Arbori binari de căutare optimali

Introducere

In cazul arborilor binari de căutare (BST – Binary Search Tree), pentru un nod (sau cheie) oarecare, timpul de căutare este proporţional cu adâncimea nodului (distanţa de la acesta la rădăcină).

Evident, unele chei sunt căutate mai des altele mai rar. Ar fi convenabil ca acele chei care sunt căutate mai des să fie plasate mai aproape de rădăcina arborelui iar acele chei care sunt căutate mai rar să fie plasate mai departe de rădăcina arborelui.

Dacă pentru fiecare dintre cheile unui arbore binar ordonat cunoaştem frecvenţa cu care această cheie va apărea în cadrul operaţiilor ulterioare de căutare, se pot construi arbori binari optimi pentru care cheile mai des căutate să fie plasate mai aproape de rădăcină. Frecvenţa de căutare poate fi privită şi ca "probabilitatea" ca o anumită cheie să fie ţinta unei operaţii viitoare de căutare în BST.

Pentru un arbore binar ordonat conţinând cheile $k_1, k_2, \ldots, k_n$, se consideră că fiecare cheie k_i are asociată o "probabilitate de apariţie" p_i, respectând proprietăţile:

$0 \le p_i \le 1$, unde i = 1 .. n;

$p_1 + p_2 + \ldots + p_n = 1$.

Arborele binar ordonat poate fi numit "optim" (**OBST**) dacă:

$$\sum_i p_i \bullet nivel(k_i) = \text{minim} \qquad (1)$$

Suma anterioară poartă numele de cost al arborelui sau drum ponderat, fiind suma drumurilor de la rădăcina la fiecare nod, ponderate cu probabilităţile de acces la noduri.

Exemplu. Pentru un BST cu nodurile de chei 4, 5, 8, 10, 11, 12, 14 există mai multe topologii care să respecte proprietatea BST, printre care şi reprezentările din figura de mai jos.

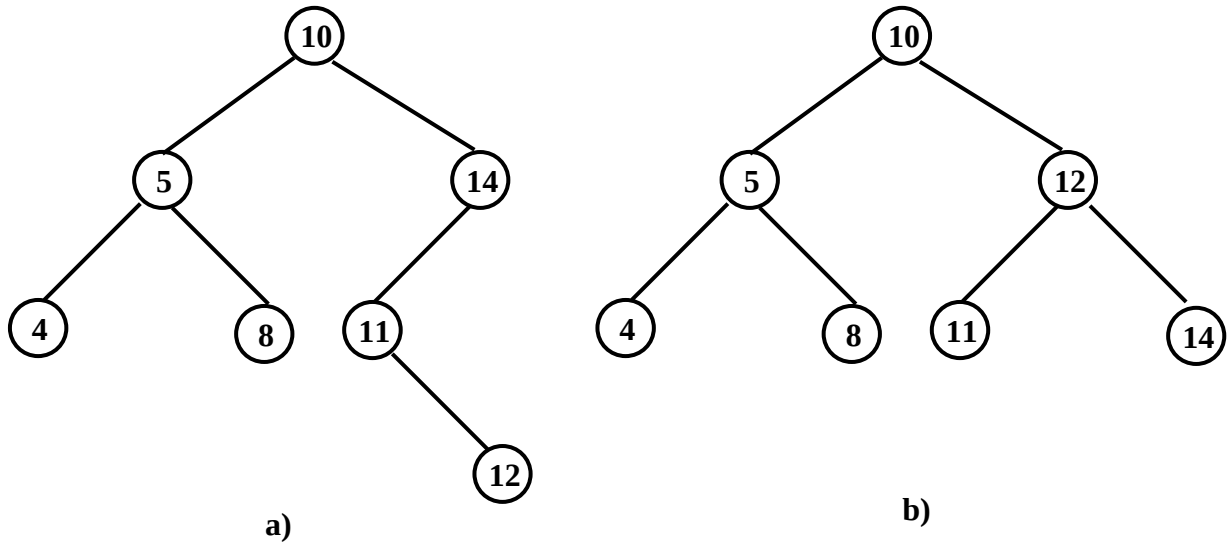

Fig. 1. Topologii ale unui BST

Deşi reprezentarea din figura b) corespunde unui AVL, acest fapt nu garantează faptul că arborele este optimal în sensul definiţiei de mai sus. Arborele din figura a) poate fi optimal numai dacă frecvenţele de acces ale nodurilor sunt mai mari pentru nodurile de pe niveluri mici şi sunt mari pentru nodurile de pe niveluri mari. Putem admite că este posibil ca un BST degenerat să fie unul optimal (*a se vedea prezentarea de la curs*!).

Mai apare o problemă: pe lângă frecvenţele de accesare a nodurilor existente in arbore (***căutări încheiate cu succes***), cum putem evidenţia căutările unor chei ce nu fac parte din arbore (***căutări eşuate***) şi cum le vom lua în considerare în construirea unui OBST?

Pentru o astfel de abordare, vom porni de la Fig. 1 a) şi vom construi arborele extins corespunzător (fig. 2).

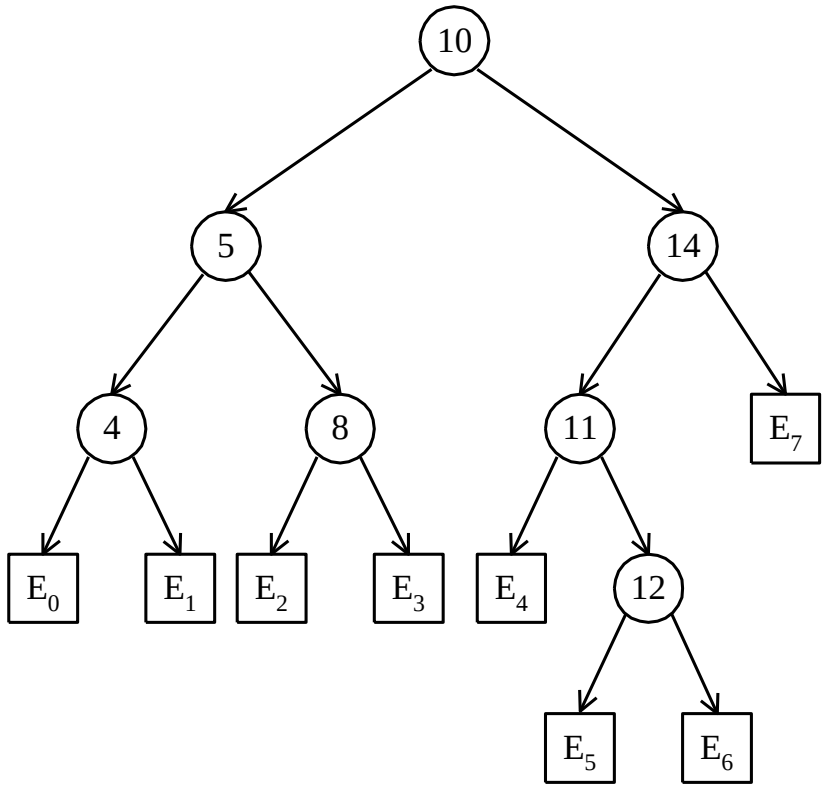

Fig. 2. Arborele OBST extins

Un BST extins se obţine dintr-un BST prin adăugarea nodurilor succesoare E_i (noduri extinse sau fictive - marcate cu pătrate in fig. 2) la toate frunzele din arborele de la care am plecat. Un nod E_i corespunde tuturor căutărilor eşuate pentru valori cuprinse în intervalul cheilor (k_i, k_{i+1}). Astfel, timpul de căutare pentru un element oarecare x este dat fie de adâncimea nodului de cheie x fie de adâncimea pseudonodului corespunzător intervalului dintre noduri care ar putea să-l conţină pe x. Notând cu q_i probabilitatea de a căuta un nod de cheie x ce satisface relaţia $k_i<x<k_{i+1}$, atunci costul unui BST se defineşte:

$$C = \sum_{i=1}^{n} p_i \bullet nivel(k_i) + \sum_{i=0}^{n} q_i \bullet (nivel(E_i) - 1) \qquad (2)$$

Expresia (nivel(E_i)-1) în a doua sumă se justifică prin faptul că încadrarea unui nod x într-un interval se decide la nivelul părintelui unui pseudonod nu la nivelul pseudonodului. Pentru un BST optimal, valoarea lui C este minimă.

Orice subarbore BST conţine cheile ordonate în domeniul k_i ….. k_j pentru $1 \le i \le j \le n$ şi va avea propriile noduri fictive E_{i-1} … E_j.

Problema construirii unui OBST revine la a construi subarbori optimali. Dacă k_r este rădăcina unui arbore optimal cu n chei, una din cheile $k_1 \ldots k_{r-1}$ va fi rădăcina subarborelui stâng şi una din cheile k_{r+1} … k_n va fi rădăcina subarborelui drept (a se vedea figura de mai jos). Dacă rădăcinile celor doi subarbori se aleg astfel încât aceştia să fie optimali avem garanţia că arborele de rădăcină k_r este optimal.

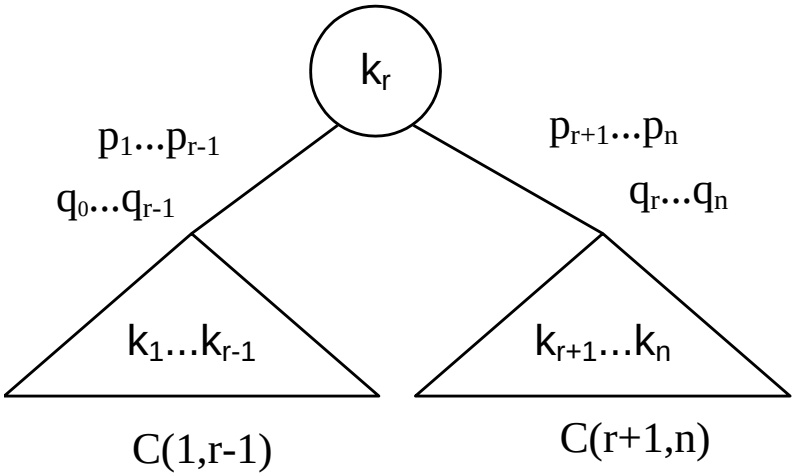

Fig. 3. Subarbore optimal

Ce se întâmplă când avem subarbori nuli? Dacă într-un subarbore cu cheile k_i ….. k_j găsim cheia k_i ca fiind rădăcina subarborelui optimal vom aprecia că subarborele stâng conţine doar nodul fictiv E_{i-1} şi deci costul unei căutări pentru o cheie de valoare mai mică decât k_i va depinde doar de q_{i-1}.

Dacă se notează cu C(i,j) costul unui subarbore optimal cu cheile k_i, . . . k_j, relaţia de calcul este:

$$\begin{aligned} C(i,j) \quad &= \min_{i \le k \le j} \left\{ p_k + \left[q_{i-1} + \sum_{m=i}^{k-1} \left(p_m + q_m \right) + C\left(i,k-1\right) \right] \right. \\ &\qquad \left. + \left[q_k + \sum_{m=k+1}^{j} \left(p_m + q_m \right) + C\left(k+1,j\right) \right] \right\} \\ &= \min_{i \le k \le j} \left\{ C\left(i,k-1\right) + C\left(k+1,j\right) + q_{i-1} + \sum_{m=i}^{j} \left(p_m + q_m \right) \right\} \end{aligned} \tag{3}$$

In relaţia (3), $i \ge 1$, $j \le n$ şi $j \ge i-1$. Cazul j=i-1 corespunde situaţiei subarborelui vid când se ia în considerare doar E_{i-1}. Ultimul cost care va fi calculat este C(1,n):

$$C(1,n) = \min_{1\le k\le n}\left\{ p_k + \left[q_0 + \sum_{i=1}^{k-1}\left(p_i + q_i\right) + C\left(1,k-1\right)\right] + \left[q_k + \sum_{i=k+1}^{n}\left(p_i + q_i\right) + C\left(k+1,n\right)\right]\right\} \quad (4)$$

Această relaţie de recurenţă evidenţiază faptul că atunci când un arbore de cost minim devine subarbore a unui alt nod, nivelul (adâncimea) fiecărui nod din subarbore creşte cu 1 iar costul calculat anterior creşte cu suma tuturor probabilităţilor din subarbore.

Dacă k_r este rădăcina unui subarbore optimal de chei $k_i, \ldots k_j$, relaţia (3) poate fi rescrisă:

$$\begin{aligned} C(i,j) \quad & = p_r + \left(C\left(i,k-1\right) + w(i,r-1)\right) + \left(C\left(k+1,j\right) + w(r+1,j)\right) \\ cu \quad & w(i,j) = w(i,r-1) + p_r + w(r+1,j) \\ rezulta \quad & C(i,j) = C(i,r-1) + C(r-1,j) + w(i,j) \end{aligned} \quad (5)$$

Relaţia de recurenţă de mai sus presupune că se ştie nodul de cheie k_r ce urmează a fi rădăcină. Alegerea nodului rădăcină care va garanta cel mai mic cost conduce la următoarea relaţie pentru calculul recursive:

$$c(i,j) = \begin{cases} \min\{C(i,r-1) + C(r+1,j) + w(i,j), & \text{daca } i \le j;\ i \le r \le j \\ q_{j-1}, & \text{daca } j=i-1 \end{cases} \quad (6)$$

Pentru a putea construi structura OBST (evidenţa legăturilor între noduri şi descendenţi) se defineşte o matrice R(i,j) ($1\le i\le j\le n$) care va memora indexul r pentru care k_r este rădăcina unui OBST de chei k_i ….. k_j.

Generarea un OBST

O metodă “primitivă” de generare a unui OBST pentru un set de chei ar fi să se genereze toţi arborii de căutare posibili, să se calculeze costul fiecăruia şi să se reţină arborele de cost minim. Evident o astfel de abordare nu este fezabilă deoarece presupune un ordin de complexitate exponenţial.

Din cele prezentate anterior este evident că un algoritm recursiv este o soluţie realistă. Pentru implementare se construiesc următoarele matrice:

$W_{i,j}$ – matricea ponderilor (termenul în engleză este ***weight*** şi a fost tradus de unii autori drept “greutate”, dar termenul consacrat in matematică/informatică este de “pondere”) care se obţine cu relaţia:

$$W_{i,j} = \sum_{k=i+1}^{j} p_k + \sum_{k=i}^{j} q_k \ .$$

Matricea W conţine pe diagonala principală elementele q_j.

$C_{i,j}$ – matricea costurilor pentru OBST, ale cărei elemente se calculează conform relaţie (6):

$$C_{i,i} = W_{i,i}$$

$$C_{i,j} = W_{i,j} + \min_{i<k\le j}(C_{i,k-1} + C_{k,j})\ ;$$

$R_{i,j}$ – matricea ce conţine indexul cheii pentru care se obţine un minimum corespunzător în matricea $C_{i,j}$.

OBST(i,j) - reprezintă arborele optimal ce conţine cheile k_i,....k_j. Fiecare subarbore OBST(i,j) va avea rădăcina $k_{Ri,j}$ şi subarborii OBST(i,k-1) şi OBST(k+1,j).

Un exemplu de generare a unui OBST

Până acum, teoria prezentată a făcut referire la probabilităţile p_i de căutare cu succes a unei chei care se află într-un arbore ordonat şi q_i probabilităţile "eşuate" de a nu găsi în arbore cheia căutată. Aceste probabilităţi se calculează uşor dacă se cunosc frecvenţele de accesare ale nodurilor (cu sau fără succes) şi numărul total de noduri. In exemplu următor, vor fi utilizate frecvenţele de căutare, lăsând în seama studentului calcularea probabilităţilor (fapt ce nu este obligatoriu).

Se va genera un arbore de căutare optimal, cu 6 noduri şi cu următoarele frecvenţe de căutare a cheilor şi "între chei":

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**
K (Valoare cheie)		3	7	10	15	20	25
p	-	10	3	9	2	0	10
q	5	6	4	4	3	8	0

Pas1. Se calculează matricea ponderilor W cu relaţiile de mai sus. Astfel se obţine:

W (0, 0) = q0 = 5
W (1, 1) = q1 = 6
W (2, 2) = q2 = 4
W (3, 3) = q3 = 4
W (4, 4) = q4 = 3
W (5, 5) = q5 = 8
W (6, 6) = q6 = 0

W (0, 1) = q0 + q1 + p1 = 5 + 6 + 10 = 21
W (0, 2) = W (0, 1) + q2 + p2 = 21 + 4 + 3 = 28
W (0, 3) = W (0, 2) + q3 + p3 = 28 + 4 + 9 = 41
W (0, 4) = W (0, 3) + q4 + p4 = 41 + 3 + 2 = 46
W (0, 5) = W (0, 4) + q5 + p5 = 46 + 8 + 0 = 54
W (0, 6) = W (0, 5) + q6 + p6 = 54 + 0 + 10 = 64
W (1, 2) = W (1, 1) + q2 + p2 = 6 + 4 + 3 = 13 .... şi rezultă:

W	0	1	2	3	4	5	6
0	5	21	28	41	46	54	64
1		6	13	26	31	39	49
2			4	17	22	30	40
3				4	9	17	27
4					3	11	21
5						8	18
6							0

Deoarece suma elementelor vectorului p este 34 şi suma elementelor vectorului q este 30, suma tuturor accesărilor este 64, iar matricea W rescrisă în termeni de probabilitate este W':

W'	0	1	2	3	4	5	6
0	0.08	0.33	0.44	0.64	0.72	0.84	1.00
1		0.09	0.20	0.41	0.48	0.61	0.77
2			0.06	0.27	0.34	0.47	0.63
3				0.06	0.14	0.27	0.42
4					0.05	0.17	0.33
5						0.13	0.28
6							0.00

Pas. 2. Elementele de pe diagonala principală a matricei de cost sunt egale cu cele ale diagonalei principale din matricea ponderilor. Elementele de deasupra diagonalei principale se calculează cu relaţiile:

C (0, 1) = W (0, 1) + (C (0, 0) + C (**1**, 1)) = 21 + 5 + 6 = 32
C (1, 2) = W (1, 2) + (C (1, 1) + C (**2**, 2)) = 13 + 6 + 4 = 23
C (2, 3) = W (2, 3) + (C (2, 2) + C (**3**, 3)) = 17 + 4 + 4 = 25
C (3, 4) = W (3, 4) + (C (3, 3) + C (**4**, 4)) = 9 + 4 + 3 = 16
C (4, 5) = W (4, 5) + (C (4, 4) + C (**5**, 5)) = 11 + 3 + 8 = 22
C (5, 6) = W (5, 6) + (C (5, 5) + C (**6**, 6)) = 18 + 8 + 0 = 26

Tot în acest pas, elementele de deasupra diagonalei principale din R se completează cu valorile indexului i din C(i,j).

Pas 3. În continuare se calculează celelalte elemente ale matricei C conform relaţiei 6 şi se completează a doua diagonală din matricea R cu indexul i aferent valorii de cost minim – acele valori sunt subliniate in relaţiile de mai jos şi corespund valorilor minime scrise cu roşu.

C (0, 2) = W (0, 2) + min (C (0, 0) + C (**1**, 2), C (0, 1) + C (2, 2)) = 28 + min (**28**, 36) = 56
C (1, 3) = W (1, 3) + min (C (1, 1) + C (2, 3), C (1, 2) + C (**3**, 3)) = 26 + min (31, **27**) = 53
C (2, 4) = W (2, 4) + min (C (2, 2) + C (**3**, 4), C (2, 3) + C (4, 4)) = 22 + min (**20**, 28) = 42
C (3, 5) = W (3, 5) + min (C (3, 3) + C (4, 5), C (3, 4) + C (**5**, 5)) = 17 + min (26, **24**) = 41
C (4, 6) = W (4, 6) + min (C (4, 4) + C (5, 6), C (4, 5) + C (**6**, 6)) = 21 + min (29, **22**) = 43

C	0	1	2	3	4	5	6
0	5	32					
1		6	23				
2			4	25			
3				4	16		
4					3	22	
5						8	26
6							0

C	0	1	2	3	4	5	6
0	5	32	56				
1		6	23	53			
2			4	25	42		
3				4	16	41	
4					3	22	43
5						8	26
6							0

Matricea costurilor după iteraţia a doua şi a treia

Calculele continuă în aceeaşi manieră şi în final se obţin matricele C şi R de mai jos:

C	0	1	2	3	4	5	6
0	5	32	56	98	118	151	188
1		6	23	53	70	103	140
2			4	25	42	75	108
3				4	16	41	68
4					3	22	43
5						8	26
6							0

R	0	1	2	3	4	5	6
0	0	1	1	2	3	3	3
1		0	2	3	3	3	3
2			0	3	3	3	4
3				0	4	5	6
4					0	5	6
5						0	6
6							0

Lungimea medie a drumului de căutare, numită şi costul arborelui de căutare sau lungimea de cale ponderată este egală cu max(C(i,j))/max(W(i,j))=C(0,6)/W(0,6).

Pas 4. Construirea arborelui. Arborele de căutare optimal obţinut este prezentat în figura 4 de mai jos şi are un drum ponderat de 2,93=188/64. Calcul poziţiilor nodurilor în arbore se face astfel:

- rădăcina arborelui optimal are indexul cheii dat de R (0, 6)=3 → K3 este cheia rădăcinii de valoare 10 şi frecvenţă de accesare 9;
- In general, dacă indexul rădăcinii unui arbore optimal este pe poziţia (i,j), atunci indexul cheii rădăcinii subarborelui stâng este pe poziţia (i,R(i,j)-1). In cazul considerat, indexul subarborelui stâng este dat de R(0, 2) =1 →K1 este cheia nodului de valoare 3 şi frecvenţă de accesare 10;
- dacă indexul rădăcinii unui arbore optimal este pe poziţia (i,j), atunci indexul cheii rădăcinii subarborelui drept este pe poziţia (R(i,j),j). In exemplul nostru, indexul subarborelui drept al rădăcinii este dat de R(3, 6) =6 →K6 este rădăcina subarborelui drept de valoare 25 şi frecvenţă de accesare 10.

Mai departe arborele se construieşte după aceleaşi reguli.

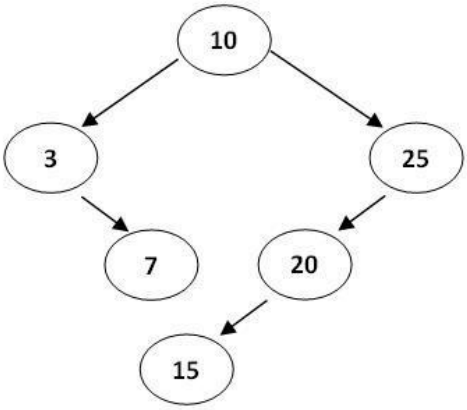

Fig. 4. OBST obţinut pentru exemplul numeric considerat

Indicaţii de implementare

In continuare sunt prezentate declaraţiile şi funcţiile propuse pentru implementarea unei aplicaţii care crează şi afişează un OBST.

```
Declaratii

#define nmax 20 //nr max de noduri (chei)

//structura arborelui optimal
struct OBST
{     char KEY;
      OBST *left, *right;
};

int C[nmax][nmax];      //matrice cost
int W[nmax][nmax];      //matrice ponderi
int R[nmax][nmax];      //radacinile subarborilor optimali. Va contine
                        //indecsii cheilor din vectorul KEYS,
                        //corespunzator radacinilor subarborilor
int q[nmax];            //frecventa de cautare intre chei=esec in ce
                        //priveste cautarea
int p[nmax];            //frecventa de cautare a cheilor
int nr_keys;            //numarul de noduri (chei)
```

```
char KEYS[nmax];        //

//Functia pentru calculul matricelor W, C si R
void Matrix()
{
      int x, min; int i, j, k, h,m;
      //construiesc matricea de ponderi W
      for(i = 0; i <= nr_keys; i++)
      { ...................
      }
      //Construesc matricea cost C si matricea R
      for(i = 0; i <= nr_keys; i++)
            //relatia pentru diagonala principala din C;
            ......................
      for(i = 0; i <= nr_keys -1; i++)
      {
            j = i + 1;
            //relatia pentru a doua diagonală din C (deasupra celei cunoscute)
            ....................................
            //relatia pentru diagonala matricei R
            ....................................
      }
      //completez matricele C si R
      for(h = 2; h <= nr_keys; h++)
            for(i = 0; i <= nr_keys -h; i++)
            {
                  j = i + h; m = R[i][j-1]; min = C[i][m-1] + C[m][j];
                  for(k = m+1; k <= R[i+1][j]; k++)
                  { //caut minim si index pentru minim (m)
                        x = C[i][k-1] + C[k][j];
                        if(x < min)
                        { m = k;
                        min = x;
                        }
                  }
                  C[i][j] = W[i][j] + min;
                  R[i][j] = m; //indexul lui k pentru care s-a obtinut
                              //minimum se pune in R
            }
            //afisez W
            cout<<"\nMatricea W\n";
.......................................
            //Afisez C
            cout<<"\n Matricea cost"<<endl;
.......................................
            //Afisez R
            cout<<"\n Matricea R:\n";
.....................................

//Construiesc BST optimal
OBST *Build_OBST(int i, int j)
{ OBST *p;
      if(i == j) p = NULL;
      else
            {     p = new OBST;                        //se aloca memorie
                  p->KEY = KEYS[R[i][j]];              //pun valoarea cheii
                  p->left = Build_OBST(i, R[i][j] -1); //subarbore sting
                  p->right = Build_OBST(R[i][j], j); //subarbore drept
            }
      return p;
}
```

Structuri de Date si Algoritmi – Lucrarea nr. 13

In main se citesc numarul de chei (nr_keys), vectorii KEYS, p (de la 1 la nr_keys) şi q (de la 0 la nr_keys) apoi se apelează funcţia Buld_OBST(0,nr_keys) şi cea de afişare indentată a OBST.

Teme

1. Să se creeze şi să afişeze un OBST cu chei de tip char, pe baza frecvenţelor de căutare.
2. Modificaţi aplicaţia astfel încât arborele optimal să fie creat pe baza probabilităţilor de accesare. Să se afişeze drumul mediu.
3. Care este ordinul de complexitate pentru generarea unui OBST? Justificaţi.