



UNIUNEA EUROPEANĂ



GUVERNUL ROMÂNIEI



Instrumente Structurale
2007-2013



Platformă de e-learning și curriculum e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

4. Scheme de algoritmi – Programare dinamica

Bibliografie

- Cormen – Introducere în Algoritmi cap. 17
- Giumale – Introducere in Analiza Algoritmilor cap 4.4 ,4.5
- <http://www.cs.umass.edu/~barring/cs611/lecture/4.pdf>
- <http://thor.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps>
- <http://www.math.fau.edu/locke/Greedy.htm>
- <http://en.wikipedia.org/wiki/Greedoid>
- <http://activities.tjhsst.edu/sct/lectures/greedy0607.pdf>
- <http://www.cse.ust.hk/~dekai/271/notes/L12/L12.pdf>



Programare dinamică

- Programare dinamică
 - Descriere generală
 - Algoritm generic
 - Caracteristici
- Exemplificare: Înmulțirea matricilor
- Exemplificare: Arbori optimi la căutare (AOC)
 - Definiții
 - Construcția AOC

Programare dinamica

- **Descriere generală**
 - Soluții optime construite iterativ asamblând soluții optime ale unor probleme similare de dimensiuni mai mici.
- **Algoritmi “clasici”**
 - Algoritmul Floyd-Warshall care determină drumurile de cost minim dintre toate perechile de noduri ale unui graf.
 - AOC
 - Înmulțirea unui șir de matrici
 - Numere catalane
 - Viterbi

Algoritm generic

- Programare dinamică (crit_optim, problema)
 - // fie problema₀ problema₁ ... problema_n astfel încât
 - // problema_n = problema; problema_i mai simplă decât problema_{i+1}
 - 1. Sol = soluții_inițiale(crit_optim, problema₀);
 - 2. **Pentru** i = 1 → n **Repetă** // construcție soluții pentru problema_i
// folosind soluțiile problemelor precedente
 - 3. Sol_i = calcul_soluții(Sol, Crit_optim, Problema_i);
// determin soluția problemei_i
 - 4. Sol = Sol U Sol_i;
// noua soluție se adaugă pentru a fi refolosită pe viitor
 - 5. s = soluție_pentru_problema_n(Sol);
// selecție / construcție soluție finală
 - 6. **Întoarce** s;

Caracteristici

- O soluție optimă a unei probleme conține soluții optime ale subproblemelor.
- Decompozabilitatea recursivă a problemei P în subprobleme similare $P = P_n, P_{n-1}, \dots, P_0$ care acceptă soluții din ce în ce mai simple.
- Suprapunerea problemelor (soluția unei probleme P_i participă în procesul de construcție a soluțiilor mai multor probleme P_k de talie mai mare $k > i$) – memoizare (se folosește un tablou pentru salvarea soluțiilor subproblemelor pentru a nu le recalcula)
- În general se folosește o abordare bottom-up, de la subprobleme la probleme.



Diferențe Greedy – programare dinamică

Programare lacomă

- Sunt menținute doar soluțiile parțiale curente din care evoluează soluțiile parțiale următoare
- Soluțiile parțiale anterioare sunt eliminate
- Se poate obține o soluție neoptimă. (trebuie demonstrat că se poate aplica)

Programare dinamică

- Se păstrează toate soluțiile parțiale
- La construcția unei soluții noi poate contribui orice altă soluție parțială generată anterior
- Se obține soluția optimă.

Diferențe divide et impera – programare dinamică

Divide et impera

- abordare top-down – problema este descompusă în subprobleme care sunt rezolvate independent
- putem rezolva aceeași problemă de mai multe ori (dezavantaj potențial foarte mare)

Programare dinamică

- abordare bottom-up - se pornește de la sub-soluții elementare și se combină sub-soluțiile mai simple în sub-soluții mai complicate, pe baza criteriului de optim
- se evită calculul repetat al aceleiași subprobleme prin memorarea rezultatelor intermediare

Exemplu: Parantezarea matricilor (Chain Matrix Multiplication)

- Se dă o șir de matrice: A_1, A_2, \dots, A_n .
- Care este **numărul minim de înmulțiri** de scalari pentru a calcula produsul:

$$A_1 \times A_2 \times \dots \times A_n ?$$

- Să se determine una dintre **parantezările care minimizează** numărul de înmulțiri de scalari.

Înmulțirea matricilor

- $A(p, q) \times B(q, r) \Rightarrow pqr$ înmulțiri de scalari.
- Dar înmulțirea matricilor este asociativă (deși nu este comutativă).
- $A(p, q) \times B(q, r) \times C(r, s)$
 $(AB)C \Rightarrow pqr + prs$ înmulțiri
 $A(BC) \Rightarrow qrs + pqs$ înmulțiri
- Ex: $p = 5, q = 4, r = 6, s = 2$
 $(AB)C \Rightarrow 180$ înmulțiri
 $A(BC) \Rightarrow 88$ înmulțiri
- **Concluzie:** Parantezarea este foarte importantă!

Soluția banală

- Matrici: A_1, A_2, \dots, A_n .
- Vector de dimensiuni: $p_0, p_1, p_2, \dots, p_n$.
- $A_i(p_{i-1}, p_i) \rightarrow A_1(p_0, p_1), A_2(p_1, p_2), \dots$
- Dacă folosim căutare exhaustivă și vrem să construim toate parantezările posibile pentru a determina minimul: $\Omega(4^n / n^{3/2})$.
- Vrem o soluție polinomială folosind P.D.



Descompunere în subprobleme

- Încercăm să definim subprobleme identice cu problema originală, dar de dimensiune mai mică.
- $\forall 1 \leq i \leq j \leq n$:
 - Notăm $A_{i,j} = A_i \times \dots \times A_j$. $A_{i,j}$ are p_{i-1} linii și p_j coloane: $A_{i,j}(p_{i-1}, p_j)$
 - $m[i, j]$ = numărul optim de înmulțiri pentru a rezolva subproblema $A_{i,j}$
 - $s[i, j]$ = poziția primei paranteze pentru subproblema $A_{i,j}$
 - Care e parantezarea optimă pentru $A_{i,j}$?
- Problema inițială: $A_{1,n}$

Combinarea subproblemelor

- Pentru a rezolva $A_{i,j}$
 - trebuie găsit acel indice $i \leq k < j$ care asigură parantezarea optimă:

$$A_{i,j} = (A_i \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_j)$$

$$A_{i,j} = A_{i,k} \times A_{k+1,j}$$

Alegerea optimală

- Căutăm optimul dintre toate variantele posibile de alegere ($i \leq k < j$)
- Pentru aceasta, trebuie însă ca și subproblemele folosite să aibă soluție optimală (adică $A_{i, k}$ și $A_{k+1, j}$)

Substructura optimală

- Dacă știm că alegerea optimală a soluției pentru problema $A_{i,j}$ implică folosirea subproblemelor ($A_{i,k}$ și $A_{k+1,j}$) și soluția pentru $A_{i,j}$ este optimală, atunci și soluțiile subproblemelor $A_{i,k}$ și $A_{k+1,j}$ trebuie să fie optime!
- **Demonstrație:** Folosind metoda cut-and-paste (metodă standard de demonstrare a substructurii optime pentru problemele de programare dinamică).
- **Observație:** Nu toate problemele de optim posedă această proprietate!
Ex: drumul maxim dintr-un graf orientat.

Definirea recursivă

- Folosind **descompunerea in subprobleme**, **combinarea subproblemelor**, **alegerea optimală** și **substructura optimală** putem să rezolvăm problema prin programare dinamică.
- Următorul pas este să definim recursiv soluția unei subprobleme.
- Vrem să găsim o formulă recursivă pentru $m[i, j]$ și $s[i, j]$.

Definirea recursivă (II)

$$m[i, j] = \begin{cases} 0 & i = j, \\ \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j) & i < j \end{cases}$$

- Cazurile de bază sunt $m[i, i]$
- Noi vrem să calculăm $m[1, n]$
- Cum alegem $s[i, j]$?
- Bottom-up de la cele mai mici subprobleme la cea inițială

Rezolvare bottom-up

$m[1, 2], m[2, 3], m[3, 4], \dots, m[n-3, n-2], m[n-2, n-1], m[n-1, n]$

$m[1, 3], m[2, 4], m[3, 5], \dots, m[n-3, n-1], m[n-2, n]$

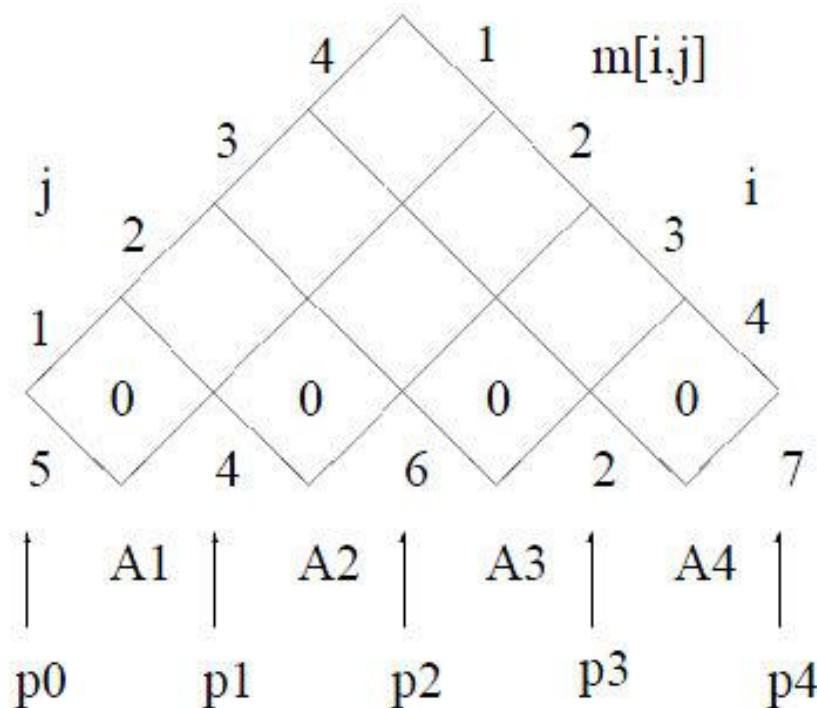
$m[1, 4], m[2, 5], m[3, 6], \dots, m[n-3, n]$

\vdots

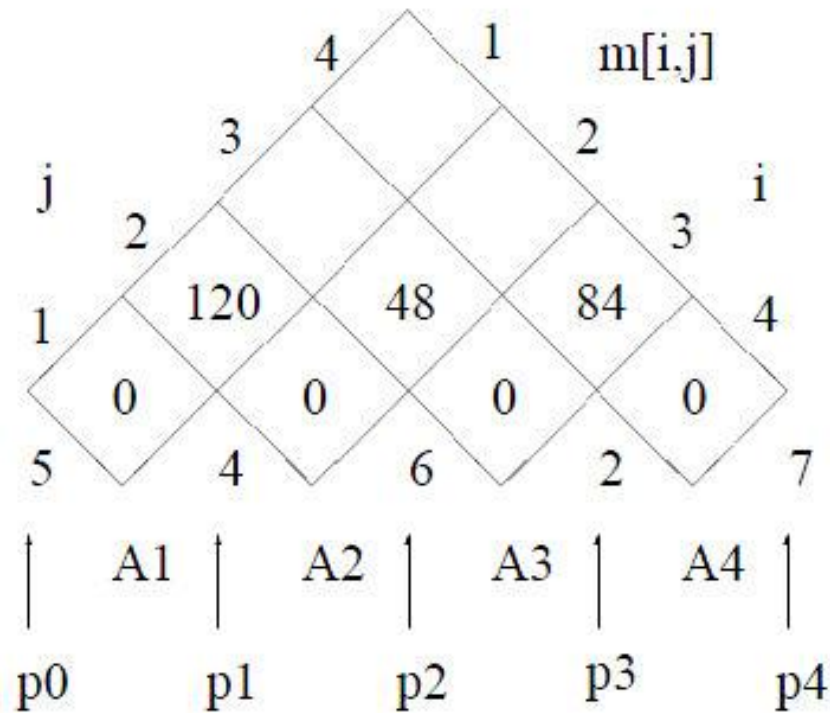
$m[1, n-1], m[2, n]$

$m[1, n]$

Rezolvare - inițializare



Rezolvare – pas intermediar



Pseudocod

```
Matrix-Chain(p, n)
{
  for (i = 1 to n) m[i, i] = 0;
  for (l = 2 to n)
  {
    for (i = 1 to n - l + 1)
    {
      j = i + l - 1;
      m[i, j] = ∞;
      for (k = i to j - 1)
      {
        q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j];
        if (q < m[i, j])
        {
          m[i, j] = q;
          s[i, j] = k;
        }
      }
    }
  }
  return m and s;
}
```

Complexitate

- Spațială: $\Theta(n^2)$
 - Pentru memorarea soluțiilor subproblemelor
- Temporală: $O(n^3)$
 - N_s : Număr total de subprobleme: $O(n^2)$
 - N_a : Număr total de alegeri la fiecare pas: $O(n)$
 - Complexitatea este de obicei egala cu $N_s \times N_a$

Arbori optimi la căutare

- **Def 2.1:** Fie K o mulțime de chei. Un **arbore binar cu cheile K** este un **graf orientat si aciclic** $A = (V, E)$ a.i.:
 - Fiecare nod $u \in V$ **conține o singură cheie** $k(u) \in K$ iar **cheile din noduri sunt distincte**.
 - Există un **nod unic** $r \in V$ a.i. $i\text{-grad}(r) = 0$ si $\forall u \neq r, i\text{-grad}(u) = 1$.
 - $\forall u \in V, e\text{-grad}(u) \leq 2$; $S(u) / D(u)$ = succesorul stânga / dreapta.
- **Def 2.2:** Fie K o mulțime de chei peste care exista o **relație de ordine** \prec . Un **arbore binar de căutare** satisface:
 - $\forall u, v, w \in V$ avem $(v \in S(u) \Rightarrow \text{cheie}(v) \prec \text{cheie}(u)) \wedge (w \in D(u) \Rightarrow \text{cheie}(u) \prec \text{cheie}(w))$

Căutare

- Caută(elem, Arb)
 - Dacă Arb = null
 - Întoarce null
 - Dacă elem = Arb.val // valoarea din nodul crt.
 - Întoarce Arb
 - Dacă elem < Arb.val
 - Întoarce Caută(elem, Arb.st)
 - Întoarce Caută(elem, Arb.dr)

● Complexitate: $\Theta(\log n)$

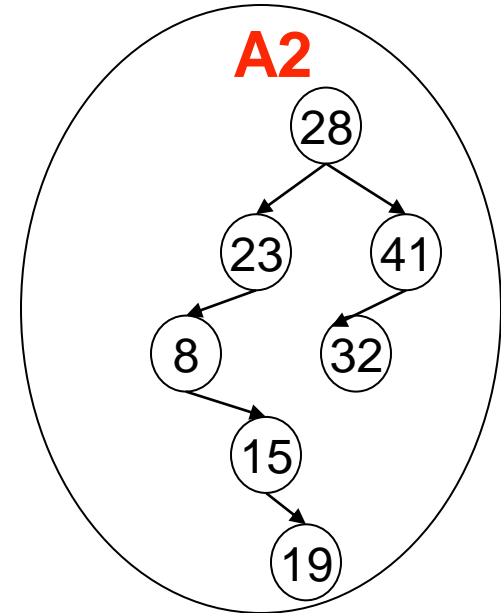
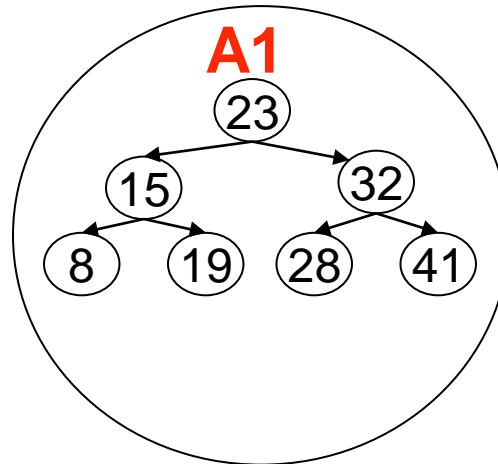
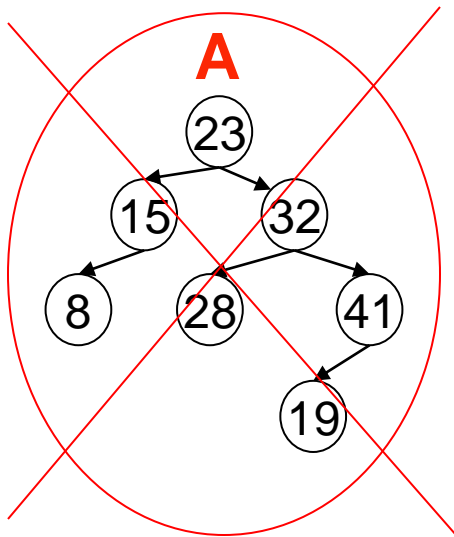
Insertie în arbore de căutare

- Inserare(elem, Arb)
 - Dacă Arb = vid // **adaug cheia in arbore**
 - nod_nou(elem, null, null) ← nod Stînga ← nod Dreapta
 - Dacă elem = Arb.val // **valoarea există deja**
 - Întoarce Arb
 - Dacă elem < Arb.val
 - Întoarce Inserare(elem, Arb.st) // **adaugă in stînga**
 - Întoarce Inserare(elem, Arb.dr) // **sau in dreapta**

● Complexitate: $\Theta(\log n)$

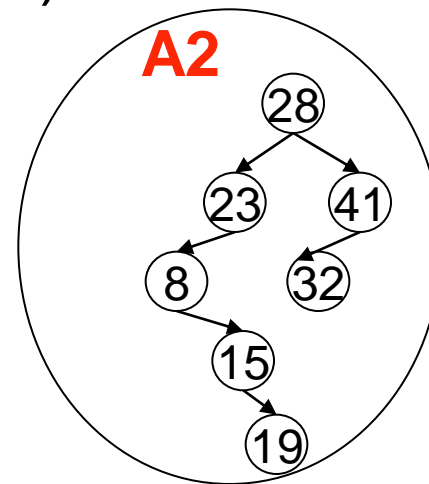
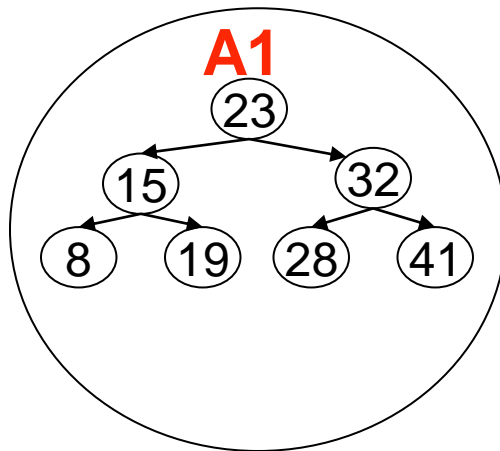
Exemplu de arbori de căutare

- Cu aceleași chei se pot construi arbori distincți



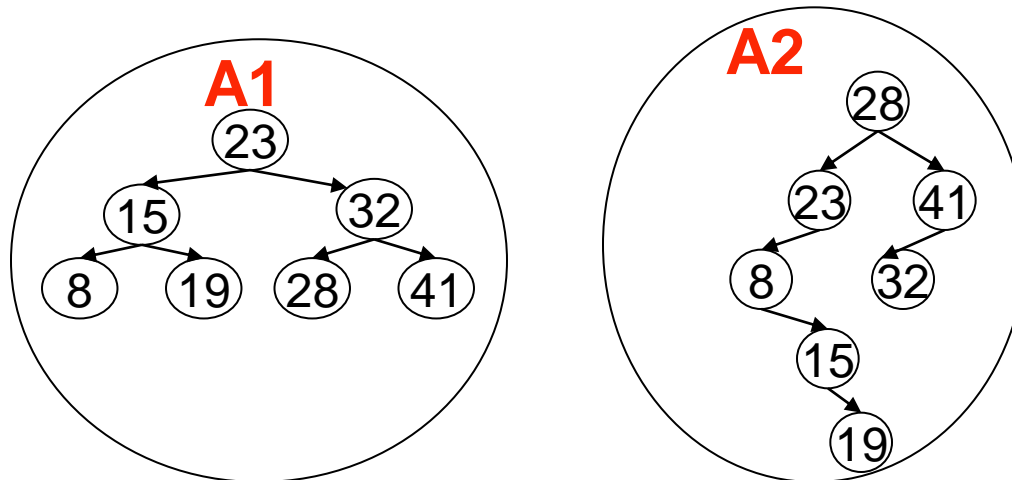
Exemplu (I)

- presupunem că elementele din A1 și A2 au **probabilități de căutare egale**:
 - numărul mediu de comparații pentru A1 va fi:
 $(1 + 2 + 2 + 3 + 3 + 3 + 3) / 7 = 2.42$
 - numărul mediu de comparații pentru A2 va fi:
 $(1 + 2 + 2 + 3 + 3 + 4 + 5) / 7 = 2.85$



Exemplu (II)

- presupunem că elementele au următoarele probabilități:
 - 8: 0.2; 15: 0.01; 19: 0.1; 23: 0.02; 28: 0.25; 32: 0.2; 41: 0.22;
 - numărul mediu de comparații pentru A1:
 - $0.02*1+0.01*2+0.2*2+0.2*3+0.1*4+0.25*3+0.22*3=2.85$
 - numărul mediu de comparații pentru A2:
 - $0.25*1+0.02*2+0.22*2+0.2*3+0.2*3+0.01*4+0.1*5=2.47$



Probleme

- Costul căutării **depinde de frecvența** cu care este căutat fiecare termen.
- → Ne dorim ca **termenii cei mai des** căutați să fie **cât mai aproape de vârful arborelui** pentru a micșora numărul de apeluri recursive.
- Dacă arborele este **construit prin sosirea aleatorie a cheilor** putem ajunge la o **simplă listă cu n elemente**

Definiție AOC

- **Definiție:** Fie A un arbore binar de căutare cu chei într-o mulțime K , fie $\{x_1, x_2, \dots, x_n\}$ cheile conținute în A , iar $\{y_0, y_1, \dots, y_n\}$ chei reprezentante ale cheilor din K ce nu sunt în A astfel încât: $y_{i-1} \prec x_i \prec y_i, i = \overline{1, n}$. Fie $p_i, i = \overline{1, n}$ probabilitatea de a căuta cheia x_i și $q_j, j = \overline{0, n}$ probabilitatea de a căuta o cheie reprezentată de y_j . Vom avea relația: $\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$. Se numește **arbore de căutare probabilistică**, un arbore cu **costul**:

$$Cost(A) = \sum_{i=1}^n (nivel(x_i, A) + 1) * p_i + \sum_{j=0}^n nivel(y_j, A) * q_j$$

- **Definiție:** Un arbore de căutare probabilistică având **cost minim** este un **arbore optim la căutare (AOC)**.

Algoritm AOC naiv

- Generarea permutărilor x_1, \dots, x_n .
- Construcția arborilor de căutare corespunzători.
- Calcularea costului pentru fiecare arbore.
- Alegerea arborelui de cost minim.
- **Complexitate: $\Theta(n!)$** (deoarece sunt $n!$ permutări).
- → căutăm altă variantă!!!



Construcția AOC – Notații

- $A_{i,j}$ desemnează un AOC cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_j\}$ în noduri și cu cheile $\{y_i, y_{i+1}, \dots, y_j\}$ în frunzele fictive.

- $C_{i,j} = \text{Cost}(A_{i,j})$. $\text{Cost}(A_{ij}) = \sum_{k=i+1}^j (\text{nivel}(x_k, A_{ij}) + 1) * p_k + \sum_{k=i}^j \text{nivel}(y_k, A_{ij}) * q_k$

- $R_{i,j}$ este indicele α al cheii x_α din rădăcina arborelui $A_{i,j}$.

- $w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k$

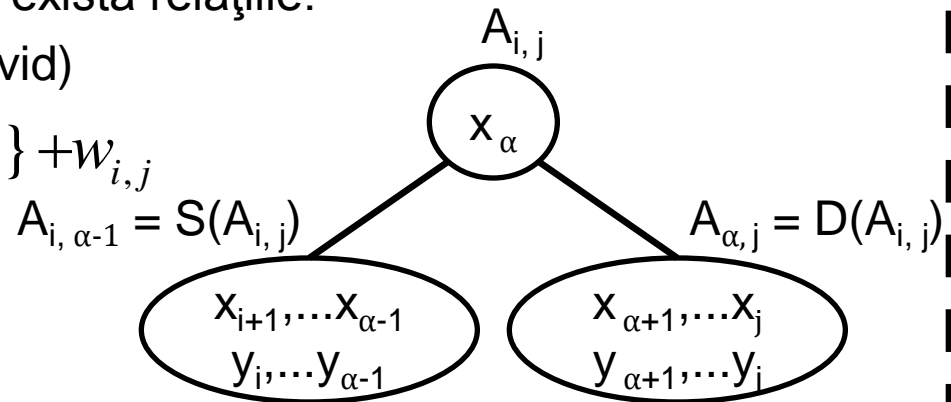
- **Observație:** $A_{0,n}$ este chiar arborele A , $C_{0,n} = \text{Cost}(A)$ iar $w_{0,n} = 1$.

Construcția AOC - Demonstrație

- **Lemă:** Pentru orice $0 \leq i \leq j \leq n$ există relațiile:

- $C_{i,j} = 0$, dacă $i = j$; (arbore vid)

- $C_{i,j} = \min_{i \leq \alpha \leq j} \{C_{i,\alpha-1} + C_{\alpha,j}\} + w_{i,j}$



- **Demonstrație:**

$$Cost(A_{ij}) = \sum_{k=i+1}^j (nivel(x_k, A_{ij}) + 1) * p_k + \sum_{k=i}^j nivel(y_k, A_{ij}) * q_k$$

$$\Rightarrow C_{i,j} = C_{i,\alpha-1} + C_{\alpha,j} + w_{i,j}$$

- $C_{i,j}$ depinde de indicele α al nodului rădăcină.
- dacă $C_{i,\alpha-1}$ și $C_{\alpha,j}$ sunt minime (costurile unor AOC) $\Rightarrow C_{i,j}$ este minim.

Construcția AOC

- 1. In etapa d , $d = 1, 2, \dots, n$ se calculează costurile și indicele cheilor din rădăcina arborilor AOC $A_{i, i+d}$, $i = 0, n-d$ cu d noduri și $d + 1$ frunze fictive.
- Arborele $A_{i, i+d}$ conține în noduri cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$, iar în frunzele fictive sunt cheile $\{y_i, y_{i+1}, \dots, y_{i+d}\}$. Calculul este efectuat pe baza rezultatelor obținute în etapele anterioare.
- Conform lemei avem
$$C_{i, i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i, \alpha-1} + C_{\alpha, i+d}\} + w_{i, i+d}$$
- Rădăcina $A_{i, i+d}$ are indicele $R_{i, j} = \alpha$ care minimizează $C_{i, i+d}$.
- 2. Pentru $d = n$, $C_{0, n}$ corespunde arborelui AOC $A_{0, n}$ cu cheile $\{x_1, x_2, \dots, x_n\}$ în noduri și cheile $\{y_0, y_1, \dots, y_n\}$ în frunzele fictive.

Algorithm AOC

AOC(x, p, q, n)

Pentru $i = 0 \rightarrow n$

$\{C_{i,i} = 0, R_{i,i} = 0, w_{i,i} = q_i\}$ // inițializare costuri AOC vid $A_{i,i}$

Pentru $d = 1 \rightarrow n$

Pentru $i = 0 \rightarrow n-d$ // calcul indice rădăcina și cost pentru $A_{i,i+d}$

$j = i + d, C_{i,j} = \infty, w_{i,j} = w_{i,j-1} + p_j + q_j$

Pentru $\alpha = i + 1 \rightarrow j$ // ciclul critic – operații intensive

Dacă $(C_{i,\alpha-1} + C_{\alpha,j} < C_{i,j})$ // cost mai mic?

$\{C_{i,j} = C_{i,\alpha-1} + C_{\alpha,j}; R_{i,j} = \alpha\}$ // update

$C_{i,j} = C_{i,j} + w_{i,j}$ // update

Întoarce $\text{gen_AOC}(C, R, x, 0, n)$ // construcție efectivă arbore $A_{0,n}$
// cunoscând indicii

Complexitate???

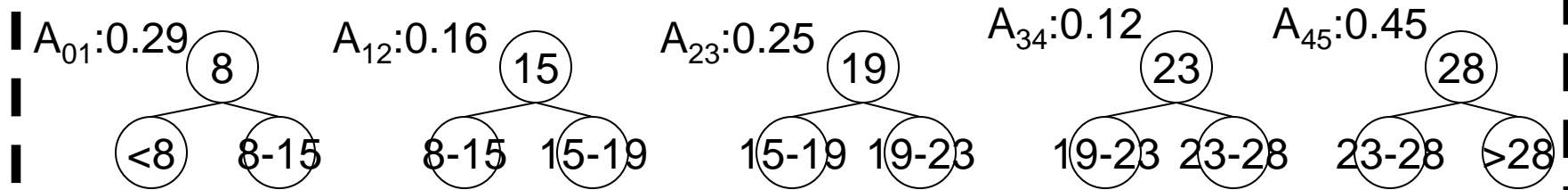


Exemplu constructie AOC (I)

- 8: 0.2; 15: 0.01; 19: 0.1; 23: 0.02; 28: 0.25; (58%)
- [0:8): 0.02; (8:15): 0.07; (15:19): 0.08; (19:23): 0.05; (23:28): 0.05; (28,∞): 0.15 (42%)
- $C_{01} = p_1 + q_0 + q_1 = 0.29$
- $C_{12} = p_2 + q_1 + q_2 = 0.16$
- $C_{23} = p_3 + q_2 + q_3 = 0.25$
- $C_{34} = p_4 + q_3 + q_4 = 0.02 + 0.05 + 0.05 = 0.12$
- $C_{45} = p_5 + q_4 + q_5 = 0.25 + 0.05 + 0.15 = 0.45$

$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$

Exemplu constructie AOC (II)

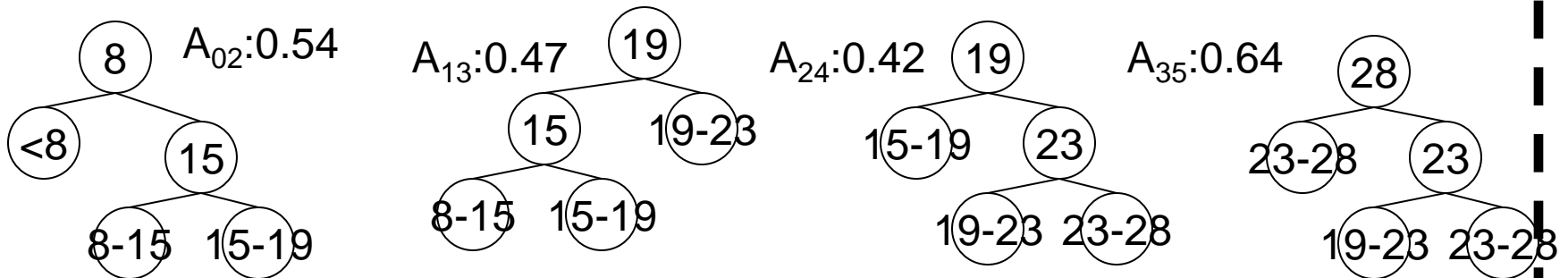


$$C_{02} = \min \{ (C_{00} + C_{12}), (C_{01} + C_{22}) \} + w_{02} = \min(0.16, 0.29) + 0.38 = 0.54 \quad R_{02} = 1 \quad (\alpha=1)$$

$$C_{13} = \min \{ C_{23}, C_{12} \} + w_{13} = \min(0.25, 0.16) + 0.31 = 0.47 \quad R_{13} = 3$$

$$C_{24} = \min \{ C_{34}, C_{23} \} + w_{24} = \min(0.12, 0.25) + 0.3 = 0.42 \quad R_{24} = 3$$

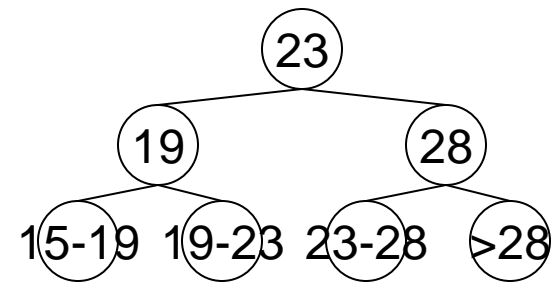
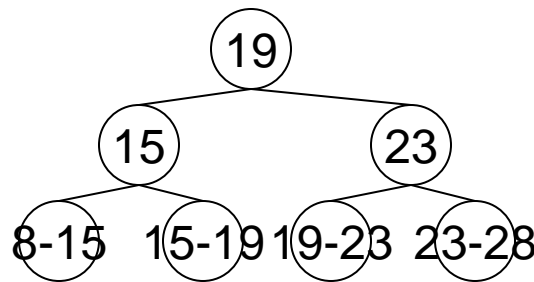
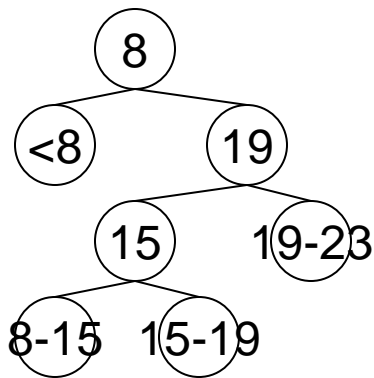
$$C_{35} = \min \{ C_{45}, C_{34} \} + w_{35} = \min(0.45, 0.12) + 0.52 = 0.64 \quad R_{35} = 5$$



$$w_{i,j} = \sum_{k=i+1}^j p_k + \sum_{k=i}^j q_k \quad C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{ C_{i,\alpha-1} + C_{\alpha,i+d} \} + w_{i,i+d}$$

Exemplu constructie AOC (III)

- $C_{03} = \min(C_{00}+C_{13}, C_{01}+C_{23}, C_{02}+C_{33}) + w_{03} = \min(0.47, 0.54, 0.54) + w_{03} \Rightarrow R_{03} = 1$
- $C_{14} = \min(C_{11}+C_{24}, C_{12}+C_{34}, C_{13}+C_{44}) + w_{14} = \min(0.42, 0.28, 0.47) + w_{14} \Rightarrow R_{14} = 3$
- $C_{25} = \min(C_{22}+C_{35}, C_{23}+C_{34}, C_{24}+C_{55}) + w_{25} = \min(0.64, 0.37, 0.42) + w_{25} \Rightarrow R_{25} = 4$



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

$$C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{ C_{i,\alpha-1} + C_{\alpha,i+d} \} + w_{i,i+d}$$

AOC – Corectitudine (I)

- **Teoremă:** Algoritmul AOC construiește un arbore AOC A cu cheile $x = \{x_1, x_2, \dots, x_n\}$ conform probabilităților de căutare $p_i, i = 1, n$ și $q_j, j = 0, n$.
- **Demonstrație:** prin inducție după etapa de calcul al costurilor arborilor cu d noduri.
- **Caz de baza:** $d = 0$. Costurile $C_{i, i}$ ale arborilor vizi $A_{i, i}, i = 0, n$ sunt 0, așa cum sunt inițializate de algoritm.
- **Pas de inducție:** $d \geq 1$. **Ip. ind.:** pentru orice $d' < d$, algoritmul AOC calculează costurile $C_{i, i+d'}$ și indicii $R_{i, i+d'}$, ai rădăcinilor unor AOC $A_{i, i+d'}$, $i = 0, n-d'$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d'}\}$. Trebuie să arătăm că valorile $C_{i, i+d}$ și $R_{i, i+d}$ corespund unor AOC $A_{i, i+d}, i = 0, n-d$ cu cheile $\{x_{i+1}, x_{i+2}, \dots, x_{i+d}\}$.

AOC – Corectitudine (II)

- Pentru d și i fixate, algoritmul calculează:

$$C_{i,i+d} = \min_{i \leq \alpha \leq i+d} \{C_{i,\alpha-1} + C_{\alpha,i+d}\} + w_{i,i+d}$$

- unde costurile $C_{i,\alpha-1}$ și $C_{\alpha,i+d}$ corespund unor arbori cu un număr de noduri $d' = \alpha - 1 - i$ în cazul $C_{i,\alpha-1}$ și $d' = i + d - \alpha$ în cazul $C_{\alpha,i+d}$.
- $0 \leq d' \leq d - 1 \rightarrow$ aceste valori au fost deja calculate în etapele $d' < d$ și conform **ipotezei inductive** \rightarrow sunt costuri și indici ai rădăcinilor unor AOC.
- Conform **Lemei** anterioare, $C_{i,j}$ este costul unui AOC. Conform algoritmului \rightarrow rădăcina acestui arbore are indicele $r = R_{i,j}$, iar cheile sunt $\{x_{i+1}, x_{i+2}, \dots, x_{r-1}\}$
 $\{x_r\} \{x_{r+1}, x_{r+2}, \dots, x_j\} = \{x_{i+1}, x_{i+2}, \dots, x_j\}$
- Pentru $d = n$, costul $C_{0,n}$ corespunde unui AOC $A_{0,n}$ cu cheile x și cu rădăcina de indice $R_{0,n}$.



AOC – Concluzii

- Câte subprobleme sunt folosite în soluția optimală într-un anumit pas?
 - AOC: 2 subprobleme
- Câte variante de ales avem de făcut pentru determinarea alegerii optime într-un anumit pas?
 - AOC: $j - i + 1$ candidați pentru rădăcină
- Informal, complexitatea = $N_s * N_a$ (N_s = număr subprobleme; N_a = număr alegeri)
 - Complexitate AOC: $O(n^2) * O(n) = O(n^3)$
 - Optimizare – Knuth: $O(n^2)$

