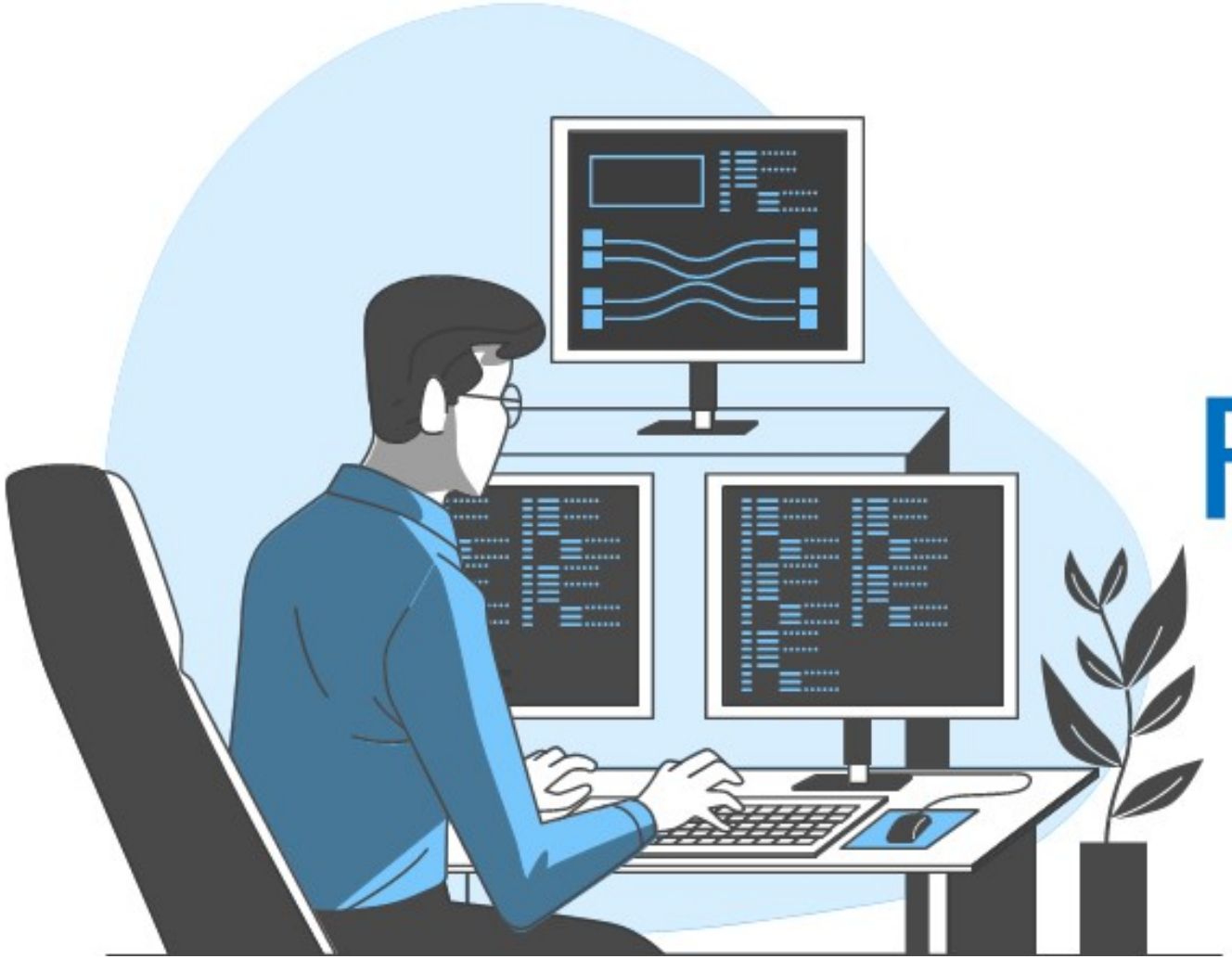
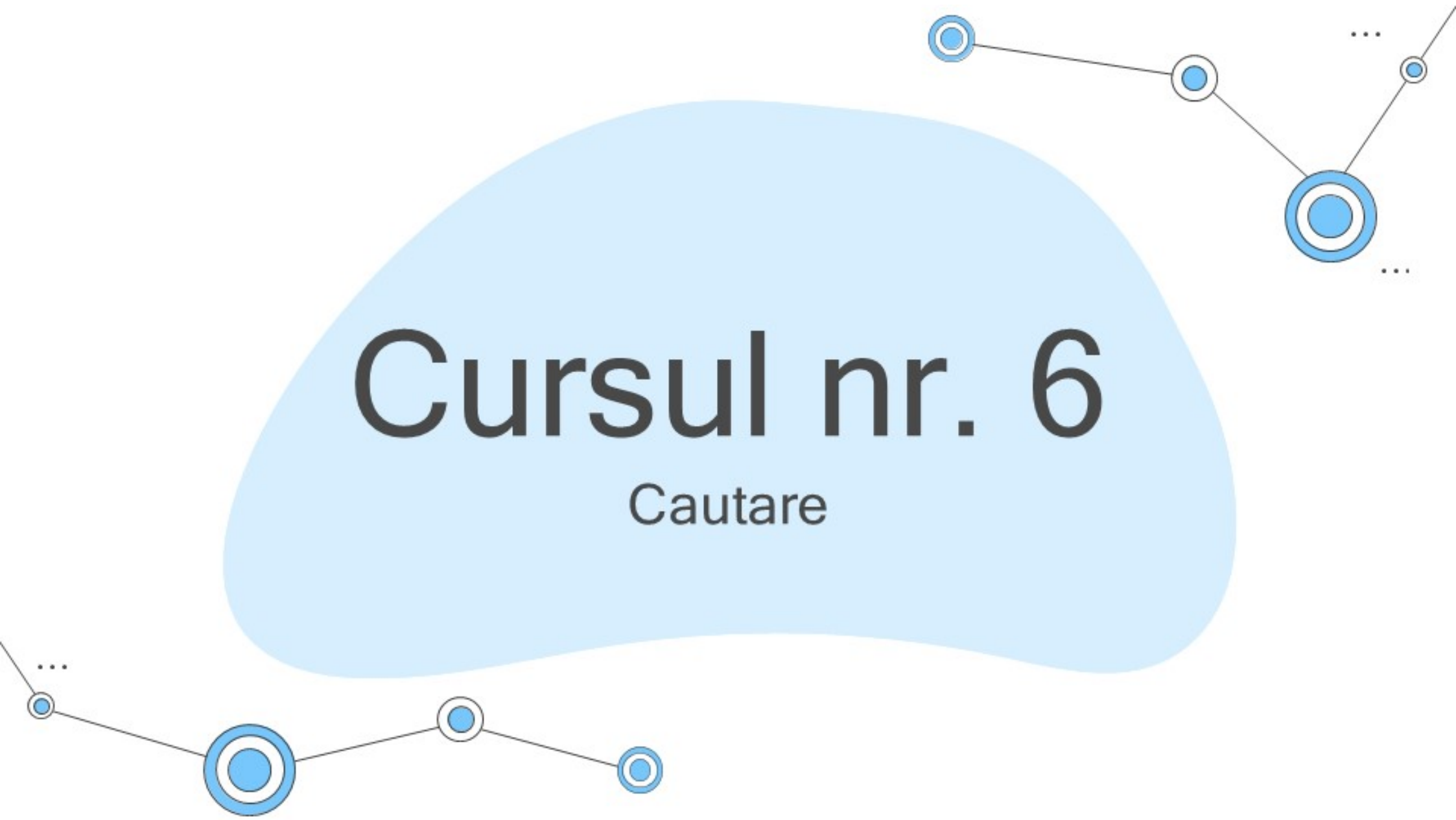


Algoritmi Fundamentali

Lector dr.
Dorin IORDACHE


Cursul nr. 6
Cautare

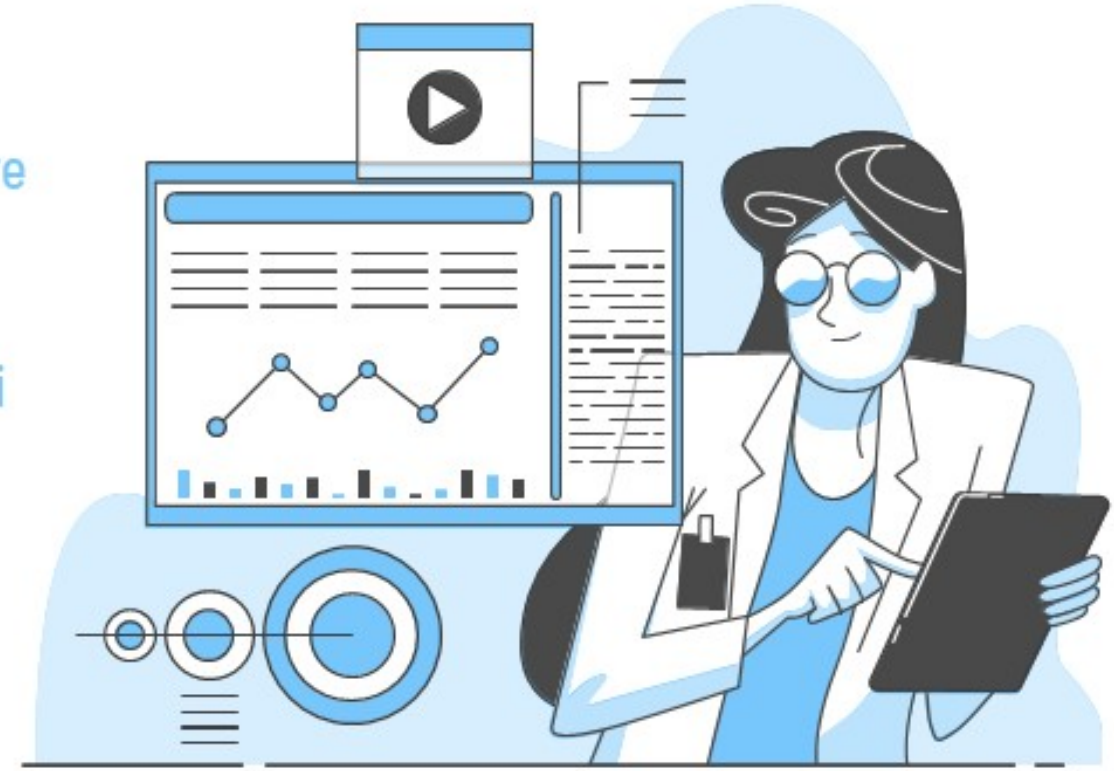
Agenda

01 Cautarea secventiala

02 Cautarea binara
Cautarea prin interpolare

03 Cautarea folosind arbori

04 Cautarea folosind
tabele de dispersie

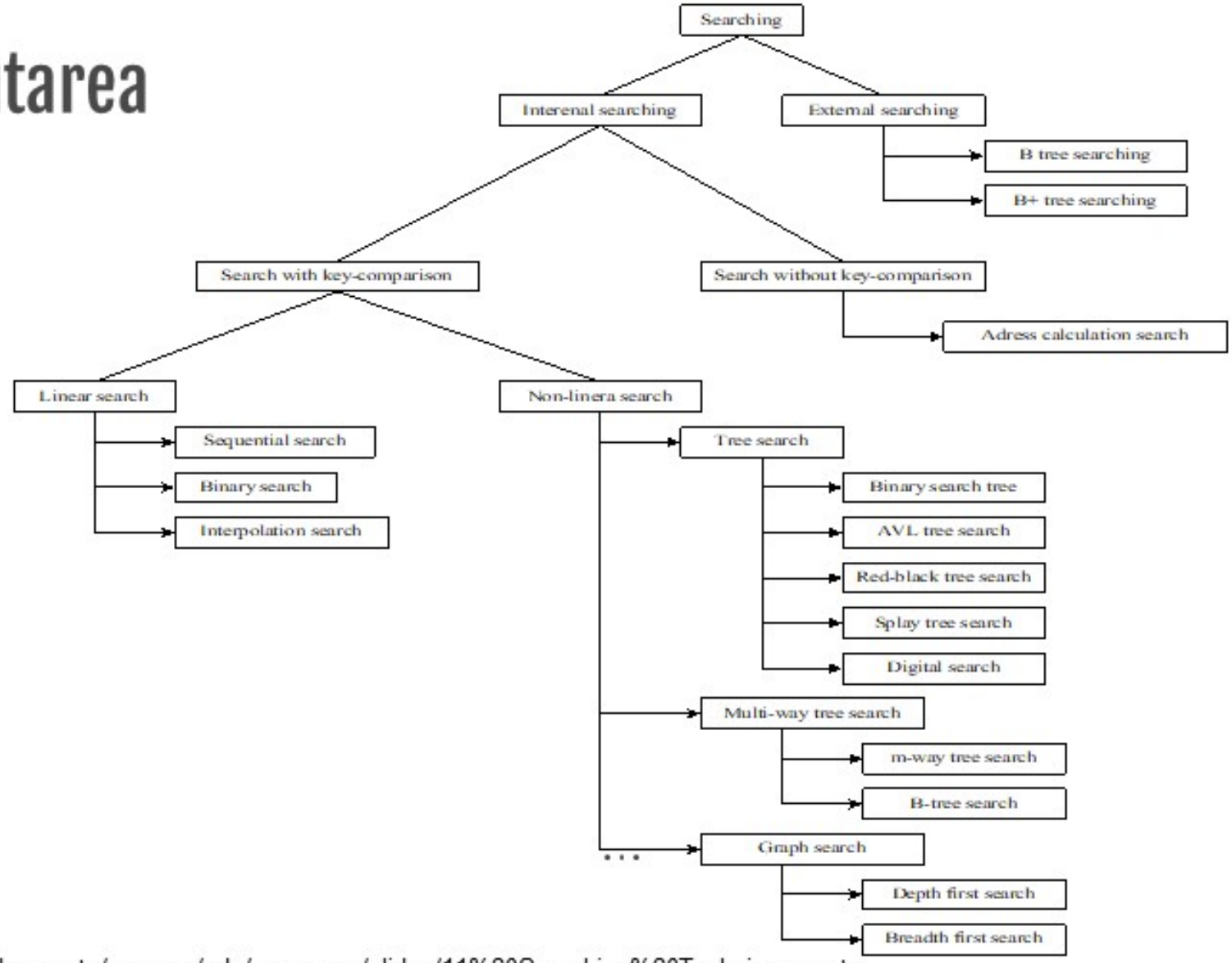


Cautarea

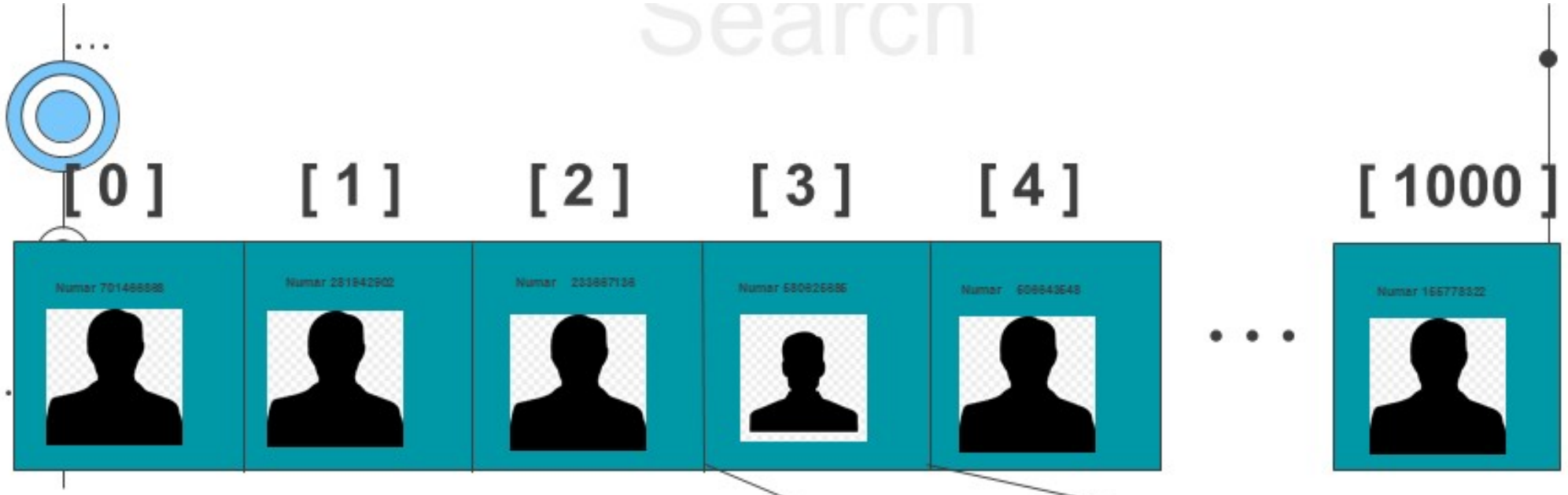
Problematica

- o listă de înregistrări
- fiecare înregistrare are o cheie asociată.
- Realizarea unui algoritm eficient pentru căutarea unei înregistrări care conține o anumită cheie.
- eficiența este cuantificată în termeni de analiză a timpului mediu (număr de comparații) pentru a prelua un articol.

Cautarea

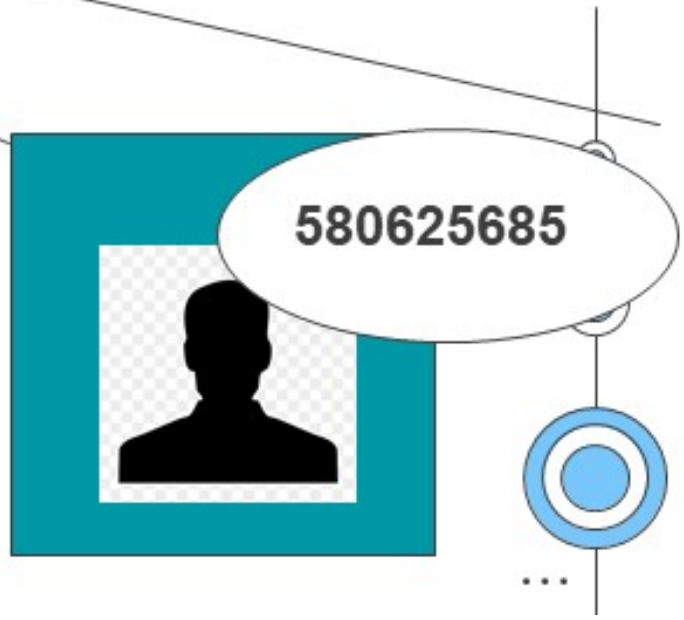


https://cse.iitkgp.ac.in/~dsamanta/courses/pds/resources/slides/11%20Searching%20Techniques.pptx



Fiecare înregistrare din listă are o cheie asociată.
În acest exemplu, cheile sunt numere de identificare.

Având în vedere o anumită cheie, cum putem prelua eficient înregistrarea din listă? …



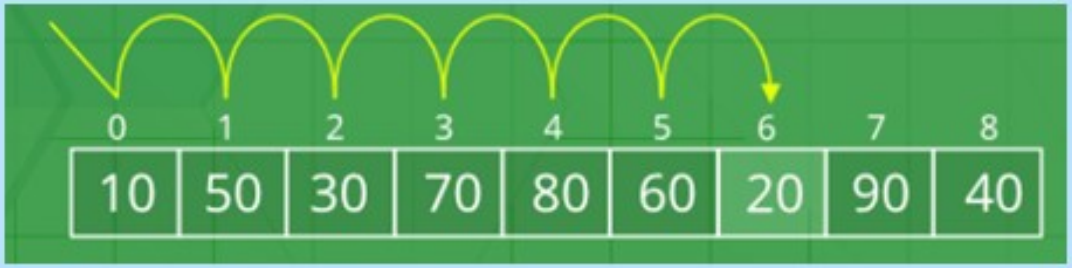
01

Cautarea secventiala

Cautarea secventiala

Căutarea secvențială este definită ca un algoritm de căutare liniară care începe la un capăt și trece prin fiecare element al unei serii de valori până când este găsit elementul dorit, în caz contrar căutarea continuă până la sfârșitul setului de date.

Căutarea valorii 20



Cautarea secventiala

- Se varifica gama de înregistrări, una câte una.
- Se cauta înregistrarea cu cheia potrivită.
- Căutarea se oprește când:

  - este găsită înregistrarea cu cheia potrivită, sau
  - când căutarea a examinat toate înregistrările fără succes.

Cautarea secventiala

Intrare: *arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170},* ***x = 110****;*
Iesire: *6*
Comentariu: *Element x is present at index 6*
Intrare: *arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170},* ***x = 175****;*
Iesire: *-1*
Comentariu: *Element x is not present in arr[]*

$gasit \leftarrow 0$
$i \leftarrow 0$
while $i < n$ **AND** $gasit == 0$ **do**
    if $x == v[i]$ **then**
        $gasit \leftarrow 1$
    else
        $i \leftarrow i + 1$
    end if
end while

```
int caut(int v[], int N, int x)
{
    int gasit=0,i;
    while (i <N && gasit == 0)
        if (v[i] == x)
            gasit = 1;
        else
            i++;

    return gasit;
}
```

Cautarea secventiala

Analiza complexitatii

Op.	Algoritm	Cost	Nr. rep.
1	$gasit \leftarrow 0$	1	1
2	$i \leftarrow 0$	1	1
3	**while** $i < n$ AND $gasit == 0$ **do**	1	$\tau_1(n) + 1$
4	**if** $x == v[i]$ **then**	1	$\tau_1(n)$
5	$gasit \leftarrow 1$	1	1
	else		
6	$i \leftarrow i + 1$	1	$\tau_1(n) - 1$
	end if		
	end while		

Cautarea secventiala

Analiza complexitatii

Obţinem $T(n) = 1 + 1 + (\tau_1(n) + 1) + \tau_1(n) + 1 + (\tau_1(n) - 1) = 3 * \tau_1(n)+3$. Şi în această situaţie vom calcula T(n) pentru cele două cazuri "extreme".

Cazul cel mai favorabil: elementul de căutat se găseşte pe prima poziţie din şir şi după o singură iteraţie se iese din ciclul while, atunci $\tau_1(n) = 1$, de unde rezultă $T(n) = 6$,
respectiv $T(n) \in O(1)$

Cazul cel mai nefavorabil: elementul de căutat nu aparţine şirului, de unde rezultă că în cadrul ciclului while vor fi n iteraţii şi $\tau_1(n) = n$. Atunci, $T(n) = 3*n+3$, $T(n) \in O(n)$

Cautarea secventiala

Cazul	Numaruld e comparatii	O( )	Comentariu
1	$T(n) = 6$	$T(n) = O(1)$	Cel mai favorabil
2	$T(n) = 3\ n+3$	$T(n) = O(n)$	Cel mai nefavorabil

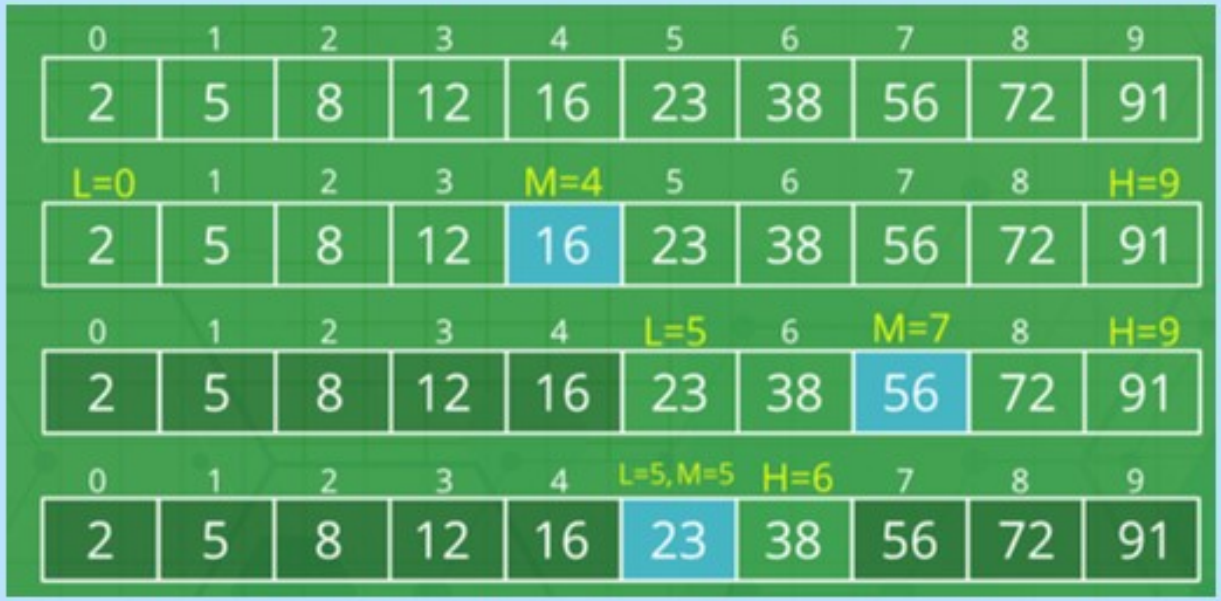
02

Cautarea binara si prin interpolare

Cautarea binara

Căutarea binară este un algoritm de căutare utilizat într-un vector sortat, împărțind în mod repetat intervalul de căutare la jumătate.

Căutarea valorii 23



23 > 16

23 < 56

23 = 23
gasit

Cautarea binara

Algoritmul de căutare binară:

- *Start cu elementul mijlociu al întregii serii ca o cheie de căutare.*
- *Dacă valoarea cheii de căutare este egală cu elementul, returnare index al cheii de căutare.*
- *Sau, dacă valoarea cheii de căutare este mai mică decât elementul din mijlocul intervalului, restrângeți intervalul la jumătatea din stanga mijlocului.*
- *Altfel, intervalul din dreapta mijlocului.*
- *Se verifica în mod repetat de la al doilea punct până când valoarea este găsită sau intervalul este gol (valoarea este inexistenta)*

Cautarea binara

Examplu: un vector.

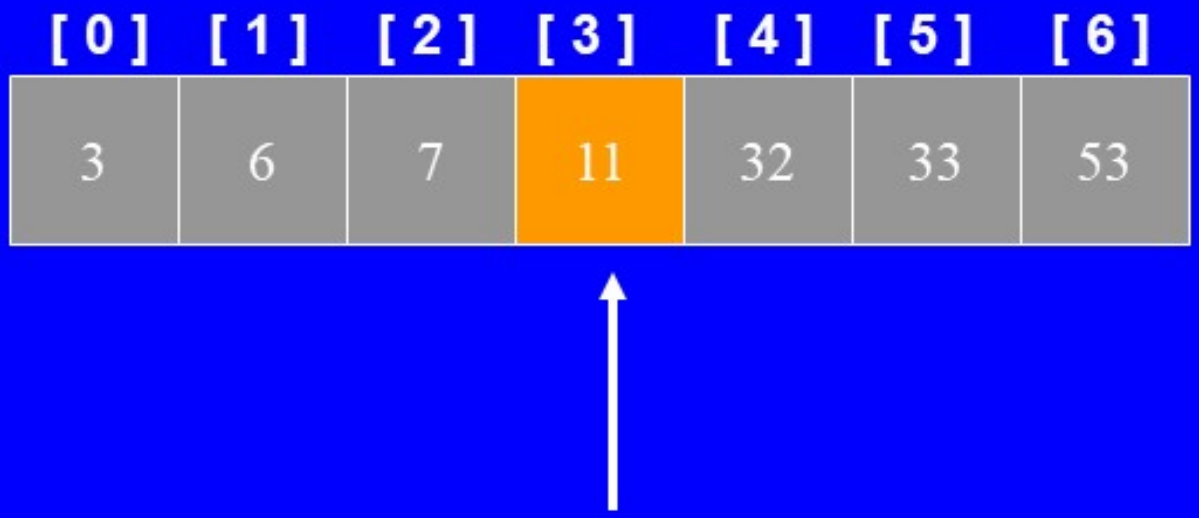
Cautare cheia 7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Cautarea binara

Examplu: un vector.

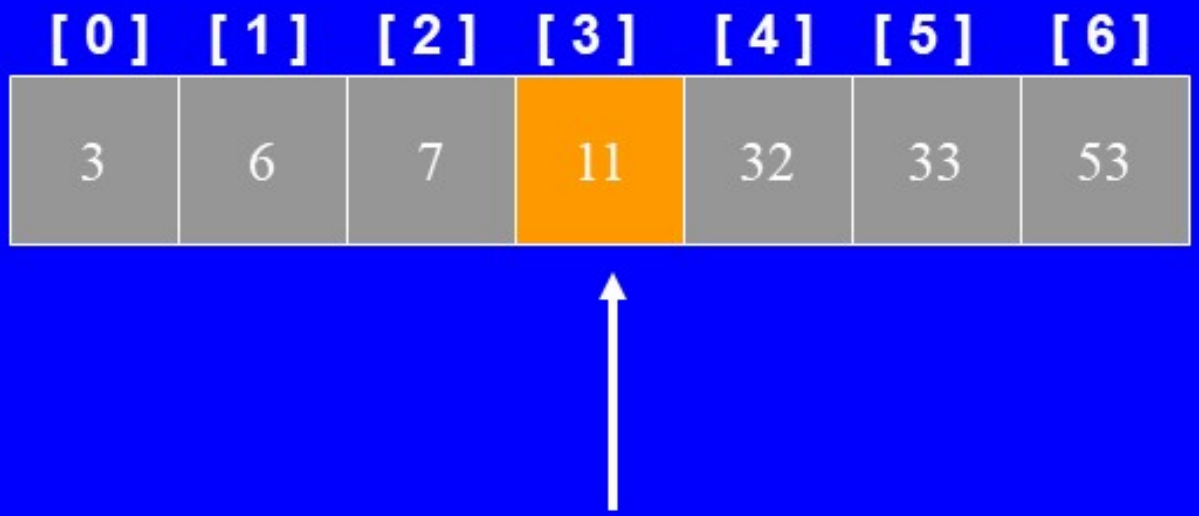
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

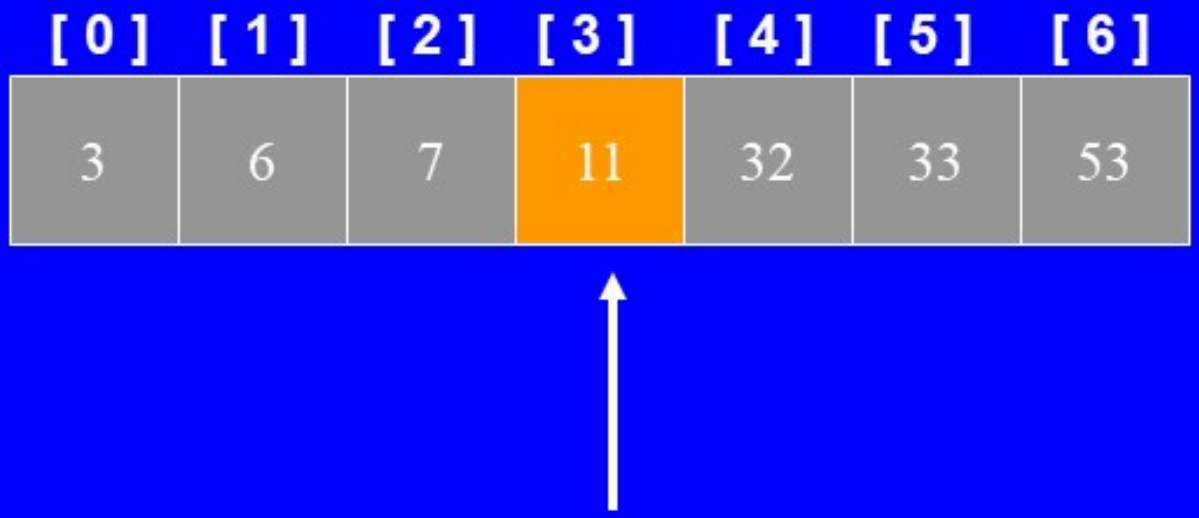
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

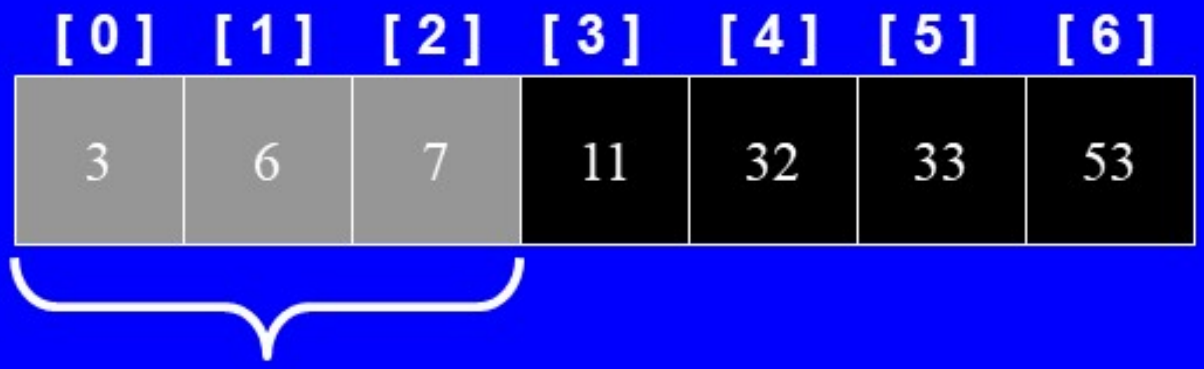
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

Cautare cheia 7.

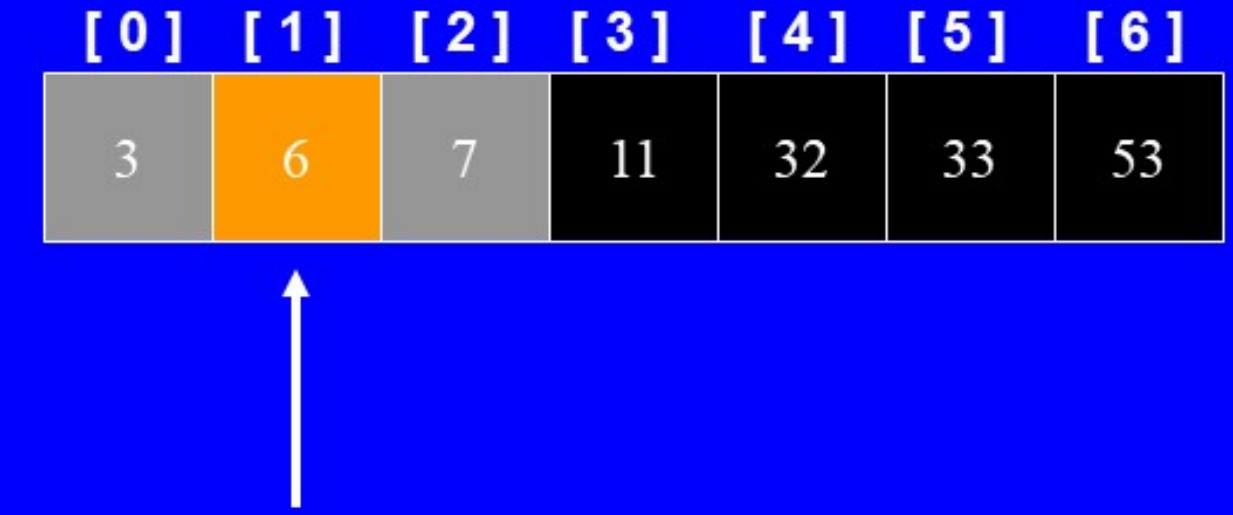


Cautarea continua cu elementele din stanga elementului din mijloc.

Cautarea binara

Examplu: un vector.

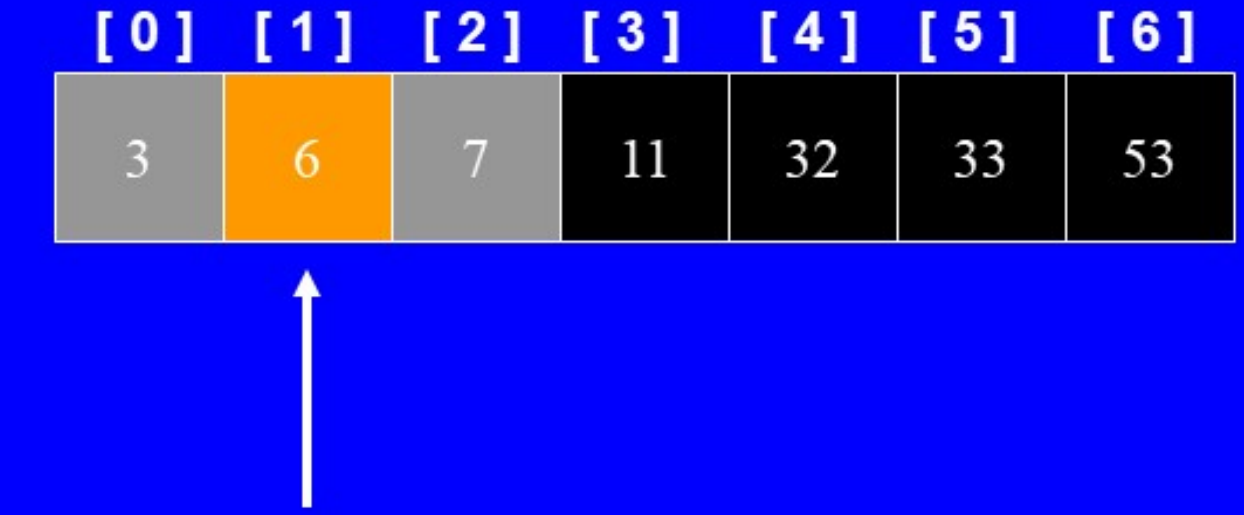
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

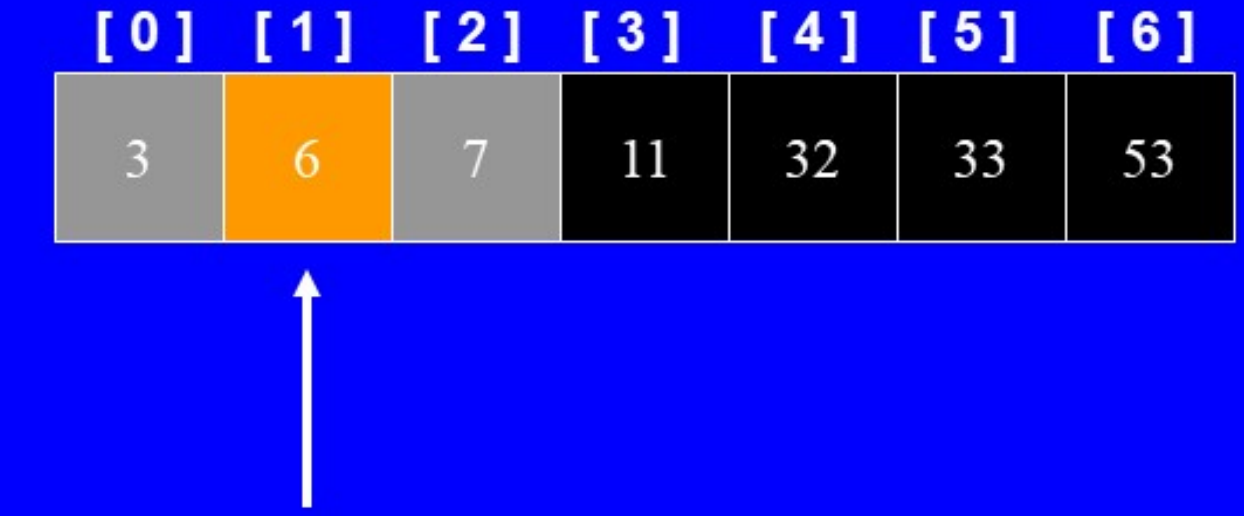
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

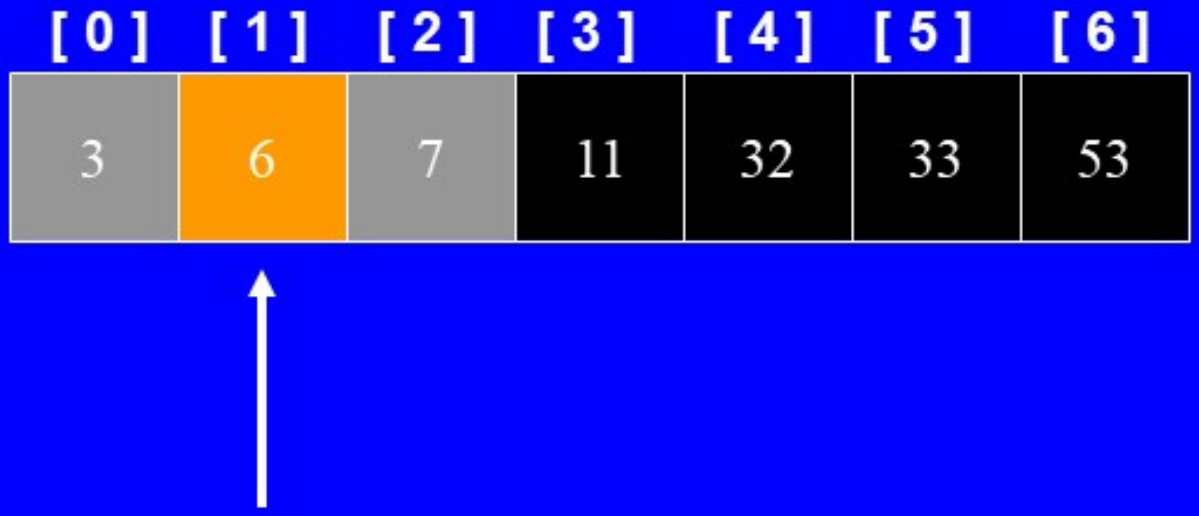
Cautare cheia 7.



Binary Search

Examplu: un vector.

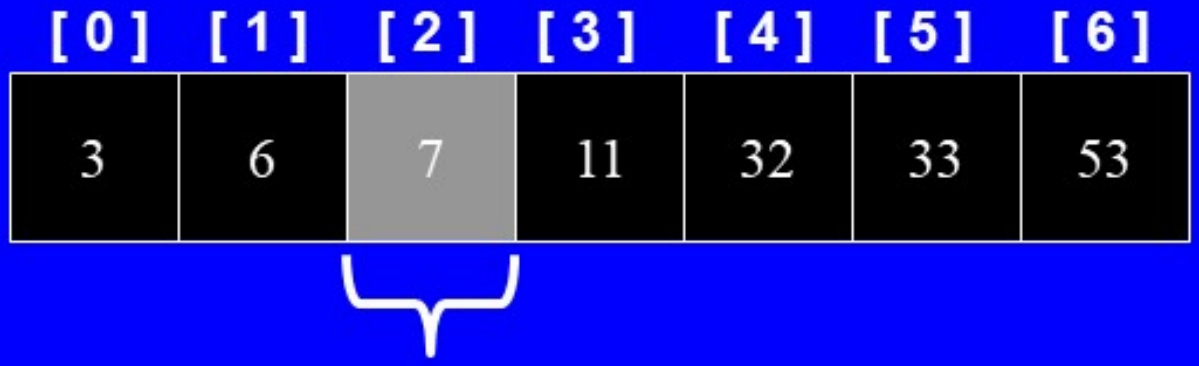
Cautare cheia 7.



Cautarea binara

Examplu: un vector.

Cautare cheia 7.

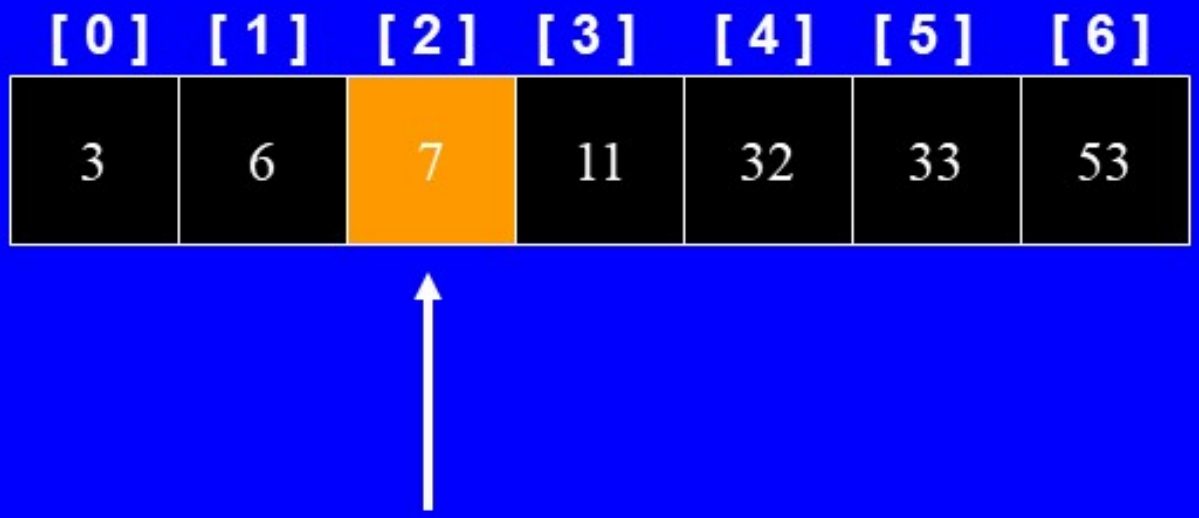


Determinare mijloc interval.

Cautarea binara

Examplu: un vector.

Cautare cheia 7.



Cautarea binara

```
cautareaBinara(arr, x, stanga, dreapta)
          repeat till stanga = dreapta
          mid = (stanga + dreapta)/2
          if (x == arr[mid]) then
                    return mid
          else
                    if (x > arr[mid]) then
                              stanga = mid + 1
                    else
                              dreapta = mid - 1
                    endif
          endif
end cautareBinara
```

Cautarea binara

Analiza complexitatii

Op.	Algoritm	Cost	Nr. rep.
1	$gasit \leftarrow 0$	1	1
2	$p \leftarrow 0$	1	1
3	$u \leftarrow n-1$	1	1
4	**while** $p <= u$ AND $gasit == 0$ **do**	1	$\tau_1(n)+1$
5	$m \leftarrow (p+u)/2$	1	$\tau_1(n)$
6	**if** $x == v[m]$ **then**	1	$\tau_1(n)$
7	$gasit \leftarrow 1$	1	1
	else		
8	**if** $x > v[m]$ **then**	1	$\tau_1(n) - 1$
9	$p \leftarrow m+1$	1	$\tau_2(n)$
	else		
10	$u \leftarrow m-1$	1	$\tau_1(n) - \tau_2(n) - 1$
	end if		
	end if		
	end while		

Cautarea binara

Analiza complexitatii

Obţinem: $T(n) = 5 * \tau_1(n) + 3$.

Cazul cel mai favorabil: elementul căutat se găseşte la mijlocul şirului iniţial, caz în care $\tau_1(n) = 1$. Rezultă: $T(n) = 8$, $T(n) \in O(1)$

Cazul cel mai nefavorabil: elementul căutat nu aparţine şirului, atunci, T(n) se poate scrie ca un şir recursiv astfel:

$T(1) = 0$

$T(n) = a * T(n/2) + b; \quad \forall \; n > 1;$

unde a şi b sunt constante reale. Prin urmare, timpul de execuţie are termenul dominant log n.

Cautarea prin interpolare

Căutarea prin interpolare găsește un anumit element calculând prin sondaj poziția. Spre deosebire de căutarea binară, folosește o estimare a poziției valorii țintă pe baza valorilor de la sfârșitul intervalului de căutare.

Algoritmul

mijl = St + ((x – A[St]) * (Dr – St) / (A[Dr] – A[St])

unde:

A = vectorul numerelor

St = indexul din stanga al listei

Dr = indexul din dreapta al listei

A[n] = valoarea stocată în vector la poziția n

Cautarea prin interpolare

Pasul 1 - Căutarea datelor începe cu elementul din mijlocul listei.
Pasul 2 - Dacă este o potrivire, returnează indexul valorii găsite.
Pasul 3 - Dacă nu, se returnează indexul mijlocului.
Pasul 4 – Se împarte lista folosind formula de interpolare și se determină noul mijloc.
Pasul 5 - Dacă datele sunt mai mari decât mijlocul, căutați în sublista superioară.
Pasul 6 - Dacă datele sunt mai mici decât mijlocul, căutați în sublista inferioară.
Pasul 7 - Repetați până se potrivește.

Cautarea prin interpolare

```
int cautareInterpolare(int vector[], int n, int x) {
    int stanga = 0, dreapta = n - 1, pos;
    while (stanga <= dreapta && x >= vector[stanga] && x <= vector[dreapta]) {
        if (stanga == dreapta) {
            if (vector[stanga] == x)
                return stanga;
            return -1;
        }
        pos = stanga + (((double)(dreapta - stanga) / (vector[dreapta] - vector[stanga])) * (x -
vector[stanga]));
        if (vector[pos] == x)
            return pos;
        if (vector[pos] < x)
            stanga = pos + 1;
        else
            dreapta = pos - 1;
    }
    return -1;
}
```

Cautarea prin interpolare

Exemplu: int vector[] = { 2, 4, **6**, 8, 10, 12, 14, 16, 18, 20}; int x = 6;

```
int poz = stanga + (((double)(dreapta - stanga) / (vector[dreapta] - vector[stanga])) * (x - vector[stanga]));

int poz = 0 + (((double)(9 - 0) / (20 - 2)) * (6 - 2));
poz = 0 + (9  / 18) * 4);
poz = 0 + (1 / 2) * 4);
poz = 2;
```

Exemplu: int vector[] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20}; int x = 200;

```
stanga <= dreapta && x >= vector[stanga] && x <= vector[dreapta])

0 <= 9 && x >= 2 && 200 <= 20)  - FALS
```

03

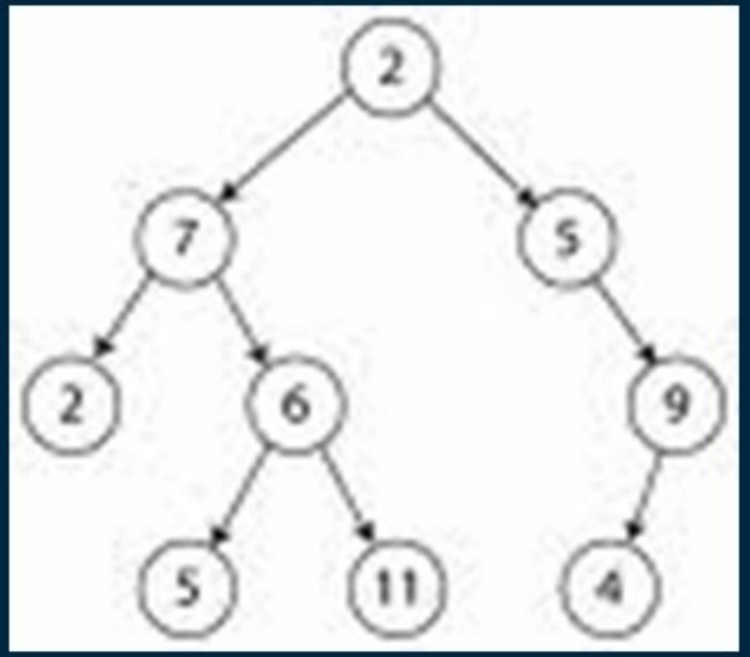
Cautarea Arbori Binari de Cautare

Operații Arbori Binar de Căutare

- Căutare

(1) Pornim de la radacina
(2) Cautam in ABC nivel cu nivel pana gasim un element sau am ajuns la o frunza.

Arbore Binar



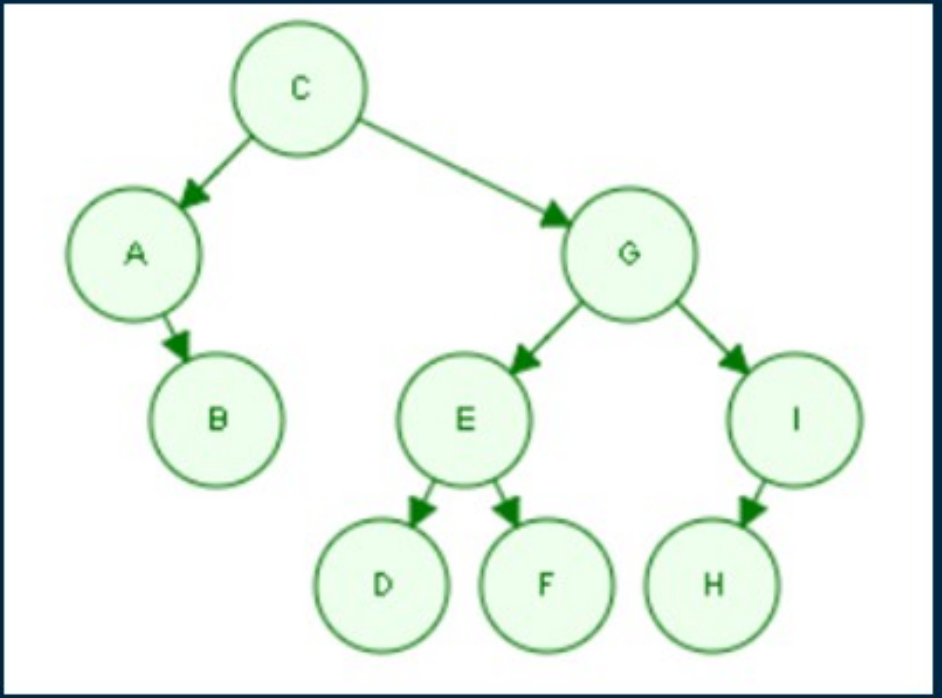
Operații Arbori Binar de Căutare

- Căutare

(1) Pornim de la radacina
(2) Comparăm valoarea de căutat cu cheia nodului.
Dacă valoarea de căutat este mai mică decât cheia nodului, vom continua căutarea în subarborele stâng,
În caz contrar, în subarborele drept
(3) Se continuă până când găsim valoarea de căutat sau nu mai există subarbori

Este aceasta cautare mai buna decat cautarea intr-o lista inlantuita?

Arbore Binar de Căutare



Operații Arbori Binar de Căutare

- Căutare

```
begin procedure caut(valoare)
        if nod == NULL then
                return NULL
        endif
        if valoare == nod >data then
                return nod >data
        endif
        if valoare < nod >data then
                return caut(nod >stânga)
        else
                return caut(nod >dreapta)
        endif
end procedure caut
```

Operații Arbori Binar de Căutare

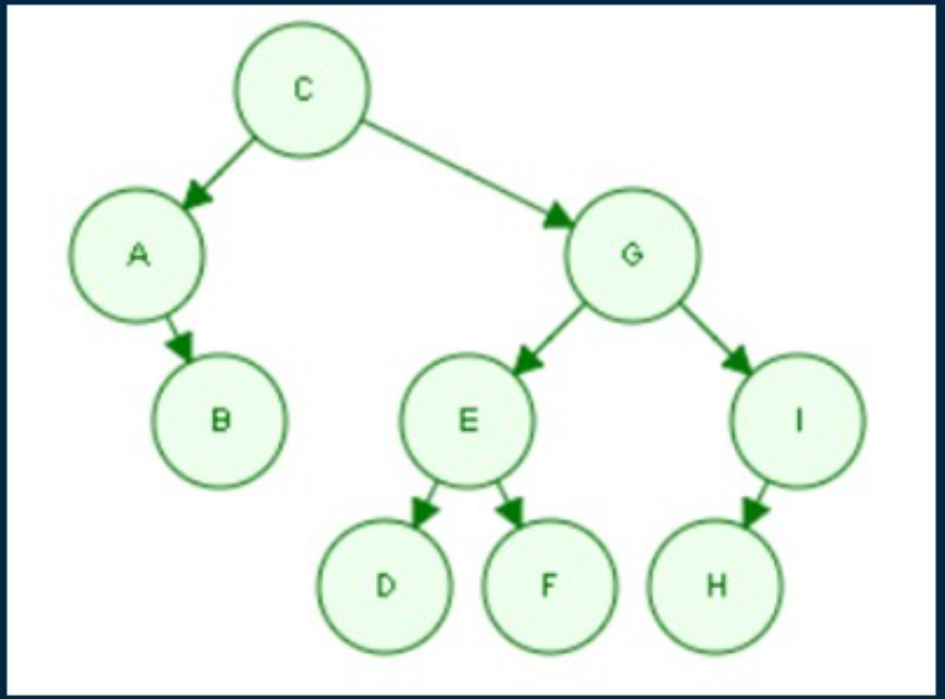
- Căutare

Arbore Binar de Căutare

Analiza complexitatii:

1. căutăm valoarea E
2. căutăm valoarea K

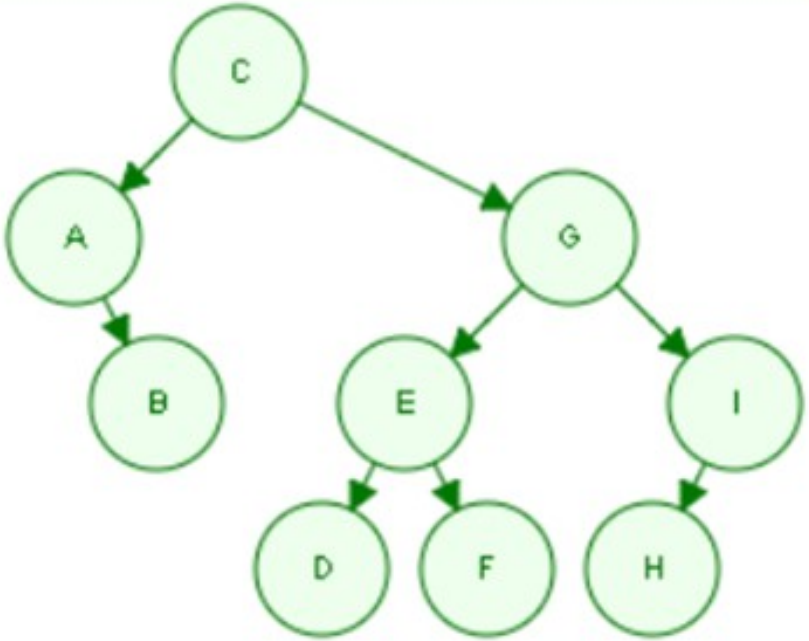
?



Analiza complexitatii – Cautare ABC

Complexitate O()

1. Cel mai favorabil caz: O(1)
   - cheia nodului de cautat este radacina
2. Cazul mediu: O(h) sau O(log n)
   - h este inaltimea arborelui
3. Cel mai nefavorabil: O(n)
   - arborele este debalansat

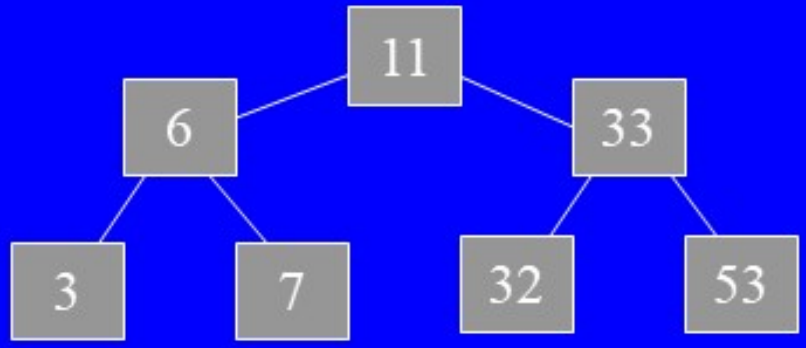


Arbore Binar de Cautare

Vectorul din exemplul anterior:

3	6	7	11	32	33	53

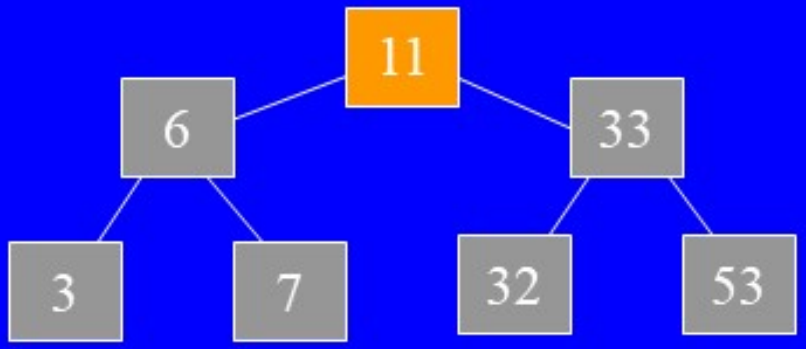
Arborele Binar de Cautare asociat



Cheia de cautat = 7

Determinare mijloc:

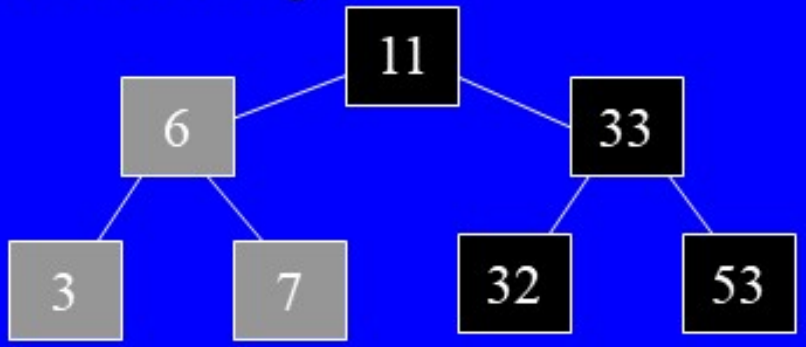
Start root:



Cheia = 7

Subvector stang:

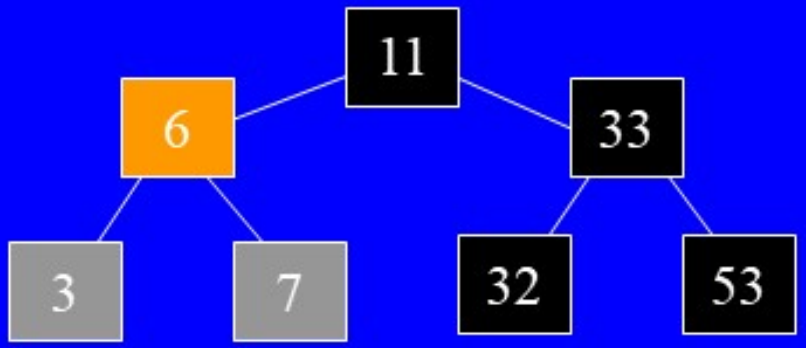
Subarbore stang:



Cheia = 7

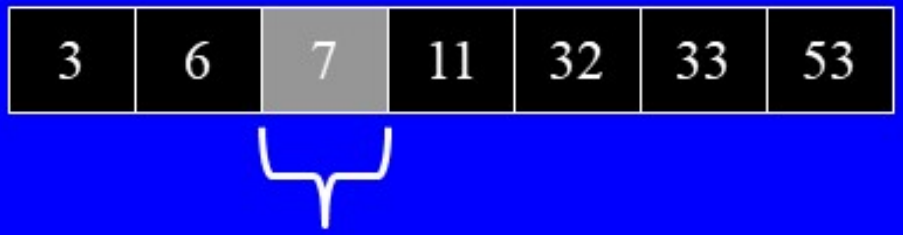
Determinare mijloc:

Radacina arborelui:

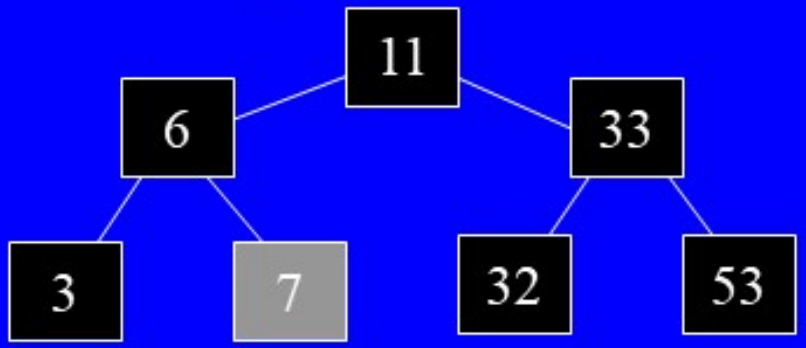


Cheia = 7

Subvector drept:



Subarbore drept:



Cheia = 7 GASIT

BST – exemplu

```
#include <iostream>
#include <set>
using namespace std;

int main() {
    set<int> bst;
    bst.insert(10);
    bst.insert(7);
    bst.insert(15);
    bst.insert(3);
    bst.insert(9);
    bst.insert(13);
    bst.insert(20);
    cout << "Dimensiunea BST: " << bst.size() << endl;
```

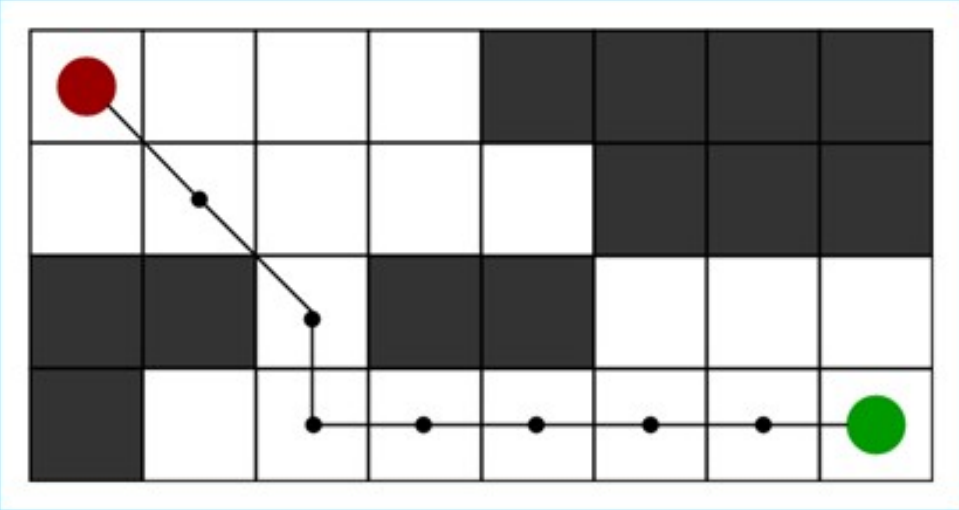
BST - exemplu

```
    cout << "parcurgere InOrder: ";
    for (auto x : bst)
        cout << x << " ";
    cout << endl;

    cout << "Caut elementul cu cheia 13: " << (*bst.find(13) == 13 ? "Gasit" : "Nu exista") << endl;
    cout << "Caut elementul cu cheia 77: " << (*bst.find(77) == 77 ? "Gasit" : "Nu exista") << endl;
    cout << "Stergere element 9: " << (bst.erase(9) == 1 ? "Sters" : "Nu exista") << endl;

    cout << "parcurgere InOrder dupa stergere traversal: ";
    for (auto x : bst)
        cout << x << " ";
    cout << endl;
    return 0;
}
```

```
Dimensiunea BST: 7
parcurgere InOrder: 3 7 9 10 13 15 20
Caut elementul cu cheia 13: Gasit
Caut elementul cu cheia 77: Nu exista
Stergere element 9: Sters
parcurgere InOrder dupa stergere traversal: 3 7 10 13 15 20
```

A* search

Pentru a aproxima calea cea mai scurtă în situații din viața reală, cum ar fi hărți, jocuri în care pot exista multe obstacole.



Algoritmul de căutare A* este unul dintre cele mai bune implementari utilizate în găsirea căilor și traversările grafurilor.

1) Distanta Manhattan

```
h = abs (current_cell.x - goal.x) +
    abs (current_cell.y - goal.y)
```

2) Distanta Diagonala

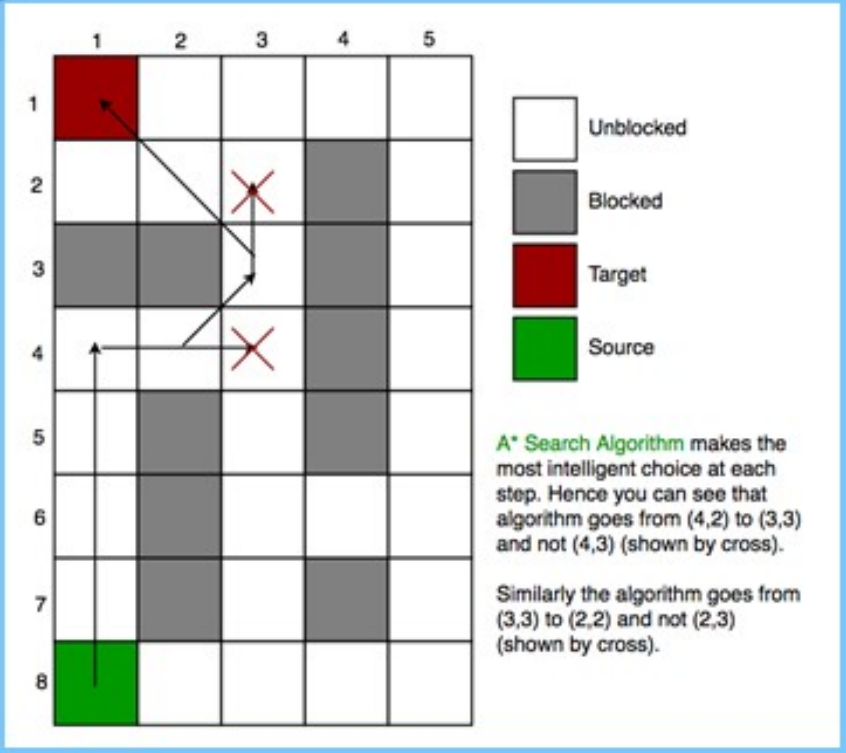
```
dx = abs(current_cell.x - goal.x)
dy = abs(current_cell.y - goal.y)
h = D * (dx + dy) + (D2 - 2 * D) *
min(dx, dy)
D = 1 iar D2 = sqrt(2)
```

3) **Distanta Euclidiana**

$$h = \sqrt{(current\_cell.x - goal.x)^2 + (current\_cell.y - goal.y)^2}$$

https://www.geeksforgeeks.org/a-search-algorithm/

A* search

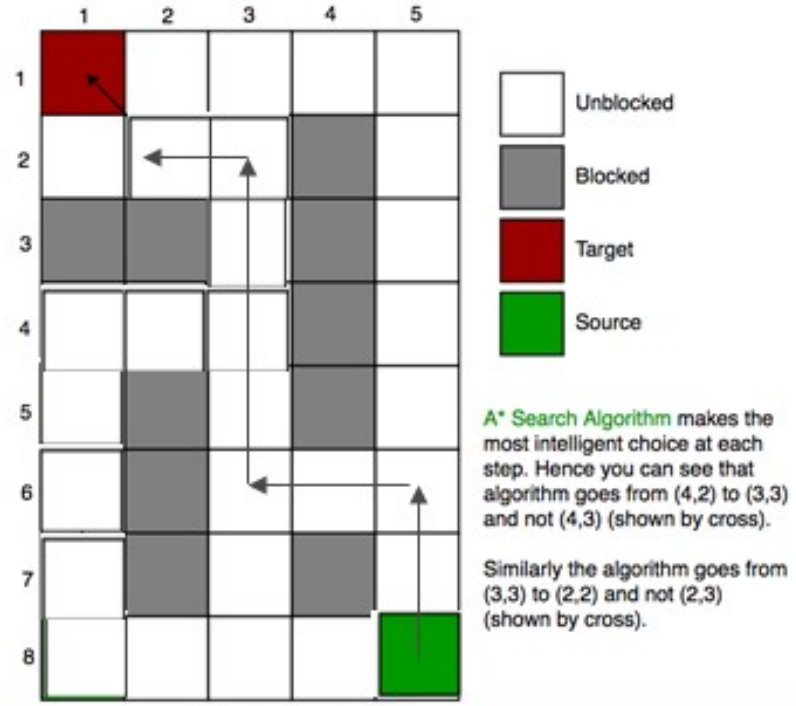



```
/* Tabla de joc
1--> Celula nu este blocata
0--> Celula este blocata */
int grid[ROW][COL]
    = { { 1, 1, 1, 1, 1 },
        { 1, 1, 1, 0, 1 },
        { 0, 0, 1, 0, 1 },
        { 1, 1, 1, 0, 1 },
        { 1, 0, 1, 0, 1 },
        { 1, 0, 1, 1, 1 },
        { 1, 0, 1, 0, 1 },
        { 1, 1, 1, 1, 1 }};
```

```
Exista drum catre celula destinatie

Drumul este -> (7,0) -> (6,0) -> (5,0) -> (4,0) -> (3,1) -> (2,2) -> (1,1) -> (0,0)
```

https://www.geeksforgeeks.org/a-search-algorithm/

A* search

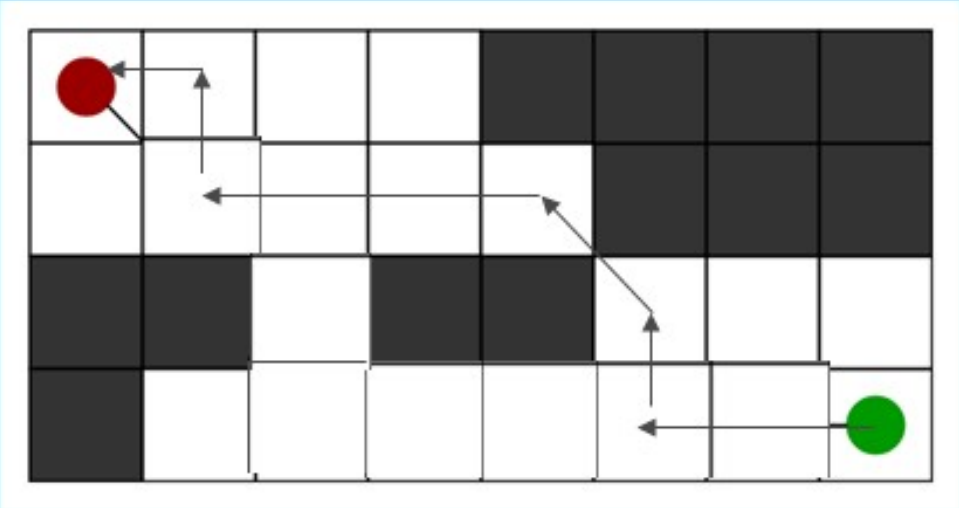



```
/* Tabla de joc
1--> Celula nu este blocata
0--> Celula este blocata */
int grid[ROW][COL]
    = { { 1, 1, 1, 1, 1 },
        { 1, 1, 1, 0, 1 },
        { 0, 0, 1, 0, 1 },
        { 1, 1, 1, 0, 1 },
        { 1, 0, 1, 0, 1 },
        { 1, 0, 1, 1, 1 },
        { 1, 0, 1, 0, 1 },
        { 1, 1, 1, 1, 1 }};
```

```
Exista drum catre celula destinatie

Drumul este -> (7,4) -> (6,4) -> (5,3) -> (4,2) -> (3,2) -> (2,2) -> (1,1) -> (0,0)
```

https://www.geeksforgeeks.org/a-search-algorithm/

A* search



```
/* Tabla de joc
1--> Celula nu este blocata
0--> Celula este blocata */
int grid[ROW][COL]
    = { { 1, 1, 1, 1, 0, 0, 0, 0 },
        { 1, 1, 1, 1, 1, 0, 0, 0 },
        { 0, 0, 1, 0, 0, 1, 1, 1 },
        { 0, 1, 1, 1, 1, 1, 1, 1 }};
```

```
Exista drum catre celula destinatie

Drumul este -> (3,7) -> (3,6) -> (2,5) -> (1,4) -> (1,3) -> (1,2) -> (0,1) -> (0,0)
```

https://www.geeksforgeeks.org/a-search-algorithm/

04

Cautarea
Tabele de dispersie

Tabele de dispersie

1. Cel mai favorabil caz:
   O(1)
2. Cazul nefavorabil
   O(c), c < n (c = nr. Coliziuni)

HASH-BASED SEARCH

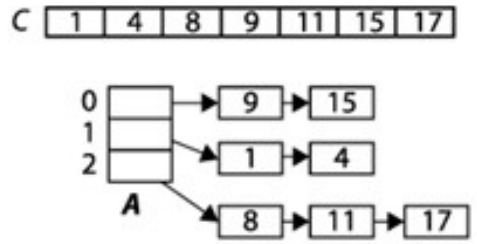
Best	Average	Worst
O (1)	O (1)	O (n)

Array

Hash

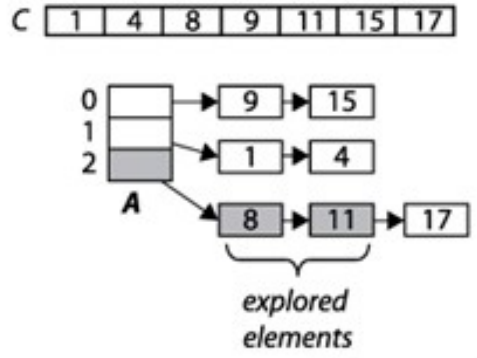
```
loadTable (size, C)
1.  A = new array of given size
2.  for i = 0 to n – 1 do
3.    h = hash(C[i])
4.    if (A[h] is empty) then
5.        A[h] = new Linked List
6.    add C[i] to A[h]
7.  return A
end

search (A, t)
1.  h = hash (t)
2.  list = A[h]
3.  if (list is empty) then
4.    return false
5.  if  (list contains t) then
6.    return true
7.  return false
end
```

A = loadTable (3, C)



A handles collisions with lists
hash (e) = remainder of e ÷ 3

search (A, 11)
Note that remainder of 11 ÷ 3 is 2



Rezumat

Căutarea secvențială:
caz mediu O(n)

Căutarea binară:
caz mediu O(log n)

Mulțumesc

pentru atenție!

dorin.iordache@365.univ-ovidius.ro