Strukture podataka - tipovi podataka

dr Davorka R. Jandrlić dr Goran Lazović

Mašinski fakultet, Univerziteta u Beogradu

Tipovi podataka

Tip podataka

Tip podataka predstavlja skup vrednosti koje neki podatak može uzeti. (Npr. podatak tipa int može imati samo vrednosti iz skupa celih brojeva. Podatak tipa boolean može imati samo vrednost true ili false).

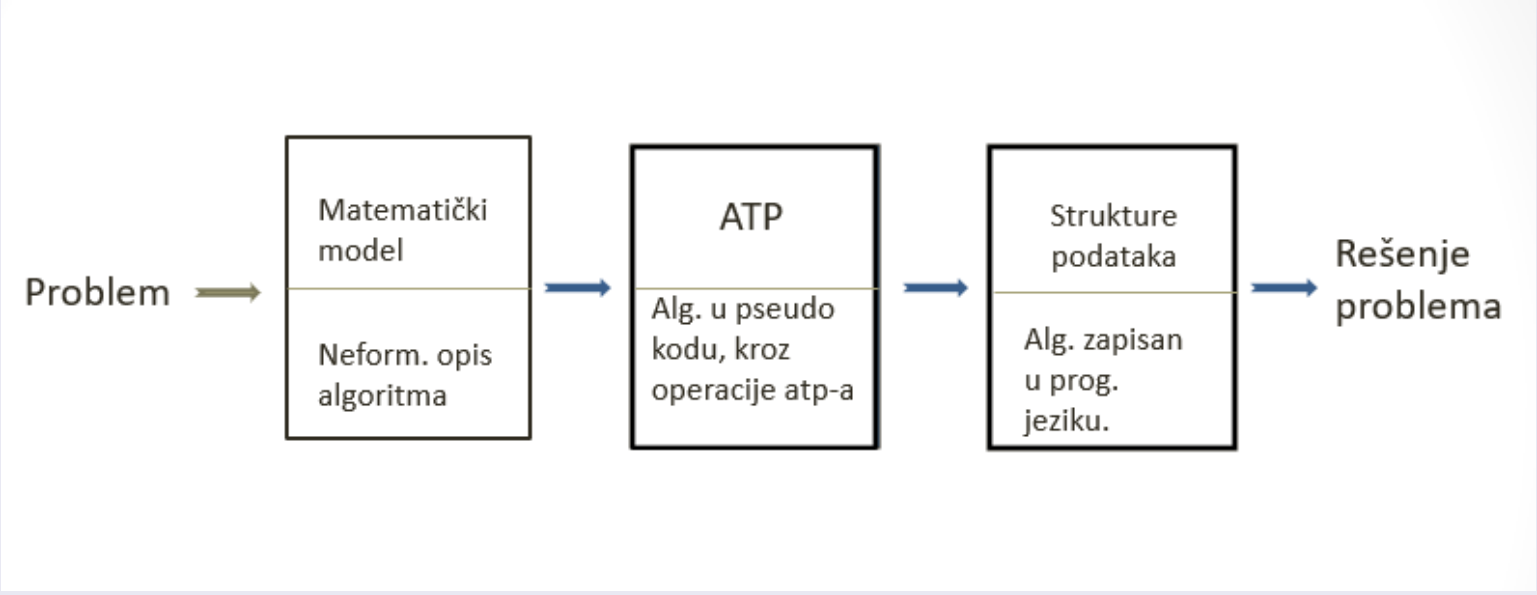
Apstraktni tip podatka

Tip podataka, zajedno sa operacijama nad njim predstavlja apstraktni tip podataka (ATP). Npr. skup celih brojeva zajedno sa operacijama unije, preseka i razlike predstavlja jedan ATP. U ATPu operacije mogu kao operande da imaju instance tog tipa za koji su definisane, kao i instance drugih definisanih tipova.

Struktura podataka predstavlja način organizacije i skladištenja podataka u računaru. Različite strukture podataka pogodne su za različite tipove problema. (Cilj: efikasan pristup podacima i njihova modifikacija).

Implementacija određenog apstraktnog tipa podataka predstavlja konkretnu realizaciju tog tipa u nekom programu. Realizacija se sastoji od definicije strukture podataka kojom se prikazuju podaci ATPa, kao i implementaciji procedura (funkcija) kojima se predstavljaju operacije. Za isti apstraktni tip podataka obično postoji više implementacija koje se razlikuju u strukturi podataka kojima se tip predstavlja i/ili algoritmima kojima se realizuju operacije.

Razvoj programa



ATP kompleksan broj

scalar – bilo koji tip za koji su definisane operacije sabiranja i množenja
complex – uređen par podataka tipa scalar.
Operacije nad kompleksnim brojevima:

ADD (c1, c2), gde su c1 = (x1, y1), c2= (x2, y2), kao rezultat daje complex z = (x, y), takav da je x = x1 + x2, y = y1 + y2.

MULT (c1, c2), gde su c1 = (x1, y1), c2= (x2, y2), kao rezultat daje complex z = (x, y), takav da je x = x1 * x2 - y1 * y2, y = x1 * y2 + x2 * y1.

Implementacija u prog. jeziku C

```
   typedef struct _complex {
    double re;
    double im;
} complex;

complex add(complex c1, complex c2) {
    complex rez = {c1.re + c2.re, c1.im + c2.im};
    return rez;
}
complex mult(complex c1, complex c2) {
    complex rez = {c1.re * c2.re - c1.im * c2.im, c1.re *
        c2.im +
                   c1.im * c2.re};
    return rez;
}
```

Elementi od kojih se grade strukture podataka

Struktura podataka predstavlja kolekciju promenljivih povezanih na različite načine. Elemente od kojih se grade strukture podataka predstavljaju: ćelije, nizovi, slogovi, pokazivači i kursori.

- **Ćelija** je osnovni element struktura podataka. Svaka ćelija ima svoj tip i adresu.
- **Niz** predstavlja sekvencu ćelija ISTOG tipa, čije su adrese zadate sekvencijalno, jedna za drugom. Broj ćelija je unapred zadat i nepromenljiv. Svaka ćelija niza određena je svojim indeksom. U prog. jeziku C indeksiranje počinje od 0 pa do 1, 2, ..., n – 1, gde je n dužina niza.

- **Slog** (structure u C-u) je kolekcija ćelija (tzv. polja), koje mogu biti različitog tipa. Broj, redosled i tip ćelija je unapred zadat i ne može se promeniti.
- **Pokazivač** je ćelija čija vrednost predstavlja adresu neke druge ćelije. (pokazuje na neku drugu ćeliju). Služi za uspostavljanje veza između delova strukture.
- **Kursor** je ćelija celobrojnog tipa koja pokazuje na element niza. Sadržaj kursora je indeks elementa u nizu.

Lista

Matematički, lista je sekvenca od nula ili više elemenata istog tipa. Podaci koji čine listu nazivaju se elementi liste. Lista se predstavlja na sledeći način:

$$(a_1, a_2, a_3, ..., a_n)$$

gde je n dužina liste. Za listu se kaže da je prazna kada je $n = 0$. Pod pretpostavkom da je $n \geqslant 0$, element a_1 je prvi element liste, dok je element a_n poslednji.

Elementi liste su linearno uređeni u odnosu na svoju poziciju. Za element a_i se kaže da je prethodnik elementa a_{i+1}, a takođe da je i sledbenik elementa a_{i-1}.

Broj elemenata u listi nije fiksan: elementi se mogu umetati u listu, i iz nje uklanjati. U nekim programskim jezicima lista je osnovni objekat od kojeg se grade svi ostali (npr. LISP – List Processing).

Da bi se matematički pojam liste pretvorio u ATP, potrebno je definisati skup operacija nad listom (takvih skupova može biti više).

Apstraktni tip podataka Lista

Pojmovi:

- tipelementa – bilo koji tip
- L – lista elemenata tipa tipelementa
- x – objekat tipa tipelementa
- p – tipa pozicija.

Tip pozicija može da varira u zavisnosti od implementacije liste. Iako se neformalno misli na celobrojnu poziciju, pozicija može imati i drugačiju reprezentaciju.

END(L) – funkcija koja vraća poziciju kraja liste (tačnije za listu dužine n, funkcija vraća poziciju koja sledi iza n-tog člana liste).

umetanje

INSERT(x, p, L) umeće x na poziciju p u listi L, pomerajući trenutni element na poziciji p jedno mesto udesno. Lista oblika (a1, a2, a3, ..., an) nakon umetanja postaje (a1, a2, ..., ap-1, x, ap, ..., an). Ako je p == END(L) tada L postaje (a1, a2, a3, ..., an, x). Ako u L ne postoji pozicija p tada je rezultat nedefinisan.

brisanje

DELETE(p, L) briše element iz liste L na poziciji p. Lista oblika (a1, a2, a3, ..., an) nakon brisanja postaje (a1, a2, ..., ap-1, ap+1, ..., an). Ako u L ne postoji pozicija p ili je p == END(L) rezultat je nedefinisan.

LOCATE(x, L) vraća poziciju elementa x u listi L. Ako se x pojavljuje više puta kao rezultat se vraća pozicija prvog pojavljivanja. U slučaju da se x ne nalazi u listi rezultat je pozicija END(L)

RETRIEVE(p, L) vraća element liste L na poziciji p. Rezultat je nedefinisan ako je p == END(L) ili ako u L ne postoji pozicija p.

FIRST(L) kao rezultat daje prvu poziciju u listi L. Ako je lista prazna rezultat je END(L).

NEXT(p, L) i PREVIOUS(p, L) vraćaju poziciju koja sledi odnosno prethodi poziciji p u listi L. Ako je p poslednja pozicija u listi L onda je NEXT(p, L) == END(L). Operacija PREVIOUS(p, L) je nedefinisana za prvu poziciju u listi L. Obe operacije su nedefinisane za nepostojeće pozicije p.

MAKENULL(L) prazni listu (uklanja sve elemente liste) i kao rezultat vraća poziciju END(L).

Implementacije Liste

Od izbora strukture podataka za implementaciju liste određene operacije nad listom su efikasnije od drugih.

Razmatraćemo implementaciju liste pomoću niza i pomoću pokazivača, u programskom jeziku C.

Implementacija Liste pomoću niza

U slučaju implementacije liste pomoću niza, elementi liste fizički se nalaze u uzastopnim ćelijama jednog niza. Takođe, osim niza, implementacija sadrži i jedan kursor, koji označava poziciju poslednjeg elementa liste. Lista ne mora biti fiksne dužine, ali ćemo zbog jednostavnosti pretpostaviti tako.

Iz ovakve implementacije liste jednostavno je primetiti da su operacije RETRIEVE, FIRST, NEXT i PREVIOUS složenosti $O(1)$. Linearna pretraga za implementaciju operacije LOCATE zahteva vreme $O(n)$. Ubacivanje i izbacivanje elementa na kraj (sa kraja) liste je jednostavno. Problem predstavlja umetanje ili brisanje elemenata na ostalim pozicijama, pošto je u tom slučaju potrebno pomeranje (ili prepisivanje) dela podataka.

```
#define MAX_LIST_SIZE 50

#define MIN_ELEMENT_LIST_POSITION 0

typedef int element_type;

typedef int position;

typedef struct _list {
    position last_position;
    element_type elements[MAX_LIST_SIZE];
} list;

position end(list L) {
    return L.last_position + 1;
}
```

```
position first(list L) {
    return MIN_ELEMENT_LIST_POSITION;
}
position next(position i, list L) {
    assert(i >= MIN_ELEMENT_LIST_POSITION && i < end(L));
    return i + 1;
}
position previous(position i, list L) {
    assert(i > MIN_ELEMENT_LIST_POSITION && i <= end(L));
    return i - 1;
}
position make_null(list *L) {
    L->last_position = MIN_ELEMENT_LIST_POSITION - 1;
}
```

```
element_type retrieve(position i, list L) {
    assert(i >= MIN_ELEMENT_LIST_POSITION && i < end(L));
    return L.elements[i];
}

position locate(element_type x, list L) {
    position i;
    for(i = MIN_ELEMENT_LIST_POSITION; i < end(L); i++) {
        if(retrieve(i, L) == x) {
           return i;
        }
    }

    return end(L);
}
```

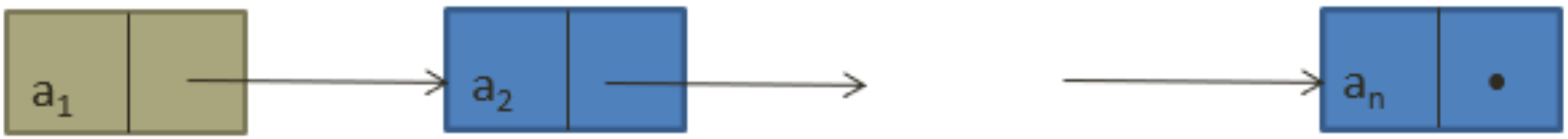
```
void insert(element_type x, position p, list *L) {
     assert(end(*L) < MAX_LIST_SIZE);
     assert(p >= MIN_ELEMENT_LIST_POSITION && p <=
         end(*L));

     position i;
     for(i = L->last_position; i >= p; i--) {
          L->elements[i + 1] = L->elements[i];
     }

     L->elements[p] = x;

     L->last_position++;
}
```

```
void delete(position p, list *L) {
     assert(p >= MIN_ELEMENT_LIST_POSITION && p < end(*L));

     position i;
     for(i = p; i < L->last_position; i++) {
          L->elements[i] = L->elements[i + 1];
     }

     L->last_position--;
}
```

Implementacija Liste pomoću pokazivača

Jednostruko-povezana lista koristi pokazivač za vezivanje uzastopnih elemenata liste. U odnosu na implementaciju liste pomoću niza dobitak predstavlja činjenica da se za čuvanje elemenata liste ne uzimaju uzastopne adrese u memoriji, već proizvoljne. Cenu predstavlja to što se za svaki novi element odvaja prostor za čuvanje pokazivača.

U ovoj reprezentaciji lista se sastoji od ćelija, od kojih svaka sadrži element liste i pokazivač na sledeću ćeliju liste. Prva ćelija liste (početak liste) naziva se "head" (glava liste). Pokazivač u poslednjoj ćeliji liste sadrži NIL pokazivač (ekvivalent NULL vrednošću u C-u).

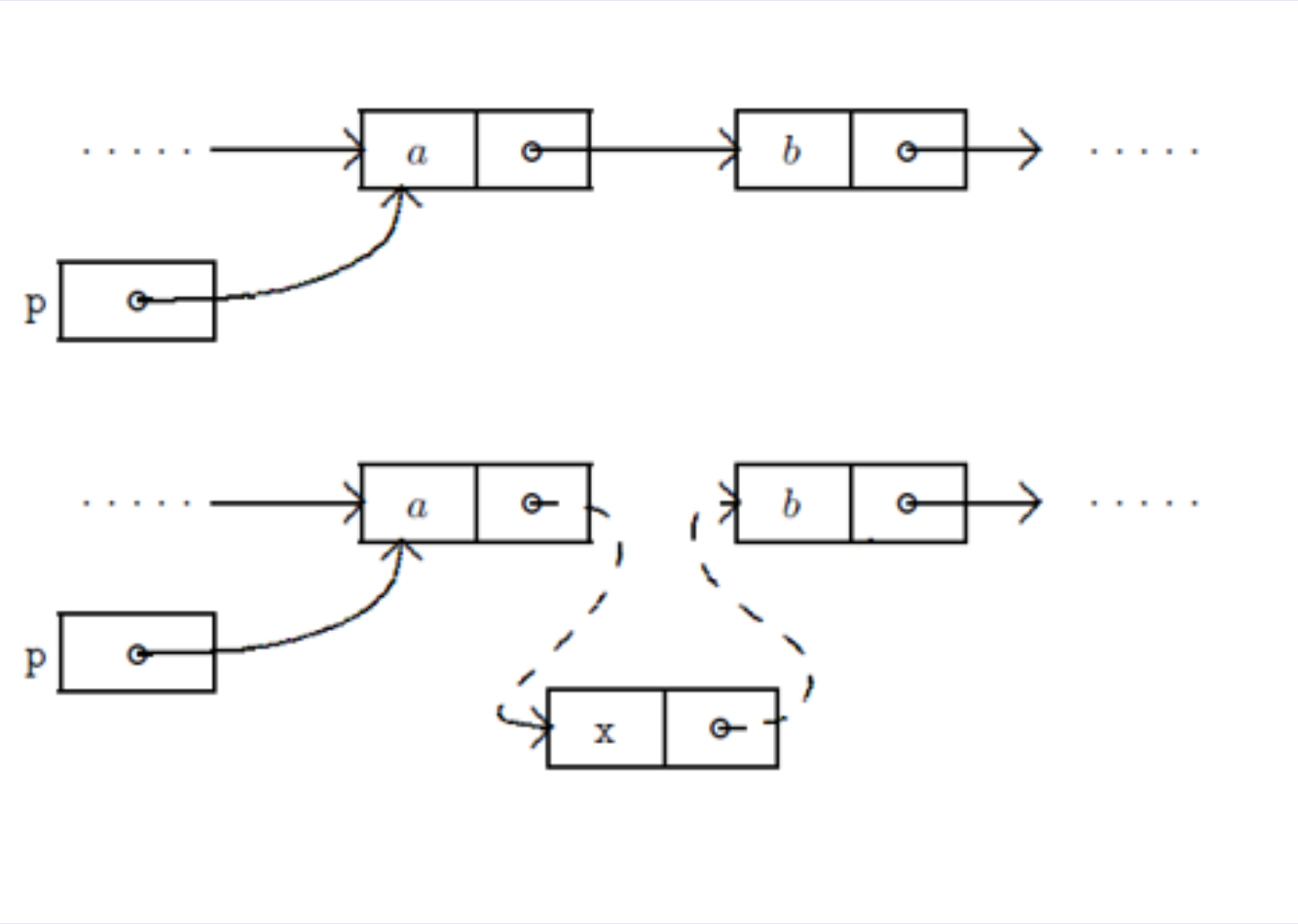


Poziciju u ovakvoj reprezentaciji predstavlja pokazivač na element liste.

Funkcija END(L) kao rezultat vraća pokazivač na poslednji element liste.

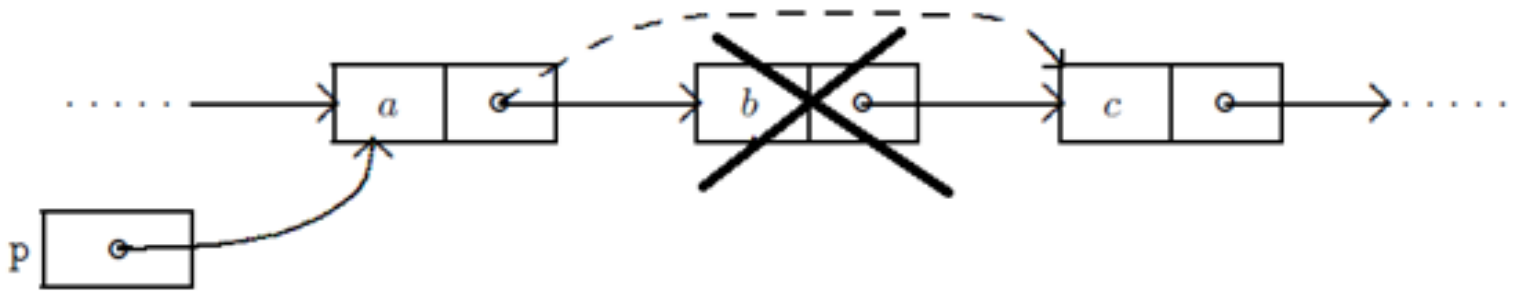
```
typedef int element_type;

typedef struct _list_element {
    element_type element;
        struct _list_element *next;
} list_element;

// Pozicija se definise kao pokazivac na element
// liste
typedef list_element* position;

//
typedef list_element* list;
```

Umetanje elementa



```
void insert(element_type x, position p) {
    list_element* new_el =
        (list_element*)
           malloc(sizeof(list_element));

    new_el->element = x;
    new_el->next = p->next;

    p->next = new_el;
}
```

Brisanje elementa



```
void delete(position p) {
     position temp = p->next;
     p->next = p->next->next;
     free(temp);
}
```

Kada izabrati implementaciju pomoću niza, a kada pomoću liste?

Implementacija pomoću niza efikasna je u slučaju kada je dohvatanje elemenata po poziciji "glavna" operacija. Operacije umetanja i brisanja su skupe zbog pomeranja elemenata na susednim pozicijama. Takođe ova implementacija zahteva i više memorije pošto je u svakom momentu rezervisana memorija za maksimalni broj elemenata (tzv. kapacitet).

Implementacija liste pomoću pokazivača pogodnija je kada su najčešće operacije umetanje i brisanje. Operacije poput END i PREVIOUS su neefikasne jer zahtevaju vreme proporcionalno dužini liste (O(n)).