ARBORI BINARI ECHILIBRAȚI

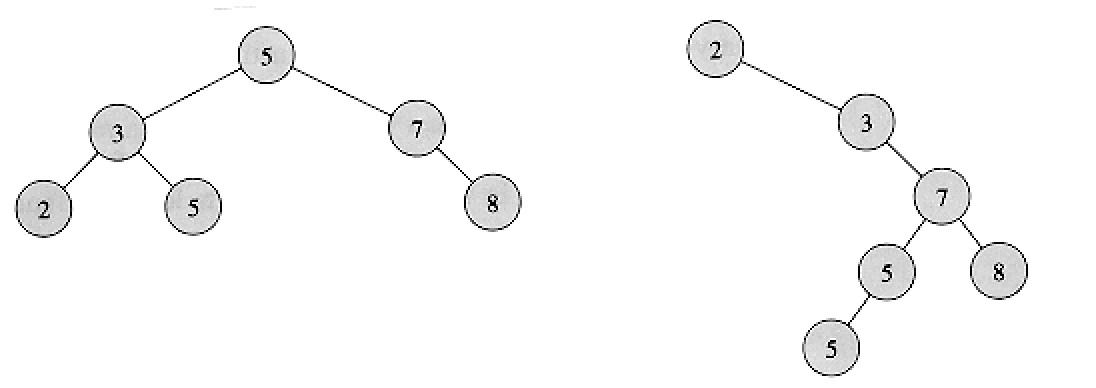
Organizatorice

Arbore binar de căutare

Un **arbore binar de căutare** este un arbore **binar** care satisface următoarea proprietate:

Pentru un nod x:

- Dacă **y** este un nod din subarborele *stâng* al lui **x**, atunci cheie[y] ≤ cheie[x]
- Dacă **y** este un nod din subarborele *drept* al lui **x**, atunci cheie[x] ≤ cheie[y]



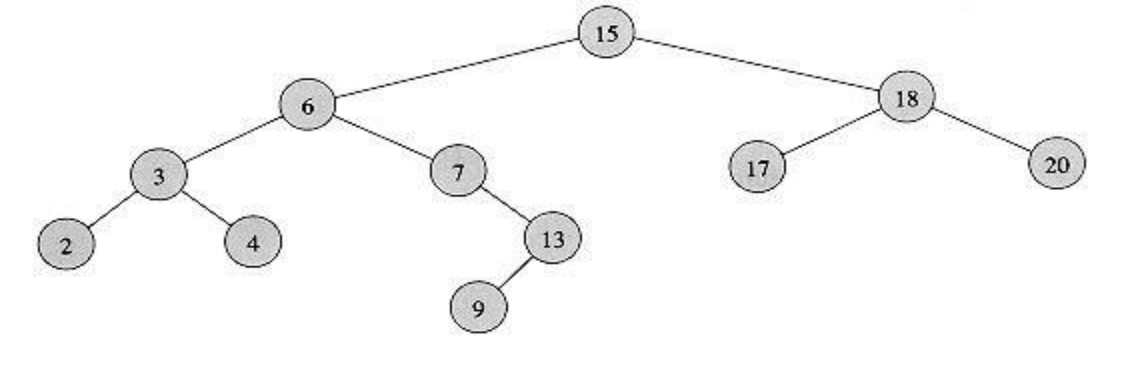
Arbori Binari de Căutare Construiți Aleator

- Amestecăm bine de tot și inserăm elementele în arborele binar de căutare. Ce înălțime va avea?

Arbori Binari de Căutare Construiți Aleator

Lema 13.3. Notăm cu T arborele care rezultă prin inserarea a n chei distincte $k_1, k_2, ..., k_m$ (în ordine) într-un arbore binar de căutare iniţial vid. Cheia k_i este un strămoş al cheii k_m în T, pentru $1 \leq i < j \leq n$, dacă şi numai dacă:

- **$k_i = \min\{ k_m : 1 \leq l \leq i$ şi $k_m > k_m \}$** // k_i este cel mai mic număr mai mare decât k_m din primele i, practic în procesul de inserare a lui j vom ajunge în k_i și vom merge în stânga
- SAU **$k_i = \max\{ k_m : 1 \leq l \leq i$ şi $k_m < k_m \}$**
  - 13 e fiu al lui 7 (până la momentul inserării lui 13, 7 era cel mai mare număr mai mic)
  - Ulterior, proprietatea e valabilă și pentru 9 și 14.
  - Ce trebuia să se întâmple ca 18 să fie fiu al lui 7?



Arbori Binari de Căutare Construiți Aleator

Demonstrație:

'⇒': Presupunem că k_i este un strămoş al lui k_m . Notăm cu T_i arborele care rezultă după ce au fost inserate în ordine cheile $k_1, k_2, \ldots, k_i$. Drumul de la rădăcină la nodul k_i în T_i este acelaşi cu drumul de la rădăcină la nodul k_i în T. De aici, rezultă că, dacă s-ar insera în arborele T_i nodul k_m, acesta (k_m) ar deveni fie fiu stâng, fie fiu drept al nodului k_i. Prin urmare (vezi exerciţiul 13.2-6), k_i este fie cea mai mică valoare dintre $k_1, k_2, ..., k_i$ care este mai mare decât k_m, fie cea mai mare valoare dintre cheile $k_1, k_2, ..., k_i$ care este mai mică decât k_m .

'⇐': Presupunem că k_i este cea mai mică valoare dintre $k_1, k_2, ..., k_i$ care este mai mare decât k_m . (Cazul când k_i este cea mai mare cheie dintre $k_1, k_2, ..., k_i$ care este mai mică decât k_m se tratează simetric). Compararea cheii k_m cu oricare dintre cheile de pe drumul de la rădăcină la k_i în arborele T produce aceleaşi rezultate ca şi compararea cheii k_i cu cheile respective. Prin urmare, pentru inserarea lui k_m, se va parcurge drumul de la rădăcină la k_i, apoi k_m se va insera ca descendent al lui k_i.

Arbori Binari de Căutare Construiți Aleator

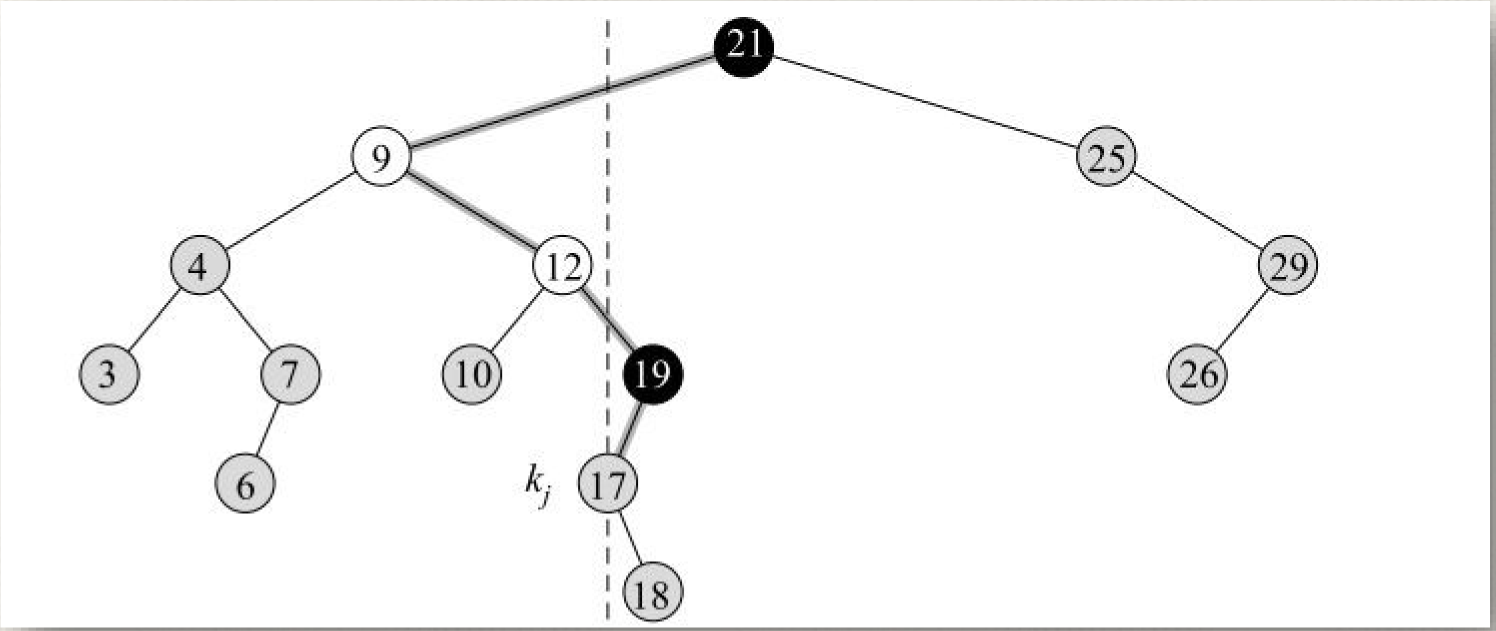
Corolarul 13.4. Fie T arborele care rezultă prin inserarea a n chei distincte k_1, k_2, ..., k_m (în ordine) într-un arbore binar de căutare inițial vid. Pentru o cheie k_m dată, cu $1 \leq j \leq n$, definim mulţimile:

- $G_m = \{ k_i : 1 \leq i < j$ şi $k_m > k_i > k_m$ pentru toţi indicii $l < i$ cu $k_m > k_m \}$
- $L_m = \{ k_i : 1 \leq i < j$ şi $k_m < k_i < k_m$ pentru toţi indicii $l < i$ cu $k_m < k_m \}$

Atunci cheile de pe drumul de la rădăcină la k_m sunt chiar cheile din $G_m \cup L_m$, iar adâncimea oricărei chei k_m din T este $d(k_m, T) = |G_m| + |L_m|$.

Arbori Binari de Căutare Construiți Aleator

Cu negru sunt nodurile care sunt, la inserarea lor, cel mai mic element mai mare decât 19 (G → greater). Similar, cele cu alb sunt elemente care, la inserarea lor, erau cele mai mari elemente mai mici decât 19 (L → lower).



Arbori Binari de Căutare Construiți Aleator

Practic, pentru a calcula înălțimea unui arbore, trebuie să calculăm $\mathbf{max}_{1 \leq j \leq n}(|G_m| + |L_m|)$.

Simplificăm și discutăm cum calculăm de câte ori se modifică, în medie, minimul, dacă inserăm **n** elemente pe rând.

Exercițiu: Care este probabilitatea ca k_i să fie minimul primelor i numere?

Arbori Binari de Căutare Construiți Aleator

Practic, pentru a calcula înălțimea unui arbore, trebuie să calculăm $\mathbf{max}_{1 \le j \le n}(|G_m| + |L_m|)$.

Simplificăm și discutăm cum calculăm de câte ori se modifică, în medie, minimul, dacă inserăm **n** elemente pe rând.

Răspuns: Probabilitatea ca k_i să fie minimul primelor i numere este **1/i**.

Prin urmare, numărul mediu de modificări este $\sum_{i=1}^{n} \frac{1}{i} = H_n$,

unde $H_m = \ln(n) + O(1)$ este al n-lea număr armonic.

→ Avem **log(n)** modificări.

Arbori Binari de Căutare Construiți Aleator

Lema 13.5. Fie $k_1, k_2, \ldots, k_m$ o permutare oarecare a unei mulţimi de n numere distincte şi fie |S| variabilă aleatoare reprezentând cardinalul mulţimii.

$S = \{ k_i : 1 \le i \le n$ şi $k_m > k_i$ pentru orice $l < i \}$ **(13.1)**

Atunci $Pr\{ |S| \ge (\beta + 1)H_m \} \le 1/(n^2)$, unde H_m este al n-lea număr armonic, iar $\beta \approx 4{,}32$ verifică ecuaţia $(\ln \beta - 1)\beta = 2$.

Prin urmare, e foarte probabil să avem maxim O(log(n)) modificări ale minimului.

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).

Arbori Binari de Căutare Construiți Aleator

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

<u>Demonstrație:</u> Fie $k_1, k_2, \ldots, k_m$ o permutare oarecare a celor n chei şi fie T arborele binar de căutare care rezultă prin inserarea cheilor în ordinea specificată, pornind de la un arbore inițial vid. Vom discuta prima dată probabilitatea ca adâncimea $d(k_m, T)$ a unei chei date k_m să fie cel puţin t, pentru o valoare t arbitrară. Conform caracterizării adâncimii $d(k_m, T)$ din ***corolarul 13.4***, dacă adâncimea lui k_m este cel puţin t, atunci cardinalul uneia dintre cele două mulţimi G_m şi L_m trebuie să fie cel puţin t/2.

Prin urmare, $\Pr\{ d(k_m, T) \geq t \} \leq \Pr\{ |G_m| \geq t/2 \} + \Pr\{ |L_m| \geq t/2 \}$.

Arbori Binari de Căutare Construiți Aleator

Să examinăm la început $\Pr\{\ |G_m| \geq t/2\ \}$. Avem

$$\Pr\{\ |G_m| \geq t/2\ \} = \Pr\{\ |\{k_i : 1 \leq i < j \text{ şi } k_m > k_i > k_m,\ \forall l < i\}| \geq t/2\ \}$$

$$\leq \Pr\{\ |\{k_i : i \leq n \text{ şi } k_m > k_i,\ \forall l < i\}| \geq t/2\ \}$$

$$= \Pr\{\ |S| \geq t/2\ \},$$

unde S este definit în relația **(13.1.)** $S = \{\ k_i : 1 \leq i \leq n \text{ şi } k_m > k_i,\ \forall l < i\ \}$.

În sprijinul acestei afirmații, să observăm că probabilitatea nu va descreşte dacă vom extinde intervalul de variație al lui i de la $i < j$ la $i \leq n$, deoarece, prin extindere, se vor adăuga elemente noi la mulțime. Analog, probabilitatea nu va descreşte dacă se renunță la condiția $k_i > k_m$, deoarece, prin aceasta, se înlocuieşte o permutare a (de regulă) mai puțin de **n** elemente (şi anume acele chei k_i care sunt mai mari decât k_m) cu o altă permutare oarecare de n elemente. Folosind o argumentare similară, putem demonstra că

$$\Pr\{\ |L_m|\ \ | \geq t/2\ \} \leq \Pr\{\ |S| \geq t/2\ \}.$$

Arbori Binari de Căutare Construiți Aleator

Folosind o argumentare similară, putem demonstra că

$$\Pr\{ |L_m| \geq t/2 \} \leq \Pr\{ |S| \geq t/2 \}$$

şi apoi, folosind inegalitatea **(13.2)**, obținem:

$$\Pr\{ d(k_m, T) \geq t \} \leq 2 * \Pr\{ |S| \geq t/2 \} .$$

Dacă alegem $t = 2(\beta + 1)H_m$, unde H_m este al n-lea număr armonic, iar $\beta \approx 4.32$ verifică ecuaţia $(\ln \beta - 1)\beta = 2$, putem aplica **lema 13.5** pentru a concluziona că

$$\Pr\{ d(k_m , T) \geq 2(\beta + 1)H_m \} \leq 2 * \Pr\{ |S| \geq (\beta + 1)H_m \} \leq 2/n^2 .$$

Arbori Binari de Căutare Construiți Aleator

Deoarece discutăm despre un arbore binar de căutare construit aleator şi cu cel mult n noduri, probabilitatea ca adâncimea oricăruia dintre noduri să fie cel puțin $2(\beta + 1)H_m$ este, folosind *inegalitatea lui Boole**, de cel mult $n*(2/n^2) = 2/n$. Prin urmare, în cel puțin $1 - 2/n$ din cazuri, înălțimea arborelui binar de căutare construit aleator este mai mică decât $2(\beta + 1)H_m$ şi în cel mult $2/n$ din cazuri înălțimea este cel mult n. În concluzie, înălțimea medie este cel mult

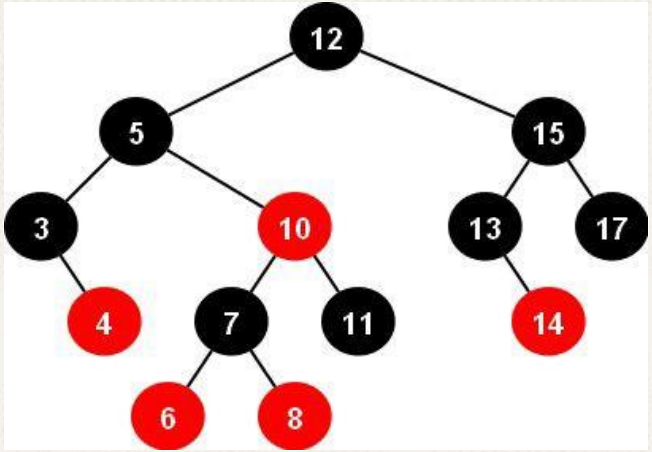
$$(2(\beta + 1)H_m)(1 - 2/n) + n(2/n) = O(\lg n).$$

**Inegalitatea lui Boole*: Fie

$A_1, A_2, \ldots, A_m$ în K cu $\bigcap_{i=1}^{n-1} A_i \neq 0$. Atunci : $Pr(\bigcap_{i=1}^{n-1} A_i) \geq (\sum_{i=1}^{n} Pr(A_i)) - n - 1$

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

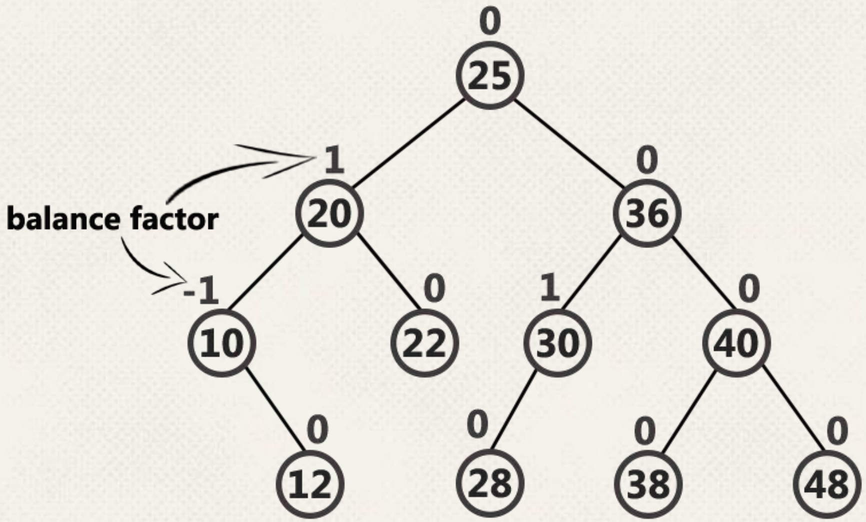
- Red Black Trees (nu veți avea la examen)
  - MIT Video
  - MIT Lecture Notes

Red Black Trees

~~Red Black Trees~~ AVL

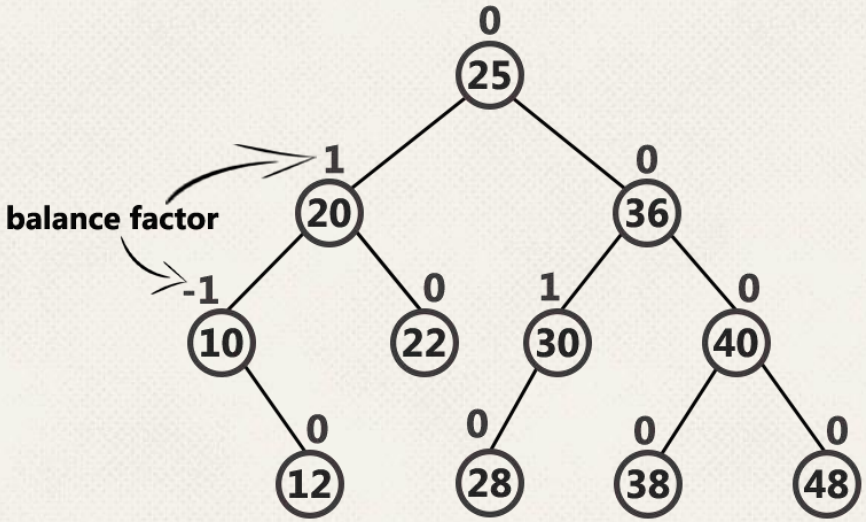
AVL

- 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 (nu veți avea la examen daaar cand spunem ca un arbore este echilibrat ne referim la definitia de la AVL si anume ca intre oricare 2 frati nu exista diferenta de inaltime mai mare ca 1)
H maxima in AVL este ~ 1.43*logn

- Video (MIT).
- Lecture Notes

Căutare binară

```
sol = 0; // Inițializăm soluția cu 0, care reprezintă indexul unde valoarea este mai mică sau egală cu x.

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

Căutare binară

```
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

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;
}
```

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

SKIP LISTS

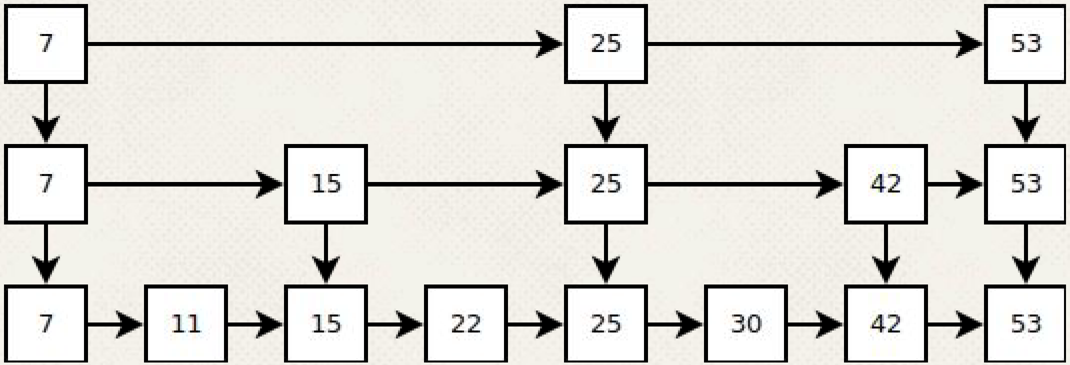
Skip Lists

- 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!

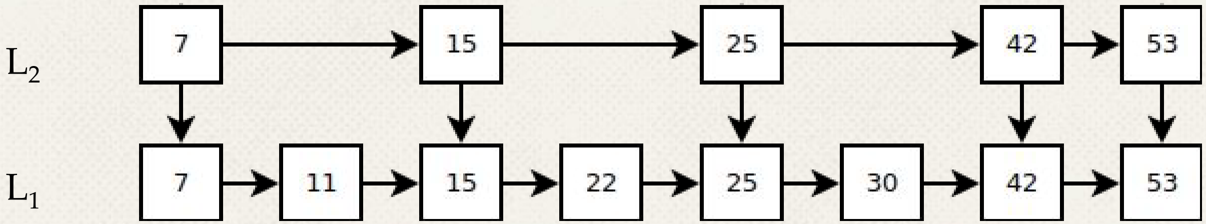
Skip Lists

- 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



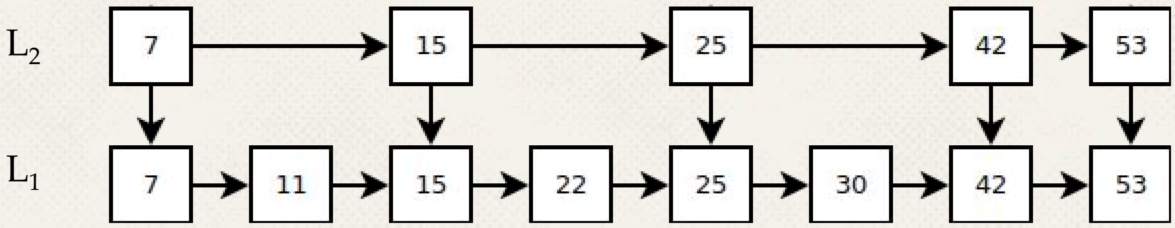
Skip Lists

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



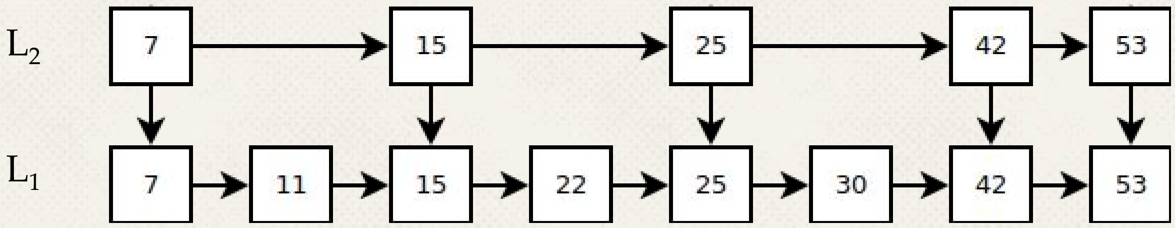
Skip Lists 2 liste

- 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

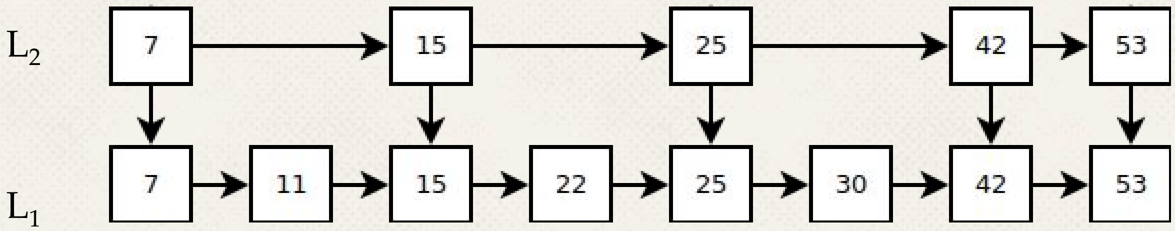
- Cum putem alege ce elemente ar trebui transferate în nivelul următor?
  - Cea mai bună metodă: elemente egal depărtate
  - Costul căutării = $|L_2| + (|L_1| / |L_2|) = |L_2| + (n / |L_2|)$
  - Când este minim acest cost?
    - Când $|L_2| = n / |L_2|$ $\Rightarrow$ $\mathbf{|L_2| = sqrt(n)}$



Skip Lists 2 liste

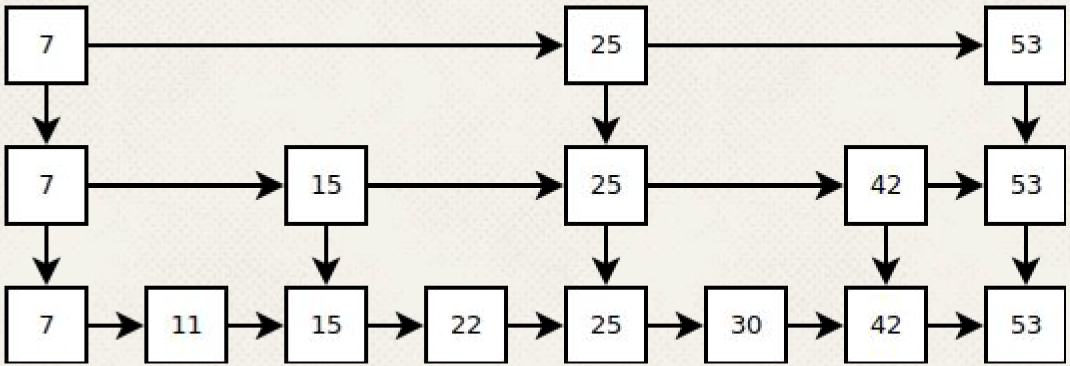
- Cum putem alege ce elemente ar trebui transferate în nivelul următor?
  - Cea mai bună metodă: elemente egal depărtate
  - Costul căutării = $|L_2| + (|L_1| / |L_2|) = |L_2| + (n / |L_2|)$
  - Când este minim acest cost?
    - Când $|L_2| = n / |L_2|$ ⇒ $\mathbf{|L_2| = sqrt(n)}$
  - Deci, costul minim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

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



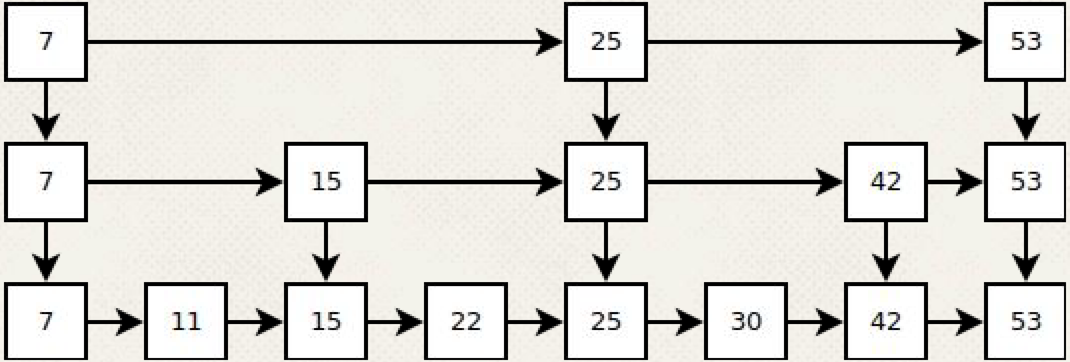
Skip Lists

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



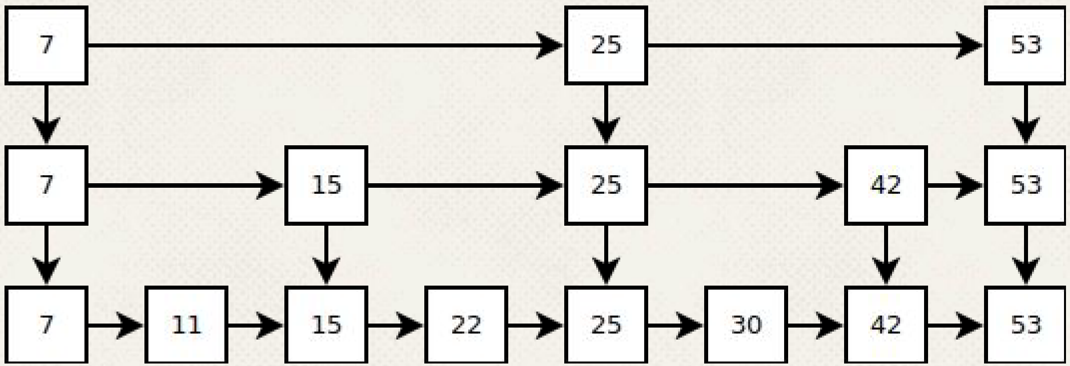
Skip Lists

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



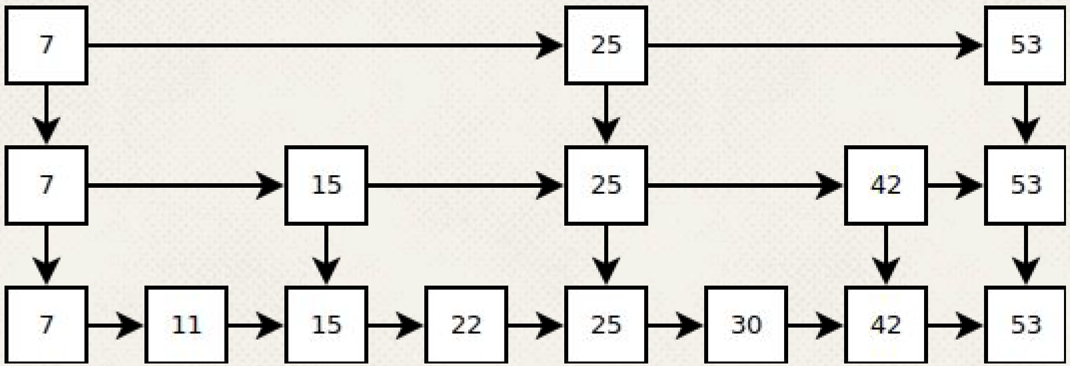
Skip Lists

- 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: $k * \sqrt[k]{n}$



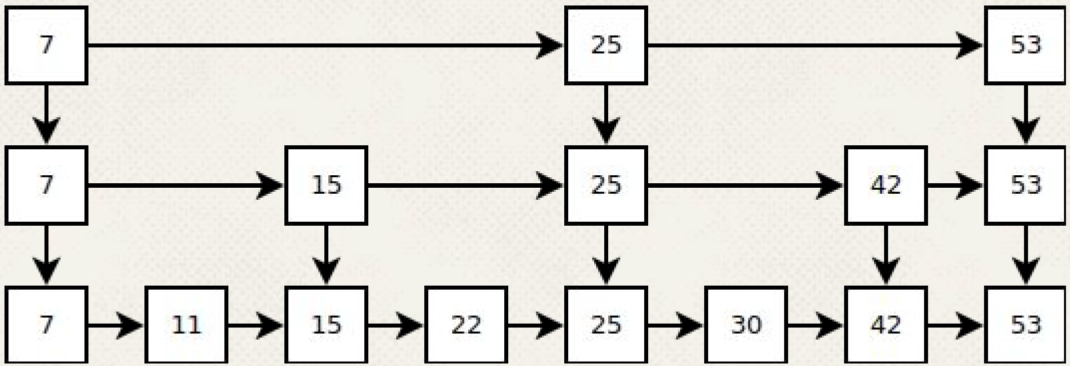
Skip Lists

- 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: $logn * \sqrt[logn]{n}$



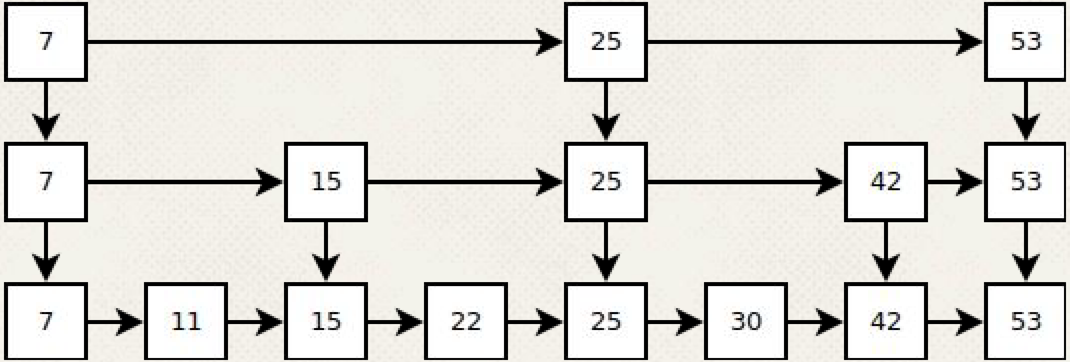
Skip Lists

- 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: $logn * \sqrt[logn]{n} = 2 * logn$ $\Rightarrow$ 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)

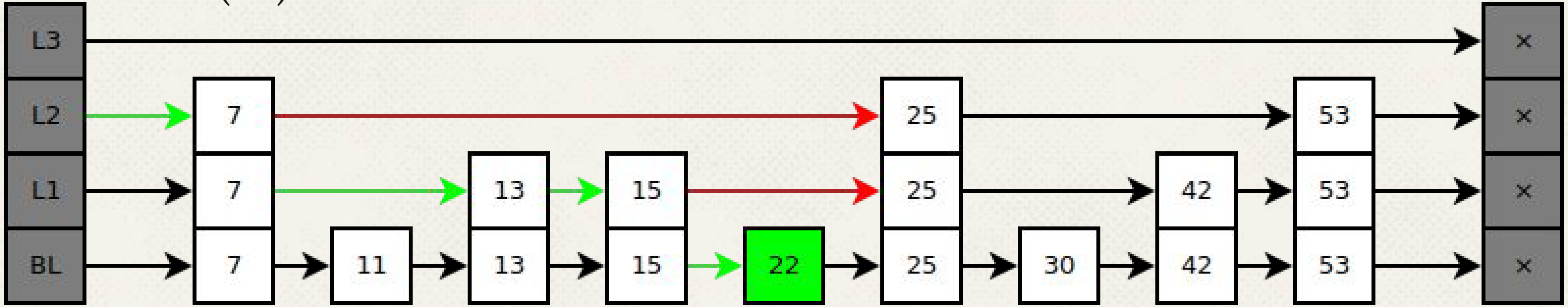
Skip Lists - Căutare

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:

Complexitate: O(logn)

search(22)



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

Final