Skupovi

Općeniti skup

- matematički model koji se često pojavljuje u algoritmima
- skup je zbirka podataka istog tipa (elemenata)
- svi elementi moraju imati različitu vrijednost
- unutar zbirke se ne zadaje eksplicitni linearni ili hijerarhijski uređaj među podacima (kao kod liste i stabla) kao ni neki drugi oblik veze među podacima
- za same vrijednosti elemenata postoji neka prirodna relacija totalnog uređaja
- može se odrediti koji je element veći, a koji manji
- primjer upotrebe skupova: mogu se odrediti skupovi imena osoba A={osobe rođene u Zagrebu}, B={studenti Fizičkog odsjeka PMF Zagreb}. Tada su osobe koje su rođene u Zg ili studiraju na PMF-FO Zg predstavljene skupom A unija B, a osobe koje su rođene u Zg i studiraju na PMF-FO skupom A presjek B. Osobe koje studiraju na PMF-FO, a nisu rođeni u Zg su tada u skupi B bez A
- skup se može definirati kao apstraktni tip podatka s operacijama uobičajenim u matematici

Apstraktni tip podataka SET

- elementtype .. Bilo koji tip s totalnim uređajem ≤
- SET … podatak ovog tipa je konačni skup čiji elementi su međusobno različiti podaci tipa elementtype
- MAKE_NULL(&A) … funkcija pretvara skup A u prazan skup
- INSERT(x,&A) … funkcija ubacuje element x u skup A, tj. mijenja skup A u A unija {x}. Ako x već je element od A, tada ova funkcija ne mijenja A
- DELETE(x,&A) … funkcija izbacuje element x iz skupa A, tj. mijenja A u A bez {b}. Ako x nije element od A, ova funkcija ne mijenja A
- MEMBER(x,A) … funkcija vraća istinu ako je x element od A, inače laž
- MIN(A), MAX(A) … funkcija vraća najmanji (najveći) element skupa A u smislu uređaja ≤. Nedefinirana ako je A prazan skup
- SUBSET(A,B) … funkcija vraća istinu ako je A podskup od B, inače laž
- UNION(A,B,&C) … funkcija pretvara skup C u uniju skupova A i B
- INTERSECTION(A,B,&C) … funkcija pretvara skup C u presjek skupova A i B
- DIFFERENCE(A,B,&C) … funkcija pretvara skup C u razliku skupova A i B

- gornji ATP je teško implementirati, efikasno obavljanje jedne vrste operacija usporava ostale operacije. Razmotrimo prvo implementacije koje omogućuju efikasno obavljanje operacija UNION(), INTERSECTION(), DIFFERENCE(), SUBSET()

Implementacija skupa pomoću bit-vektora

- Može se uzeti, bez gubitka općenitosti, da je elementtype = {0,1,2,…,N-1}
- Skup se prikazuje poljem bitova ili byte-ova (char-ova)
- Bit s indeksom *i* je 1 (0) ako i samo ako *i*-ti element pripada (ne pripada) skupu

```
#define N … /* dovoljno velika konstanta */
typedef char *SET  /* početna adresa char polja duljine N */
```

pojedina varijabla tipa SET se inicijalizira dinamičkim alociranjem memorije za polje ili tako da se poistovjeti s početnom adresom već deklariranog polja

```
SET A;                          ili      char bv[N]
A = (char*) malloc(N);                   SET A = bv;
```

od operacija iz ATP SET funkcije INSERT(), DELETE() i MEMBER() zahtijevaju konstantno vrijeme, jer se svode na direktan pristup *i*-tom bitu (bytu). Funkcije UNION(), INTERSECTION(), DIFFERENCE(), SUBSET() zahtijevaju vrijeme proporcionalno N

- Pseudokod za funkcije UNION() i INTERSECTION() će biti :

```
 void UNION(SET A, SET B, SET *C_ptr){
  int i;
  for (i = 0; i < N; i++)
    (*C_ptr)[i] = (A[i] == 1) || (B[i] == 1);
}

void INTERSECTION(SET A, SET B, SET *C_ptr){
  int i;
  for (i = 0; i < N; i++)
    (*C_ptr)[i] = (A[i] == 1) && (B[i] == 1);
}
```

Ova implementacija nije efikasna za vrlo veliki N.

Implementacija skupa pomoću sortirane vezane liste

- Skup se prikazuje kao lista ostvarena pomoću pokazivača
- Dinamičko rezerviranje memorije je prikladno jer veličina skupa dobivena operacijama UNION(); INTERSECTION() i DIFFERENCE() može jako varirati
- Za efikasnije obavljanje operacija potrebno je da lista bude sortirana u skladu s ≤
- Tipovi celltype i LIST su definirani kao kod liste:

```
typedef struct cell_tag{
        elementtype element;
        struct cell_tag *next;
 } celltype;
 typedef celltype *LIST;
```

- Podaci tipa SET se definiraju kao

```
typedef celltype *SET;
```

- Napisat ćemo funkciju INTERSECTION(), koja računa skup C, prikazan sortiranom vezanom listom čiji je pokazivač na glavu chp, kao presjek skupova A i B, prikazanih sortiranim vezanim listama čije glave pokazuju pokazivači ah i bh
- Funkcije UNION(), DIFFERENCE() i SUBSET() su slične. Vrijeme njihovog izvršavanja je proporcionalno duljini veće od listi

```
void INTERSECTION(SET ah, SET bh, SET *chp) {
celltype *ac, *bc, *cc     /* trenutne ćelije u listi za A i B, te zadnja ćelija u listi za C */
 *chp = (celltype*) malloc(sizeof(celltype));  /* glava liste za C */
 ac = ah->next;
 bc = bh->next;
 cc = *chp;
 while ((ac != NULL) && (bc != NULL)) { /* uspoređuje trenutne elemente liste A i B */
    if ((ac->element) == (bc->element)) { /* dodaje element u C koji je presjek */
          cc->next = (celltype*) malloc(sizeof(celltype));
          cc = cc->next;
          cc->element = ac->element;
          ac = ah->next;
          bc = bh->next;
      } else if ((ac->element) < (bc->element)) /* element iz A je manji */
          ac = ac->next;
          else /* element iz B je manji */
          bc = bc->next;
   }  cc->next = NULL;
 }
```

Rječnik

- U primjenama vrlo često nisu potrebne operacije na skupovima nego se zapisuju podaci jednog skupa te se obavljaju povremena ubacivanja i izbacivanja elemenata u skup i traži se zadani element u skupu
- Takav skup se naziva rječnik
- ATP DICTIONARY se definira slično kao ATP SET, s time da se popis operacija smanji na funkcije MAKE_NULL(), INSERT(), DELETE() i MEMBER()
- Primjeri takvog skupa su:

pravopis je popis ispravno napisanih riječi nekog jezika, provjerava se da li je neka riječ zapisana, povremeno se ubacuju i izbacuju riječi

višekorisničko računalo prepoznaje korisnike na osnovu njihovog imena zapisanog na popisu dopuštenih korisnika

- Implementacija rječnika pomoću bit-vektora je identična onoj za općeniti skup, operacije INSERT(), DELETE() i MEMBER() se obavljaju u konstantnom vremenu. Ova implementacija je neupotrebljiva ako je elementtype velik ili složen tip podataka

Implementacija rječnika pomoću liste

- Skup se može shvatiti kao lista (sortirana u skladu s ≤ ili nesortirana) prikazana pomoću polja ili pokazivača
- Promotrit ćemo sortiranu listu prikazanu kao polje
- Pogodno za statičke skupove (često obavljanje funkcije MEMBER() a rjeđe INSERT() i DELETE()
- Pretraživanje se vrši algoritmom binarnog traženja, pa je vrijeme izvršavanja MEMBER() proporcionalno $\log_2 n$ (n je duljina liste)
- Tip LIST se definira kao kod ATP LIST:

```
typedef struct {
int last;
elementtype elements[MAXLENGTH];
} LIST;
```

Tip SET se deklarira kao

```
typedef LIST SET;
```

Funkcija MEMBER() će tada biti:

```
int MEMBER(elementtype x, SET A) {
   int f, l;  /* indeks prvog i zadnjeg elementa razmatrane podliste */
   int m;   /* indeks srednjeg elementa razmatrane podliste */
   f = 0;
   l = A.last;
   while (f <= l) {
      m = (f+l)/2;
      if (A.elements[m] == x)
          return 1;
      else if (A.elements[m] < x)
          f = m+1;
      else  /* A.elements[m] > x */
          l = m-1;
     }
  return 0;
}
```

- Da bi lista ostala sortirana, INSERT() mora ubaciti novi element na pravo mjesto, što zahtijeva prepisivanje do n elemenata (isto za DELETE()).
- Ako se za rječnik često obavljaju INSERT() i DELETE() pogodnije su ostale tri varijante implementacije pomoću liste. S njima se može izbjeći prepisivanje elemenata, ali se treba pretraživati po listi, pa je vrijeme izvršavanja proporcionalno s n

Implementacija rječnika pomoću rasute (hash) tablice

- Implementacija pomoću bit-vektora je bijekcija oblika elementtype → memorija (1→1 preslikavanje): svakoj mogućoj vrijednost tipa elementtype se pridružuje njena zasebna ćelija memorije
- Prednost: funkcije INSERT(), DELETE(), MEMBER() se obavljaju u jednom koraku
- Mana: zauzima preveliki dio memorije od koje se najveći dio ne koristi
- Kompromis: koristiti umjesto bijekcije surjekciju na manji dio memorije: za svaki dio memorije postoji bar jedan ili više vrijednosti elementtype (obično više)
- Funkcija rasipanja (hash funkcija) h() je potprogram koji surjektivno preslikava skup elementtype na skup cijelih brojeva između 0 i B-1 gdje je B cjelobrojna konstanta. Rasuta (hash) tablica je polje od B ćelija koje zovemo pretinci s indeksima 0, 1, …, B-1
- Implementacija rječnika pomoću rasute tablice se bazira na tome da se element x spremi u pretinac s indeksom h(x) i kasnije se i traži u tom pretincu
- Da bi ideja funkcionirala, bitno je da h() jednoliko raspoređuje vrijednosti iz skupa elementtype po pretincima 0, 1,…,B-1

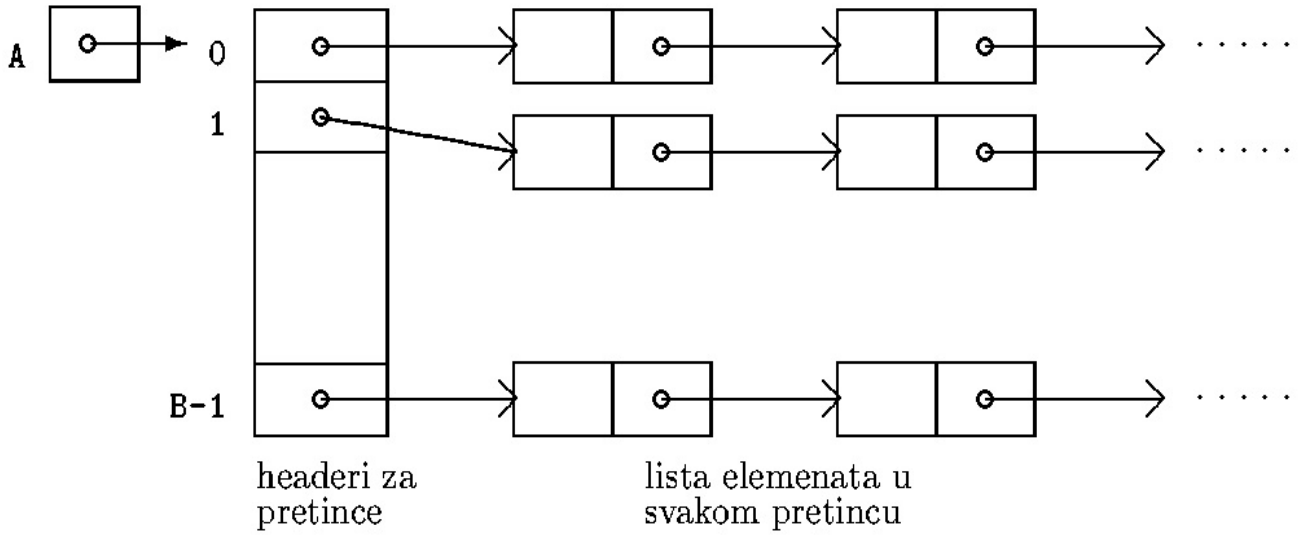
- Primjer: ako je elementtype skup svih nizova znakova duljine 10, može se definirati ovakav h():

```
int h(elementtype x) {
int i, sum = 0;
for (i = 0; i < 10; i++) sum += x[i];
return sum % B;
}
```

- Postoje razne varijante implementacije pomoću rasute tablice, a razlikuju se po građi pretinca tablice

Otvorena rasuta tablica

- Pretinac je građen kao vezana lista, kada treba ubaciti novi element u *i*-ti pretinac *i*-ta vezana lista se produlji još jednim zapisom
- Kapacitet jednog pretinca je neograničen i promjenljiv



```
#define B …
typedef struct cell_tag {
     elementtype element;
     struct cell_tag *next;
} celltype;
typedef celltype **DICTIONARY;  /* početna adresa polja glava (headera) */

void MAKE_NULL(DICTIONARY *Ap) {
 int i;
 for (i = 0; i < B; i++) (*Ap)[i] = NULL;
}

int MEMBER(elementtype x, DICTIONARY A)  {
 celltype *current;
 current = A[h(x)];  /* glava x-ovog pretinca */
 while (current != NULL) {
    if (current->element == x)  return 1;
    else        current = current->next;
     } return 0; /* element nije nađen */
}
```

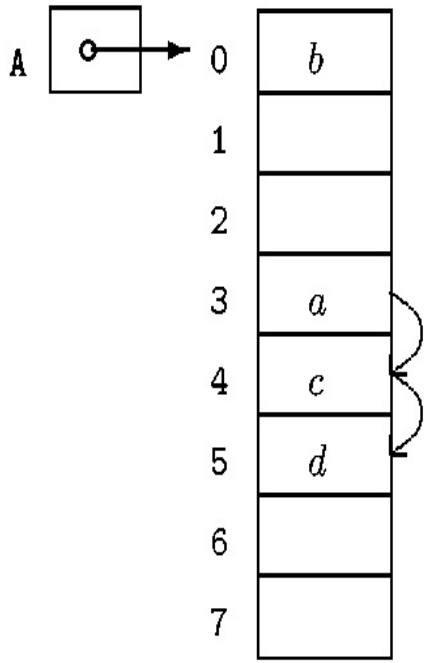
```
void INSERT(elementtype x, DICTIONARY *Ap) { /* ubacuje novi element na početak
    pretinca */
 int bucket;
 celltype *oldheader;
 if (MEMBER(x,*Ap) == 0) {
   bucket = h(x);
   oldheader = (*Ap)[bucket];
   (*Ap)[bucket] = (celltype*) malloc(sizeof(celltype));
   (*Ap)[bucket]->element = x;
   (*Ap)[bucket]->next = oldheader; }
}
```

```
void DELETE(elementtype x, DICTIONARY *Ap) {
celltype *current, *temp;
int bucket;
bucket = h(x);
if ((*Ap)[bucket] != NULL)  {
  if ((*Ap)[bucket]->element == x) {  /* x je u prvoj ćeliji */
     temp = (*Ap)[bucket];
     (*Ap)[bucket] = (*Ap)[bucket]->next;
     free(temp);
    }
```

```
  else {  /* x, ako postoji, nije u prvoj ćeliji */
     current = (*Ap)[bucket];  /* current pokazuje na prethodnu ćeliju */
     while (current->next != NULL)  {
        if (current->next->element == x) {
            temp = current->next;
            current->next = current->next->next;
            free(temp);
            return 1;
         }
           else  /* x još nije nađen  */
           current = current->next;
        }
     }
  }
}
```

Zatvorena rasuta tablica

- Pretinci imaju fiksni kapacitet
- možemo uzeti da u svaki stane jedan element
- Rasuta tablica je tada polje ćelija tipa elementtype
- Ako se ubacuje element x, a ćelija h(x) je već puna, tada se pokušava s alternativnim lokacijama hh(1,x), hh(2,x), … sve dok se ne nađe prazna ćelija
- Najčešće se hh(i,x) zadaje kao: hh(i,x) = (h(x) + 1) % B
- To je postupak linearnog rasipanja
- Primjer: B = 8, rječnik se gradi ubacivanjem elemenata a,b,c,d te funkcija h() daje redom vrijednosti 3, 0, 4, 3; koristi se linearno rasipanje
- U prazne pretince upisana posebna vrijednost EMPTY različita od svih elemenata koji se ubacuju
- Traženje x u tablici se provodi provjerom pretinaca h(x), hh(1,x), hh(2,x),… dok se ne nađe x ili EMPTY



- Ovaj postupak traženja je ispravan samo ako nema brisanja elemenata
- Da bi se omogućilo brisanje uvodi se dodatna rezervirana vrijednost DELETED, pa se element briše tako da se umjesto njega u njegov pretinac upiše DELETED
- Ta ćelija se kasnije može upotrijebiti kod novog ubacivanja elemenata
- Pseudokod za ovu izvedbu:

```
#define EMPTY …
#define DELETED …
#define b …
typedef elementtype *DICTIONARY;  /* početna adresa polja elemenata */

void MAKE_NULL(DICTIONARY *Ap) {
  int i;
  for (i = 0; i < B; i++) (*Ap)[i] = EMPTY;
}
```

- Slijedeća funkcija prolazi tablicom od pretinca h(x) nadalje, sve dok ne nađe x ili prazni pretinac ili dok se ne vrati na mjesto polaska. Funkcija vraća indeks pretinca na kojem je stala iz bilo kojeg od ovih razloga

```
int locate(elementtype x, DICTIONARY A) {
int initial, i;  /* prvi čuva h(x), drugi broji pređene pretince */
initial = h(x);
i = 0;
while ((i < B) && (A[(initial+i)%B] != x) && (A[(initial+i)%B] !=EMPTY))
      i++;
return((initial+i)%B);
}
```

Slična funkcija, ali stane i kad naiđe na DELETED:

```
int locate1(elementtype x, DICTIONARY A) {
int initial, i;  /* prvi čuva h(x), drugi broji pređene pretince */
initial = h(x);
i = 0;
while ((i < B) && (A[(initial+i)%B] != x) && (A[(initial+i)%B] != EMPTY) &&
   (A[(initial+i)%B] != DELETED))
      i++;
return((initial+i)%B);
}
```

```
int MEMBER(elementtype x, DICTIONARY A) {
 if ( A[locate(x,A)] == x) return 1;
 else return 0;
}

void INSERT(elementtype x, DICTIONARY *Ap) {
int bucket;
if (MEMBER(x,*Ap)) return 0; /* x je već u A */
bucket = locate1(x,*Ap);
if (((*Ap)[bucket] == EMPTY) || ((*Ap)[bucket] == DELETED))
     (*Ap)[bucket] = x;
else error("Tablica je puna");
}

void DELETE(elementtype x, DICTIONARY *Ap) {
int bucket;
bucket = locate(x,*Ap);
if ((*Ap)[bucket] == x)
(*Ap)[bucket] = DELETED;
}
```

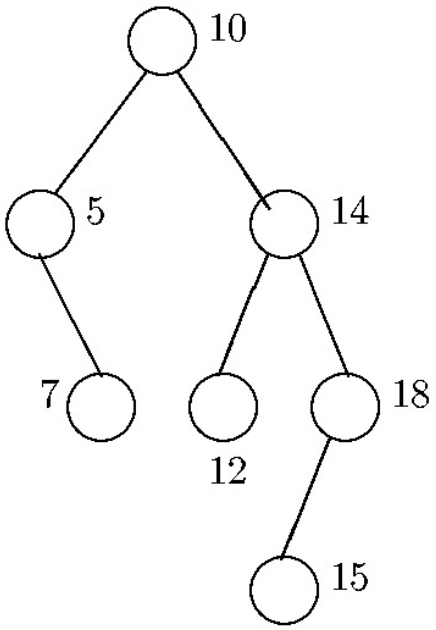
- Kod obje varijante tablice rasipanja važno je da tablica bude dobro dimenzionirana u odnosu na rječnik koji se pohranjuje
- za n elemenata u rječniku, preporučuje se za otvorenu rasutu tablicu $n \leq 2*B$, a kod zatvorene rasute tablice $n \leq 0.9*B$
- Tada se bilo koja od operacija INSERT(), DELETE(), MEMBER() može obaviti uz svega par čitanja iz tablice
- Ako se tablica previše napuni, treba je prepisati u veću

Implementacija rječnika pomoću binarnog stabla traženja

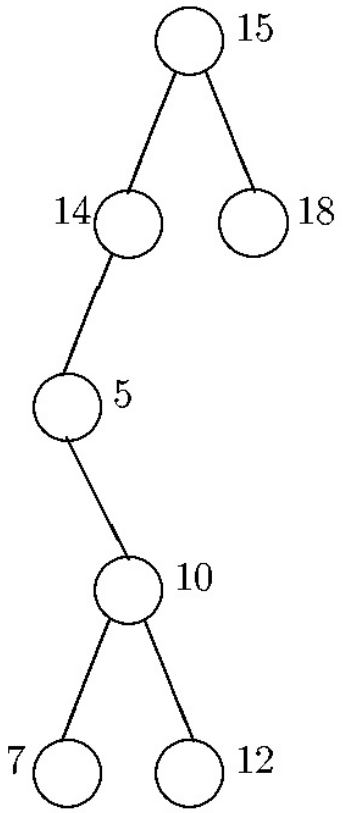
- Binarno stablo T je binarno stablo traženja ako su ispunjeni uvjeti:
  - čvorovi od T su označeni podacima nekog tipa na kojem je definiran totalni uređaj
  - neka je *i* bilo koji čvor od T. Tada su oznake svih čvorova u lijevom (desnom) podstablu od *i* manje (veće ili jednake) od oznake od *i*.
- Ideja implementacije: rječnik se prikazuje binarnim stablom traženja
- Svakom elementu rječnika odgovara točno jedan čvor stabla i obratno (bijekcija)
- Element rječnika služi kao oznaka odgovarajućeg čvora stabla: element je spremljen u određenom čvoru
- Svi čvorovi stabla će imati različite oznake, iako se to ne zahtjeva u definiciji binarnog stabla traženja (izlazi iz definicije rječnika)
- Prikaz skupa pomoću binarnog stabla traženja nije jedinstven

- Primjer: skup A = {5,7,10,12,14,15,18}, prikaz ovisi o redoslijedu upisa elemenata

npr. (10, 14, 5, 12, 18, 7, 15)



npr. (15, 14, 5, 18, 10, 12, 7)



- Obilaskom stabla algoritmom INORDER() (obilazi T_1, pa r, pa T_2, …) dobivaju se elementi skupa u sortiranom redoslijedu

- Binarno stablo se prikazuje pokazivačem.
- Definicije strukture podataka:

```
typedef struct cell_tag {
   elementtype element;
   struct cell_tag *leftchild;
   struct cell_tag *rightchild;
} celltype;
typedef celltype *DICTIONARY; /* pokazivač na korijen */
```

Funkcija MAKE_NULL() je trivijalna, pokazivaču se pridružuje vrijednost NULL

Funkcija MEMBER(), vraća 1 ako je element u rječniku, inače 0:

```
int MEMBER(elementtype x, DICTIONARY A) {
if (A == NULL) return 0;
else if (x == A->element) return 1;
else if (x < A->element) return(MEMBER(x,A->leftchild));
else return(MEMBER(x,A->rightchild));
}
```

```
void INSERT(elementtype x, DICTIONARY *Ap) {
if (A == NULL) {
 *Ap = (celltype*)malloc(sizeof(celltype));
 (*Ap)->element = x;
 (*Ap)->leftchild = (*Ap)->rightchild = NULL;
 }
 else if (x < (*Ap)->element) INSERT(x,&((*Ap)->leftchild));
 else if (x > (*Ap)->element) INSERT(x,&((*Ap)->rightchild));
/* ako je x == (*Ap)->element, ne radi ništa jer je x već u rječniku */
}
```

- Operacija DELETE(x,&A) je složenija. Tri su mogućnosti:
  1. x je u listu: jednostavno se izbaci list iz stabla
  2. x je u čvoru koji ima samo jedno dijete: nadomjesti se čvor od x njegovim djetetom
  3. x je u čvoru koji ima oba djeteta: nađe se najmanji element y u desnom podstablu čvora od x i izbaci se njegov čvor (svodi se na gornje slučajeve). U čvor od x se spremi y umjesto x i time se briše x

- Potrebna je pomoćna funkcija DELETE_MIN(&A) koja iz nepraznog binarnog stabla traženja A izbacuje čvor s najmanjim elementom, te vraća taj element

```
elementtype DELETE_MIN(DICTIONARY *Ap) {
celltype *temp;
elementtype minel;
if ((*Ap)->leftchild == NULL) {  /*    (*Ap) pokazuje najmanji element    */
   minel = (*Ap)->element;
   temp = (*Ap);
   (*Ap) = (*Ap)->rightchild;
   free(temp);
 }
 else    /* čvor kojeg pokazuje (*Ap) ima lijevo dijete */
 minel = DELETE_MIN(&((*Ap)->leftchild));
return minel;
}

void DELETE(elementtype x, DICTIONARY *Ap) {
celltype *temp;
if (*Ap != NULL)  {
  if ( x < (*Ap)->element)
    DELETE(x, &((*Ap)->leftchild));
  else if (x > (*Ap)->element)
    DELETE(x,&((*Ap)->rightchild));
```

```
 /* ako dođemo do ovdje, tada je x u čvoru kojeg pokazuje *Ap     */
 else if (((*Ap)->leftchild == NULL) && ((*Ap)->rightchild == NULL))  {
       /* izbaci list koji sadrži x */
       free (*Ap);
       *Ap = NULL;
   }
   else if ((*Ap)->leftchild == NULL) {
      /* nadomjesti se čvor od x s njegovim desnim djetetom */
    temp = *Ap;
    *Ap = (*Ap)->rightchild;
    free(temp);
    }
   else if ((*Ap)->rightchild == NULL) {
      /*   nadomjesti se čvor od x s njegovim lijevim djetetom */
     temp = *Ap;
     *Ap = (*Ap)->leftchild;
     free(temp);
   }
  else  /* postoje oba djeteta */
   (*Ap)->element = DELETE_MIN(&((*Ap)->rightchild));
 }
}
```

- Funkcije MEMBER(), INSERT(), DELETE() prolaze jednim putem od korijena do traženog čvora, pa je vrijeme njihovog izvršavanja ograničeno visinom stabla
- Ako rječnik ima n elemenata, visina stabla može biti između $(\log_2(n+1))-1$ (za potpuno binarno stablo) i n-1 (koso stablo)
- Vrijeme izvršavanja ovih funkcija varira od O(log n) do O(n)
- Može se pokazati da za slučaj izgradnje binarnog stabla traženja od praznog stabla n-strukom primjenom operacije INSERT(), uz uvjet da su svi redoslijedi ubacivanja elemenata jednako vjerojatni, očekivano vrijeme izvršavanja za operacije INSERT(), DELETE(), MEMBER() je proporcionalno s log n.
- Opisana implementacija je jedna iz porodice sličnih gdje se rječnik prikazuje pomoću neke strukture zasnovane na stablu
- Postoje stabla kod kojih se automatski stablo izgrađuje tako da mu visina bude što manja čime se omogućava efikasno obavljanje osnovnih operacija, ali je programski kod koji to omogućuje znatno kompliciraniji, jer je potrebno stablo pregraditi ako se ono isuviše iskosi

Prioritetni red

- Neki problemi zahtijevaju algoritme koji rade sa skupovima čijim su elementima pridruženi cijeli ili realni brojevi – prioriteti
- Na takvim skupovima se definiraju operacije ubacivanja novog elementa te izbacivanje elementa s najmanjim prioritetom
- Takav skup se naziva prioritetni red
- Primjeri: red čekanja pacijenata kod liječnika na hitnoj gdje najteži bolesnik ima prednost ili stvaranje prioritetnog reda procesa koji čekaju na izvršavanje u računalu
- Ovaj problem se svodi na jednostavniji problem ubacivanja elemenata u skup, te izbacivanje najmanjeg elementa
- Promatraju se uređeni parovi (prioritet(x),x)
- za njih se definira leksikografski uređaj: (prioritet(x_1),x_1) je manji ili jednak od (prioritet(x_2),x_2) ako je (prioritet(x_1) manji od prioritet(x_2)) ili (prioritet(x_1) jednak prioritet(x_2) i x_1 manji ili jednak od x_2)
- Time se omogućuje sljedeća definicija apstraktnog tipa podataka:

Apstraktni tip podatak PRIORITY_QUEUE

- elementtype … bilo koji tip s totalnim uređajem ≤
- PRIORITY_QUEUE … podatak ovog tipa je konačni skup čiji elementi su međusobno različiti podaci tipa elementtype
- MAKE_NULL(&A) … funkcija pretvara skup A u prazni skup
- EMPTY(A) … funkcija vraća istinu ako je A prazan skup, inače vraća laž
- INSERT(x,&A) … funkcija ubacuje element x u skup A, tj. mijenja A u A unija x
- DELETE_MIN(&A) … funkcija iz skupa A izbacuje najmanji element, te vraća taj izbačeni element. Nedefinirana ako je A prazan skup

- ATP PRIORITY_QUEUE se može smatrati restrikcijom ATP SET
- Funkcija DELETE_MIN() je kombinacija funkcija MIN() i DELETE()
- Prioritetni red se može izvesti pomoću **sortirane vezane liste**:

  funkcija DELETE_MIN() izbacuje prvi element u listi pa ima konstantno vrijeme izvođenja

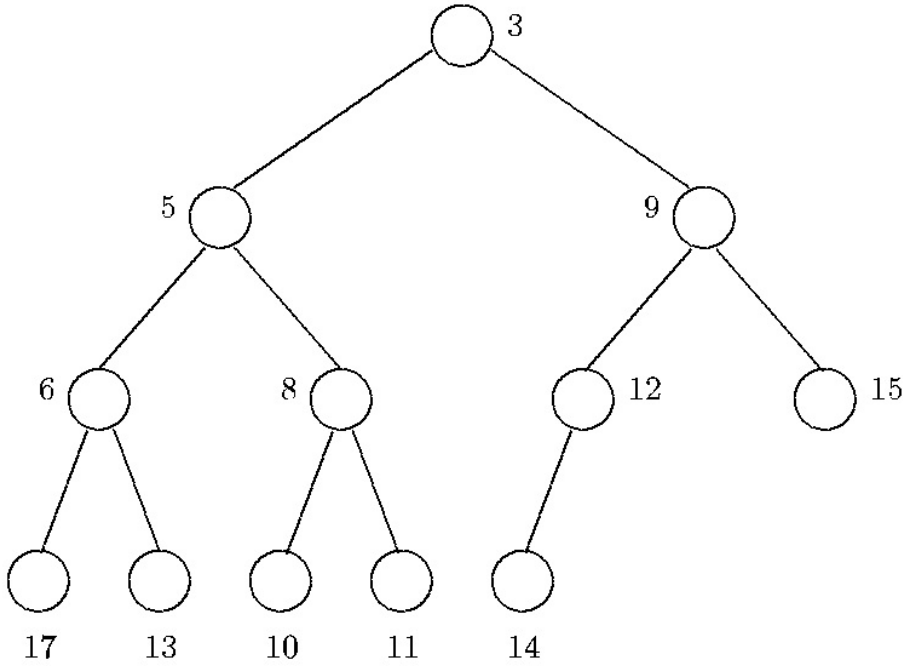
  funkcija INSERT() mora ubaciti novi element na pravo mjesto što znači da vrijeme izvršavanja ovisi o broju elemenata u redu
- Implementacija prioritetnog reda pomoću **binarnog stabla traženja** je ista kao za rječnik, mogu se upotrijebiti isti potprogrami INSERT(), DELETE_MIN(), MAKE_NULL()

Implementacija prioritetnog reda pomoću hrpe

- Potpuno binarno stablo T je hrpa (heap) ako su ispunjeni uvjeti:
  - čvorovi od T su označeni podacima nekog tipa za koje je definiran totalni uređaj
  - neka je *i* bilo koji čvor od T. Tada je oznaka od *i* manja ili jednaka od oznake bilo kojeg djeteta od *i*

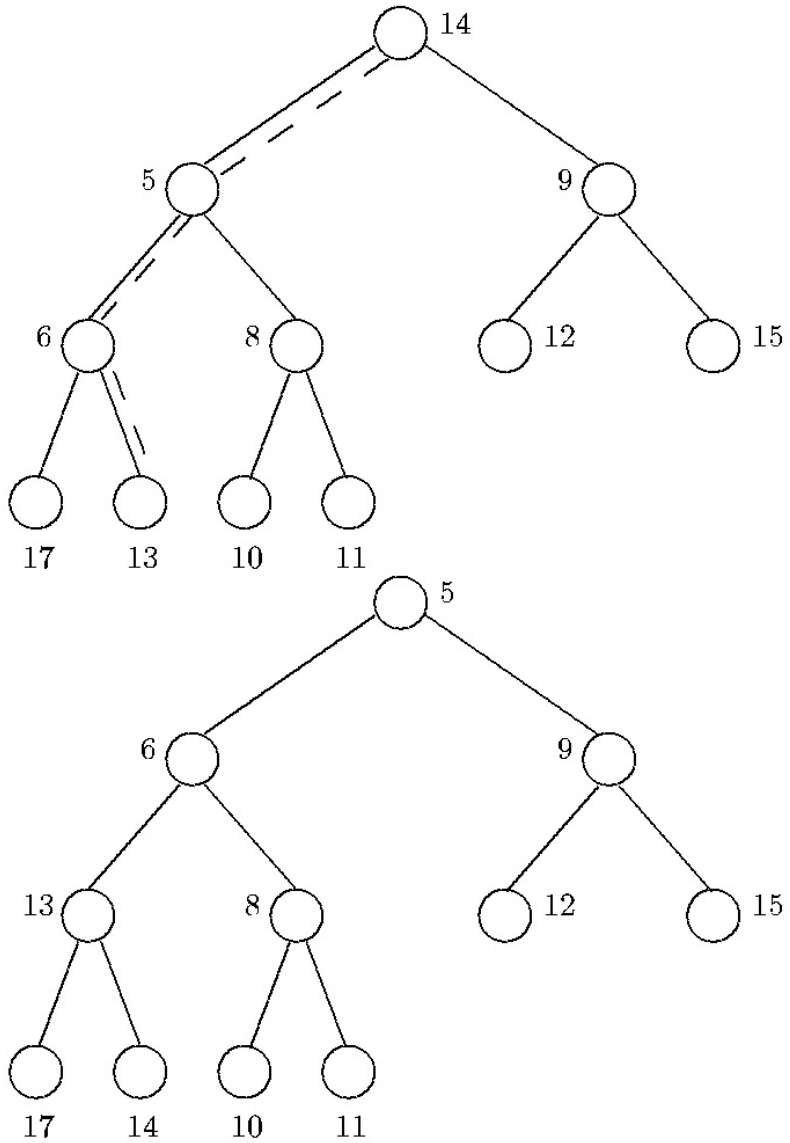
- Prioritetni red se prikazuje hrpom tako da svakom elementu reda odgovara točno jedan čvor hrpe i obratno, element reda služi kao oznaka odgovarajućeg čvora hrpe, tj. element je spremljen u čvoru
- Čvorovi ove hrpe imaju svi različite oznake, iako se to ne zahtijeva u definiciji hrpe
- Prikaz prioritetnog reda pomoću hrpe nije jedinstven
- Iz svojstava hrpe slijedi da je najmanji element u korijenu
- Ponovimo: hrpa je potpuno binarno stablo

- Primjer: prioritetni red A = {3,5,6,8,9,10,11,12,13,14,15,17}



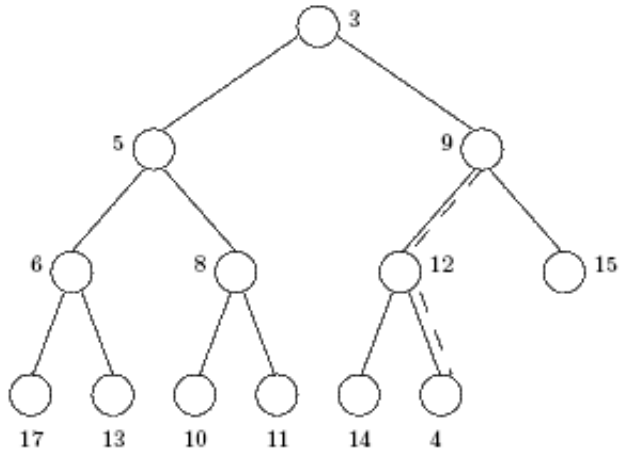
- Operacija DELETE_MIN(): korijen se ne može jednostavno izbaciti jer bi se stablo raspalo. Postupak: izbacuje se zadnji čvor na zadnjem nivou, a njegov element se stavi u korijen, što kvari svojstvo hrpe. Zamijene se element u korijenu i manji element u korijenovom djetetu, zatim se zamijene element u djetetu i manji element u djetetovom djetetu itd najdalje do nekog lista

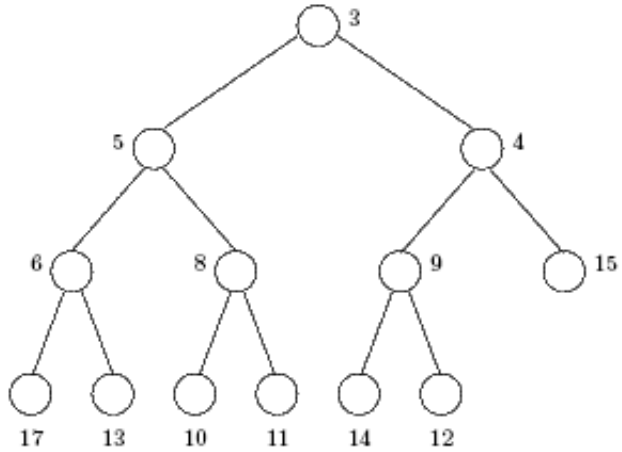
- Djelovanje DELETE_MIN() na hrpu iz primjera:



- Obavljanje operacije INSERT(): napravi se novi čvor na prvom slobodnom mjestu zadnjeg nivoa i stavi se novi element u taj novi čvor. Time se kvari svojstvo hrpe. Popravak: zamijene se element u roditelju i novi element (ako je novi element manji), pa element i roditelj roditelja itd sve do korijena ako je potrebno

Primjer za ubacivanje elementa 4 u hrpu

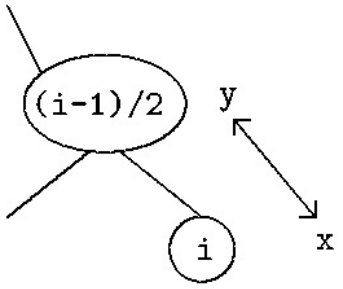




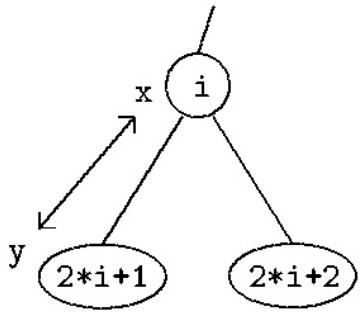
Struktura podataka za prikaz hrpe u prikazu pomoću polja:

```
#define MAXSIZE …
typedef struct {
   elementtype elements[MAXSIZE];
   int last; } PRIORITY_QUEUE;
```

```
void INSERT(elementtype x, PRIORITY_QUEUE *Ap) {
int i;
elementtype temp;
if (Ap->last >= MAXSIZE-1)  error("Prioritetni red je pun");
else  {
   (Ap->last)++;
   Ap->elements[Ap->last] = x;  /* novi čvor s elementom x  */
   i = Ap->last  /* i je indeks čvora u kojem je x  */
   while ((i > 0) && (Ap->elements[i] < Ap->elements[(i-1)/2])) {
         temp = Ap->elements[i];
         Ap->elements[i] = Ap->elements[(i-1)/2];
         Ap->elements[(i-1)/2] = temp;
         i = (i-1)/2; }}}
```



```
elementtype DELETE_MIN(PRIORITY_QUEUE *Ap) {
 int i,j;
 elementtype minel, temp;
 if (Ap->last < 0) error("Prioritetni red je prazan");
 else  { minel = Ap->elements[0]  /* najmanji element je u korijenu  */
   Ap->elements[0] = Ap->elements[Ap->last];  /* zadnji čvor ide u korıjen  */
   (Ap->last)--;  /* izbacuje se zadnji čvor  */
   i = 0;  /* i je indeks čvora u kojem se nalazi element iz izbačenog čvora  */
   while (i <= (Ap->last-1)/2) {   /* pomiče se element u čvoru i prema dolje  */
     if ((2*i+1 == Ap->last) || (Ap->elements[2*i+1] < Ap->elements[2*i+2]))
         j = 2*i+1;
     else
         j = 2*i+2;   /* j je dijete koji sadrži manji element  */
     if (Ap->elements[i] > Ap->elements[j])  {   /* zamijeni elemente iz čvorova i,j  */
          temp = Ap->elements[i];
          Ap->elements[i] = Ap->elements[j];
          Ap->elements[j] = temp;
          i = j; } else return(minel); /* nije potrebno dalje pomicanje */
          } return (minel); /* došli smo do lista */
}}
```



- Funkcije MAKE_NULL() i EMPTY() su trivijalne (izjednačavanje Ap->last s 0 i provjera da li je Ap->last 0)
- Prikazana funkcija INSERT() ne provjerava da li je element x već upisan u neki čvor, pa je potrebno prije njenog pozivanja provjeriti da li je x već u prioritetnom redu ili preurediti INSERT() tako da vrši provjeru
- Vrijeme izvršavanja operacija INSERT() i DELETE_MIN() koje obilaze jedan put u potpunom binarnom stablu je u najgorem slučaju proporcionalno s log n gdje je n broj čvorova u stablu
- Kod implementacije pomoću binarnog stabla traženja je prosječno vrijeme izvršavanja proporcionalno s log n
- Izvedba s hrpom je dakle brža i efikasnija
- Ali izvedba s binarnim stablom traženja omogućuje efikasnu izvedbu dodatnih operacija MEMBER(), DELETE(), MIN() i MAX() što nije slučaj za izvedbu s hrpom

Preslikavanje i relacija

- Često je potrebno pamtiti pridruživanja među podacima koja se mogu opisati matematičkim pojmom funkcije (preslikavanja)
- Definicija: preslikavanje M je skup uređenih parova oblika (d,r) gdje su svi d-ovi podaci jednog tipa, a svi r-ovi podaci drugog tipa. Pritom, za zadani d u M postoji najviše jedan par (d,r).
- Prvi tip podataka se naziva domena ili područje definicije
- Drugi tip podataka je kodomena ili područje vrijednosti
- Ako za zadani d preslikavanje M sadrži par (d,r), tada taj jedinstveni r označavamo s r = M(d) i kažemo da je M(d) definirano. Ako M ne sadrži (d,r) onda M(d) nije definirano.
- Primjeri:

telefonski imenik je preslikavanje čiju domenu čine imena ljudi a kodomenu telefonski brojevi (slično: rječnici, indeks pojmova …)

evidencija koju o osobama vodi državna administracija (MUP …) skup zapisa o osobi koji se razlikuju na osnovu JMBG: domena JMBG, kodomena ostali podaci

matematički pojam vektor duljine n može se shvatiti kao preslikavanje iz domene {1,2,..,N} u kodomenu koja je skup realnih brojeva (slično: matrica …)

Apstraktni tip podataka MAPPING

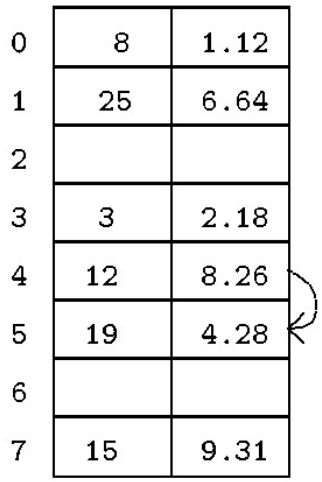
- domain … bilo koji tip (domena)
- range … bilo koji tip (kodomena)
- MAPPING … podatak ovog tipa je preslikavanje čiju domenu čine podaci tipa domain, a kodomenu podaci tipa range
- MAKE_NULL(&M) … funkcija pretvara preslikavanje M u nul-preslikavanje tj. takvo koje nije nigdje definirano
- ASSIGN(&M,d,r) … funkcija definira M(d) tako da bude M(d) jednako r, bez obzira da li je M(d) prije bilo definirano ili nije
- DEASSIGN(&M,d) … funkcija uzrokuje da M(d) postane nedefinirano, bez obzira da li je M(d) bilo prije definirano ili nije
- COMPUTE(M,d,&r) … ako je M(D) definirano, tada funkcija vraća istinu i pridružuje varijabli r vrijednost M(d), inače vraća laž

- Za implementaciju preslikavanja se koriste strukture podataka polje, lista, rasuta tablica, binarno stablo traženja
- Preslikavanje M se pohranjuje kao skup uređenih parova oblika (d,M(d)) gdje d prolazi svim podacima za koje je M(d) definirano
- Mjesto uređenog para (d,M(d)) u strukturi se određuje samo na osnovu d, tj. onako kao kad bi se samo on pohranjivao
- moguće pronaći (d,M(d)) na osnovu zadanog d
- Operacije MAKE NULL( ), ASSIGN( ), DEASSIGN( ), COMPUTE( ) iz ATP MAPPING implementiraju se analogno kao MAKE NULL( ), INSERT( ), DELETE( ), MEMBER( ) iz ATP DICTIONARY

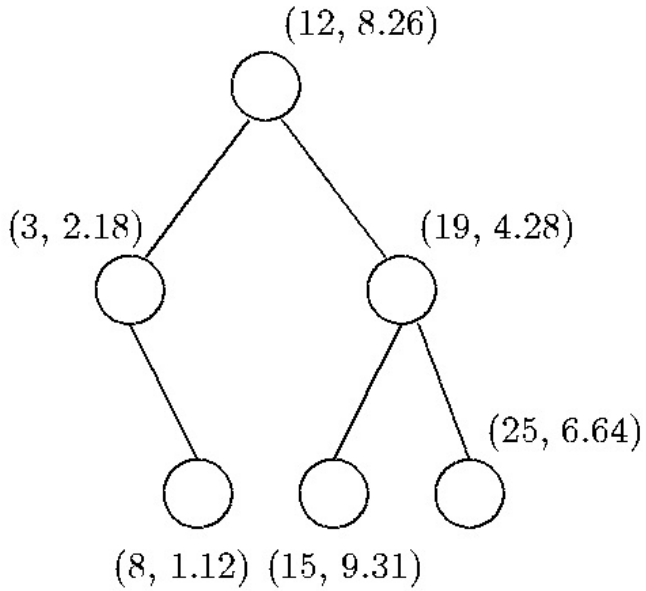
Implementacija preslikavanja pomoću rasute tablice ili binarnog stabla

- Primjer: preslikavanje M čija je domena tip int, a kodomena tip float. M je zadana tablicom. Za cijele brojeve koji se ne pojavljuju u tablici M(d) nije definirana. M se shvaća kao skup M = {(3,2.18), (8,1.12), … , (25,6.64) }
- Taj skup se shvaća kao rječnik i prikazuje na načine opisane za rječnik
- Npr. može se koristiti prikaz pomoću zatvorene rasute tablice s B pretinaca, gdje se u svaki pretinac stavi jedan uređeni par
- Funkcija rasipanja h() preslikava područje definicije od M na skup {0, 1, 2, …, B-1}
- Uređeni par (d,M(d)) se sprema u pretinac h(d) i kasnije se traži u njemu
- Možemo uzeti B = 8 i h(x) = x % 8
- U slučaju da je pretinac za x već popunjen, primjenjuje se linearno rasipanje

d	M(d)
3	2.18
8	1.12
12	8.26
15	9.31
19	4.28
25	6.64



- Skup M se može također prikazati pomoću binarnog stabla traženja gdje se uređeni parovi (d,M(d)) smještaju u čvorove stabla
- grananje u stablu se provodi samo na osnovu d: vrijedi uređaj za oznake čvorova (d_1,M(d_1)) manje ili jednako (d_2,M(d_2)) ako i samo ako je d_1 manji ili jednak d_2



- Često se radi s pridruživanjima među podacima koja su općenitija od onih obuhvaćenih pojmom preslikavanja. To su **relacije.**
- Definicija: **binarna relacija** R je skup uređenih parova oblika (d_1,d_2), gdje se kao d_1 pojavljuju podaci jednog tipa, a kao d_2 podaci drugog tipa. Ova dva tipa se nazivaju prva i druga domena. Ako relacija R za zadani d_1 i d_2 sadrži uređeni par (d_1,d_2), tada se kaže da je d_1 u relaciji R s d_2 i piše se d_1 R d_2. U protivnom d_1 nije u relaciji s d_2.
- Primjeri: studenti upisuju izborne kolegije, čime se uspostavlja relacija čije domene su skup studenata i skup kolegija. Može se odrediti koje sve kolegije je upisao zadani student ili koji sve studenti su upisali zadani kolegij
- slični primjeri su ljudi i časopisi, filmovi i kina, odredišta i avionske kompanije
- Proizvodi se sastavljaju od dijelova, može se tražiti koji su sve dijelovi potrebni za zadani proizvod ili u kojim se sve proizvodima pojavljuje zadani dio
- slično tome: programi su sastavljeni od potprograma, jelo se sastoji od namirnica itd

Apstraktni tip podataka RELATION

- domain1 … bilo koji tip (prva domena)
- domain2 … bilo koji tip (druga domena)
- set1 … podatak tipa set1 je konačan skup podataka tipa domain1
- set2 … podatak tipa set2 je konačan skup podataka tipa domain2
- RELATION … podatak ovog tipa je binarna relacija čiju prvu domenu čine podaci tipa domain1, a drugu domenu podaci tipa domain2
- MAKE_NULL(&R) … funkcija pretvara relaciju R u nul-relaciju, tj. takvu u kojoj ni jedan podatak iz set1 nije u relaciji ni s jednim podatkom iz set2
- RELATE(&R,d1,d2) … funkcija postavlja da je d1 R d2 bez obzira da li je to već bilo postavljeno ili nije
- UNRELATE(&R,d1,d2) … funkcija postavlja da d1 nije u relaciji R s d2, bez obzira da li je to već bilo postavljeno ili nije
- COMPUTE2(R,d1,&S2) … za zadani d1 funkcija skupu S2 pridružuje kao vrijednost {d2 | d1 R d2}
- COMPUTE1(R,&S1,d2) … za zadani d2 funkcija skupu S1 pridružuje kao vrijednost {d1 | d1 R d2}

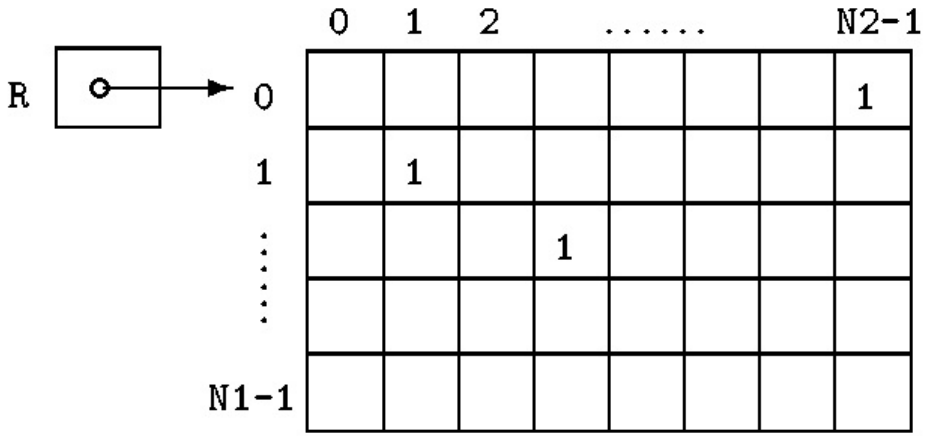
- Za implementaciju relacije se upotrebljavaju iste strukture kao za rječnik
- Relacija R se pohranjuje kao skup uređenih parova oblika (d_1,d_2) takvih da je d_1 R d_2.
- Operacije MAKE_NULL(), RELATE(), UNRELATE() iz ATP RELATION se implementiraju analogno kao MAKE_NULL(), INSERT(), DELETE() iz ATP DICTIONARY
- Za efikasno obavljanje operacija COMPUTE1() i COMPUTE2() potrebne su promjene i nadopune takve strukture

Implementacija binarne relacije pomoću bit-matrice

- Nastaje na osnovu izvedbe skupa pomoću bit-vektora
- Ovdje se 1D polje (vektor) presloži u 2D (matricu), uzima se domain1 = {0,1, … ,N1-1} i domain2 = {0,1, …,N2-1} gdje su N1, N2 dovoljno velike konstante
- Relacija se prikazuje 2D poljem bitova ili byte-ova ili znakova (char)

```
#define N1 …
#define N2 …
typedef char[N1][N2] *RELATION    /* početna adresa char polja veličine N1 x N2 */
```

- (*i*,*j*)-ti bit je 1 ako je *i*-ti podatak u relaciji s *j*-tim
- Time se operacije COMPUTE1(), COMPUTE2() svode na čitanje retka i stupca polja



Izvedba binarne relacije pomoću multi-liste

- Sliči na izvedbu skupa pomoću vezane liste, no umjesto jedne vezane liste sa svim uređenim parovima (d_1,d_2) koristi se mnogo malih listi koje odgovaraju rezultatima operacija COMPUTE2() i COMPUTE1()
- Jedan uređen par (d_1,d_2) se prikazuje zapisom:

```
typedef struct cell_tag {
domain1 element1;
domain2 element2;
struct cell_tag *next1;
struct cell_tag *next2;
} celltype;
```

- Jedan zapis je istovremeno uključen u vezanu listu prve vrste koja povezuje sve zapise s istim d_1 i vezanu listu druge vrste koja povezuje sve zapise s istim d_2
- Za povezivanje listi prve (druge) vrste služe pokazivači next1 (next2)
- Liste mogu biti sortirane i nesortirane
- Pokazivač na početak liste prve (druge) vrste zadaje se preslikavanjem čija domena je domain1 (domain2) a kodomena pokazivači na celltype
- Ova dva preslikavanja se mogu izvesti na razne načine (kao za skup i rječnik)

- Operacije COMPUTE2() i COMPUTE1() svode se na pronalaženje pokazivača za zadani $d_1$ili d_2 i prolazak odgovarajućom vezanom listom prve tj. druge vrste
- Primjer: relacija R = {(0,1), (0,3), (1,2), (1,3), (2,1), (2,3), (2,4)}