

Algoritmi in podatkovne strukture 2

Sprehodi po grafih

Luka Fürst

Graf

- $G = (V, E)$
 - uporabljali bomo tudi $G.V$ in $G.E$ (Cormen *et al.*)
- $V = \{0, 1, \dots, n - 1\}$: množica vozlišč
- $E \subseteq V \times V$: množica povezav
- Omejimo se na **enostavne** grafe
 - brez zank in vzporednih povezav
- Graf je lahko **usmerjen** ali **neusmerjen**
- V tem poglavju so grafi **neuteženi**

Predstavitev grafa

- Seznam sosednosti

- tabela n tabel
- $Adj[v] = [v_1, \dots, v_k] \iff$ sosedje v so v_1, \dots, v_k
- najboljša predstavitev za večino algoritmov

- Matrika sosednosti

- dvojiška matrika $n \times n$
- $A[u, v] = 1 \iff (u, v) \in E$
- učinkovita za preverjanje obstoja povezave med podanima vozliščema

Iskanje v širino (BFS)

- Pričnemo v izbranem vozlišču v_0 in preiskujemo vozlišča po naraščajočih razdaljah od v_0
- Vzdržujemo **vrsto** nazadnje odkritih vozlišč
- Vrsta na začetku vsebuje vozlišče v_0
- V vsaki iteraciji odvezamemo vozlišče iz vrste in vanjo dodamo vse še neodkrite sosedo
- *v.obiskano*
 - false na začetku
 - true, ko vozlišče obiščemo
- *v.predhodnik*
 - predhodnik vozlišča v na poti od v_0 do v
- BFS za vsako vozlišče v odkrije **najkrajšo pot** of v_0 do v

Iskanje v širino (BFS)

```
function BFS( $v_0$ )  
   $Q \leftarrow$  INICIALIZIRAJ-VRSTO()  
  DODAJ-V-VRSTO( $Q, v_0$ )  
   $v_0.obiskano \leftarrow$  true  
  while  $\neg$ JE-VRSTA-PRAZNA( $Q$ ) do  
     $u \leftarrow$  ODSTRANI-IZ-VRSTE( $Q$ )  
    PRINT( $u$ )  
    for all  $v \in Adj[u]$  do  
      if  $\neg v.obiskano$  then  
         $v.predhodnik \leftarrow u$   
         $v.obiskano \leftarrow$  true  
        DODAJ-V-VRSTO( $Q, v$ )
```

Iskanje v globino (DFS): začetna različica

- DFS v izbranem vozlišču u poženemo tako, da
 - označimo vozlišče u kot obiskano
 - poženemo DFS iz vsakega od neobiskanih sosedov vozlišča u

```
function DFS1( $u$ )  
   $u.obiskano \leftarrow \text{true}$   
  for all  $v \in Adj[u]$  do  
    if  $\neg v.obiskano$  then  
       $v.predhodnik \leftarrow u$   
      DFS1( $v$ )
```

Prva nadgradnja: označevanje vozlišč

- Na začetku so vsa vozlišča bela
- Ko vozlišče obiščemo, postane sivo
- Ko za neko vozlišče obdelamo vse njegove sosedo, postane črno

```
function DFS1(u)  
  u.barva ← siva  
  for all v ∈ Adj[u] do  
    if v.barva = bela then  
      v.predhodnik ← u  
      DFS1(v)  
  u.barva ← črna
```

Druga nadgradnja: čas odkritja in čas zaključka

- »Čas« je preprosto globalen števec, inicializiran na 0
- $v.d$: trenutek, ko vozlišče v postane sivo (čas odkritja)
- $v.f$: trenutek, ko vozlišče v postane črno (čas zaključka)

function DFS1(u)

$čas \leftarrow čas + 1$

$u.d \leftarrow čas$

$u.barva \leftarrow siva$

for all $v \in Adj[u]$ **do**

if $v.barva = bela$ **then**

$v.predhodnik \leftarrow u$

 DFS1(v)

$u.barva \leftarrow črna$

$čas \leftarrow čas + 1$

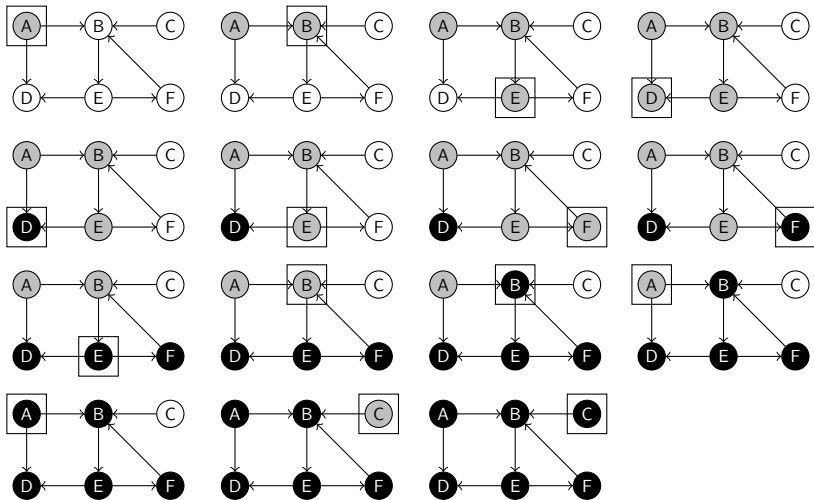
$u.f \leftarrow čas$

Tretja nadgradnja

- Pri povezanem neusmerjenem grafu bo funkcija DFS1 odkrila vsa vozlišča
- Pri usmerjenih ali nepovezanih grafih se to (morda) ne bo zgodilo
- DFS1 zato poganjamo, dokler obstajajo neobiskana vozlišča

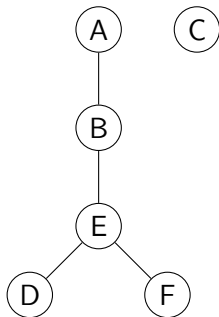
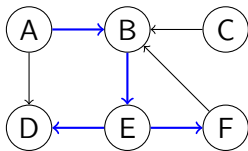
```
function DFS( $G$ )  
  for all  $u \in G.V$  do  
     $u.barva \leftarrow$  bela  
  for all  $u \in G.V$  do  
    if  $u.barva =$  bela then  
      DFS1( $u$ )
```

Primer



DFS-gozd

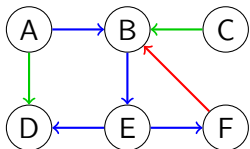
- množica dreves, določena z relacijo **predhodnik**
- $V = G.V$
- $E = \{(u, v) \mid u = v.\text{predhodnik}\}$



Klasifikacija povezav

- **Drevesne povezave**
 - povezave, ki tvorijo drevesa v DFS-gozdu
 - sivo vozlišče → belo vozlišče
- **Povratne povezave**
 - povezave med vozliščem in njegovim prednikom v DFS-gozdu
 - sivo vozlišče → sivo vozlišče
- **Ostale povezave**
 - povezave med vozliščem in njegovim potomcem v DFS-gozdu
 - povezave med različnimi drevesi v DFS-gozdu
 - sivo vozlišče → črno vozlišče

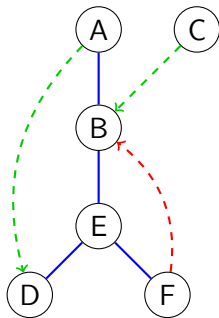
Klasifikacija povezav



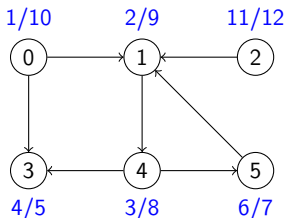
————— drevesne

————— povratne

————— ostale



Čas odkritja in čas zaključka



Lastnost 1

Za vsak par vozlišč u in v velja eno od sledečega:

- $[v.d, v.f] \subset [u.d, u.f]$ in v je potomec u v DFS-gozdu;
- $[u.d, u.f] \subset [v.d, v.f]$ in u je potomec v v DFS-gozdu;
- $[u.d, u.f] \cap [v.d, v.f] = \emptyset$, u in v pa nista v relaciji prednik-potomec v DFS-gozdu.

Dokaz lastnosti 1

- Naj bo $u.d < v.d$ (alternativa je povsem simetrična)
- **Možnost 1:** $v.d < u.f$
 - u je bilo ob času odkritja v sivo
 - v je zato potomec u v DFS-gozdu
 - DFS najprej rekurzivno preišče vso soseščino v -ja in v počrni, šele potem se vrne na u
 - zato je $v.f < u.f$ in $u.d < v.d < v.f < u.f$
 - $[v.d, v.f] \subset [u.d, u.f]$
- **Možnost 2:** $v.d > u.f$
 - po definiciji je $u.d < u.f$ in $v.d < v.f$
 - torej je $u.d < u.f < v.d < v.f$
 - $[v.d, v.f] \cap [u.d, u.f] = \emptyset$
 - DFS je odkril v , ko je bilo u že črno, zato v ni potomec u v DFS-gozdu
 - seveda tudi u ni potomec v

Še nekaj lastnosti

Lastnost 2

Vozlišče v je potomec vozlišča u v DFS-gozdu natanko v primeru, če je $u.d < v.d < v.f < u.f$.

Lastnost 3

Vozlišče v je potomec vozlišča u v DFS-gozdu natanko v primeru, če ob času $u.d$ obstaja »bela pot« od u do v (pot, sestavljena iz samih belih vozlišč).

Dokaz lastnosti 3

- v je potomec $u \implies$ obstaja bela pot od u do v
 - za vse potomce vozlišča u v DFS-gozdu velja $v.d > u.d$, zato morajo biti ob času odkritja vozlišča u (ko u postane sivo) biti bela
- Obstaja bela pot od u do $v \implies v$ je potomec u
 - recimo, da obstaja »bela pot« $u \rightsquigarrow w \rightarrow v$ in da w postane potomec u -ja v DFS-gozdu, v pa nima te sreče
 - velja $u.d < w.d < w.f < u.f$ (lastnost 2)
 - DFS obišče v (in zaključi z njim), preden zaključi z w , zato je tudi $w.d < v.d < v.f < w.f$
 - od tod sledi $u.d < v.d < v.f < u.f$
 - zato je v dejansko potomec u

Še zadnja . . .

Lastnost 4

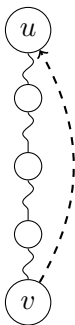
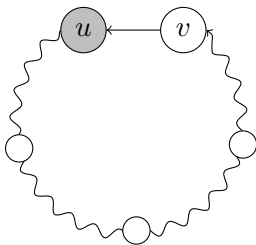
Pri izvajanju DFS na **neusmerjenem** grafu je vsaka povezava bodisi drevesna bodisi povratna.

- Naj bo (u, v) poljubna povezava v grafu in naj bo $u.d < v.d$
- DFS zaključi z vozliščem v , preden zaključi z vozliščem u
- Zato je vozlišče v potomec vozlišča u v DFS-gozdu
- Če DFS obiše povezavo (u, v) v smeri $u \rightarrow v$, je to drevesna povezava (u je tedaj sivo, v pa belo)
- Če DFS obiše povezavo (u, v) v smeri $v \rightarrow u$, je to povratna povezava (obe vozlišči sta tedaj sivi)

Detekcija cikla

Trditev

Graf ima cikel natanko v primeru, če pri izvedbi DFS vsaj ena povezava postane povratna.



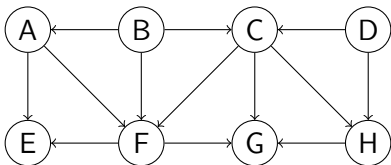
Dokaz trditve

- Graf ima cikel \implies vsaj ena povratna povezava
 - Naj bo v_1, \dots, v_k cikel v grafu
 - (BŠS) Naj bo v_1 prvo vozlišče v ciklu, ki ga DFS obišče
 - Ko DFS obišče v_1 , je v_1 sivo, v_2, \dots, v_k pa bela
 - Vozlišča v_2, \dots, v_k po izreku o belih poteh postanejo potomci vozlišča v_1 v DFS-gozdu
 - Ko DFS obišče v_k , bo v_1 še vedno sivo, zato bo $v_k \rightarrow v_1$ povratna povezava
- Vsaj ena povratna povezava \implies graf ima cikel
 - Pot po drevesu $u \rightsquigarrow v$ skupaj s povratno povezavo $v \rightarrow u$ tvori cikel

Topološko urejanje

- Naj bo G usmerjen acikličen graf
- Topološki vrstni red vozlišč
 - če obstaja pot $u \rightsquigarrow v$, je $u \prec v$
 - če obstaja pot $v \rightsquigarrow u$, je $v \prec u$
 - če ne obstaja niti prva niti druga pot, lahko izberemo bodisi $u \prec v$ bodisi $v \prec u$

Primer



- Topološki vrstni red:

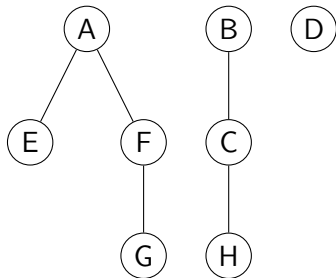
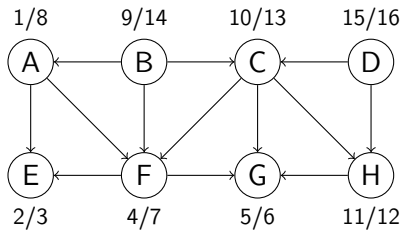
D \prec B \prec C \prec H \prec A \prec F \prec G \prec E

Algoritem

function TOPOLOŠKO-UREDÍ(G)

DFS(G)

uredi vozlišča po padajočih časih zaključka ($v.f$)



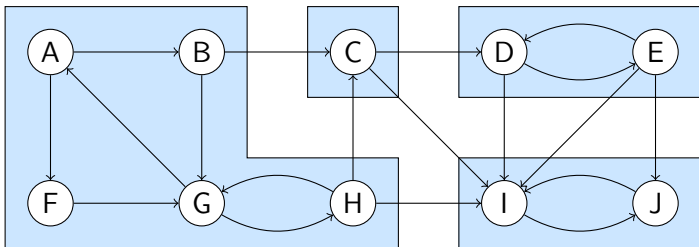
Zakaj deluje?

- Recimo, da imamo povezavo $u \rightarrow v$
- Recimo, da je DFS pravkar odkril u
- Vozlišče v je lahko belo ali črno
 - če bi bilo sivo, bi v DFS-gozdu imeli povratno povezavo $v \rightarrow u$ in s tem cikel
- Če je v belo, je $u.d < v.d < v.f < u.f$ in potemtakem $v.f < u.f$
- Če je v črno, je $v.f < u.d < u.f$ in potemtakem $v.f < u.f$
- Torej bo v vsakem primeru veljalo $v.f < u.f$

Krepko povezane komponente

- Naj bo G usmerjen graf
- Vozlišči u in v pripadata isti **krepko povezani komponenti** natanko v primeru, če obstajata poti $u \rightsquigarrow v$ in $v \rightsquigarrow u$
- Cilj problema je poiskati vse krepko povezane komponente v podanem grafu

Primer



Algoritem

function KREPKO-POVEZANE-KOMPONENTE(G)

DFS(G)

$\langle v_1, \dots, v_n \rangle \leftarrow$ vozlišča, urejena po padajočih $v.f$

izdelaj G^T

poženi DFS(G^T), vendar pa v glavni zanki obravnavaj
vozlišča po vrstnem redu v_1, \dots, v_n

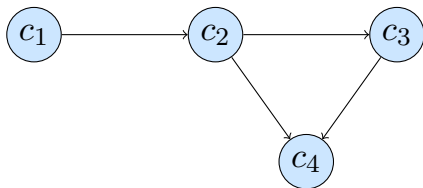
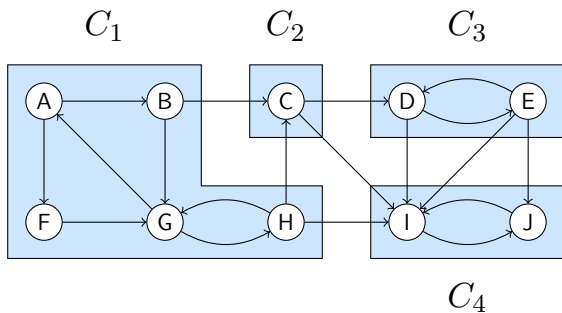
drevesa v nastalem DFS-gozdu predstavljajo posamezne
krepko povezane komponente

- Graf G^T (**transponirani graf**) je enak grafu G , le da so vse povezave obrnjene v nasprotno smer
 $((u, v) \in G.E \iff (v, u) \in G^T.E)$

Komponentni graf

- Naj bodo C_1, C_2, \dots, C_k krepko povezane komponente grafa G
- **Komponentni graf** je usmerjen graf $G^{\text{SCC}} = (V^{\text{SCC}}, E^{\text{SCC}})$
- $V^{\text{SCC}} = \{c_1, \dots, c_k\}$
- $(c_i, c_j) \in E^{\text{SCC}} \iff \exists u \in C_i, v \in C_j: (u, v) \in G$

Komponentni graf

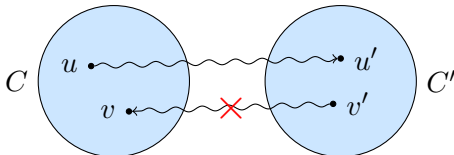


Lastnosti komponentnega grafa

Trditev 1

Komponentni graf je acikličen.

- Naj bosta C in C' različni krepko povezani komponenti
- Naj bo $u, v \in C$ in $u', v' \in C'$
- Recimo, da obstaja pot $u \rightsquigarrow u'$
- Pokazati moramo, da pot $v' \rightsquigarrow v$ ne obstaja
- Res je: če bi pot $v' \rightsquigarrow v$ obstajala, bi obstajala tudi pot $v \rightsquigarrow u \rightsquigarrow u' \rightsquigarrow v'$ in bi v in v' pripadala isti komponenti



Lastnosti komponentnega grafa

- Naj bo $f(C) = \max_{v \in C} v.f$

Trditev 2

Naj bo $u \in C$ in $u' \in C'$ ($C \neq C'$). Če obstaja povezava (u, u') , potem je $f(C') < f(C)$.

- Če DFS obiše u pred u' ($u.d < u'.d$), bo u sivo in u' belo, zato bo u' potomec u v DFS-gozdu in bo $u.d < u'.d < u'.f < u.f$
- Če DFS obiše u' pred u ($u'.d < u.d$), bo z u' zaključil, preden bo sploh prispel do u , saj ne obstaja pot $u' \rightsquigarrow u$
- Vnovič bo $u'.f < u.f$

Lastnostni komponentnega grafa

Trditev 3

Naj bo $u \in C$ in $u' \in C'$ ($C \neq C'$). Če je $f(C) > f(C')$, potem G^T nima povezave $u \rightarrow u'$.

- Če bi G^T imel povezavo $u \rightarrow u'$, bi G imel povezavo $u' \rightarrow u$ in bi po trditvi 2 veljalo $f(C) < f(C')$

Pravilnost delovanja algoritma

Trditev

Vsako DFS-drevo, ki ga tvori algoritem pri preiskovanju grafa G^T , predstavlja natanko eno krepko povezano komponento

- Indukcija: vsako od prvih k DFS-dreves, ki jih tvori algoritem, predstavlja po eno krepko povezano komponento
- Naj bo u koren DFS-drevesa $k + 1$ in naj u pripada krepko povezani komponenti C
- Vsa ostala vozlišča v C so bela, zato postanejo potomci vozlišča u v DFS-drevesu $k + 1$
- Vseh k komponent, ki smo jih že obiskali, ima manjši čas zaključka
- Po trditvi 3 vse povezave iz komponente C vodijo do komponent, ki smo jih že obiskali
- Drevo $k + 1$ torej tvori natanko komponento $k + 1$

Vprašanje

- V drugi izvedbi poženemo DFS na **transponiranem** grafu, pri čemer vozlišča obravnavamo po **padajočih** časih zaključka
- Zakaj ne bi v drugi izvedbi enostavno pognali DFS na **originalnem** grafu, vozlišča pa bi obravnavali po **naraščajočih** časih zaključka?