Teorija izračunljivosti

Ivan Prokić
kabinet 117, F blok
prokic@uns.ac.rs
http://imft.ftn.uns.ac.rs/∼iprokic/

Novi Sad

Tema 1

Algoritmi za množenje matrica

Množenje matrica: vreme rada $\Theta(n^3)$

Problem: Pomnožiti dve matrice formata $n \times n$.

Podsećanje: Množenjem matrica $A = [a_{ij}]_n$ i $B = [b_{ij}]_n$ dobijamo matricu $C = [c_{ij}]_n$ čiji element c_{ij} predstavlja skalarni proizvod vektora i-te vrste matrice A i vektora j-te kolone matrice B, tj.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

Algorithm 1: Matrice (A, B)

$n =$ broj kolona matrice A
Neka je C nova matrica formata $n \times n$
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    $c_{ij} = 0$
    for $k = 1$ **to** n **do**
      $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

Množenje matrica: divide and conquer

Množenjem matrica $A = [a_{ij}]_n$ i $B = [b_{ij}]_n$ dobijamo matricu $C = [c_{ij}]_n$. Pretpostavimo sada da je $n = 2^m$. Ako A, B i C predstavimo kao blok matrice množenje matrica formata $n \times n$ možemo definisati (rekurzivno) preko množenja matrica formata $n/2 \times n/2$:

$$AB = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = C,$$

gde su

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned}$$

Množenje matrica: divide and conquer

Na osnovu prethodnog možemo definisati sledeći algoritam.

Algorithm 2: MatriceRek (A, B)

$n =$ broj kolona matrice A
Neka je C nova matrica formata $n \times n$
if $n == 1$ **then**
  $c_{11} = a_{11}b_{11}$
else
  Podeli matice A, B i C na podmatrice opisane prethodno

$$\begin{aligned} C_{11} &= \text{MatriceRek}(A_{11}, B_{11}) + \text{MatriceRek}(A_{12}, B_{21}) \\ C_{12} &= \text{MatriceRek}(A_{11}, B_{12}) + \text{MatriceRek}(A_{12}, B_{22}) \\ C_{21} &= \text{MatriceRek}(A_{21}, B_{11}) + \text{MatriceRek}(A_{22}, B_{21}) \\ C_{22} &= \text{MatriceRek}(A_{21}, B_{12}) + \text{MatriceRek}(A_{22}, B_{22}) \end{aligned}$$

return C

Analiza vremena rada MatriceRek algoritma

- $c_{11} = a_{11}b_{11}$ računanje - $\Theta(1)$ vremena;
- podela na podmatrice - $\Theta(n^2)$ vremena ako kopiramo u nove matirce ili $\Theta(1)$ ako ih delimo pomoću indeksa;
- rekurzivni poziv se dešava 8 puta za veličinu ulaza $n/2$;
- sabiranja matrica - $\Theta(n^2)$.

Ukupno

$$T(n) = \Theta(1) + \Theta(n^2) + 8T(n/2) + \Theta(n^2) = 8T(n/2) + \Theta(n^2).$$

Ako primenimo Master metod da rešimo rekurentnu relaciju imamo $a = 8, b = 2$ i $f(n) = \Theta(n^2)$, a kako je $n^{\log_b a} = n^3$ (polinomno veće od $f(n)$), vidimo da imamo prvi slučaj Master teoreme, te je $T(n) = \Theta(n^3)$.

Ništa brže od prvog algoritma!

Množenje matrica: divide and conquer - Štrasenov algoritam

Množenje $A \cdot B = C$ matrica $n \times n$ možemo definisati (rekurzivno) preko množenja matrica $n/2 \times n/2$:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{bmatrix},$$

gde su

$$\begin{aligned} S_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ S_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ S_3 &= (A_{11} - A_{21})(B_{11} + B_{12}) \\ S_4 &= B_{22}(A_{11} + A_{12}) \\ S_5 &= A_{11}(B_{12} - B_{22}) \\ S_6 &= A_{22}(B_{21} - B_{11}) \\ S_7 &= B_{11}(A_{21} + A_{22}) \end{aligned}$$

Štrasenov algoritam - analiza vremena rada

- Umesto prethodnih 8 sada imamo 7 množenja matrica $n/2 \times n/2$;
- Umesto prethodnih 4 sabiranja imamo 18 sabiranja.

Vreme rada je

$$T(n) = \Theta(1) + \Theta(n^2) + 7T(n/2) + 18\Theta(n^2) = 7T(n/2) + \Theta(n^2).$$

Ako primenimo Master metod da rešimo rekurentnu relaciju imamo $a = 7, b = 2$ i $f(n) = \Theta(n^2)$, a kako je $n^{\log_b a} = n^{\log_2 7}$ (polinomno veće od $f(n)$), vidimo da opet imamo prvi slučaj Master teoreme, te je $T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2,81..})$.

Asimptotski brže od prva dva algoritma!

Tema 2

Algoritmi za proveru zadovoljivosti iskaznih formula

Iskazne formule

Podsećanje:

1. Iskazne formule (Bulovi izrazi) se dobijaju na sledeći način
   - $\top, \bot$, iskazna slova $x, y, z, \ldots$;
   - Ako su φ i ϕ iskazne formule onda su to i $\neg\varphi, \varphi \wedge \phi, \varphi \vee \phi, \varphi \Rightarrow \phi$ i $\varphi \Leftrightarrow \phi$;
   - Iskazne formule se dobijaju konačnom primenom prethodna dva pravila.
2. Svaka iskazna formula ϕ se može zapisati u konjuktivnoj normalnoj formi (KNF)

$$\phi = \bigwedge_{i=1}^{n} (c_{i_1} \vee c_{i_1} \vee \ldots \vee c_{i_n}),$$

gde je c_{i_j} literal, tj. iskazno slovo ili njegova negacija. (Inače, može i u DNF, i to ne mora biti na jedinstven način. Može i u SDNF i SKNF, i to na jedinstven način.)

3. Iskazna formula ϕ je zadovoljiva ako postoji valuacija $\tau : X_\phi \to \{\bot, \top\}$, gde je X_ϕ skup svih promeljivih u ϕ, u odnosu na koju je formula tačna, tj. $v_\tau(\phi) = \top$.

SAT problem: Za iskaznu formulu ϕ datu u KNF utvrditi da li je zadovoljiva.

SAT problem

- Naivni (”brute-force”) algoritam za SAT problem bi bio da za datu iskaznu formulu ϕ isprobamo sve evaluacije (npr. formiramo istinitosnu tablicu) i vidimo da li je u nekoj formula tačna. Pošto svih evaluacija formule od n iskaznih slova ima 2^n, takav algoritam bi bio **eksponencijalne složenosti**.
- Postoje efikasniji algoritmi od ovog (npr. DPLL algoritam), ali su svi eksponencijalne složenosti (za najgori slučaj).
- Ono što možemo da uradimo u polnomnom (zapravo u linearnom) vremenu u odnosu na dužinu date formule jeste da **proverimo da li je u određenoj evaluaciji ona tačna ili ne**.

Za ovakve algoritme kažemo da su **u klasi složenosti NP**.

Digresija - o klasama vremenske složenosti

Probleme izračunavanja možemo podeliti u sledeće **klase vremenske složenosti**:

- Klasa **P** - problemi rešivi u najviše polinomnom vremenu (npr. množenje matrica, računanje n-tog člana Fibonačijevog niza, itd.);
- Klasa **EXPTIME** - problemi rešivi u najviše eksponencijalnom vremenu (npr. generalizovana igra dama (eng. checkers)).

Uz ove imamo i jednu posebnu klasu složenosti:

- Klasa **NP** - problemi za koje ne postoji (poznat) algoritam koji ih rešava u polinomnom vremenu (već samo u eksponencijalnom), ali čije svako rešenje možemo da proverimo u polinomnom vremenu (npr. SAT problem).

Za klase vremenske složenosti važi sledeći odnos:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}.$$

P=**NP**? Ovo je pitanje od milion dolara i jedno od najvećih pitanja današnje matematike. Da osvojite novac potrebno je ”samo” da rešite SAT problem u polinomnom vremenu (ili da dokažete da takav algoritam ne postoji).

Nazad na SAT problem - definicija HORNSAT problema

Dakle, SAT problem je ”težak” (po Kuk-Levinovoj teoremi, najteži iz klase **NP**). Međutim, postoje specijalni slučajevi SAT problema (koji rešavaju problem samo za instance u nekom određenom obliku) koji su rešivi u polinomnom vremenu. Jedan takav je HORNSAT problem koji odgovara na pitanje zadovoljivosti KNF **Hornovih formula**.

Definicija (Hornova formula)

KNF formula ϕ se zove Hornova formula ako je

$$\phi = \bigwedge_{i=1}^{n} (c_{i_1} \vee c_{i_1} \vee \ldots \vee c_{i_n}),$$

gde za svako i postoji najviše jedan literal bez negacije.

HORNSAT problem

Problem: Naći (polinomni) algoritam za proveru zadovoljivosti HORN formule.

Primetimo prvo da sve konjukte u Hornovoj formuli možemo zapisati kao implikacije:

- ako su u konjuktu svi literali sa negacijom, tj. $\neg x_1 \vee \ldots \vee \neg x_n$ onda je on ekvivalentan sa $(x_1 \wedge \ldots \wedge x_n) \Rightarrow \bot$;
- ako je konjukt oblika x onda je on ekvivalentan sa $\top \Rightarrow x$;
- ako su u konjuktu svi literali sa negacijom sem jednog, tj. $\neg x_1 \vee \ldots \vee \neg x_n \vee y$ onda je on ekvivalentan sa $(x_1 \wedge \ldots \wedge x_n) \Rightarrow y$;

Pored toga posmatrajmo svaku valuaciju $\tau : X_\phi \to \{\bot, \top\}$, gde je X_ϕ skup svih promeljivih u ϕ, kao skup $T = \{x \in X_\phi \,:\, \tau(x) = \top\}$, tj. T je skup svih iskaznih slova iz ϕ koja uzimaju vrednost $\top$ u valuaciji τ.

Algoritam za HORNSAT problem

Algorithm 3: HORNSAT(ϕ)

$T = \emptyset$
while T *ne zadovoljava* ϕ **do**
  uzmi jedan netačan konjukt $P \Rightarrow Q$
  if Q *je iskazno slovo* **then**
    $T = T \cup \{Q\}$
  else
    return ϕ je kontadikcija
return T

Kako radi algoritam:
1. Postavi sva iskazna slova na $\perp$ i proveri zadovoljivost ϕ. Ako je zadovoljiva vrati $T = \emptyset$. U ovoj valuaciji ako postoji konjukt koji je netačan onda je on oblika $\top \Rightarrow x$ i tada promeni valuaciju slova x na $\top$ i vrati se nazad u **while** da proveriš zadovoljivost formule;
2. U svakom sledećem prolasku kroz petlju prvo proveri zadovoljivost ϕ za trenutnu valuaciju T. Ako ne zadovoljava, nađi jedan konjukt koji je netačan. Taj konjukt može biti

- $\top \Rightarrow x$: postupi kao što smo već objasnili;
- $(x_1 \wedge \ldots \wedge x_n) \Rightarrow y$: moguće samo ako su svi x_i već u T, a $y \notin T$ - postavi y na $\top$;
- $(x_1 \wedge \ldots \wedge x_n) \Rightarrow \perp$: moguće samo ako su svi x_i već u T - formula je kontradikcija.

Algoritam za HORNSAT problem

Zadatak

Ispitati zadovoljivost sledeće Hornove formule koristeći istinitosnu tablicu, a zatim za istu formulu primeniti HORNSAT algoritam

$$\phi = (\neg x \vee \neg y) \wedge (\neg y \vee z) \wedge z.$$

Rešenje. Na času. $\diamond$

Korektnost HORNSAT algoritma

Jasno je da **while** petlja može iterirati najviše onoliko puta koliko u formuli ϕ ukupno ima konjukata oblika $\top \Rightarrow x$ i $(x_1 \wedge \ldots \wedge x_n) \Rightarrow y$ (plus još jedna iteracija za konjukt oblika $(x_1 \wedge \ldots \wedge x_n) \Rightarrow \bot$). Takođe, iz algoritma možemo primetiti da je T najmanji od svih mogućih skupova koji predstavljaju valuacije u kojima je formula zadovoljiva.

Lema

Neka je za formulu ϕ skup T dobijen HORNSAT algoritmom. Za svaki skup T' koji predstavlja valuaciju formule ϕ u kojoj je ona tačna važi $T \subseteq T'$.

Dokaz

Dokažimo da je tvrđenje leme invarijanta **while** *petlje u HORNSAT algoritmu.*

- **Inicijalizacija petlje**: *Tada je $T = \emptyset$, pa $T \subseteq T'$ važi.*
- **Očuvanje**: *Pretpostavimo da je $T \subseteq T'$ pre i-te iteracije. U i-toj iteraciji posmatramo konjukt $P \Rightarrow Q$ koji u valuaciji T nije tačan. Pošto T' zadovoljava ϕ, konjukt $P \Rightarrow Q$ je u T' tačan. Razlikujemo dva slučaja za $P \Rightarrow Q$*
  1. $\top \Rightarrow x$: *tada $x \notin T$ ali $x \in T'$, pa nakon i-tog prolaska kroz petlju $T \cup \{x\} \subseteq T'$ važi.*
  2. $(x_1 \wedge \ldots \wedge x_n) \Rightarrow Q$: *tada $\{x_1, \ldots, x_n\} \in T \subseteq T'$, pa kako je konjukt tačan u T' sledi da je $Q = y, y \in T'$ i $y \notin T$. Nakon i-tog prolaska kroz petlju $T \cup \{y\} \subseteq T'$ važi.*

Korektnost i složenost HORNSAT algoritma

Koristeći prethodnu lemu možemo dokazati glavno tvrđenje:

Teorema

Hornova KNF formula ϕ je zadovoljiva ako i samo ako je tačna u valuaciji T dobijenoj iz HORNSAT algoritma.

Dokaz

Ako je ϕ tačna u T valuaciji, jasno da je zadovoljiva. Pretpostavimo sada da ϕ je tačna u nekoj valuaciji T' ali nije u T i pokažimo da to nije moguće. Iz pretpostavke da ϕ nije tačna u T zaključujemo da jedan od njenih konjukata nije tačan u T. Kako je T dobijeno tako su u njemu da svi konjukti oblika $\top \Rightarrow x$ i $(x_1 \wedge \ldots \wedge x_n) \Rightarrow y$ tačni, zaključujemo da postoji konjukt oblika $(x_1 \wedge \ldots \wedge x_n) \Rightarrow \bot$ koji u T nije tačan. Sledi $\{x_1, \ldots, x_n\} \in T$. Kako je po prethodnoj lemi $T \subseteq T'$ imamo da $\{x_1, \ldots, x_n\} \in T'$, odakle je konjukt $(x_1 \wedge \ldots \wedge x_n) \Rightarrow \bot$ iz ϕ netačan i u T', što je u kontradikciji sa pretpostavkom da je ϕ tačno u T'. Zaključujemo da ako je ϕ zadovoljiva onda je ona tačna u T dobijenom iz HORNSAT algoritma.

Složenost. Ako je n dužina formule ϕ tada jednu njenu evaluaciju možemo izračunati u $O(n)$ vremena, i broj iteracija **while** petlje takođe možemo ograničiti sa $O(n)$. HORNSAT algoritam je složenosti $O(n^2)$.

Tema 3

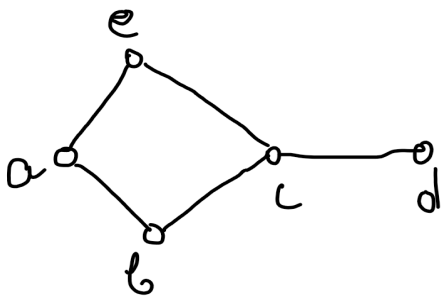
Algoritmi na grafovima

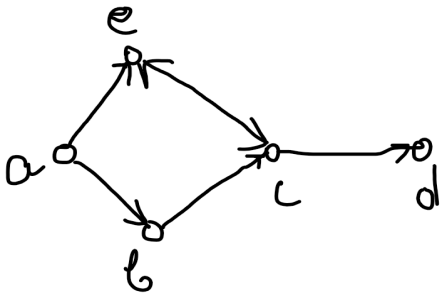
Grafovi

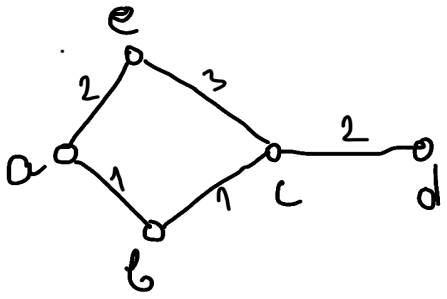
Definicija

Graf *je uređeni par* $G = (V, E)$*, gde je* V *neprazan (konačan) skup, a* E *je binarna relacija na skupu* V*. Elementi skupa* V *se zovu* **čvorovi***, a elementi skupa* E **grane***. Grafovi mogu biti:*

- **neorijentisani***, kada je* E *simetrična relacija, a grane su neusmerene;*
- **orijentisani** *(ili digrafovi), grane su usmerene.*
- **težinski***, kada je* $G = (V, E, w)$*, gde je* $w : E \to X$ *težinska funkcija koja svakoj grani dodeljuje težine iz* X *(najčešće broj).*



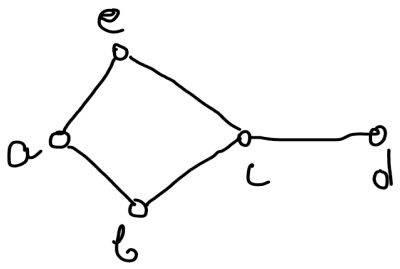




Grafovi - reprezentacija

Matrica susedstva: Ako graf G ima n čvorova, dodelimo mu kvadratnu matricu A_G formata $n \times n$, gde je $a_{ij} = 1$ ukoliko postoji grana (i, j), a inače je $a_{ij} = 0$. Ovaj način troši $O(n^2)$ prostora.

Liste susedstva: Ako graf G ima n čvorova i m grana, za svaki čvor formiramo listu njegovih suseda (kod orijentisanog dve liste: jedna za ulazne, druga za izlazne grane). Ovaj način troši $O(n + m)$ prostora ($O(n + 2m)$ za orijentisani graf), te je bolji za grafove koji nemaju puno grana.



$$A_G = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} -a \\ -b \\ -c \\ -d \\ -e \end{array}$$

u	L_u
a	$\{b, e\}$
b	$\{a, c\}$
c	$\{b, e, d\}$
d	$\{c\}$
e	$\{a, c\}$

Dostiživost u grafovima (eng. reachability)

Problem: Za dati graf G i dva njegova čvora s i t (od eng. source i target) proveri da li postoji put koji ih povezuje.

Rešenje: pretraga u širinu - BFS algoritam (breath-first search). BFS daje sve čvorove dostižive iz s i njihovu (minimalnu) udaljenost od s (funkcijom $dist(u)$). U algoritmu koristimo *listu* L tipa *red* (eng. queue) u kojoj možemo pristupiti samo prvom članu liste ($head(L)$, pri čemu drugi element liste postaje prvi), izbrisati prvi član ($pull(L)$) i dodati podatke na kraj liste ($push(L)$). Takođe, koristimo Bulovu funkciju new da obeležimo posećene čvorove.

Algorithm 4: BFS(V, E, s)

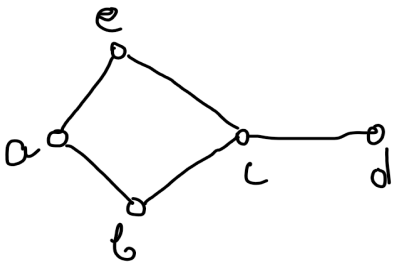
$dist(s) = 0;\ new(s) = \bot;\ L = [s]$
for $a \in V \setminus \{s\}$ **do**
  $dist(a) = \infty;\ new(a) = \top$
while $L \neq \emptyset$ **do**
  $a = head(L);\ pull(L)$
  for $b \in Susedi(a)$ **do**
    if $new(b) = \top$ **then**
      $dist(b) = dist(a) + 1;\ new(b) = \bot;\ push(L, b)$

BFS algoritam - primer

Zadatak

Primeniti BFS algoritam na graf sa slike dole, uzeti da je $s = a$.

Rešenje. Na času. ◇



$$A_G = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ -a \\ -b \\ -c \\ -d \\ -e \end{array}$$

u	L_u
a	$\{b, e\}$
b	$\{a, c\}$
c	$\{b, e, d\}$
d	$\{c\}$
e	$\{a, c\}$

BFS algoritam - korektnost

Lema

U BFS algortimu u svakom koraku važi: ako je $L = [a_1, \ldots, a_n]$ *tada je*

$$dist(a_1) \leq \ldots \leq dist(a_n) \leq dist(a_1) + 1.$$

Dokaz

Dokažimo da je tvrđenje leme invarijanta **while** *petlje BFS algortima.*

- **Inicijalizacija***:* $L = [s]$ *pa imamo* $dist(a_1) \leq dist(a_1) + 1$*.*
- **Očuvanje***: ako je pre* i*-te iteracije* **while** *petlje* $L = [a_1, \ldots, a_n]$ *i važilo je* $dist(a_1) \leq \ldots \leq dist(a_n) \leq dist(a_1) + 1$*, tada će nakon* i*-tog prolaska kroz petlju* $L = [a_2, \ldots, a_n, b_1, \ldots, b_m]$*, za* $b_1, \ldots, b_m$ *susede od* a_1 *koji ranije nisu bili posećeni. Za novododate čvorove imamo* $dist(b_i) = dist(a_1) + 1$*, pa sledi*

$$dist(a_2) \leq \ldots \leq dist(a_n) \leq dist(b_1) \leq \ldots \leq dist(b_m) \leq dist(a_2) + 1.$$

BFS algoritam - korektnost

Teorema

Kao rezultat poziva BFS(V, E, s) za svako $a \in V$ ako važi $dist(a) = k$ tada je k minimalno rastojanje čvora s do čvora a. Ako je $dist(a) = \infty$, tada s i a nisu povezani.

Dokažimo prvo da ako je $dist(a) = k$ tada postoji put od s do a dužine k, indukcijom po k. Za $k = 0$ tvrđenje je tačno jer jedino za s važi $dist(s) = 0$. Ako je $k > 0$ i $dist(a) = k$, tada je a mogao biti označen jedino u **while** petlji pri čemu je a označen kao sused od nekog b za kog je važilo $dist(b) = k - 1$. Iz indukcijske pretpostavke dobijamo da postoji put od s do b dužine $k - 1$, pa je put od s do a kroz b dužine k.

Još treba da dokažemo da ako je $dist(a) = k$, tada ne postoji put od s do a koji je kraći od k.

Pretpostavimo suprotno, neka postoji čvor a za koji važi $dist(a) = k$ i da postoji put od s do a koji je minimalne dužine l za $l < k$ i neka je k minimalno sa tom osobinom. Posmatrajmo taj put od s do a minimalne dužine l i neka je b pretposlednji čvor na tom putu. Tada je deo tog puta od s do b takođe minimalne dužine, i to $l - 1$. S obzirom da smo k izabrali da bude minimalan za koji važi $dist(c) = k$ i postoji put od s do c dužine kraće od k, sledi $dist(b) = l - 1$. Ako je b "otkriven" BFS algoritmom kao sused od a, tada bi važilo $dist(b) = dist(a) + 1 = k + 1$, što nije tačno. Ako je a "otkriven" BFS algoritmom kao sused od b, tada bi važilo $dist(a) = dist(b) + 1 = l$, što nije tačno. Preostaje mogućnost da su se a i b u nekom momentu našli zajedno na listi L u BFS algortimu. Kako je $dist(a) - dist(b) = k - l + 1 > 2$, po prethodnoj lemi dobijamo kontradikciju.

BFS algoritam - složenost

Neka je $G = (V, E)$, pri čemu je n broj čvorova u V, a m broj grana u E.

- prvi red troši $\Theta(1)$ vremena;
- prva **for** petlja troši $\Theta(n)$ vremena;
- rad **while** petlje možemo ograničiti sa $O(n^2)$, međutim možemo i bolje sa $O(m)$ jer će svih iteracija ukupno biti jednako dvostrukom broju svih grana komponente povezanosti čvora s. Intutivno, za svaku granu (a, b) iz komponente povezanosti čvora s: ako je a prvi "obrađeni" tada će a biti skinut sa liste L, a b dodat u L kao sused i označen kao posećen. Zatim, u jednoj od sledećih iteracija će se obrađivati susedi od b kada će se za a samo ustanoviti da je već bio posećen. Dakle, uslov u **if**-u će biti proveren $2l$ puta, dok će poslednji red algoritma zapravo biti izvršen tačno l puta, gde je l broj svih grana komponente povezanosti čvora s.

Dakle, složenost BFS algoritma je $O(n + m)$.

BFS algoritam - najkraći put od s do t

Problem: Za dati graf $G = (V, E)$ i dva njegova čvora s i t: ako su s i t povezani nađi jedan put minimalne dužine koji ih povezuje.

Rešenje:

- Primeni BFS(V, E, s);
- ako je $dist(t) = k$ iz skupa suseda od t nađi jedan b_1 za koji važi $dist(b_1) = k - 1$;
- iz skupa suseda od b_1 nađi jedan b_2 za koji važi $dist(b_2) = k - 2$;
- ...
- iz skupa suseda od b_{i-1} nađi jedan b_i za koji važi $dist(b_i) = k - i$;

Tada je $s, b_{k-1}, \ldots, b_1, t$ jedan put minimalne dužine od s do t.

BFS2 algoritam - verzija bez lista

Algorithm 5: BFS2(V, E, s)

```
dist(s) = 0;  Pos = {s};  Akt = {s}
for a ∈ V\{s} do
    dist(a) = ∞
while Akt ≠ ∅ do
    B = ∅
    for a ∈ Akt do
        for b ∈ Susedi(a) do
            if b ∉ Pos then
                dist(b) = dist(a) + 1;  Pos = Pos ∪ {b};  B = B ∪ {b}
    Akt = B
```

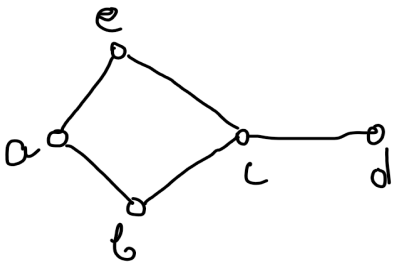
Dokazati da u i-tom prolasku kroz **while** petlju za sve posećene čvorove postoje putevi minimalne dužine i od s do njih. Dokazati korektnost BFS2 algoritma i proceniti njegovu složenost.

BFS2 algoritam - primer

Zadatak

Primeniti BFS2 algoritam na graf sa slike dole, uzeti da je $s = a$.

Rešenje. Na času. ◇



$$A_G = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ -a \\ -b \\ -c \\ -d \\ -e \end{array}$$

u	L_u
a	{b, e}
b	{a, c}
c	{b, e, d}
d	{c}
e	{a, c}

Tema 4

Dajkstrin algoritam

Dajkstrin algoritam

Problem: Dat je težinski graf $G = (V, E, w)$ i jedan čvor s. Odrediti težinu najlakšeg puta od s do svakog drugog čvora.
Težinsku funkciju ćemo definisati kao $w : V \times V \to \mathbb{N} \cup \{\infty\}$, gde je $w(a, a) = 0$, $w(a, b) = \infty$ ako $(a, b) \notin E$ i $a \neq b$. Tešina puta se definiše kao zbir težina pojedinačnih grana koje čine taj put.

Algorithm 6: Dajkstra(V, E, w, s)

$dist(s) = 0;\ Pos = \{s\}$
for $a \in V \backslash \{s\}$ **do**
  $dist(a) = w(s, a)$
while $Pos \neq V$ **do**
  nađi $a \in V \backslash Pos$ takvo da je $dist(s, a)$ minimalno
  $Pos = Pos \cup \{a\}$
  for $b \in V \backslash Pos$ **do**
    $dist(b) = min(dist(b), dist(a) + w(a, b))$

Dajkstrin algoritam

Složenost: Neka je n broj čvorova, a m broj grana. Prva **for** pelja troši $\Theta(n)$ vremena. U **while** petlji: provera uslova, prva linija i **for** petlja troše $\Theta(n)$, a druga linija proši $\Theta(1)$ vremena. Takođe svakom iteracijom se u Pos dodaje po jedan čvor, pa je broj iteracija **while** petlje $\Theta(n)$. Dakle, ukupna složenost je $\Theta(n^2)$.

Zadatak

Primeniti Dajkstrin algoritam na težinski graf sa slike dole, uzeti da je $s = a$.