Zadaća 7

Zadatak 1

Napišite nerekurzivnu verziju FIND-SET sa kompresijom putova.

In [ ]:
```python
def find_set(x):
    root = x
    #nadi root
    while root.parent != root:
        root = root.parent

    #kompresiraj puteve
    while x != root:
        parent = x.parent
        x.parent = root
        x = parent

    return root
```

Zadatak 2

Pokažite kako izgleda rezultirajuća struktura podataka i odgovori koje vrati FIND-SET operacija u sljedećem programu. Koristite reprezentaciju disjunktnih skupova sumom disjunktnih skupova sa unijom po rangu i kompresijom puteva.

In [ ]:
```javascript
for(let i = 1; i <= 16; i++)
    MAKE-SET(i);
for(let i = 1; i <= 15; i += 2)
    UNION(i, i + 1);
for(let i = 1; i <= 13; i += 4)
    UNION(i, i + 2);
UNION(1, 5);
UNION(11, 13);
UNION(1, 10);
FIND-SET(2);
FIND-SET(9);
```

1. Originalno imamo 16 skupova koje unosimo petljom 1, svaki sadrži i = 1,2,..16.
2. Unijom susjeda u petlji 2 dobivamo: **{1,2},{3,4},{5,6},{7,8},{9,10},{11,12},{13,14},{15,16}**
3. Unijom svakog trećeg elementa u petlji 3 dobivamo: **{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}**
4. Unijama na linijama 7 i 8 dobivamo po 2 od gornjih skupova spojeno: **{1,2,3,4,5,6,7,8},{9,10,11,12,13,14,15,16}**

5. Napokon, linijom 8 dobivamo jedan skup s vrijednostima 1-16:
   {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}

FIND-SET(2) i FIND-SET(9) će oba vratiti pointer na isti root(jer su oba elementa na istom setu). Ako pretpostavimo da uvijek uzimamo root lijevog skupa kao root unije, oni će vratiti pointer na jedinicu (jer krećemo spajati od jedinice koja je sama sebi root)

Zadatak 3

Za dani niz od m MAKE-SET, UNION i FIND-SET operacija, od kojih je n MAKE-SET, pokažite da je potrebno Ω(m lg n) vremena kada koristimo uniju po rangu.

1. Napravimo n MAKE-SET operacija, odnosno napravimo n setova sa samo jednim elementom.
2. Zatim, kako bismo napravili uniju svih setova u jedan veliki set, koristimo 2^{floor(log_2 n)} - 1 UNION operacija unijom po rangu (bez kompresije puteva). Tako ćemo kreirati binomijalnu hrpu dubine floor(log_2 n). To vrijedi jer unija 2 binomijalne hrpe dubine k daje binomijalnu hrpu dubine k + 1, pa će dubine redom po unijama ići 2,3,4... (UNION(3,4) npr daje hrpu dubine 2, ako su hrpe {3} i {4} dubina 1).
3. Zasad smo iskoristili n + 2^{floor(log_2 n)} -1 operacija. Kako je ova potencija broja 2 omeđena s n od gore zbog svojstva logaritma, a 1 je konstanta, možemo ovo omeđiti na 2n operacija od gore.
4. Ova hrpa sad sigurno ima čvor na dubini floor(log_2 n). Neka taj čvor ima vrijednost k. Taj čvor je reprezentant (nemamo heuristiku težinske unije pa uvijek spajamo s lijeva na desno).
5. Sljedećih m - 2n poziva će dakle imati vremensku složenost O(lg n), jer je čvor na dubini floor(lg_2 n). Kako je m >= n iz uvjeta zadatka, m - 2n možemo omeđiti odozgo sa m. Tako će ukupna složenost algoritma biti od dolje omeđena s Ω(m lg n).

Zadatak 4

Napišite pseudokod za MAKE-SET, FIND-SET i UNION koristeći povezanu listu kao reprezentaciju i heuristiku težinske unije.

In [ ]:
```
def make_set(x):
    o = {} #nekakav objekt
    l = [] #linked lista
    l.head = l.tail = o
```

```
    o.next = None
    o.set = l
    o.value = x #reprezentant

    l.size = 1 # velicina liste je bitna
    return l
```

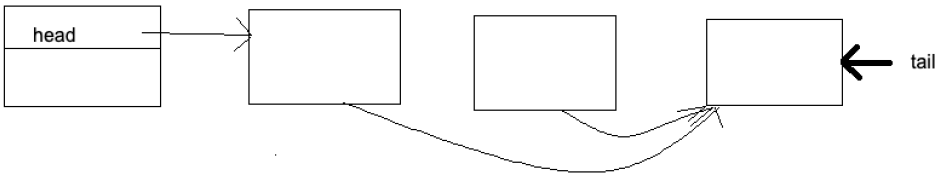
In [ ]:
```
def find_set(x):
    #o je objekt iz make_set
    #ovo radi jer je head liste o, a njegov value je reprezentant
    return o.set.head.value
```

In [ ]:
```
def union(x,y):
    L1= x.set
    L2 = y.set

    if L1.size < L2.size:
        union(y,x)
    l2s = L2.size

    L1.tail.next = L2.head
    z = L2.head
    while z.next != None:
        z.set = L1
    L1.tail = L2.tail

    L1.size += L2.size
    return L1
```

Zadatak 5

Profesor Gompers pretpostavlja da bi bilo moguće zadržati samo jedan pokazivač u svakom objektu skupa, umjesto dva (head i tail), zadržavajući broj pokazivača u svakom elementu liste na dva. Pokažite da je profesorova pretpostavka dobro utemeljena tako što ćete opisati kako predstaviti svaki skup povezanom listom tako da svaka operacija ima isto vrijeme izvršavanja kao i prije. Također opišite kako operacije funkcioniraju. Vasa bi shema trebala omogućiti heuristiku težinske unije, s istim učinkom kao i prije.

In [ ]:
```
def transform(set):
    # postavi root skupa na njegov tail
    set.root = set.tail
    for node in set:
        # umjesto na objekt, pokazuj na tail
        node.set_begin = set.tail
        # ovo mice potrebu za tail pointerom na samom objektu

# ovako bismo tail mogli i dalje dohvatiti u konstantnom vremenu jer prat
# MAKE-SET: ostaje nepromijenjen
# FIND-SET: ostaje nepromijenjen
# UNION: trik je da manji skup prikacimo slijeva većem.
# Razlog tomu je što će sad reprezentant biti tail, pa ćemo sve elemente
# im je reprezentant reprezentant većeg skupa. Kako je skup manji, radimo
```


head
tail