Konstrukcija algoritama indukcijom

Inspiracija:

An analogy between proving mathematical theorems and designing computer algorithms provides an elegant methodology for designing algorithms, explaining their behavior, and understanding their key ideas.

Udi Manber *USING INDUCTION TO DESIGN ALGORITHMS*

Dakle, da bi se rešio neki problem, treba rešiti neki njegov mali slučaj, a zatim pokazati kako se rešenje zadatog problema može konstruisati polazeći od (rešenih) manjih verzija istog problema. Imajući na umu ovo, pažnja se može skoncentrisati na nalaženje načina za svođenje problema na manje probleme.

Miodrag Živković *ALGORITMI*

Da li je induktivni pristup uvek vodi do konstrukcije najefikasnijeg algoritma? Slede primeri.

Zadaci

1. Izračunavanje vrednosti polinoma

Problem: izračunavanje vrednosti polinoma u tački x

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

Rešenje 1. Najjednostavniji pristup je induktivni pristup: znamo da izračunamo vrednost polinoma

$$P_{n-1}(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$

pa pomoću njega izračunavamo Pn(x) formulom

$$P_n(x) = a_n x^n + P_{n-1}(x)$$

Glavni nedostatak ovog algoritma je neefikasnost, jer je potrebno

$n + (n-1) + \ldots + 1 = \frac{n\cdot(n+1)}{2}$ množenja i n sabiranja.

Rešenje 2:

Konstruisati algoritam za izračunavanje vrednosti polinoma zasnovan na dekompoziciji. Koliki je broj sabiranja i množenja potreban?

$P(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} + a_n x^n = P_p(x^2) + xP_n(x^2)$ gde je

$P_p(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-1}x^{(n-1)/2}$

$P_n(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_n x^{(n-1)/2}.$

Algoritam Vrednost polinoma dekomp(a, n. x)
Ulaz: a = a0, a1, ..., an (koeficijenti polinoma), n (stepen polinoma oblika 2^k -1), x (realan broj)
Izlaz: P (vrednost polinoma u tački x)

```
begin
if (n = 0) then P := a0;
else
  xnovo := x * x;
  Pp := Vrednost polinoma dekomp(aparno, (n - 1)/2, xnovo);
  Pn := Vrednost polinoma dekomp(aneparno; (n - 1)/2, xnovo);
  P := Pp + x * Pn;
end
```

a_{parno} označava sve članove niza a koji imaju parne indekse (vrši se preimenovanje ovih indeksa za novi poziv algoritma).

Time se izračunavanje vrednosti polinoma stepena n svodi na izračunavanje vrednosti dva polinoma stepena $(n-1)/2$, dva množenja i jedno sabiranje.
Ako $T(n)$ označava broj operacija potrebnih za izračunavanje vrednosti polinoma stepena n - 1 (polinoma sa n koeficijenata), onda je
$T(2^k) = 2T(2^{k-1}) + c$, gde je $c = 2$ za broj množenja, a $c = 1$ za broj sabiranja.
Početne vrednosti su
$T(0) = 0$ i $T(1) = 1$.
Rešenje ove diferencne jednačine je
$T(2^k) = 2^k - 1$ (sabiranja, za $c = 1$), odnosno
$T(2^k) = 2^k + 2^{k-1} - 2$ (množenja, za $c = 2$).

Dakle, za izračunavanje vrednosti polinoma stepena $n = 2^k - 1$ potrebno je izvršiti n sabiranja i $(3n-4)/2$ množenja.
Taj broj množenja je veći nego kod Hornerove šeme.

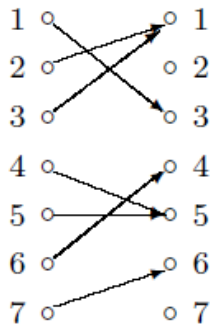
Rešenje 3:
Najefikasniji algoritam za izračunavanje vrednosti polinoma je Hornerova šema. Za izvršavanje algoritma potrebno je n množenja i n sabiranja.

Hornerova šema(a, n, x)
Ulaz: $a = a_0, a_1,...,a_n$ (koeficijenti polinoma), n, x (realan broj)
Izlaz: P (vrednost polinoma u tački x)
begin
  $P := a_n$;
  for i = 1 to n
    $P := x * P + a_{n-i}$
end

Zadatak za vežbu

Ilustrovati prethodne algoritme na računanju vrednosti polinoma
$P_3(x) = 1 + 2x + 3x^2 + 4x^3$ u tački $x = 2$.

2. Nalaženje bijekcije
Neka je f funkcija koja preslikava konačan skup A u samog sebe, $f : A \rightarrow A$. Odrediti podskup $S \subset A$ sa najvećim mogućim brojem elemenata tako da važi $f(S) \subset S$ i da restrikcija f na S bude bijekcija.
Rešenje:
Da li je funkcija zadata sledeći dijagramom bijekcija? Zašto?



Dakle, uočimo da f(2)=f(3)=1

Ako je neka zadata funkcija f bijekcija, onda kompletan skup A zadovoljava uslove problema, te je maksimalni podskup $S = A$.
U suprotnom ako postoje a,b takvi da $a \neq b$ i da je $f(a) = f(b)$, onda S ne može da sadrži i a i b. Na primer, skup S u problemu sa slike ne može da sadrži oba elementa 2 i 3, jer je $f(2) = f(3) = 1$. Međutim, nije svejedno koji će elemenat od ova dva biti eliminisan.
1. Ako na primer eliminišemo 3, onda se zbog $f(1) = 3$ mora eliminisati i 1. Zatim se na isti način zbog $f(2) = 1$ mora eliminisati i 2. Dobijeni podskup sigurno nije maksimalan, jer je moguće eliminisati samo elemenat 2 umesto 1, 2 i 3.
Rešenje problema na slici je podskup $\{1, 3, 5\}$ što se može ustanoviti npr. proverom svih mogućih podskupova $S \subset A$.
Postavlja se pitanje nalaženja opšteg metoda za odlučivanje koje elemente treba uključiti u S.

Koristićemo sledeću jednostavnu induktivnu hipotezu.
Induktivna hipoteza. Znamo da rešimo problem za skupove sa *n* - 1 elemenata.

```
// A - ceo skup
// n - velicina skupa
bijekcija(f, A, n) {
    if (n == 1) {   // Baza indukcije:
        return A;
    }

    a = element u koji se nijedan drugi ne slika

    if (a == null)
        return A;
    else
        return bijekcija(A \ {a}, n - 1);

}
```

Baza indukcije: ako skup *A* ima jedan element, on se mora preslikavati u sebe, pa je funkcija *f* na *A* bijekcija.

Pretpostavimo dalje da imamo proizvoljan skup A od n elemenata i da tražimo podskup S koji zadovoljava uslove problema.
Tvrdjenje: ako elemenat a ne pripada skupu $f(A)$, onda a ne može pripadati S;
Drugim rečima: elemenat a za koji u odgovarajući čvor na desnoj strani dijagrama ne ulazi ni jedna grana, ne može biti u S.
Dokaz:
Zaista, iz uslova problema sledi da je $f(S) = S$, a to je nemoguće, ako se pretpostavi da $a \in S$, jer po pretpostavci $a \notin f(S)$.

Dakle, ako postoji takav elemenat a, mi ga eliminišemo iz skupa.
Sada imamo skup $A' = A \setminus \{a\}$ sa n - 1 elementom, koji funkciju f preslikava u samog sebe i prema induktivnoj hipotezi mi znamo da rešimo problem sa skupom A'.

Ako takvo a ne postoji, onda je preslikavanje bijekcija, i problem je rešen.

Suština ovog rešenja je da se a mora ukloniti. Dokazali smo da a ne može pripadati S. U tome je snaga indukcije: onog trenutka kad se elemenat ukloni i time smanji veličina problema, zadatak je rešen.

Algoritam Bijekcija(f, n)

Ulaz f (niz prirodnih brojeva između 1 i n) i n

Izlaz S (podskup ulaznog skupa takav da je f bijekcija na S)

begin

$S := A; \{A = \{1, 2, ..., n\}\}$

for $j := 1$ to n do $c[j] := 0$;

for $j := 1$ to n do $c[f[j]] := c[f[j]] + 1$;

for $j := 1$ to n do

if $(c[j] = 0)$ then dodaj j u Listu;

while (Lista nije prazna) do

skini i sa vrha Liste;

$S := S \setminus \{i\}$;

$c[f[i]] := c[f[i]] - 1$;

if $(c[f[i]] = 0)$ then dodaj $f[i]$ u Listu;

end

Napomena:
c[i] označava broj elemenata koji se preslikavaju u i
c[f[j]] je broj elemenata koji se slikaju u f(j)
Ako nakon prva dva ciklusa se pojavi element c[j]=0, onda u odgovarajući čvor i sa desne strane slike se niko ne slika finkcijom f i moramo izbaciti čvor *i* iz skupa S

Složenost: Za uvodni deo (inicijalizaciju u prvom ciklusu i inkrementaciju u drugom ciklusu) treba izvršiti $O(n)$ koraka.
Svaki elemenat se u listu može staviti najviše jednom, a operacije potrebne da se elemenat ukloni iz liste izvršavaju se za vreme ograničeno konstantom. Prema tome, ukupan broj koraka je $O(n)$.

C-like rešenje u pseudo kodu (bez rekurzije)

```
bijekcija(f, A, n) {
    // A = {1, 2, …, n}
    S = A;
    c = niz nula duzine n;

    for (j in { 1, … , n }) {
        c[f[j]]++;
    }

    // lista = j takvi da je c[j] == 0
    lista = {};
    for (j in { 1, … , n }) {
        if (c[j] == 0) {
            lista.dodaj(j);
        }
    }

    while (!lista.prazna()) {
        i = lista.skiniSaVrha();
        S.skloni(i);
        c[f[i]]--;

        if (c[f[i]] == 0) {
```

```
                lista.dodaj(f[i]);
            }
        }

        return S;
    }
```

3. Razmotriti algoritam Bijekcija. Da li je moguće da skup S postane prazan na kraju izvršavanja algoritma?
Odgovor: Ne
Obrazloženje:
Tvrđenje da za $A = \{1, 2,..., n\}$ i proizvoljnu funkciju $f : A \rightarrow A$ postoji neprazan podskup $S \subset A$ takav da je na njemu f bijekcija, može se dokazati indukcijom po n, malom izmenom induktivne konstrukcije algoritma.
Baza indukcije: Tvrđenje je tačno za $n = 1$.
IH: Ako je tvrđenje tačno za brojeve *manje od* n, onda ili su u skupu $A = \{1, 2,..., n\}$ svi elementi slike nekih elemenata iz A (tada je $S = A \neq \{\}$), ili postoji takav elemenat $a \in A$ u koji se ne preslikava ni jedan elemenat A.
Tada je $f : A\backslash\{a\} \rightarrow A\backslash\{a\}$ funkcija definisana na podskupu $A\backslash\{a\} \subset A$, pa po induktivnoj hipotezi postoji
podskup $S \subset A\backslash\{a\} \subset A$, $S \neq \{\}$, na kome je f bijekcija.

4. Rastojanje izmedju dva čvora stabla je broj čvorova na jedinstvenom putu koji ih povezuje uključujući i ta dva čvora. Dijametar stabla `T = (V, E)` je najveće medju rastojanjima neka dva njegova čvora.

Konstruisati algoritam linearne vremenske složenosti za određivanje dijametra datog binarnog stabla.

Uputstvo: Dijametar stabla može da odgovara putu kroz koren ili ne. Ako izbacimo koren stabla, dobijamo dva binarna podstabla.

Važi:

- ako put koji odgovara dijametru stabla ne sadrži koren, tada je dijametar stabla takođe dijametar nekog od njegovih podstabala
- ako put koji odgovara dijametru sadrži koren, tada ovaj put povezuje dva čvora iz levog i desnog podstabla koja su najudaljenija od korena.

Ovo nas vodi do rešavanja problema **korišćenjem potpune indukcije**: pretpostavimo da znamo da odredimo dijametar za stablo sa manje od `n` čvorova i kako da nadjemo maksimalnu udaljenost od korena.

Baza indukcije: trivijalna (dijametar praznog stabla je 0)

Induktivni korak: Za stablo sa `n` čvorova označimo sa `v` koren stabla i rešavamo induktivno problem za oba podstabla datog stabla. Da bismo našli rastojanja od čvora `v` treba dodati `1` većem od rastojanja najudaljenijih čvorova. Pored toga dijametar dobijamo kada poredimo pronađene dijametre podstabla sa sumom dva najveća rastojanja od čvora `v`.

Složenost predloženog algoritma je O(n).

Dajemo rešenje u kom je implementirana funkcija diametar čija složenost je $O(n^2)$

```
#include <stdio.h>
#include <stdlib.h>
/* binarno stablo */
struct node
{
    int data;
    struct node* left, *right;
};

/* kreiranje novog cvora stabla */
struct node* newNode(int data);

/* vraca najveci od dva cela broja a,b */
int max(int a, int b);

/* izracunavanje visine stabla  */
int height(struct node* node);

/* algoritam za racunanje dijametra slozenosti O(n*n) */
int diameter(struct node * tree)
{
   /* baza: prazno stablo */
   if (tree == NULL)
     return 0;

  /* saznajmo visinu desnog i levog podstabla */
  int lheight = height(tree->left);
  int rheight = height(tree->right);


/* saznajmo dijametar desnog i levog podstabla */
  int ldiameter = diameter(tree->left);
  int rdiameter = diameter(tree->right);

  /* izracunajmo maksimum sledece 3 vrednosti
   1) Dijametar levog podstabla
```

```
   2) Dijameter desnog podstabla
   3) Visina levog podstabla + visina desnog podstabla + 1 */
  return max(lheight + rheight + 1, max(ldiameter, rdiameter));
}


/*  izracunavanje visine stabla. Visina stabla je broj cvorova na najduzoj putanji od cvora ka listu*/
int height(struct node* node)
{
   /* baza: prazno stablo */
   if(node == NULL)
       return 0;


   /* Ako stablo nije prazno, onda visina je  = 1 + max(visina levog podstabla, visina desnog
podstabla) */
   return 1 + max(height(node->left), height(node->right));
}

struct node* newNode(int data)
{
  struct node* node = (struct node*)
                      malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;


  return(node);
}

/* vraca maksimum dva cela broja */
int max(int a, int b)
{
  return (a >= b)? a: b;
}

/* Algoritam slozenosti O(n)
Drugi parametar funkcije se prenosi po referenci, jer cuva
   tekucu visinu stabla. Na taj nacin, ne pozivamo funkciju height zasebno kao u prethodnom
rešenju, već računamo visinu stabla unutar potprograma diametarOpt
    */
```

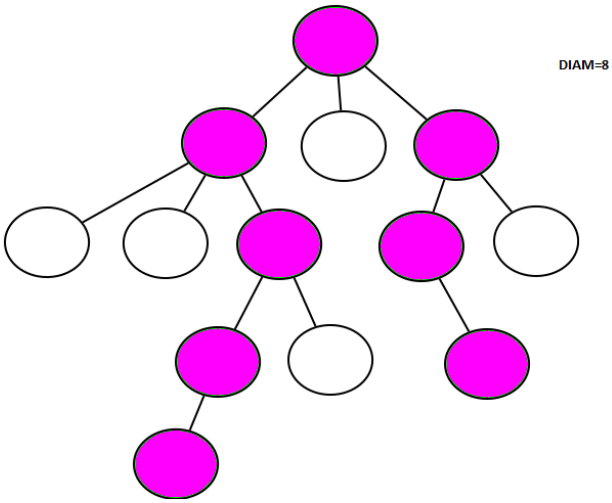
```
int diameterOpt(struct node *root, int* height)
{
  /* lh --> visina levog podstabla
     rh --> visina desnog podstabla */
  int lh = 0, rh = 0;

  /* ldiameter  --> dijametar levog podstabla
     rdiameter  --> dijametar desnog podstabla */
  int ldiameter = 0, rdiameter = 0;

  if(root == NULL)
  {
    *height = 0;
     return 0; /* dijametar ima vrednosti 0 */
  }

  /* pamtiti visinu levog i desnog podstabla u lh i rh
     pamtiti rezutujuci dijametar u  ldiameter i rdiameter */
  ldiameter = diameterOpt(root->left, &lh);
  rdiameter = diameterOpt(root->right, &rh);

  /* visina tekuceg cvora */
  *height = max(lh, rh) + 1;

  return max(lh + rh + 1, max(ldiameter, rdiameter));
}

/* Program za testiranje */
int main()
{

  /* Dato binarno stablo je
           1
         /   \
        2      3
      /  \
    4     5
  */
  struct node *root = newNode(1);
```

```
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);
  int visina=0;

  printf("Dijametar datog binarnog stabla je %d\n", diameterOpt(root, &visina));


  return 0;
}
```

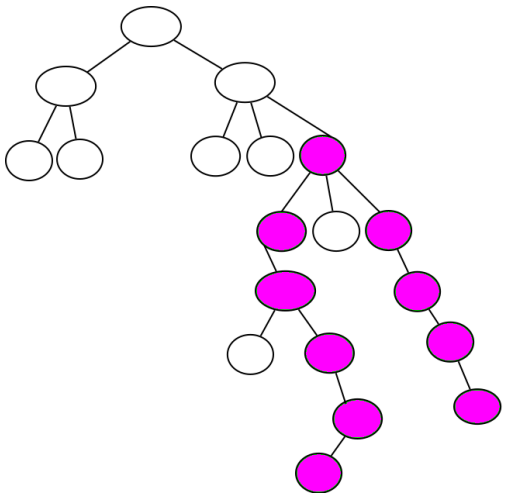
5. Konstruisati algoritam koji će odrediti diametar n-arnog stabla. Složenost?

Rešenje:

Slika 1 – dijametar=8 (ubraja se i koren)



Slika 2 – dijametar=10 (ne ubraja se koren)



Ideja:
Rešenje je najveći broj čvorova na putu koji počinje od nekog čvora (list) i
1) penjemo se naviše do najnižeg zajedničkog pretka drugog čvora (lista) i spuštamo se naniže do najdubljeg čvora tekućeg podstabla i tražimo rešenje kao suma visina dva najveća podstabla +1
ILI
2) tražimo najveći dijametar deteta tekućeg čvora