Borna Gajić

Zadatak 1.
Napišite pseudokod za Make-Set, *Find − Set* i *Union* operacije ako se koristi reprezentacija disjunktnih skupova povezanom listom i weighted-union heuristika. Specificirajte atribute koje imaju objekti u listi.

Rješenje

```
def Make-Set(L, x):
    L.head = x
    L.tail = x
    L.size = 1
    x.next = NIL
    x.list = L
```

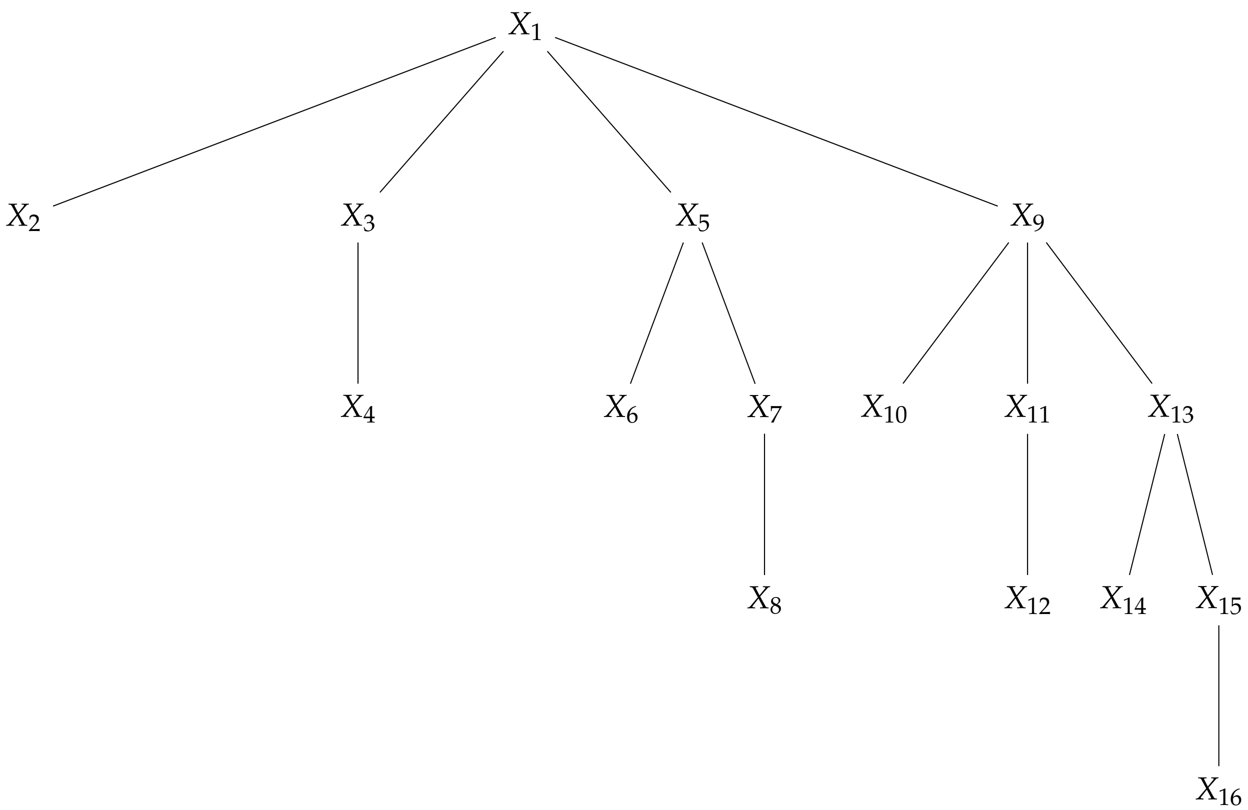
```
def Find-Set(x):
    return x.list.head
```

```
def Union(x, y):
    if x.list.size > y.list.size then
        a = x
        b = y
    else
        a = y
        b = x
    end if

    a.list.tail.root = b.list.tail
    a.list.size = a.list.size + b.list.size
    iter = b.list.head

    while iter != NIL do
        iter.list = b.list
        iter = iter.next
    end while
```

L je lista koja će postati disjunktni skup, gdje je *L.head* pokazivač na prvi član liste (predstavnika skupa), *L.tail* pokazivač na zadnji element liste i *L.size* atribut koji prati duljinu skupa. U čvoru su enkapsulirani atributi *x.next* koji sadrži pokazivač na idući element iste liste, i *x.list* koji predstavlja pokazivač na listu u kojoj se nalazi.

Zadatak 2.
Pokažite kako izgleda rezultirajuća struktura podataka te što vrati operacija *FindSet* u danom programu. Koristite reprezentaciju disjunktnih skupova sa unijom po rangu i kompresijom putova.

Rješenje



Procedura $Find - Set(X_2)$ će vratiti isto što i poziv procedure na čvor X_9, a to je X_1.

Zadatak 3. Koristeći činjenicu da svaki čvor ima rang $\lfloor lgn \rfloor$, dajte jednostavan dokaz da je vrijeme izvršavanja operacija na šumi disjunktnih skupova koja koristi uniju po rangu, ali ne i kompresiju putova jednako $O(m \cdot lgn)$.

Rješenje
$Find - Set$ procedura radi u $O(lgn)$ vremenu jer ako ju pozovemo na zadnjem čvoru u stablu moramo se vračati po parentima do samog roota stabla koji je predstavnik skupa u kojem se isti čvor nalazi. Procedura *Union* spaja root stabla s manjim rangom na root stabla s večim rangom; pošto ta procedura prima dva parametra, u i v, koji ne moraju biti predstavnici skupova u kojima se ti čvorovi nalaze, mora se dva puta pozvati procedura $Find - Set$ što u konačnici daje vrijeme izvršavanja od $O(lgn)$. Takvih m operacija *Union* i $Find - Set$ nam daje $O(m \cdot lgn)$ vrijeme.

Zadatak 4.
Problem offline minimuma zahtijeva održavanje dinamičkog skupa T elemenata iz domene $1, 2, ..., n$ na kojem se mogu izvršavati operacije $Insert$ i $Extract - Min$. Dan je niz S od n $Insert$ i m $Extract - Min$ poziva, gdje je svaki ključ iz skupa $1, 2, ..., n$ ubačen točno jednom. Želimo odrediti koji ključ je vraćen kojom $Extract - Min$ operacijom. Točnije, želimo popuniti polje $extracted[1, ..., m]$ elementima tako da $extracted[i]$ za $i = 1, ..., m$ predstavlja ključ koji vrati i-ti Extract-Min poziv. Problem je "offline" u smislu da smijemo procesuirati cijeli niz operacija S prije nego odredimo koje će točno ključeve operacije $Extract - Min$ vratiti.

(a) U sljedećoj instanci problema offline minimuma, svaka operacija $Insert(i)$ je reprezentirana vrijednošću i umetnutog ključa, a svaka operacija $Extract - Min$ je reprezentirana slovom E: 4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5. Popunite polje $extracted$ odgovarajućim vrijednostima.

(b) Kako bismo mogli razviti algoritam za ovaj problem, rastavit ćemo niz S u homogene podnizove. Točnije, S će biti predstavljen kao niz $I_1, E, I_2, E, ..., I_m, E, I_{m+1}$, gdje svako slovo E predstavlja jedan poziv operacije $Extract - Min$, a svaki I_j predstavlja (možda i prazan) niz poziva $Insert$ operacije. Za svaki podniz I_j prvo se ključevi koji se trebaju umetnuti ubace u skup K_j, koji je prazan ukoliko je I_j prazan, a zatim se radi sljedeći algoritam:

```
function Off-Line-Minimum(m, n)
    for i = 1 to n do
        odredi j takav da je i ∈ K_j
        if j ≠ m + 1 then
            extracted[j] ← i
            neka je l najmanja vrijednost veća od j za koju postoji K_l
            K_l ← K_j ∪ K_l, pri čemu se K_j uništi
        end if
    end for
    return extracted
end function
```

Objasnite zašto je polje extracted koje vrati Off-Line-Minimum algoritam ispravno.

(c) Opišite kako biste efikasno implementirali $Off - Line - Minimum$ algoritam koristeći disjunktne skupove kao strukturu podataka. Dajte čvrstu ogradu na najgore vrijeme izvršavanja te implementacije.

Rješenje

(a)

Iteracija	Extracted
1.	4
2.	3
3.	6
4.	2
5.	8
6.	1

(b) **Dokaz**

Neka su n i m gornje granice operacija *Insert* i *Extract*, te neka je polje *Extracted* duljine m rezervirano za elemente koji će na i-tim pozicijama predstavljati i-tu *Extract* operaciju.
U svakoj iteraciji for petlje uzima se najmanji element $i \in K_j$ pri čemu K_j predstavlja j-ti skup *Insert* operacija.
Razlikujemo dva slučaja, kada je $j = m + 1$, i $j \neq m + 1$:

Ako je $j \neq m + 1$, element i stavljamo u polje *Extracted* nakon čega odradimo uniju nad skupovima K_l i K_j, $K_l = K_l \cup K_j$, i uništimo skup K_j; K_l predstavlja najbliži skup nakon K_j. Skup K_l smo uništili jer smo iz njega izvukli najoptimalnije rješenje te unirali sa idućim radi ponavljanja istog postupka.

Ako je $j = m + 1$ procedura terminira i vraća polje *Extracted* jer je broj operacija *Insert* premašio broj operacija *Extract*; pa se do tog trenutka polje *Extracted* napunilo.

Pošto ne postoji treća opcija, algoritam $Off - Line - Minimum$ je korektan jer smo do trenutka $j = m + 1$ uzimali najoptimalnije elemente. $\square$

(c) Efikasnije bi implementirao $Off - Line - Miminum$ korištenjem disjunktnih skupova (predstavljeni pomoću povezane liste)
Prvo se procedurom $Make - Set$ od ključeva treba napraviti jednočlani skupovi, nakon čega se trebaju ubaciti u skupove K_j, što bi odradio s procedurom $Make - Set$. Treći korak je traženje j, takav da je $i \in K_j$, kojeg određuje sporo rastuća funckija $\lambda(n)$. Pozvao bi proceduru $Find - Set(j)$ koja vraća pokazivač na list u kojoj se i nalazi. Sljedeći korak je unija skupova K_l i K_j koju odrađuje procedura $Union(l, j)$.
Ubacivanje elemenata je u konstantnom vremenu te ne predstavlja nikakvu težinu u vremenskom smislu.

Vremenska analiza

Union u skupove K_j zajedno s $Make - Set$ radimo prije izvršavanj algoritma.

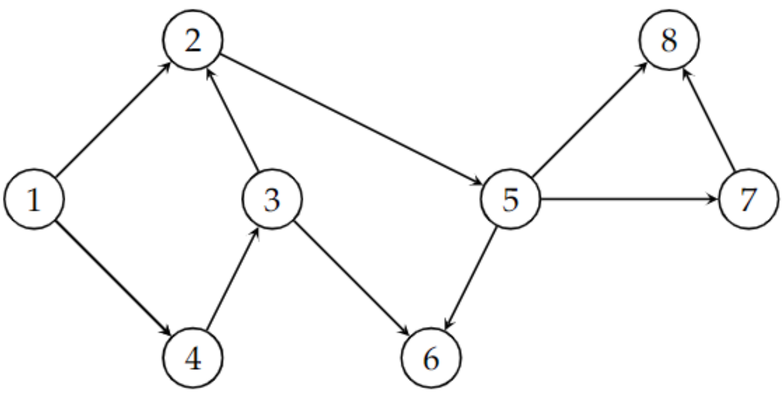
a) Pošto for petlja ide od $1, ..., n$ procedura koja traži j se poziva maksimalno n puta, stoga je vrijeme izvršavanja $O(n\lambda(n))$

b) Unija skupova K_j i K_l izvodi se u $O(minj.list.size, l.list.size) = O(n)$, no postupno će se smanjivati (eksponencijalno) broj skupova stoga je za n poziva procedure $Union(l, j)$ vrijeme u asimptotskom vremenu izvršavanja $O(1)$

Vrijeme izvršavanja efikasnije implementacije algoritma $Off - Line - Miminum$ je $O(n \cdot \lambda(n))$.

Zadatak 5.

Napišite koje su d i π vrijednosti za svaki vrh u grafu sa slike nakon izvršavanja BFS algoritma iz početnog čvora 1.



Rješenje

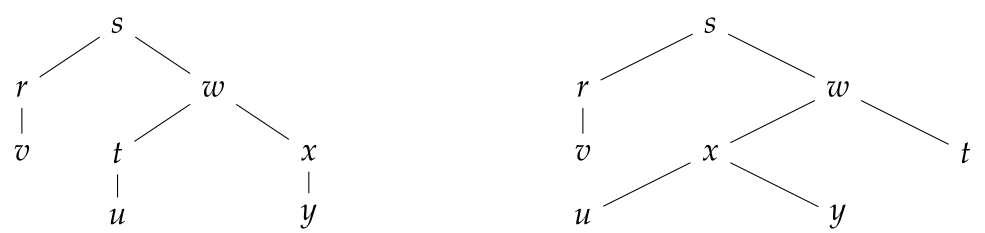
čvor	1	2	3	4	5	6	7	8
d	0	1	2	1	2	3	3	3
π	NIL	1	4	1	2	3	5	5

Zadatak 6.
Pokažite da je korištenje jednog bita za spremanje svake boje vrha dovoljno u smisluda će BFS algoritam vratiti isti rezultat i ako se uklone retci 5 i 14 u pseudokodu za BFS. (Pseudokod se nalazi na stranici 595 u knjizi Introduction to Algorithms.)

Rješenje
Ako za spremanje podatka boje čvora pretvorimo u tip podatka *Char* onda trošimo svega 1 bit memorije, gdje prvo slovo svake boje predstavlja tu boju. Izbacivanjem linija 5 i 14 u kodu BFS-a nismo ništa promijenili jer nam je jedino bitan podatak jel čvor bijel, odnosno jel ikada bio posjećen. Pa je iz tog razloga tip podatka koji sprema boju čvora mogao isto tako biti i *Bool* jer su nam bitne samo dvije boje (u kodu sa stranice 595), a to su crna i bijela. Algoritam BFS radi ne promijenjeno.

Zadatak 7.
Pokažite da je u BFS algoritmu vrijednost $u.d$ koja se dodijeli svakom vrhu u grafu neovisna o redoslijedu u kojem se vrhovi pojavljuju u listi susjedstva. Na primjeru grafa sa slike, pokažite da stablo dobiveno BFS-om može ovisiti o redoslijedu u kojem su vrhovi poslagani u listu susjedstva (pri tome je s početni čvor za BFS).

Rješenje
Vrijednost $u.d$ predstavlja najkraći put od početnog čvora s do čvora u i ono ne ovisi o poziciji čvorova unutar liste susjedstva jer ako više čvorova, jednako udaljeni od s, dijele u kao susjda; udaljenost od s do u će biti jendaka koji god čvor prije dođe na red za obilaženje svojih susjeda.
Primjer sa slike je:
$s \to w \to t \to u$ ili $s \to w \to x \to u$.
U suštini, algoritam BFS je tako osmišljen da ne vodi račun o listi susjedstva pojedinog čvora jer će svakako preskočiti one koji su već prethodno obilaženi (oni bliži početnom čvoru će prvi obići traženi čvor). Sve prijašnje tvrdnje vrijede uz pretpostavku da je u dio skupa u kojem se s nalazi.
Drugi dio zadatka:



Zadatak 8. Promjer (diameter) stabla $T = (V, E)$ definira se kao $max_{u,v \in V} \delta(u, v)$, tj. kao najdulji od svih najkraćih putova u stablu. Dajte efikasan algoritam za računanje promjera stabla i prokomentirajte vrijeme izvršavanja tog algoritma.

Rješenje

```
def Diameter (T):

    BFS(T, T.root)
    max = - infty.

    for each vertex u in T.v do
        if u.d >= max then
            max = u.d
        end if
    end for

    return max
```

Vrijeme izvršavanja: $O(V + E) + O(V) = O(2V + E) = O(N)$.
Možemo reći da je vremenski linearno s obzirom na veličinu grafa.