Algoritmi in podatkovne strukture 1

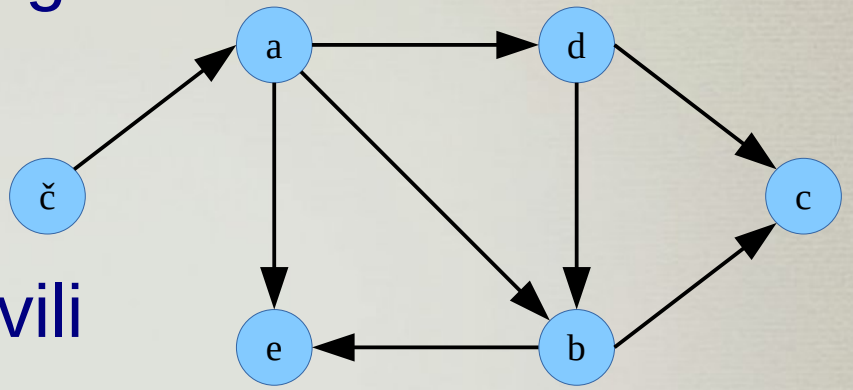
Visokošolski strokovni študij Računalništvo in informatika

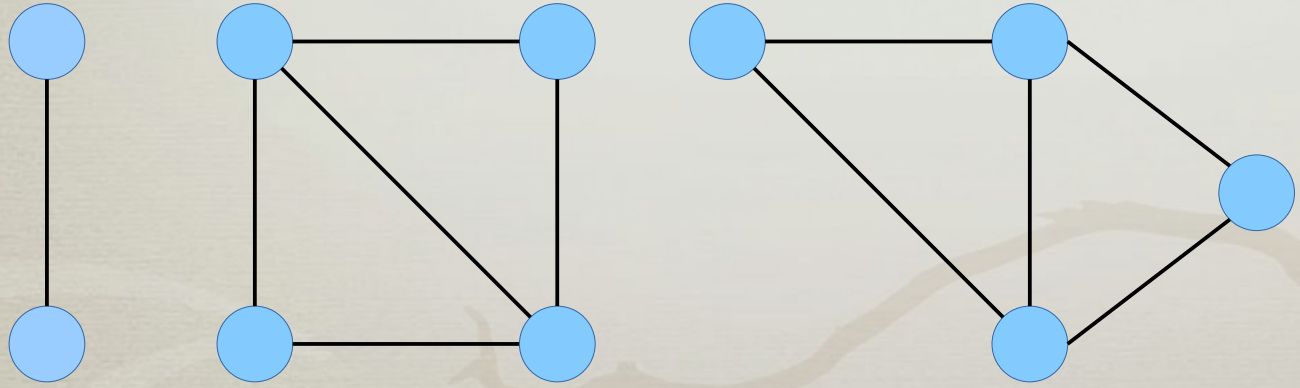
Algoritmi na grafih: obhodi grafov

Jurij Mihelič, UniLj, FRI

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
    - $v \rightarrow N(v) \rightarrow N(N(v)) \rightarrow \ldots$
  - 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
  - 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

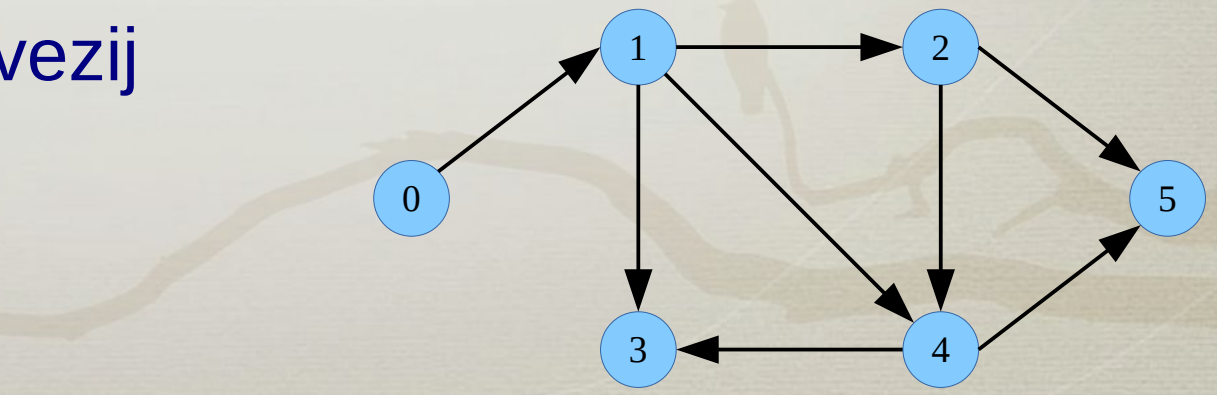
- Cikličnost grafa
  - Ali je graf cikličen / acikličen?
  - neusmerjeni ali usmerjeni graf
  - ideja
    - malenkost popravimo algoritem
    - če med preiskovanjem naletimo na že obiskanega soseda, potem ima graf cikel

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

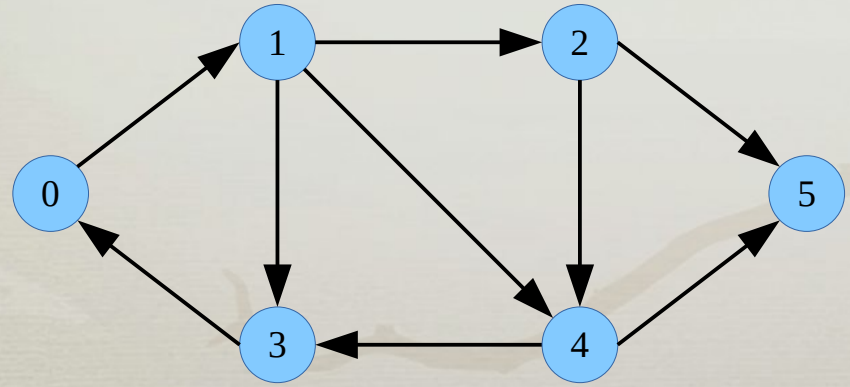
- Motivacija
  - razvrsti opravila, da slediš odvisnostmi
    - razvrščanje strojnih ukazov v procesorju
  - razreševanje odvisnosti v aritmetičnih izrazih
    - prevajalniki, zbirniki, tolmači, Excel preglednice
  - prevajanje izvorne kode
    - simbolne odvisnosti pri povezovanju (linker)
    - odvisnosti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



Topološko urejanje

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

- Primer
  - kje je težava?

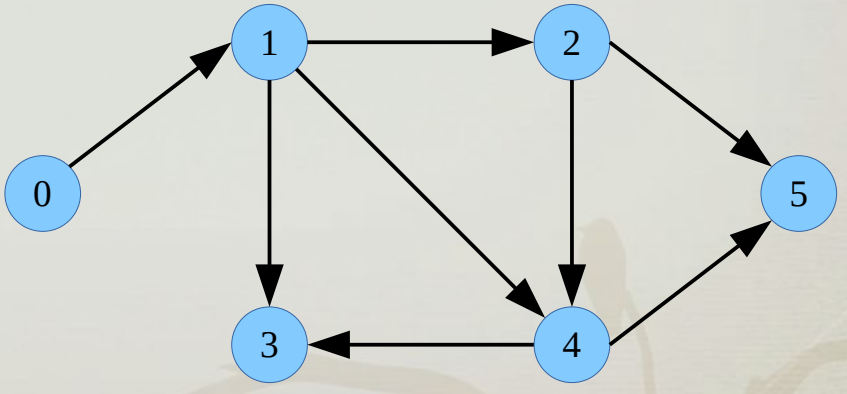


DAG – directed acyclic graph

Topološko urejanje

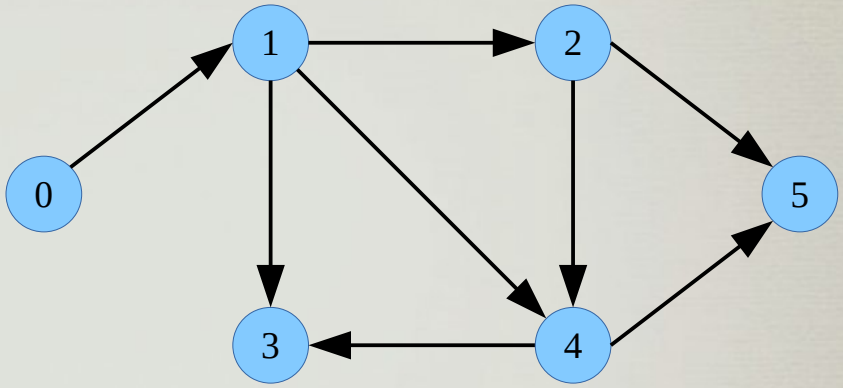
- Algoritem preko DFS
  - izvedemo DFS na celotnem grafu
    - povezava do že obiskanega vozlišča ⇒ cikel
  - **izstopni** vrstni red obiskovanja
  - v **obratnem** vrstnem redu

- 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