izstopni vrstni red** obiskovanja
  - **transponiraj** graf (obrni povezave)
  - v **obrnjenem** izstopnem vrstnem redu zaporedoma izvajaj dfs(v)
    - vsaka uporaba dfs(v) označi eno komponento



Krepko povezane komponente

- Tarjan-ov algoritem
  - predelava dfs(v)
    - mark … čas prvega obiska (vstopni vrstni red)
    - low … najmanjši čas obiska v naknadno obiskanih vozliščih
  - morebitni popravki vrednost low
    - po obisku neoznačenega sosednjega vozlišča
    - po detekciji povratne povezave na vozlišče na skladu