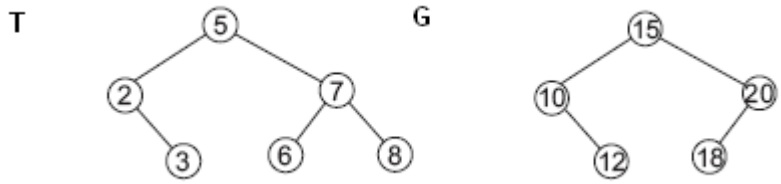
ALGORITMI I STRUKTURE PODATAKA

IV čas

1. Konkatenacija je operacija nad dva skupa koja zadovoljavaju uslov da su svi ključevi u jednom skupu manji od svih ključeva u drugom skupu. Rezultat konkatenacije je unija skupova.
Primeniti algoritam sa prethodnog časa na data dva binarna stabla pretrage. Vremenska složenost algoritma mora biti $O(h)$ u najgorem slučaju, gde je h veća od visina dva stabla.



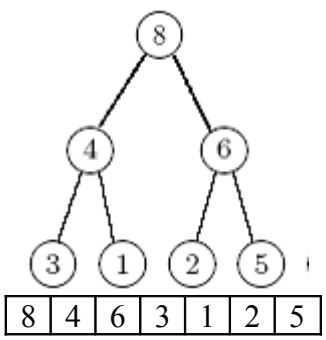
4.1 Hip (eng. *heap*)

Hip je binarno stablo koje zadovoljava uslov hipa: ključ svakog čvora veći je ili jednak od ključeva njegovih sinova.

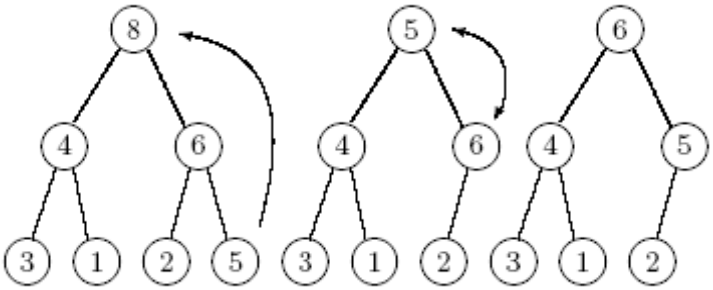
Hip se može realizovati implicitno i eksplicitno.
Mi ćemo se baviti kao kod profesora u knjizi **implicitnim** realizacijama hipa. Dakle, ako hip ima n elemenata, onda se za smeštanje elemenata koriste lokacije u nizu A sa indeksima A[1..n], tako da ako element indeksa ***i*** predstavlja čvor stabla, tada
- element indeksa ***2*i*** predstavlja levo dete čvora,
- element indeksa ***2*i+1*** predstavlja desno dete čvora.
Na primer, deca čvora A[1] su elementi A[2], A[3].

Podsetite se algoritma iz profesorove knjige **Skini_max_sa_hipa (A,n)** za uklanjanje najvećeg elementa sa hipa A dimenzije n.

Neka je dat hip.



Treba ukloniti ključ A[1], a potom transformisati niz A tako da opet zadovoljava svojstvo hipa.
Najzgodnije je u A[1] zapisati krajnji element niza i smanjiti veličinu hipa. Problem je što tada možda nije zadovoljeno pravilo hipa.



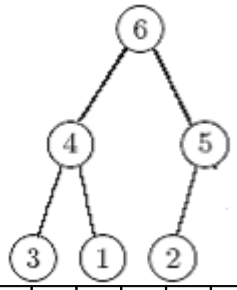
Dovođenje niza u stanje koje zadovoljava pravilo hipa vrši se prema sledećem algoritmu:

Algoritam: uređenje hipa kada vrh hipa nije u skladu s pravilom hipa
1. Započni s indeksom koji predstavlja vrh hipa i smesti taj element u promenljivu x. Nadalje se uzima da je vrh hipa "prazan".
2. Analiziraj decu praznog čvora i odredi koje dete je vece.
3. Ako je ključ od x veci od ključa veceg deteta smesti x u prazan čvor i završi,
inače upiši element veceg deteta u prazan čvor i postavi da čvor veceg deteta bude prazan čvor. Ponovi korak 2.

Rezultat je hip:

6	4	5	3	1	2

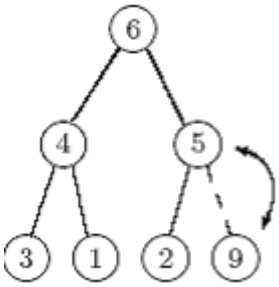
Podsetite se algoritma iz profesorove knjige **Upis_u_hip (A,n,x)** za upis elementa x u hip A dimenzije n.
Neka je dat hip



6	4	5	3	1	2

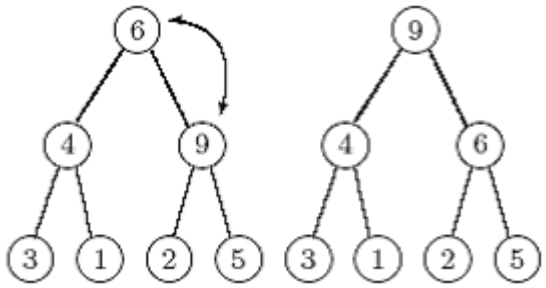
i neka treba ubaciti ključ 9.

Ako želimo dodati element u prioritetni red možemo ga dodati na kraj niza A. To je ekvivalentno dodavanju krajnjeg desnog lista u stablu. Ta operacija ima za posledicu da se N uveća za jedan, ali i da možda stablo ne zadovoljava pravilo hipa.



Da bi se zadovoljilo pravilo hipa koristi se postupak opisan u sledećem algoritmu:

Algoritam: Umetanje
1. Element iza krajnjeg elementa niza se tretira kao prazan (to je sledeći krajnji desni list stabla). Podrazumeva se da je kapacitet niza veći od broja elemenata u nizu.
2. Ako je ključ roditelja praznog čvora veći od ključa elementa x, tada se element x upisuje u prazni čvor. Time je postupak završen.
3. Ako je ključ roditelja praznog čvora manji od ključa elementa x, tada se element roditelja kopira u prazni čvor, a čvor u kojem je bio roditelj se uzima kao prazni čvor. Postupak se ponavlja korakom 2.



Rezultat je hip:

9	4	6	3	1	2	5

Primene hip-a

Lista sa prioritetom: dinamički skup čiji elementi se uklanjaju po redosledu veličina, počev od najvećeg. Na primer, objekat sa najvećim prioritetom se nalazi u korenu heap-a i trivijalno se uzima iz strukture. Ali ako se ukloni koren, tada ostaju dva podstabla i moraju se *efikasno* spojiti u jedno stablo koje će ponovo imati ***heap*** svojstvo.
Korist od upotrebe heap strukture zasniva se na svojstvu da se moze izvaditi objekat najvišeg prioriteta i ubaciti za **O(log*n*)** vremena.

Heap Sort: od kolekcije elemenata kreira se heap za (O(n)) vremena i potom se uklanjaju elementi iz heap-a, a svako uklanjanje ima složenost proporcionalnu visini heap-a, tj. (O(log n)), te ukupna složenost je (O(n log n)).

1. Odrediti izgled implicitno predstavljenog hipa koji se dobija umetanjem redom brojeva 5, 1, 3, 9, 6, 4, 7, 8 polazeći od praznog hipa. Prikazati izgradnju hipa tabelom, čiji svaki red odgovara jednoj promeni hipa. Zatim odrediti izgled hipa dobijenog uklanjanjem najvećeg elementa.

Korak 1 – dodaj 5

5

Korak 2 - dodaj 1

5	1

Korak 3: - dodaj 3

5	1	3

Korak 4 - dodaj 9 i dva podkoraka preuređivanja hipa u I i III redu:

5	**1**	3	**9**
5	9	3	1
5	**9**	3	1
9	5	3	1

Korak 5 – dodaj 6 i jedan podkorak preuređivanja hipa

9	**5**	3	1	**6**
9	6	3	1	5

Korak 6 – dodaj 4 i jedan podkorak preuređivanja hipa

9	6	**3**	1	5	**4**
9	6	4	1	5	3

Korak 7 – dodaj 7 i jedan podkorak preuređivanja hipa

9	6	**4**	1	5	3	**7**
9	6	7	1	5	3	4

Korak 7 – dodaj 8 i dva podkoraka preuređivanja hipa u I i III redu:

9	6	7	**1**	5	3	4	**8**
9	6	7	8	5	3	4	1
9	**6**	7	**8**	5	3	4	1
9	8	7	6	5	3	4	1

Uklanjanje max elementa 9

Korak 1 - ukloni 9

8	7	6	5	3	4	1

Korak 2 – kopiraj poslednji član tj. 1 u koren hipa

1	8	7	6	5	3	4

Korak 3 – preuređivanje hipa, max (8,7) zameni sa 1

1	**8**	7	6	5	3	4
8	1	7	6	5	3	4

Korak 4 – preuređivanje hipa, max (6,5) zameni sa 1

8	**1**	7	**6**	5	3	4
8	6	7	1	5	3	4

2. Neka je *A* vektor koji služi za implicitno predstavljanje hipa. Koliki najmanji broj elemenata hipa može da zauzme niz dužine 16?

REŠENJE: Kako je zauzeto polje na poziciji 16 => mora biti zauzeto i polje na poziciji 8 za čuvanje oca čvora 16. Zato što po definiciji implicitne reprezentacije hipa preko niza, element A[16] odgovara čvoru čiji predak je čvor pridružen elementu A[8]. Sličnim rezonom, dolazi se do zaključka da moraju biti zauzete za pretke i pozicije 4, 2, 1. Dakle, minimalan broj elemenata je 5. ($\log_2 16 + 1$)

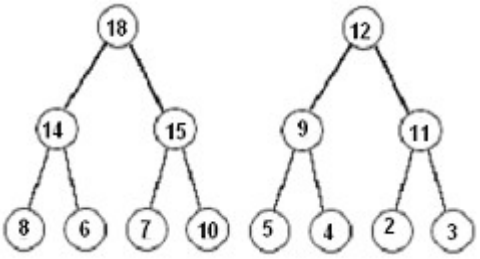
3. Konstruisati algoritam za formiranje hipa koji sadrži sve elemente dva hipa veličine *n* i *m*. Hipovi su predstavljeni eksplicitno (svaki čvor ima pokazivače na svoja dva sina). Vremenska složenost treba da bude u najgorem slučaju $O(\log(m + n))$.

REŠENJE: K1. uklanja se proizvoljan element iz nekog hipa (npr. veći koren) i izvedu se popravke hipa na način koji je opisan u sekciji uklanjanja elementa iz hipa

K2. izabrani element se umeće kao koren novog hipa, tako da njegovi sinovi budu koreni dva stara hipa

Popravka dobijenog hipa (stabla) uz eventualno premeštanje naniže elementa iz korena zahteva O(log(m+n)) koraka zbog dimenzije novog hipa.

4. Primeniti algoritam iz prethodnog zadatka na dva data hipa.



4.2 Hash tabele

Dobra heš funkcija treba da transformiše skup ključeva ravnomerno u skup slučajnih lokacija iz skupa $\{0, \ldots, m-1\}$.

1. Neka je dinamički skup *S* predstavljen tabelom sa direktnim adresiranjem dužine *m* (element sa ključem *k* smešta se na poziciju *k*). Opisati proceduru za nalaženje maksimalnog elementa skupa *S*. Koja je složenost u najgorem slučaju?
Rešenje:
Krećemo od poslednje pozicije i prvi element na koji naiđemo je maksimalni element skupa *S*.
U najgorem slučaju potrebno je pregledati celu tabelu, te je složenost O(m).

2. Pretpostavimo da pretražujemo povezanu listu dužine *n*, gde svaki element sadrži ključ *k* zajedno sa vrednošću $h(k)$. Svaki ključ je dugi niz karakatera.
Kada možemo iskoristiti prednost heš vrednosti pri prolazu kroz listu u potrazi za elementom sa datim ključem?
Rešenje:

Pretraga liste dužine n, gde svaki element sadrži ključ k zajedno sa malom heš vrednosti $h(k)$ može se optimizovati na sledeći način: najpre treba porediti heševe. Ako je to uspešno, poredimo ključeve.

3. Pretpostavimo da heširamo elemente skupa ključeva U u m lokacija. Pokazati da ako je $|U| > (n-1)m$, onda postoji podksup skupa U veličine n koji se sastoji od ključeva koji se svi preslikavaju heš funkcijom u istu lokaciju, te je stoga vreme pretrage za heširanje sa nizanjem u najgorem slučaju $\Theta(n)$.
Rešenje:
Mapiranje $(n-1)m+1$ ključeva u tabelu veličine m rezultira u najmanje jednoj lokaciji sa n ključeva ili više. Ključ sa rednim brojem $(n-1)m+1$ mora da ide na neku lokaciju na kojoj se već nalazi bar n-1 ključeva. Stoga je vreme pretrage za heširanje sa nizanjem u najgorem slučaju $\Theta(n)$.

4. Skicirajte hash tabelu nakon smeštanja ključeva iz skupa {22, 1, 13, 11, 24, 33, 18, 42, 31 } u zadatom poretku. Pretpostavite da je tabela veličine n=11, da je primarna hash funkcija h1(x) = x % 11, kao i da se za razrešavanje kolizija koristi lančanje, otvoreno odresiranje sa linearnim pretraživanjem, otvoreno adresiranje sa duplim heširanjem sekundarnom hash funkcijom g(x) = 1+ x %10.

Ubaciti u hash tabelu ključeve skupa u zadatom poretku koristeći sledeće hash metode:

a. L (LANCANJE (chaining)): funkcijom h(k) = h_1(k)
b. LP (Linear-Probing, linearno pretrazivanje) funkcijom h(k,i) = (h_1(k)+i) % n, i=0..n-1
c. DH(Double-Hashing, dvostruko hesiranje) funkcijom h_1 kao hash funkcijom i funkcijom g kao sekundarnom h(k,i) = (h_1(k) + ig(k)) % n, gde i=0..n-1

Rešenje:

	L	LP	DH
0	33 → 11→ 22	22	22
1	1	1	1
2	24 → 13	13	13
3		11 jer h(k,3)=(11%11+3) %11	
4		24	11 jer h(k,2)=(11%11+2*g(11)) %11=4
5		33	18
6			31
7	18	18	24
8			33
9	31→ 42	42	42
10		31	

5. Razmisliti da li u prethodnom zadatku bi bilo dobro kao primarna funkcija koristi funkcija g, a kao sekundarna funkcija h_1?
6. Skicirajte hash tabelu nakon smeštanja kljuceva 10, 22, 31, 4,15, 28, 17, 88, 59 u zadatom poretku. Pretpostavite da je tabela veličine n=11, da je primarna hash funkcija $h(x) = x \%11$, kao i

da se za razrešavanje kolizija koristi otvoreno odresiranje sa duplim heširanjem sekundarnom hash funkcijom $g(x) = 1 + x \% 10$.

Rešenje:

10 % 11=10

22%11=0

31%11=9

4%11=4

*15%11=4, ali zbog zauzeca koristi se h(k,1) = (h(k) + 1*g(k)) % n=(15%11 + 1*(1+15%10)) %11= (4 +6)%11=10, ali zbog zauzeća*

*koristi se h(k,2) = (h(k) + 2*g(k)) % n=(15%11 + 2*(1 + 15%10)) %11= (4 +2*6)%11=5*

DALJE, završite sami.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
22				4	15				31	10

7. Pretpostavimo da se u hash tabeli umesto povezanih listi koriste binarna stabla pretraživanja za razrešavanje kolizija odvojenim ulančavanjem.
(a) Koliko je vreme izvršavanja operacija umetanja i pretraživanja u najgorem slučaju?
(b) Koliko je očekivano vreme izvršavanja istih operacija?