Binarna relacija (Relation)

Binarna relacija R je skup uređenih parova (d_1, d_2) gdje se kao d_1 pojavljuju podaci jednog tipa, a kao d_2 podaci drugog tipa. Ako u R postoji par (d_1, d_2), kažemo da je d_1 u relaciji R s d_2 i pišemo $d_1 \, R \, d_2$.

a.t.p. Relation

```
domain1                 . . .
domain2                 . . .
set1                    . . .
set2                    . . .
Relation                . . .
ReMakeNull(&R)          . . .
ReRelate(&R,d1,d2)      . . .
ReUnrelate(&R,d1,d2)    . . .
ReCompute2(R,d1,&S2)    . . .
ReCompute1(R,&S1,d2)    . . .
```

Pogodna implementacija binarne relacije je implementacija pomoću multi-liste. Jedan uređeni par (d_1, d_2) spremljen je u element (zapis) koji je istodobno uključen u dvije vezane liste:

(i) lista prve vrste: povezuje sve zapise s istom prvom komponentom d_1

(ii) lista druge vrste: povezuje sve zapise s istom drugom komponentom d_2

Pokazivač na početak liste prve (odn. druge) vrste zadaje se preslikavanjem čija domena je `domain1` (odn. `domain2`), a kodomenu čine pokazivači. Ta preslikavanja se mogu implementirati na razne načine. Rezultat operacija `Compute1` i `Compute2` je skup koji se također može implementirati na razne načine.

Primjer. $R = \{(1, a), (2, c), (2, b), (3, a), (3, b)\}$

Zadatak 5.

Razradite implementaciju relacije pomoću multi-liste. Zbog jednostavnosti uzmite da su prva i druga domena jedan isti tip `domain`. Implementacija relacije treba biti neovisna o implementaciji preslikavanja i skupova.

Rješenje.

```
typedef struct cell {
    domain element1, element2;
    struct cell *next1, *next2;
} celltype;

typedef ... Set;    //  neka pogodna definicija za skup podataka tipa domain
typedef ... Mapping;   //  neka pogodna definicija za preslikavanje domain -> celltype*

typedef struct {
    Mapping list1;  //  preslikava domain1 u celltype*
    Mapping list2;  //  preslikava domain2 u celltype*
} Relation;

void ReCompute2(Relation R, domain d1, Set* S2) {
    celltype* current;
    int exists;
    SeMakeNull(S2);  // operacija iz atp Set
    exists = MaCompute(R.list1, d1, &current); // operacija iz atp Mapping
    if (exists)
        while (current != NULL) {
            SeInsert(current->element2, S2);
            current = current->next;
        }
}
```

Zadatak 6.

Na nekom fakultetu studenti upisuju kolegije. U tablici je prikazano koji student je upisao koji kolegij. Nacrtajte dijagram na kojem se vidi prikaz te binarne relacije pomoću: (a) bit-matrice, (b) multi-liste.

	CS101	CS202	CS303
Alan			x
Alex	x	x	
Alice			x
Amy	x		
Andy		x	x
Ann	x		x

Rješenje.

Napomena. U prethodnoj strukturi (multi-listi) postoji puno redundancije jer se imena studenata i kolegija upisuju u razne zapise. Ekonomičnija varijanta je pomoću tzv. prstenova.

Zadatak 7.

Binarna relacija je skup uređenih parova, a za prikaz skupa može se koristiti hash-tablica. Razradite implementaciju binarne relacije pomoću hash-tablice.

Rješenje. Odaberemo otvorenu hash-tablicu s B pretinaca, gdje je B oblika $B = 2^r$. Dakle, hash-adrese su nizovi od točno r bitova. Rastavimo $r = r_1 + r_2$. Biramo hash-funkciju oblika:

$$h(d_1, d_2) = h_1(d_1) \cdot 2^{r_2} + h_2(d_2),$$

gdje su h_1 i h_2 hash-funkcije oblika:

$$h_1 \colon domain_1 \to \{0, 1, \ldots, 2^{r_1} - 1\}$$

$$h_2 \colon domain_2 \to \{0, 1, \ldots, 2^{r_2} - 1\}$$

Tako hash-adresa uređenog para (d_1, d_2) ovisi i o d_1 i o d_2. Pritom gornjih r_1 bitova te adrese ovisi od d_1, a donjih r_2 bitova ovisi o d_2.

Operacije `MakeNull`, `Relate` i `Unrelate` obavljaju se na očigledan način, tj. isto kao `MakeNull`, `Insert` i `Delete` kod implementacije rječnika pomoću otvorene hash-tablice. Operacija `Compute2` zahtijeva da za zadani d_1 u tablici pronađemo sve parove (d_1, d_2). Budući da je d_1 zadan, možemo pomoću funkcije h_1 odrediti gornjih r_1 bitova u hash adresi za takve parove. Donjih r_2 bitova ostaje nepoznato, dakle, moramo sekvencijalno pretražiti 2^{r_2} pretinaca, tj. $(1/2^{r_1})$-ti dio tablice. Analogno, operacija `Compute1` zahtijeva da pretražimo 2^{r_1} pretinaca, tj. $(1/2^{r_2})$-ti dio tablice.

Konkretno, promatrajmo relaciju između studenata i kolegija iz prošlog zadatka. Za implementaciju pomoću hash-tablice odaberemo $B = 32 = 2^5$. Rastavimo $5 = 3 + 2$, dakle imamo $2^3 = 8$, $2^2 = 4$. Zadajemo hash-funkcije:

$$h_1(\text{ime studenta}) = (\text{redni broj u abecedi zadnjeg slova u imenu})\%8$$

$$h_2(\text{oznaka kolegija}) = (\text{zadnja znamenka u oznaci})\%4$$

$$h(\text{ime studenta}, \text{oznaka kolegija}) = h_1(\text{ime studenta}) \cdot 4 + h_2(\text{oznaka kolegija})$$

Dakle, hash-adrese su binarni brojevi duljine 5 bitova. Pritom ime studenta određuje prva 3 bita, a oznaka kolegija zadnja 2 bita.

Naprimjer, $h(\text{Alice}, \text{CS303}) = h_1(\text{Alice}) \cdot 4 + h_2(\text{CS303}) = 5 \cdot 4 + 3 = 23$. Taj par sprema se u 23. pretinac. Ako bismo htjeli odrediti koje sve kolegije je upisala Alice, pretraživali bismo pretince s adresama $h_1(\text{Alice}) \cdot 4 + i$ za $i = 0, 1, 2, 3$, dakle, pretince 20, 21, 22 i 23, što je 1/8 tablice. Ako bismo htjeli odrediti koji sve studenti su upisali kolegij CS303, pretraživali bismo pretince s adresama $i \cdot 4 + h_2(\text{CS303})$ za $i = 0$, 1,..., 7, dakle pretince 3, 7, 11, 15, 19, 23, 27 i 31, što je 1/4 tablice.

Ovako definirana hash-funkcija zove se *podijeljena (distribuirana) hash-funkcija.*