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

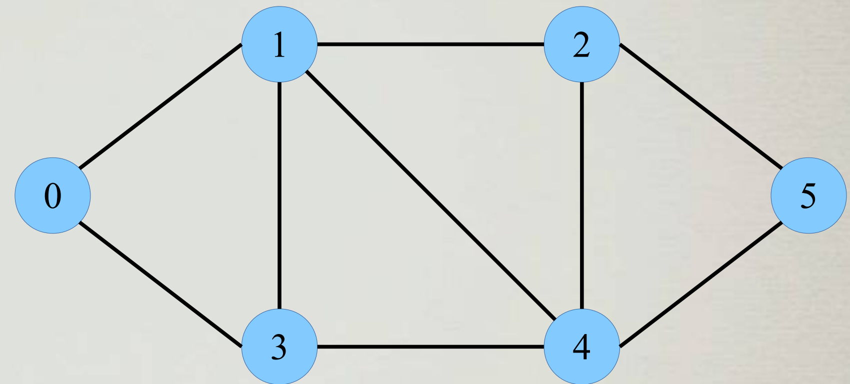
    $(v_1,v_2), (v_2,v_3), (v_3,v_4), \ldots, (v_{l-1},v_l)$ oz. $v_1, v_2, v_3, v_4, \ldots, v_l$
  - začetek naslednje povezave je enak koncu prejšnje
- Steza (*trail*)
  - sprehod, kjer vsaka povezava nastopa le enkrat
- Pot (*path*)
  - sprehod oz. steza, kjer vsako vozlišče nastopa le enkrat

Poti v grafih

- Dolžina sprehoda
  - število povezav v sprehodu
- Cena sprehoda
  - vsota cen povezav v sprehodu
- Cikel
  - zaprta pot

Število sprehodov

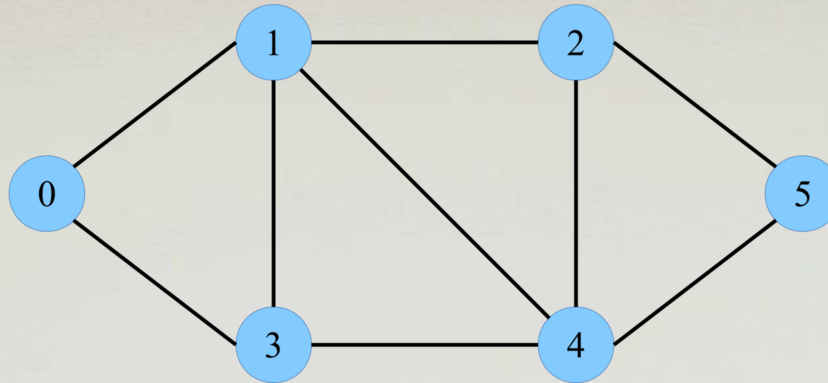
- Problem
  - za **vsak par** vozlišč u in v
    poišči **število sprehodov dolžine** l
    s pričetkom v u in koncem v v



Št. sprehodov dolžine 3
med 2 in 3:
- 2103
- 2143
- 2413
- 2543

Število sprehodov

- Problem
  - vhod



A	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	–

  - izhod

A^2	0	1	2	3	4	5
0	2	1	1	1	2	0
1	1	4	1	2	2	2
2	1	1	3	2	2	1
3	1	2	2	3	1	1
4	2	2	2	1	4	1
5	0	2	1	1	1	2

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

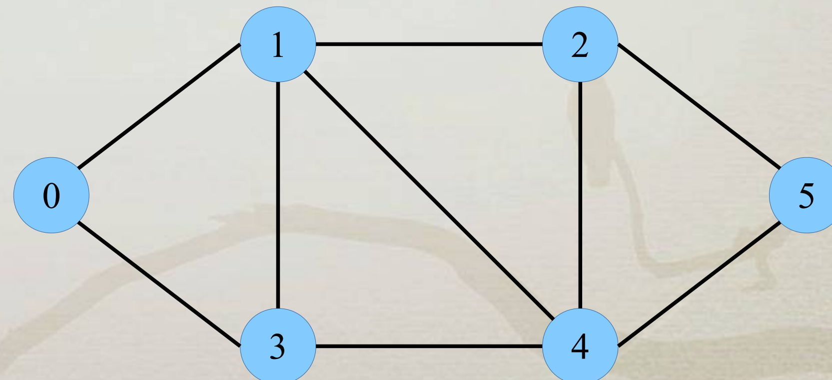
Število sprehodov

- Množenje matrike sosednosti
  - $A, A^2, A^3, \ldots$
  - $A^l = A^{l-1} \cdot A$
  - A^l … št. sprehodov dolžine natanko l

- Časovna zahtevnost
  - $l \cdot O(\mathrm{MM})$
  - MM … matrično množenje

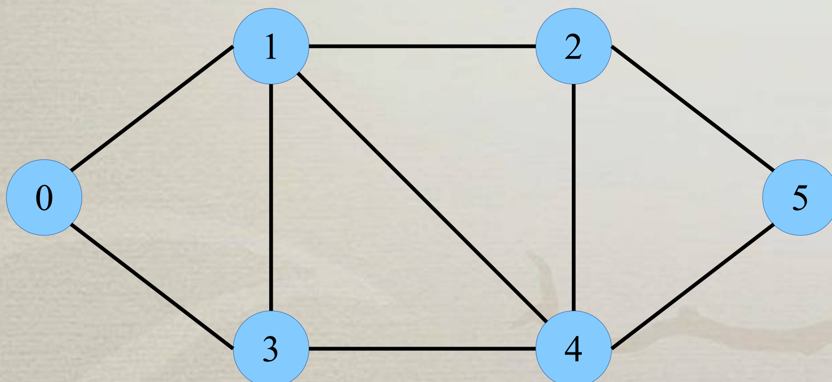
Dosegljivost

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



Število trikotnikov

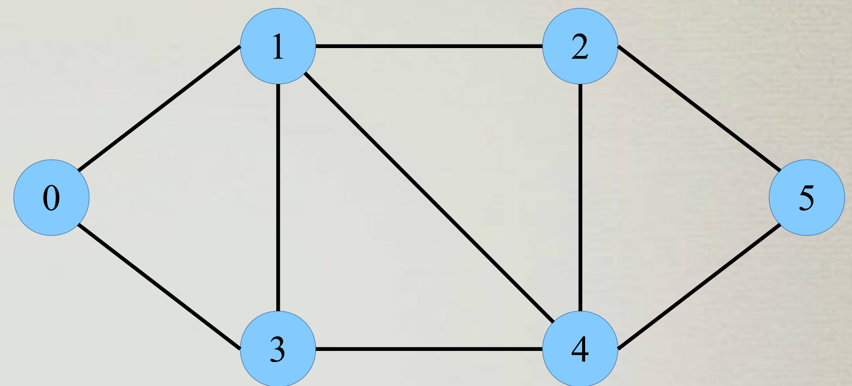
- Motivacija
  - iskanje vzorcev v grafih (družabna omrežja)
    - grafek, graflet (angl. graphlet) = majhen graf
  - koeficient gručenja (angl. clustering coefficient)
    - visok: predstavlja tesno povezane skupnosti
    - cc(v) = število povezanih sosedov / število vseh možnih povezav med sosedi
    - št. povezanih sosedov = št. trikotnikov v vozlišču / 2



cc(2) = 2 / 3
N(2) = { 1, 4, 5 }
povezave med sosedi: 14, 45
možne povezave med sosedi: 14, 15, 45

Število trikotnikov

- Problem
  - v danem grafu preštej trikotnike

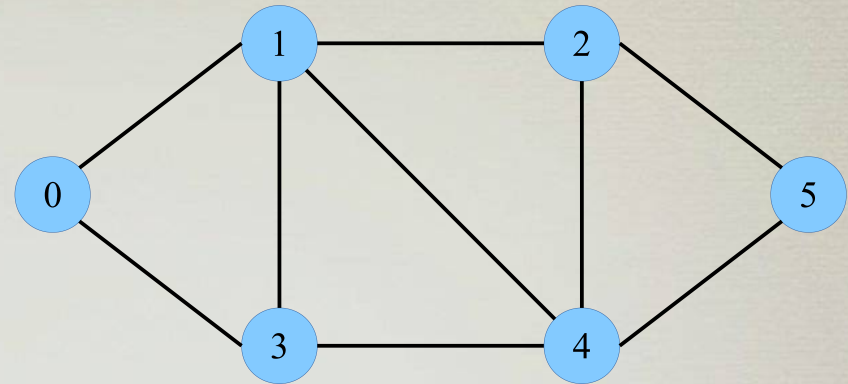


- Vprašanja
  - Kaj je trikotnik?
  - Kakšen sprehod je trikotnik?
  - Koliko trikotnikov se začne vozlišču 0?
  - Koliko trikotnikov vsebuje vozlišče 0?
  - Katero matriko A, A^2, A^3, … potrebujemo?
  - Kje v izbrani matriki najdemo št. trikotnikov?

Število trikotnikov

- Ideja
  - uporabimo sled matrike

$$tr(A^3)/6 = 1/6 \sum_{i=0}^{n-1} a_{ii}^3$$

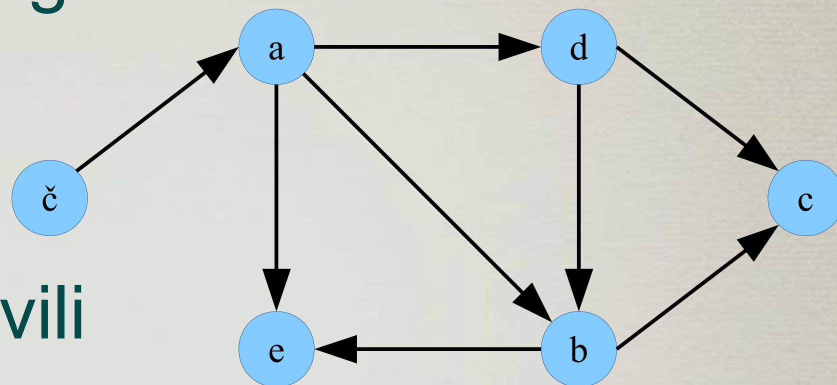


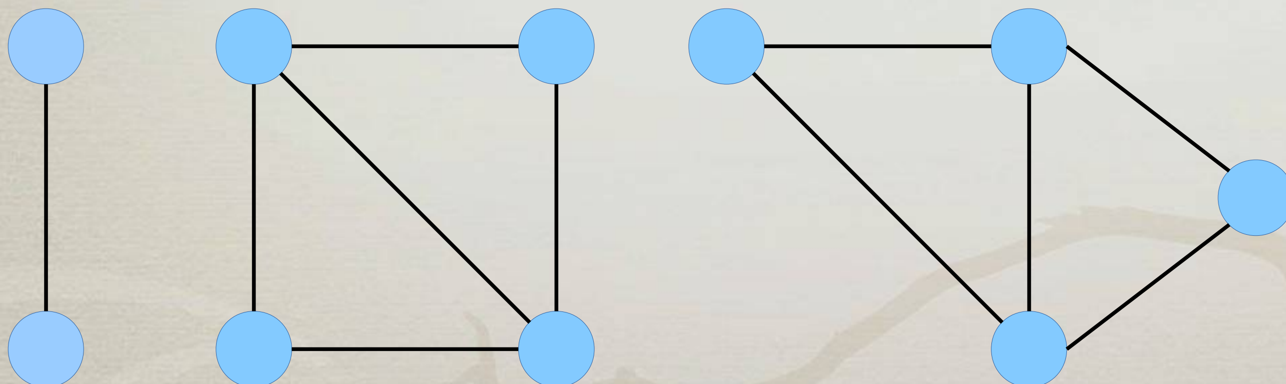
- Algoritem
  - izračunaj A^3
  - izračunaj sled matrike
    - seštej diagonalne vrednosti
  - deli s 6

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

- Motivacija
  - sistematičen pregled vozlišč grafa
  - iskanje poti in ciklov v grafu
  - iskanje dostopnih vozlišč
  - iskanje odvisnosti med opravili
  - iskanje povezanih komponent





Obhodi grafov

- Iskanje v globino (*depth-first search*)
  - načelo **poguma**
  - gremo naprej, če le lahko
  - lahko hitro pridemo
    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
  - ponavljamo
    - poiščemo **poljubnega ne-obiskanega** soseda
    - ga rekurzivno obiščemo
    - če takšnega soseda ni, zaključimo zanko in se vrnemo en nivo višje

Iskanje v globino

- Psevdokoda

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

```
fun dfs(v) is
    println("Vstop: " + v)

    // označimo vozlišče v
    time += 1
    mark[v] = time

    // rekurzivno obiščemo neoznačene sosede
    forall u in N(v) do
        if mark[u] == 0 then dfs(u)

    println("Izstop: " + v)
```

Iskanje v globino

- Vrstni red vozlišč
  - vstopni – ko vstopimo v vozlišče
  - izstopni – ko iz vozlišča izstopimo

- Katera vozlišča obišče $dfs(v)$?
  - $dfs(v)$ obišče vsa iz v dosegljiva vozlišča
  - zakaj?
    - $dfs(v)$ obišče vozlišče v in tudi vse sosede od v
    - $v \quad N(v) \quad N(N(v)) \quad \dots$
  - kolikokrat obiščemo vsako vozlišče?

Iskanje v globino

- Kako obiskati vsa vozlišča grafa?

```
fun dfs_full() is
    dfs_init()
    forall v in V do
        if mark[v] == 0 then dfs(v)
```

- Gozd iskanja v globino
  - sestoji iz **dreves iskanja v globino**
  - kdaj je rezultat drevo in kdaj gozd?
  - kaj pa na usmerjenih grafih?

Iskanje v globino

- Časovna zahtevnost
  - opazimo, da se $dfs(v)$ kliče natanko enkrat za vsako vozlišče
    - ker klic $dfs(v)$ vedno varujemo s pogojem $\mathrm{mark[v]} == 0$
  - zanka forall v $dfs(v)$ se izvede $deg(v)=|N(v)|$ krat
  - torej vsi klici $dfs(v)$ porabijo

$$\sum_{v \in V} |N(v)| = 2\mathrm{m} = O(m)$$

  - Kaj pa $dfs_full()$?
    - $O(n + m)$

Iskanje v širino

- Ideja algoritma
  - začnemo v **poljubnem ne-obiskanem** vozlišču
  - najprej obdelamo oz. izpišemo vse sosede
  - nato jih še obiščemo

- Katero podatkovno strukturo potrebujemo?

Iskanje v širino

- Psevdokoda

```
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)
```

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

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

Iskanje v širino

- Gozd iskanja v širino
  - sestoji iz **dreves iskanja v širino**
  - kdaj je rezultat drevo in kdaj gozd?
- Izrek
  - $\text{bfs}(v)$ obišče vsa iz v dosegljiva vozlišča
- Vrstni red obiskovanja
  - ob dodajanju v vrsto

Iskanje v širino

- Časovna zahtevnost
  - $O(n+m)$
  - Zakaj?
    - vsako vozlišče v vrsto dodamo kvečjemu enkrat
    - v zanki preverimo vse sosede
    - skupno je to torej O(m), kot pri DFS
    - celoten algoritem pa je O(n+m)

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
  - Je iz vozlišča u vozlišče v dosegljivo?
  - neusmerjeni ali usmerjeni graf
  - ideja
    - poženemo DFS ali BFS iz u
    - preverimo ali smo označili vozlišče v

Uporaba DFS / BFS

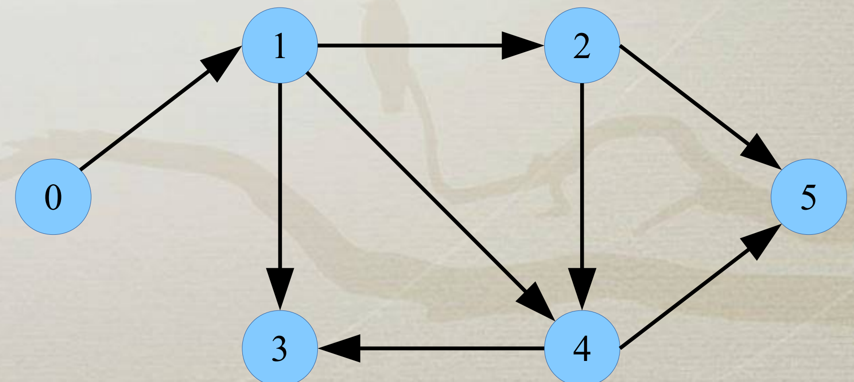
- Cikličnost grafa
  - Ali je graf cikličen / acikličen?
  - neusmerjeni ali usmerjeni graf
  - ideja
    - malenkost popravimo algoritem
    - če med preiskovanjem naletimo na že obiskanega soseda, potem ima graf cikel

Uporaba DFS / BFS

- Najkrajša pot
  - iskanje najkrajše poti
    od začetnega vozlišča do ostalih
    - dolžina poti je enaka številu povezav na poti
  - Ali DFS najde najkrajšo pot?
  - Ali BFS najde najkrajšo pot?
  - ideja
    - popravimo algoritem, da beleži tudi oddaljenost od začetnega vozlišča

Topološko urejanje

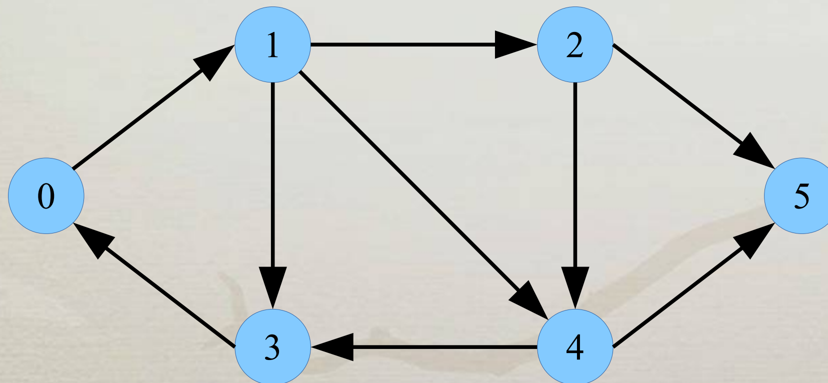
- Motivacija
  - razvrsti opravila, da slediš odvisnostmi
    - razvrščanje strojnih ukazov v procesorju
  - razreševanje odvisnosti v aritmetičnih izrazih
    - prevajalniki, zbirniki, tolmači, Excel preglednice
  - prevajanje izvorne kode
    - simbolne odvisnosti pri povezovanju (linker)
    - odvisnosti med knjižnicami, moduli ipd.
  - sinteza logičnih vezij



Topološko urejanje

- Problem
  - topološko razvrsti vozlišča usmerjenega grafa
  - **topološka ureditev**
    - če uv a E, potem je u pred v

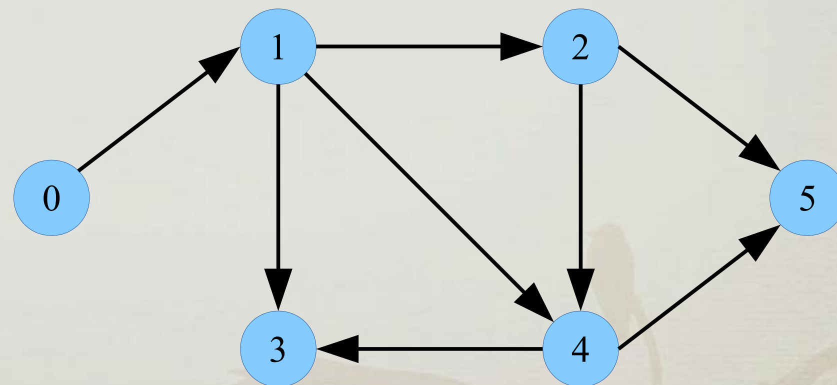
- Primer
  - kje je težava?



DAG – directed acyclic graph

Topološko urejanje

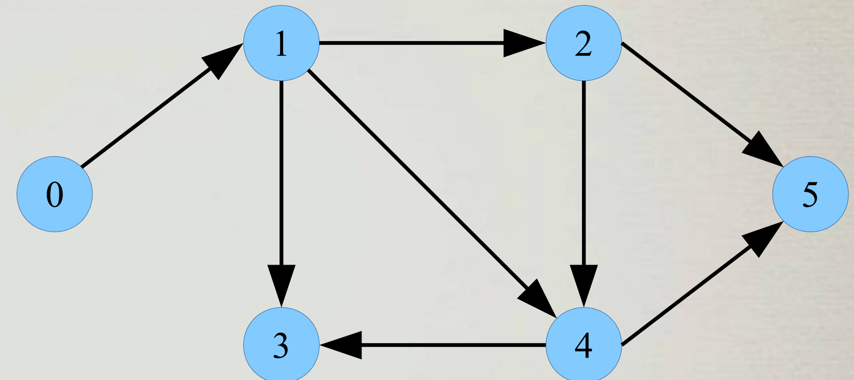
- Algoritem preko DFS
  - izvedemo DFS na celotnem grafu
    - povezava do že obiskanega vozlišča ⇒ cikel
  - **izstopni** vrstni red obiskovanja
  - v **obratnem** vrstnem redu



- Pravilnost
- Detekcija ciklov

Topološko urejanje

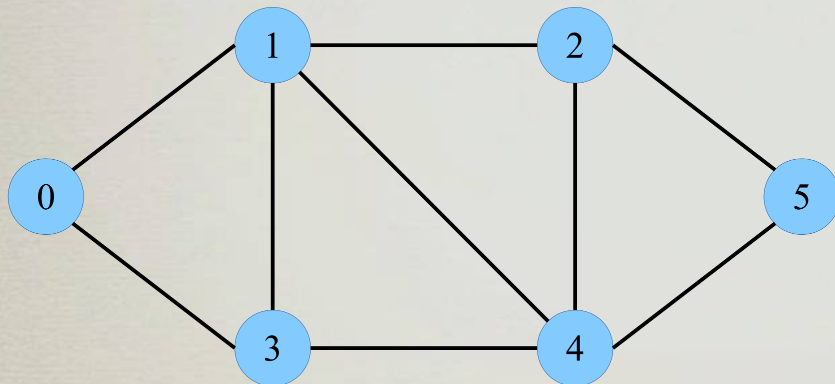
- Algoritem z odstranjevanjem vozlišč
  - odstranimo vsa vozlišča z vhodno stopnjo 0
  - jih dodamo v seznam
  - ponavljamo postopek

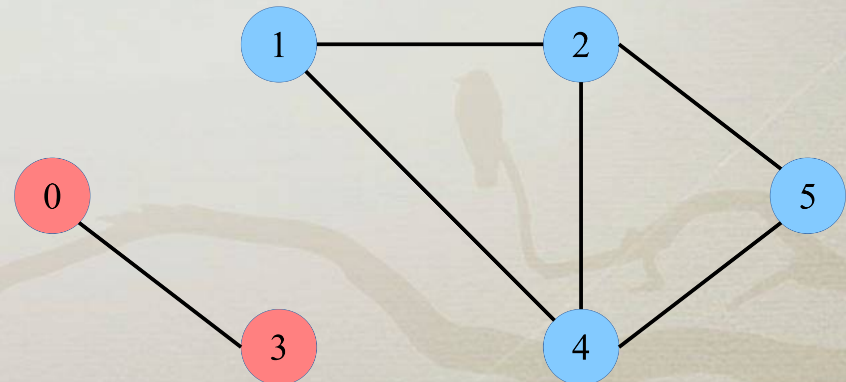


- Izrek
  - vsak DAG ima vsaj eno vozlišče z vhodno stopnjo 0

Povezanost neusmerjenega grafa

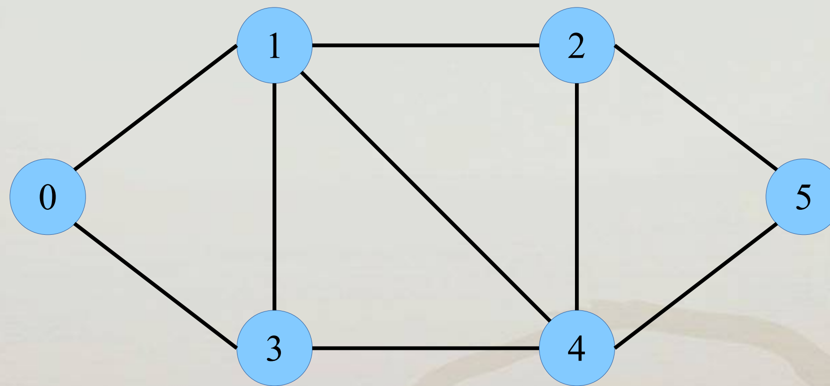
- Neusmerjeni graf
  - je **povezan**, če
    med vsakim parom vozlišč
    obstaja pot





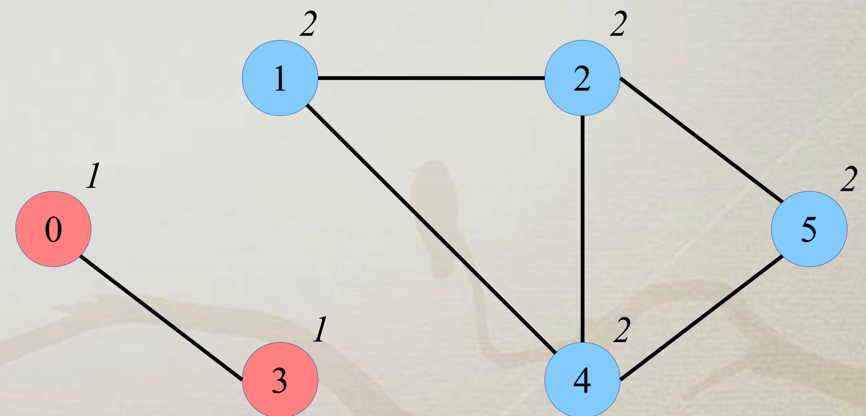
Povezanost neusmerjenega grafa

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



Povezanost neusmerjenega grafa

- Povezane komponente
  - povezana komponenta je **največji povezani podgraf**
  - problem
    - detekcija komponent
    - vozlišča v isti komponenti enako označimo
  - ideja
    - spremenimo *dfs_full*()
    - vsaka uporaba dfs*(v)* drugače označuje vozlišča



Povezanost usmerjenega grafa

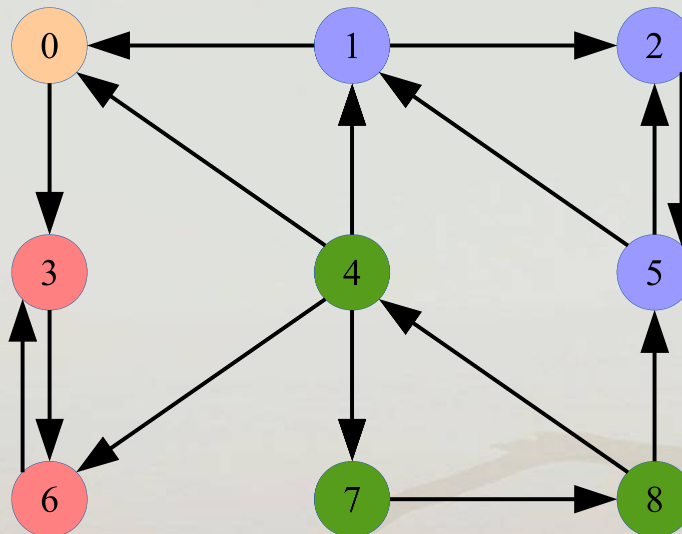
- Usmerjeni graf
  - je **šibko povezan**, če je ustrezen neusmerjeni graf povezan
    - vse usmerjene povezave spremenimo v neusmerjene
  - je **povezan**, če za vsak par vozlišč u in v obstaja pot iz u v v ali v v u
  - je **krepko povezan**, če za vsak par vozlišč u in v obstaja pot iz u v v in v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

Povezanost usmerjenega grafa

- Algoritmi
  - šibka povezanost
    - usmerjeni graf spremenimo v neusmerjenega
    - in poženemo detekcijo povezanosti slednjega
  - povezanost
    - iz vsakega vozlišča poženemo DFS (od začetka)
    - beležimo dosegljive pare
      - npr. štejemo kolikokrat smo dosegli vozlišče
  - krepka povezanost
    - glej naprej

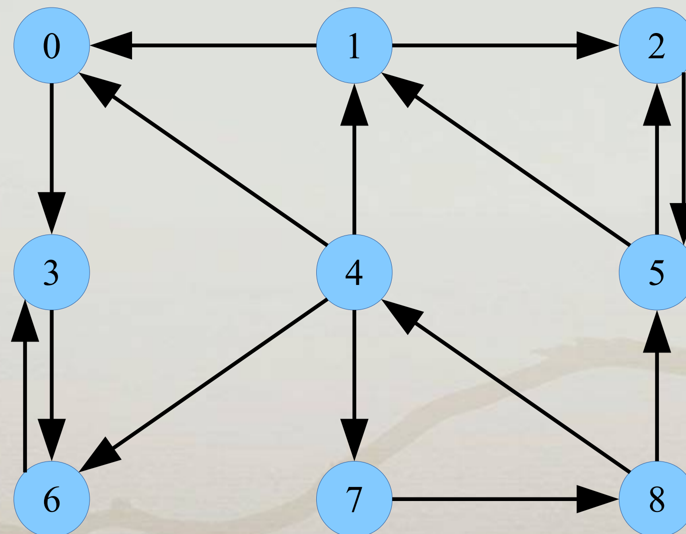
Krepko povezane komponente

- Problem
  - 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 $\mathrm{dfs}(v)$
    - vsaka uporaba $\mathrm{dfs}(v)$ 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