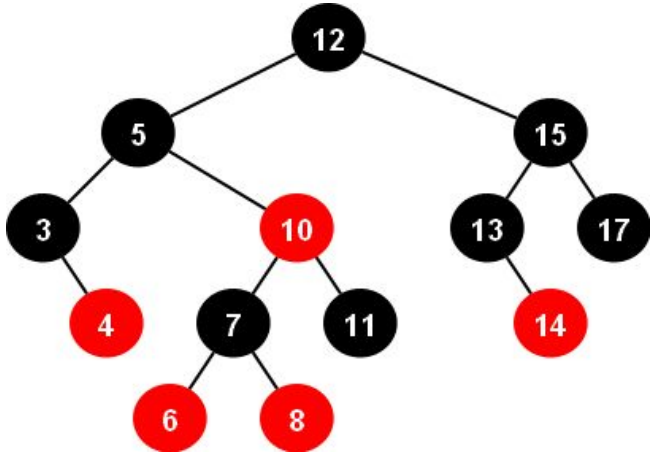
ARBORI BINARI ECHILIBRAȚI

Arbori Binari de Căutare Construiți Aleator

Teorema 13.6. Înălţimea medie a unui arbore binar de căutare construit aleator cu n chei distincte este O(lg n).

Red Black Trees

- Reguli:
  - Fiecare nod e fie roșu, fie negru
  - Rădăcina e mereu neagră
  - Nu putem avea două noduri adiacente roșii
  - Orice drum de la un nod la un descendent NULL are același număr de noduri negre



Red Black Trees

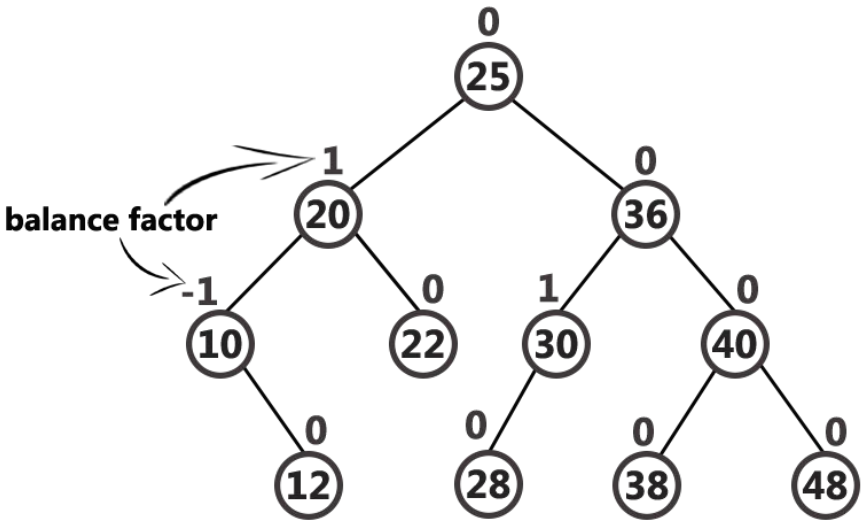
- Red Black Trees (nu veți avea la examen)
  - MIT Video
  - MIT Lecture Notes

Red Black Trees

~~Red Black Trees~~ AVL

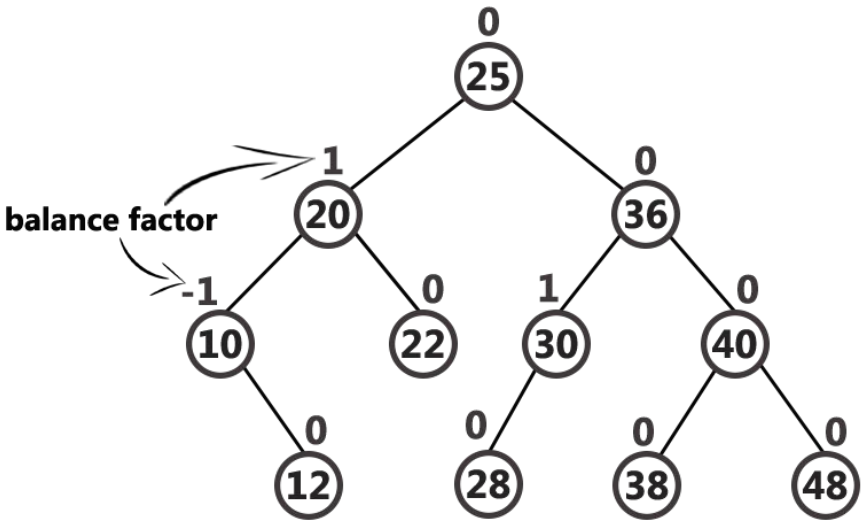
AVL

- Construcția AVL-urilor:
  - pentru fiecare nod, diferența dintre înălțimile fiilor drept și stâng trebuie să fie maxim 1



AVL

- Factorul de echilibru al unui nod:
  - `BF(X) = h(subarbore_drept(X)) - h(subarbore_stang(X))`



AVL - Reechilibrare

- Rotații:

1) **Rotație stânga-stânga**
   - când un nod este inserat în stânga subarborelui stâng
   - se realizează o rotație la dreapta
2) **Rotație dreapta-dreapta**
   - când un nod este inserat în dreapta subarborelui drept
   - se realizează o rotație la stânga
3) **Rotație dreapta-stânga**
   - când un nod este inserat la dreapta subarborelui stâng
   - se realizează două rotații
4) **Rotație stânga-dreapta**
   - când un nod este inserat la stânga subarborelui drept
   - se realizează două rotații

Mai multe informații: https://www.guru99.com/avl-tree.html

AVL

AVL (veți avea la examen)

- Video (MIT).
- Lecture Notes

SKIP LISTS

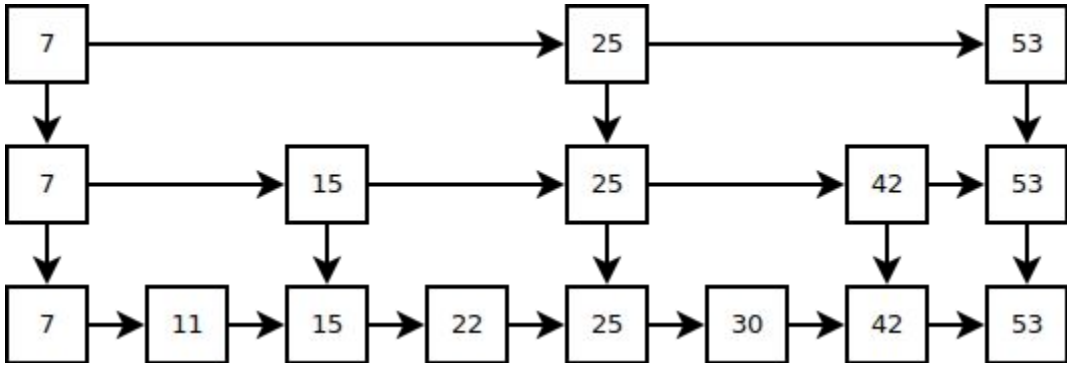
Skip Lists

- Sunt structuri de date echilibrate
- Alte structuri de date eficiente (**log n** sau mai bun):
  - Tabele de dispersie (hash tables) - nu sunt sortate
  - Heap-uri - nu putem căuta în ei
  - Arbori binari echilibrați (AVL, Red Black Trees)

- Ajută la o căutare rapidă
- Elementele sunt sortate!

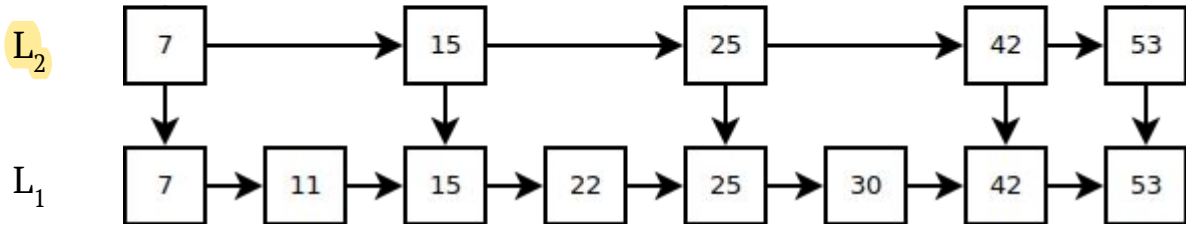
Skip Lists

- Sunt implementate ca liste înlănțuite
- Ideea de implementare:
  - este extinsă pe mai multe nivele (mai multe liste înlănțuite)
  - la fiecare nivel adăugat, **sărim peste o serie de elemente** față de nivelul anterior
  - nivelele au legături între ele



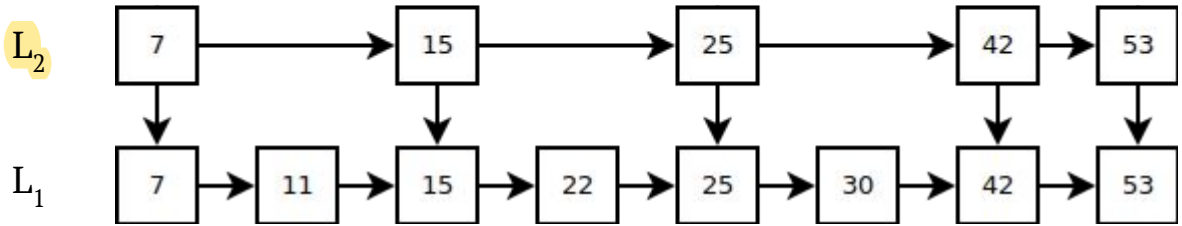
Skip Lists

- Să presupunem că avem doar 2 liste
  - Cum putem alege ce elemente ar trebui transferate în nivelul următor?



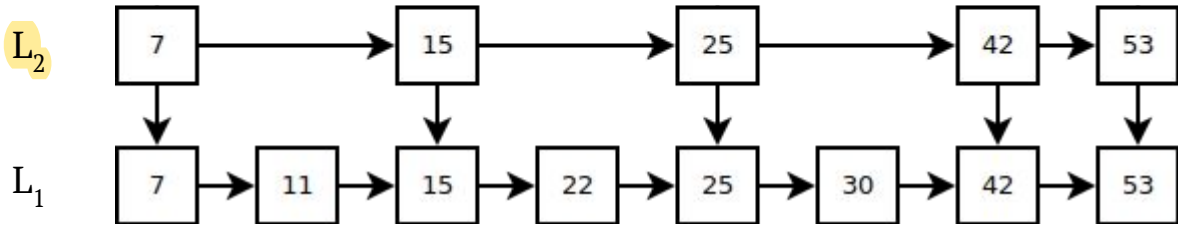
Skip Lists - 2 liste

- Cum putem alege ce elemente ar trebui transferate în nivelul următor?
  - Cea mai bună metodă: elemente egal depărtate
  - Costul căutării = $|L_2| + (|L_1| / |L_2|) = |L_2| + (n / |L_2|)$
  - Când este minim acest cost?



Skip Lists - 2 liste

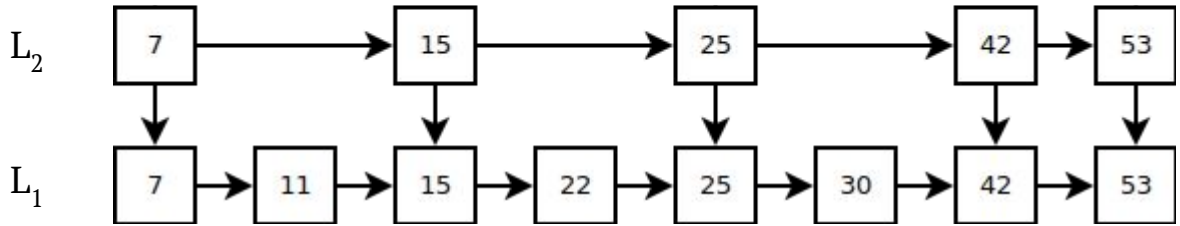
- Cum putem alege ce elemente ar trebui transferate în nivelul următor?
  - Cea mai bună metodă: elemente egal depărtate
  - Costul căutării = $|L_2| + (|L_1| / |L_2|) = |L_2| + (n / |L_2|)$
  - Când este minim acest cost?
    - Când $|L_2| = n / |L_2|$ ⇒ **$|L_2| = \text{sqrt}(n)$**



Skip Lists - 2 liste

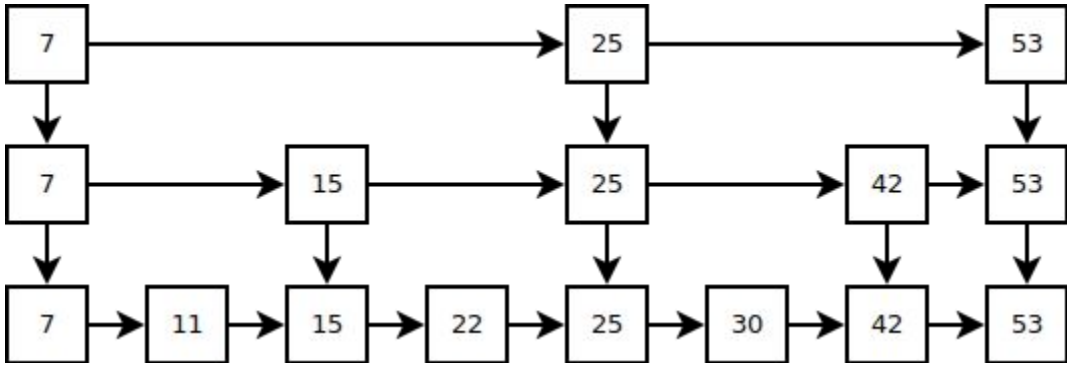
- Cum putem alege ce elemente ar trebui transferate în nivelul următor?
  - Cea mai bună metodă: elemente egal depărtate
  - Costul căutării = $|L_2| + (|L_1| / |L_2|) = |L_2| + (n / |L_2|)$
  - Când este minim acest cost?
    - Când $|L_2| = n / |L_2|$ ⇒ **$|L_2| = sqrt(n)$**
  - Deci, costul minim pentru căutare este sqrt(n) + n / sqrt(n) = 2*sqrt(n)

  - Complexitate: O(sqrt(n)) → seamănă un pic cu **Batog**



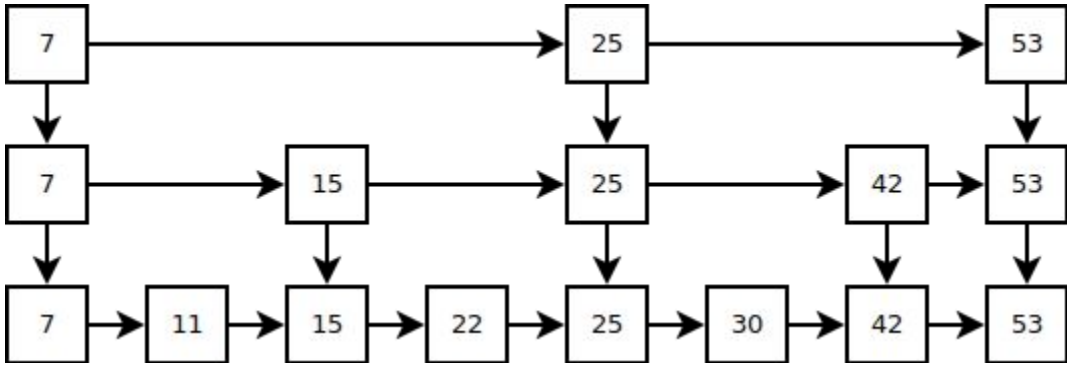
Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?



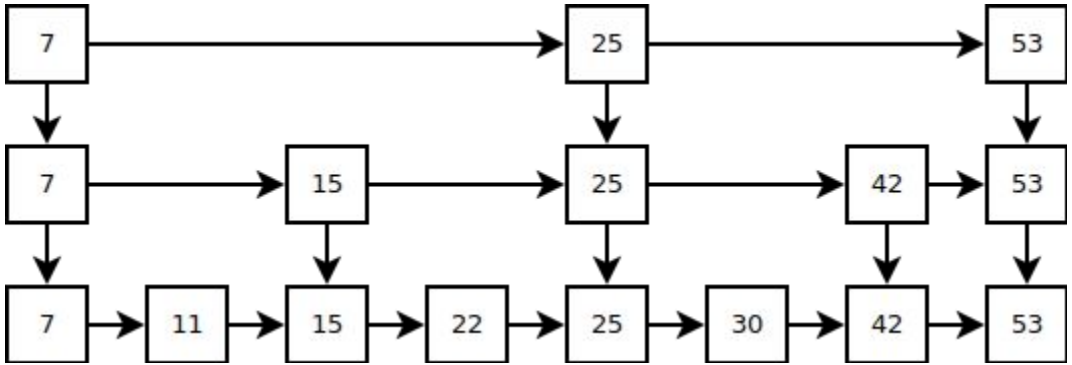
Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?
  - Costul căutării se modifică
  - 2 liste: $2 * \sqrt{n}$
  - 3 liste: ?



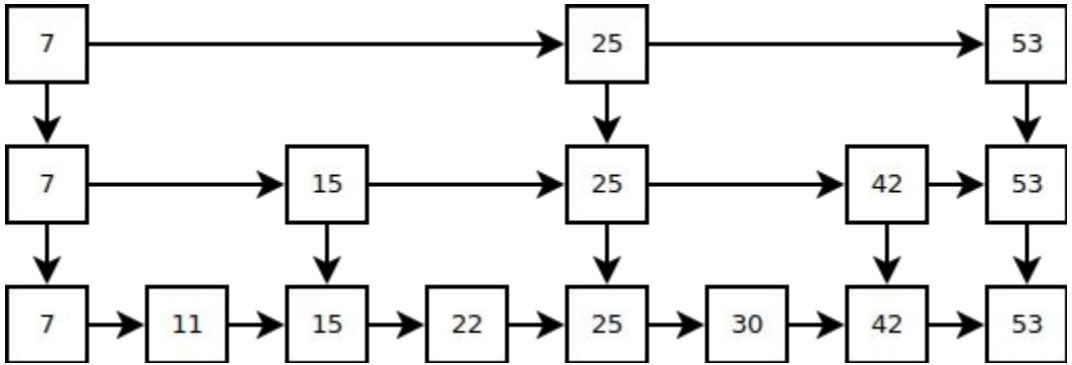
Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?
  - Costul căutării se modifică
  - 2 liste: $2 * \sqrt{n}$
  - 3 liste: $3 * \sqrt[3]{n}$
  - k liste: $k * \sqrt[k]{n}$



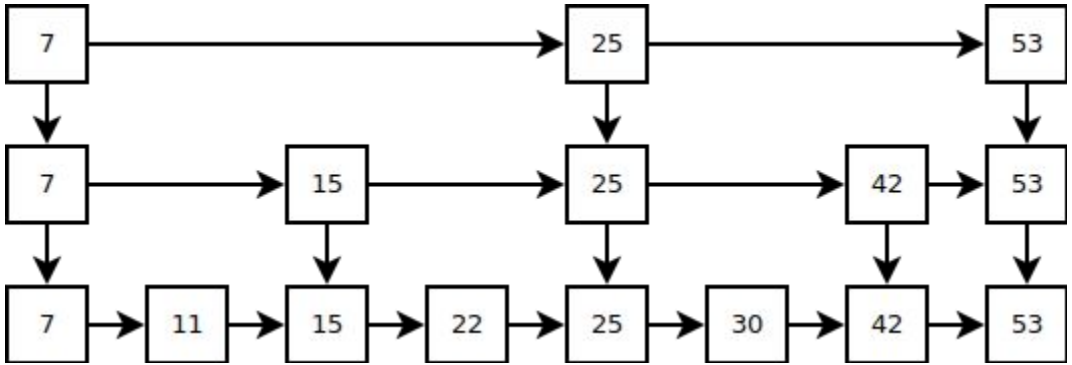
Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?
  - Costul căutării se modifică
  - 2 liste: $2 * \sqrt{n}$
  - 3 liste: $3 * \sqrt[3]{n}$
  - k liste: $k * \sqrt[k]{n}$
  - logn liste: $logn * \sqrt[logn]{n}$



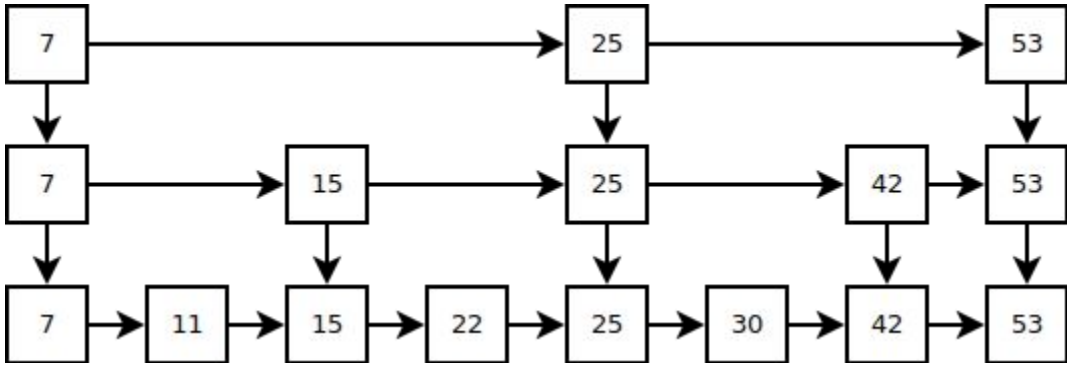
Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?
  - Costul căutării se modifică
  - 2 liste: $2 * \sqrt{n}$
  - 3 liste: $3 * \sqrt[3]{n}$
  - k liste: $k * \sqrt[k]{n}$
  - logn liste: $logn * \sqrt[logn]{n}$ = ? Cu cât este egal ? $\sqrt[logn]{n}$



Skip Lists

- Ce se întâmplă când avem mai mult de 2 liste înlănțuite?
  - Costul căutării se modifică
  - 2 liste: $2 * \sqrt{n}$
  - 3 liste: $3 * \sqrt[3]{n}$
  - k liste: $k * \sqrt[k]{n}$
  - logn liste: $logn * \sqrt[logn]{n}$ = $2 * logn$ ⇒ Complexitate: **O(logn)** !



Skip Lists - Căutare

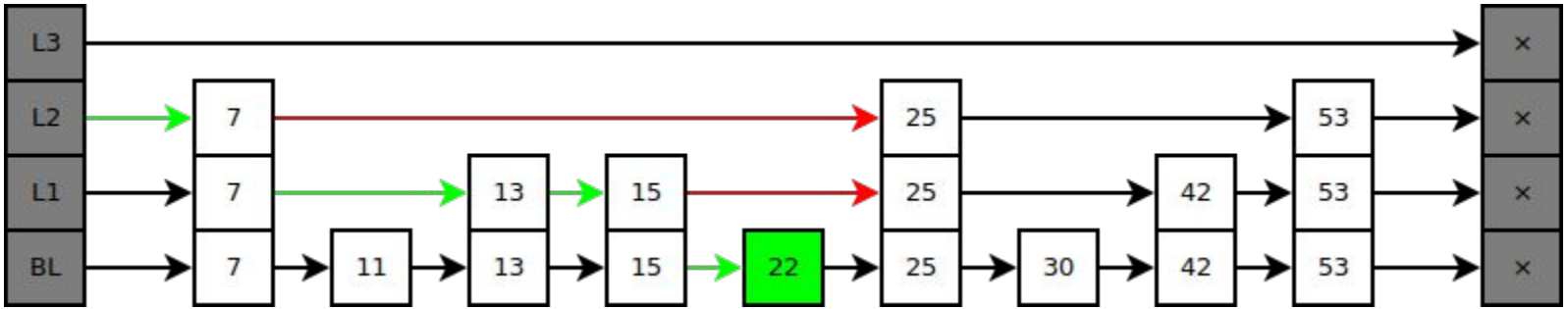
1) Începem căutarea cu primul nivel (cel mai de sus)
2) Avansăm în dreapta, până când, dacă am mai avansa, am merge prea departe (adică elementul următor este prea mare)
3) Ne mutăm în următoarea listă (mergem în jos)
4) Reluăm algoritmul de la pasul 2)

Skip Lists - Căutare

1) Începem căutarea cu primul nivel (cel mai de sus)
2) Avansăm în dreapta, până când, dacă am mai avansa, am merge prea departe (adică elementul următor este prea mare)
3) Ne mutăm în următoarea listă (mergem în jos)
4) Reluăm algoritmul de la pasul 2)

Exemplu: search(22)

Complexitate: O(logn)



Skip Lists - Inserare

- Vrem să inserăm elementul x
- **Observație:** Lista de jos trebuie să conțină toate elementele!

- x trebuie să fie inserat cu siguranță în nivelul cel mai de jos
  - căutăm locul lui x în lista de jos → search(x)
  - adăugăm x în locul găsit în lista cea mai de jos
- Cum alegem în câte liste să fie adăugat?

Skip Lists - Inserare

- Vrem să inserăm elementul x
- x trebuie să fie inserat cu siguranță în nivelul cel mai de jos
- Cum alegem în ce altă listă să fie adăugat?
  - Alegem metoda probabilistică:
    - aruncăm o monedă
      - dacă pică Stema - o adăugăm în lista următoare și aruncăm din nou moneda
      - dacă pică Banul - ne oprim
    - probabilitatea să fie inserat și la nivelul următor: ½
- În medie:
  - ½ elemente nepromovate
  - ¼ elemente promovate 1 nivel
  - ⅛ elemente promovate 2 nivele
  - etc.
- Complexitate: O(logn)

Skip Lists - Ștergere

- Ștergem elementul x din toate listele care îl conțin
- Complexitate: O(logn)

Skip Lists

- Articol
- Video MIT
- Notes

Bibliografie

http://ticki.github.io/blog/skip-lists-done-right/

https://www.guru99.com/avl-tree.html

https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

MIT lecture notes on skip lists

Esoteric Data Structures: Skip Lists and Bloom Filters - Stanford University

???

```
sol = 0;
for (t = 1 << 30; t > 0; t>>=1) {
    if (sol + t < v.size() && v[sol + t] <= x)
        sol += t;
}
```

???

```
sol = 0;
for (t = 1 << 30; t > 0; t>>=1) {
    if (sol + t < v.size() && v[sol + t] <= x)
        sol += t;
}
```

x= 32

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	7	11	20	24	28	30	32	44	49	62	68	82	84	93	97

???

```
sol = 0; x = 32;
for (t = 1 << 30; t > 0; t>>=1) {
    if (sol + t < v.size() && v[sol + t] <= x)
        sol += t;
}
```

0	1	2	3	**4**	5	**6**	**7**	8	9	10	11	12	13	14	15
3	7	11	20	24	28	30	**32**	44	49	62	68	82	84	93	97

Căutare binară

```
sol = 0; x = 32;
for (t = 1 << 30; t > 0; t>>=1) {
    if (sol + t < v.size() && v[sol + t] <= x)
        sol += t;
}
```

Complexitate?

0	1	2	3	**4**	5	**6**	**7**	8	9	10	11	12	13	14	15
3	7	11	20	24	28	30	**32**	44	49	62	68	82	84	93	97

Căutare binară

```
sol = 0; x = 32;
for (t = 1 << 30; t > 0; t>>=1) {
    if (sol + t < v.size() && v[sol + t] <= x)
        sol += t;
}
```

Complexitate **O(log n)** - recomand cu căldură :)

0	1	2	3	**4**	5	**6**	**7**	8	9	10	11	12	13	14	15
3	7	11	20	24	28	30	**32**	44	49	62	68	82	84	93	97