

Konstrukcija i analiza algoritama 2

Primer pismenog dela ispita

Svaki zadatak nosi po 10 poena. Pismeni deo ispita sastoji se od teorijskog dela (3 zadatka) i praktičnog dela (3 zadatka). Zadatke iz praktičnog dela potrebno je dopuniti da budu tačni, dok se teorijski zadaci rešavaju od nule. Teme koje dolaze u obzir su sve teme radjene na vežbama uživo, kao i teme sa okačenih video snimaka vežbi. Ovde su kao primer ispita predložene neke oblasti, no na ispitu ne mora da se dogodi da budu baš ove oblasti.

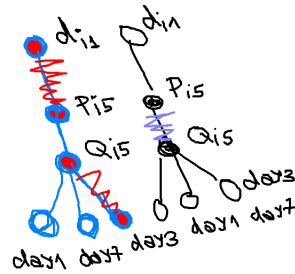
1 Primer ispita - teorijski deo

1. Medicinskoj firmi “Doktori bez vikenda” potrebna je pomoć oko usaglašavanja radnog vremena doktora u velikoj bolnici. Postoji k prazničnih perioda, od kojih svaki obuhvata nekoliko uzastopnih dana. Neka je D_j skup svih prazničnih dana koji ulaze u praznični period j . U bolnici je zaposleno n doktora, i za doktora i poznat je skup prazničnih dana S_i kada je doktor dostupan (imati na umu da ovi dani ne moraju biti iz istog prazničnog perioda, da doktor ne mora biti slobodan sve dane u jednom prazničnom periodu, kao i da tokom nekih prazničnih perioda doktor može biti konstantno zauzet). Dati polinomijalni algoritam koji u skladu sa datim informacijama daje odgovor na pitanje da li moguće da organizujemo rad doktora tako da ispoštujemo sledeća ograničenja:

- Za dati parametar c , svaki doktor treba da radi ukupno c prazničnih dana i to samo u danima kada je slobodan
- Za svaki praznični period j , svaki doktor treba da radi najviše jedan dan iz skupa D_j

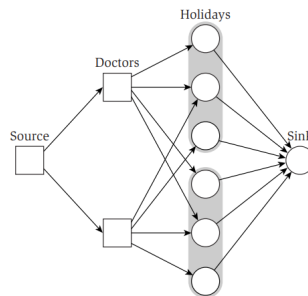
REŠENJE: (I NAČIN) Zadatak možemo svesti na problem pronalaženja optimalnog uparivanja (uparivanje sa najvećim brojem grana). S obzirom da svaki doktor treba da radi tačno c dana, za svakog doktora napravimo c kopija - za doktora 1 na primer ti čvorovi će biti označeni sa $d_{11}, d_{12}, d_{13}, \dots, d_{1c}$ (**2 poena**). Za svakog doktora i , uvešćemo čvorove P_{ij} koji nam predstavljaju praznične periode u kojima je dati doktor slobodan. Svaku kopiju doktora povezaćemo sa ovim čvorovima (**1 poen**). Za svaki čvor P_{ij} uvešćemo veštački dodat čvor Q_{ij} koji ćemo da spojimo

sa P_{ij} (**2 poena**). Poslednje što nam ostaje jeste da veštački dodat čvor Q_{ij} povežemo sa čvorovima S_{ij} koji predstavljaju praznične dane u j -tom periodu kada je doktor i slobodan (**1 poen**).

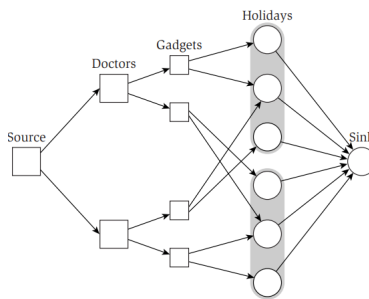


Na slici su prikazana dva načina na koja uparivanje u jednom delu grafa može da se odvije. Na levom primeru, čvor d_{i1} (prva kopija doktora i) uparen je sa prazničnim periodom P_{i5} , što znači da će doktor raditi taj praznični period. Takodje, spojeni su čvorovi Q_{i5} i $day3$ što znači da će u datom prazničnom periodu doktor i raditi treći dan (ovakva struktura grafa obezbeđuje da doktor u svakom prazničnom periodu u kome je slobodan radi najviše jedan dan). Na desnom primeru spojeni su P_{i5} i Q_{i5} što znači da doktor i neće raditi u prazničnom periodu 5 (**2 poena**). Optimalno uparivanje će težiti levom primeru (tu dobijamo dve grane, na desnom dobijamo jednu). Kada pronadjemo optimalno uparivanje, ostaje da proverimo da li su sve kopije doktora uparene. Ako jesu, to je traženi raspored, ako nisu, nije moguće naći raspored za dane koje su doktori naveli kao svoje slobodne (**2 poena**).

REŠENJE: (II NAČIN) Iako se u ovom zadatku ne pominje upotreba transportnih mreža, razlog zašto možemo da naslutimo da je to ovde moguće jeste taj što pokušavamo da povežemo jedan skup (doktore) sa drugim skupom (praznični dani). Da nemamo drugo ograničenje koje se tiče toga da svaki doktor mora da radi najviše jedan dan u svakom D_j , rešenje bi bilo klasično svodjenje na problem transportnih mreža:



Postupili bismo kao na slici (**2 poena**). Svaki doktor bi bio predstavljen čvorom u_i , svaki praznični dan čvorom v_j . Spojili bismo doktore sa danima kada su dostupni. Izvor (veštački dodati čvor) bismo povezali sa doktorima granama kapaciteta c (**1 poen**). Kapacitet svake grane koja spaja doktora sa prazničnim danom bi bio 1 (**1 poen**). Praznične dane treba povezati sa ponorom (takodje veštački dodatim čvorom). Kapacitet grana od jednog dana ka ponoru postavimo na n -broj doktora, jer svakog dana može da radi maksimalno onoliko doktora koliko ih je zaposleno u bolnici (**1 poen**).



Kako sada da prilagodimo problem našem traženom? Pošto imamo ograničenje da za svaki praznični period i svakog doktora, doktor u tom periodu može da radi najviše jedan dan, onda za svakog doktora i svaki praznični period uvodimo po jedan čvor (Gadget na slici) koji spajamo sa doktorom i sa odgovarajućim prazničnim danima (**2 poena**). Kapacitet grane od doktora do gadgeta iznosi 1, isto kao i kapacitet grane od gadgeta do prazničnog dana (**1 poen**).

Nakon što konstruišemo mrežu na opisani način, koristimo neki algoritam za maksimalni tok. Ako maksimalni tok iznosi tačno $n \cdot c$, onda postoji raspored koji zadovoljava sve postavljene uslove. U suprotnom, raspored nije moguć (**2 poena**).

- Korisćenjem samo funkcije **rand2()** koja vraća 0 i 1 sa istom verovatnoćom, napisati **rollDice()** funkciju koja simulira bacanje fer kockice (dobijanje brojeva 1,2,...,6 sa istom verovatnoćom).

REŠENJE: S obzirom da 6 nije stepen dvojke, možemo generisati vrednosti do 6, a one koje su veće odbaciti (**1 poen**). Tri poziva funkciji **rand2** daju nam tri bita koja zajedno predstavljaju zapis brojeva od 0 do 7 (**2 poena**). Ukoliko dobijemo broj od 0 do 5, povećaćemo ga za 1, i dobijeni broj predstaviti kao rešenje, a ako dobijemo broj 6 ili broj 7, njega ćemo odbaciti i pokušaćemo ponovo da generišemo novi binarni zapis (**2 poena**). S obzirom da postoji 8 mogućih ishoda, a 6 prihvatamo, verovatnoća prihvatanja rešenja je $6/8 = 3/4$ (ovo važi po definiciji klasične verovatnoće jer su svi ishodi podjednako verovatni) (**2 poena**).

Algorithm 1 rollDice

```
1: function ROLLDICE
2:   while true do
3:      $bit_1 \leftarrow \text{RAND2}$ 
4:      $bit_2 \leftarrow \text{RAND2}$ 
5:      $bit_3 \leftarrow \text{RAND2}$ 
6:      $num \leftarrow bit_1 * 4 + bit_2 * 2 + bit_3$ 
7:     if  $num < 6$  then
8:       return  $num + 1$ 
9:     end if
10:  end while
11: end function
```

Pseudo kod nosi 3 poena.

3. Pretpostaviti da su vrednosti niza L dužine n^2 distribuirane duž dvodimenzionalne mreže sastavljene od $p = n^2$ procesora (slaganjem red po red). Odrediti procesor sa najmanjim indeksom i u nizu L , za koji je $L(i) = X$ gde je X tražena vrednost. Opisati paralelni algoritam za traženje X na mreži i ilustrovati ga na primeru 3×3 za $X = f$. Može se pretpostaviti da svi procesori već znaju vrednost X . Algoritam treba da bude linearne složenosti i rešenje treba smestiti u procesor sa indeksom 0.

```
[c] --- [g] --- [f]
|       |       |
[a] --- [t] --- [f]
|       |       |
[c] --- [f] --- [c]
```

REŠENJE: Pretpostavimo da je indeks procesora na poziciji (i, j) jednak $n * i + j$ (**1 poen**). Osnovna ideja jeste da svaki procesor markiramo kao potencijalnog kandidata sa njegovom pozicijom (u slučaju kada je vrednost koju procesor čuva jednaka X) ili sa nekom posebnom vrednošću (možemo da je označimo sa M) koja je veća od svih dostupnih pozicija (u slučaju kada je vrednost koju procesor čuva različita od X) (**2 poena**). Procesori izvedu paralelno izračunavanje minimuma po mreži, najpre po redovima, a potom po kolonama (**2 poena**). Na konkretnom primeru, u prvom koraku porede se redovi cfc sa atf i to tako što se istovremeno porede a i c, t i f, f i c (njihove pozicije i M vrednosti). Rezultati se skladište u procesorima reda atf jer želimo da se primaknemo poziciji 0. Kada se a i c uporede (vrednosti su im M i M jer nisu isti kao $X=f$), u procesoru koji čuva a, beleži se M vrednost (uz a, a se ne briše) (**2 poena**). Kada se t i f uporede (vrednosti su im M i 8), broj 8 se beleži u procesoru koji čuva t. Kada

se `f` i `c` uporede (vrednosti su `im` 5 i `M`), 5 se beleži u procesoru koji čuva `f` (**1 poen**). Dalje nastavljamo tako što poredimo redove `cgf` i `atf`, i to tako što istovremeno poredimo `c` i `a`, `g` i `t`, `f` i `f`. Kada završimo redove, prelazimo na kolone. Obilazimo ih zdesna na levo, dakle prvo poredimo `ffc` i `gtf`, a potom `gtf` i `cac` (**1 poena**). Složenost je linearna, jer se poredi $n-1$ parova redova u vremenu $O(1)$, a potom kasnije ponovo $n-1$ parova kolona u vremenu $O(1)$ (**1 poena**).

2 Primer ispita - praktični deo

1. **Zadatak test1.cpp:** U ovom zadatku potrebno je implementirati strukturu podataka *sufiksno stablo* za dati ulazni string. Sufiksno stablo se koristi za rešavanje problema vezanih za pretragu podstringa, pronalaženje ponavljajućih obrazaca i sličnih operacija nad stringovima.

Dat je skelet koda u kome su definisani osnovni tipovi podataka (struktura koja predstavlja čvor sufiksnog stabla) i osnovne metode (kao što su `build` i `search`). Vaš zadatak je da dovršite implementaciju na sledeći način:

(a) Izgradnja sufiksnog stabla:

- (**2 poena**) Implementirati funkciju `insertSuffix(int start)` koja ubacuje jedan sufiks u stablo.
- (**1 poen**) Funkcija `build()` treba da pozove `insertSuffix(i)` za sve indekse i u opsegu od 0 do `text.size()-1`.
- (**2 poena**) Voditi računa o tome da, u slučaju nepodudaranja ili delimičnog poklapanja na nekoj grani, odgovarajuće podelite granu kreiranjem novih unutrašnjih čvorova u stablu.

(b) Pretraga obrasca:

- (**2 poena**) Implementirati funkciju `search(const string pattern)` koja pretražuje ulazni obrazac unutar sufiksnog stabla.
- (**2 poena**) Ako se obrazac poklapa sa delom stabla, potrebno je prikupiti početne indekse svih pojavljivanja obrasca u originalnom tekstu.
- (**1 poen**) Ukoliko obrazac nije pronađen, funkcija treba da vrati prazan vektor.

Dodatne napomene

- Skelet koda sadrži osnovne strukture (kao što su `SuffixTreeNode` i `SuffixTree` klase) i početne komentare koji objašnjavaju šta treba implementirati.
- Nije neophodno implementirati napredne algoritme poput Ukkonenovog algoritma. Može se koristiti jednostavniji pristup (npr. ubacivanje svih sufiksa jedan po jedan) pod uslovom da je rešenje ispravno.

- Pazite na efikasnost i modularnost rešenja; kod mora biti pregledan i jasan.
- Objasnite u komentarima osnovnu ideju izgradnje sufiksnog stabla i pretrage obrasca.

2. **Zadatak test2.cpp:** Vaš zadatak je da kompletirate implementaciju AVL stabla u programskom jeziku C++. AVL stablo je samobalansirajuće binarno stablo pretrage koje garantuje da će visina stabla biti $O(\log n)$ nakon svakog umetanja, čime se osigurava efikasnost operacija pretrage, umetanja i brisanja.

Data vam je skelet (osnovni) kod sa implementiranim nekim funkcijama. Vaš zadatak je da dovršite implementaciju sledećih funkcija:

- Funkcija leftRotate** — koja izvršava levu rotaciju oko čvora.
- Funkcija insert** — koja umeće novi čvor u AVL stablo, održava binarno stablo pretrage, ažurira visine čvorova, i izvršava potrebne rotacije radi održavanja svojstva balansiranosti stabla.

Zahtevi zadatka:

Implementacija funkcije leftRotate (3 boda)

Opis: Funkcija treba da izvrši levu rotaciju oko datog čvora x .

Podela poena:

- **(2 poena)** Prevezivanje pokazivača
- **(0.5 poena)** Ažuriranje visina
- **(0.5 poena)** Ispravno vraćen novi koren podstabla.

Implementacija funkcije insert (7 bodova)

Opis: Funkcija treba da umeće novi čvor sa ključem key u AVL stablo.

Podela poena:

- **(2 poena) - Standardna BST logika:** Kreiranje novog čvora i rekursivno umetanje u levo ili desno podstablo u zavisnosti od vrednosti ključa.
- **(1 poen) - Ažuriranje visine:** Nakon umetanja, ažurirajte visinu trenutnog čvora
- **(1 poen) -Izračunavanje balans faktora:**
- **(3 poena) - Balansiranje stabla:** Ako je balans faktor veći od 1 ili manji od -1, prepoznajte odgovarajući slučaj:

3. **Zadatak test3.cpp:** Dat je neusmereni graf koji je stablo sa N čvorova označenih brojevima od 1 do N i $N - 1$ grana. Svaka grana povezuje par čvorova. Vaš zadatak je da dopunite implementaciju u C++-u koja Vam je data tako da pronalazi optimalno uparivanje u datom stablu.

Definicija: Uparivanje u stablu predstavlja skup grana tako da nikoje dve nisu incidentne sa istim čvorom. Optimalno uparivanje je ono uparivanje koje sadrži najveći mogući broj grana. Na engleskom jeziku je termin za optimalno uparivanje Maximum matching.

Program treba da sadrži sledeće funkcionalnosti:

- (a) **Struktura stabla:** Koristiti listu susedstva za predstavljanje stabla.
- (b) **Funkcija AddEdge:** Dodaje granu izmedju čvorova u i v (grana se dodaje u oba smera).
- (c) **Funkcija Matching_dfs:** Implementirati rekursivnu DFS funkciju koja obilazi stablo i računa optimalno uparivanje.
- (d) **Funkcija maxMatching:** Inicira DFS obilazak (npr. od čvora 1) i ispisuje broj grana u optimalnom uparivanju.

Podela poena u rešenju:

- Iteriranje kroz sve susede posmatranog čvora u - **(2 poena)** za dobar obilazak suseda + **(2 poena)** za neodlazak u već vidjene čvorove
- Nakon procesiranja suseda čvora u , proveravamo da li su on i njegov roditelj neupareni - **(3 poena)** za proveru neuparenosti i da li roditelj nije još obradjen + **(3 poena)** pravilno ažuriranje brojača grana i markiranje uparenih čvorova koje će sprečiti da čvorovi budu ponovo upareni