Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare

Proiectarea algoritmilor (PA)

- seria CD -

Andrei Mogoș - Suport de curs

Curs 2: Greedy (Programare lacomă)

Observație

Suportul de curs de la seria CD (pentru cele 14 cursuri) se bazează pe slide-urile de la PA din anii precedenți (2007 – 2016) de la seriile CA, CB, CC (titulari de curs: Ș. Trăușan, T. Rebedea, C. Chiru)

Bibliografie

- Cormen – **Introducere în Algoritmi**: cap. Algoritmi Greedy (17)
- Giumale – **Introducere în Analiza Algoritmilor**: cap. 4.4
- https://people.cs.umass.edu/~barring/cs611/lecture/4.pdf
- http://profs.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps
- http://euler.math.fau.edu/locke/Greedy.htm
- http://en.wikipedia.org/wiki/Greedoid

Greedy (1)

- Metodă de rezolvare eficientă a unor probleme de optimizare.
- Soluția trebuie să satisfacă un criteriu de optim global (greu de verificat) → optim local mai ușor de verificat.
- Se aleg soluții parțiale ce sunt îmbunătățite repetat pe baza criteriului de optim local până ce se obțin soluții finale.
- Soluțiile parțiale ce nu pot fi îmbunătățite sunt abandonate → proces de rezolvare irevocabil (fără reveniri)!

Greedy (2)

- Schema generală de rezolvare a unei probleme folosind Greedy (programarea lacomă):

- Rezolvare_lacomă(Crit_optim, Problemă)

```
1   sol_parțiale = sol_inițiale(Problemă); // determinarea soluțiilor parțiale
2   sol_fin = Φ;
3   Cât timp (sol_parțiale ≠ Φ)
4       Pentru fiecare (s în sol_parțiale)
5           Dacă (s este o soluție a problemei) { // dacă e soluție
6               sol_fin = sol_fin U {s}; // finală se salvează
7               sol_parțiale = sol_parțiale \ {s};
8           } Altfel // se poate optimiza?
9               Dacă (optimizare_posibilă (s, Crit_optim, Problemă))
10                  sol_parțiale = sol_parțiale \ {s} U
                                   optimizare(s, Crit_optim, Problemă) // da
11              Altfel
                    sol_parțiale = sol_parțiale \ {s}; // nu
12  Întoarce sol_fin;
```

Problema rucsacului (1)

- Trebuie să umplem un rucsac de capacitate maximă M kg cu obiecte care au masa m_i şi valoarea v_i. Putem alege mai multe obiecte din fiecare tip cu scopul de a maximiza valoarea obiectelor din rucsac.
- Variante:
  - Varianta 1: putem alege fracţiuni de obiect – "problema continuă"
  - Varianta 2: putem alege doar obiecte întregi – "problema 0-1"

Problema rucsacului (2)

- Varianta 1: Algoritm Greedy
  - sortăm descrescător obiectele după raportul v_i/m_i;
  - adăugăm fracţiuni din obiectul cu cea mai mare valoare per kg până epuizăm stocul şi apoi adăugăm fracţiuni din obiectul cu valoarea următoare.

  - Exemplu: M = 50; m_1 = 10 kg, v_1 = 60, m_2 = 30 kg, v_2 = 120, m_3 = 20 kg, v_3 = 100
  - Obiectele sortate: m_1, m_3, m_2 (v_i/m_i: 6, 5, 4)
  - Soluție: (m_1 , v_1) 10 kg, (m_3,v_3) 20 kg şi 20kg din (m_2,v_2)
    - valoarea totală: 60 + 100 + 80 = 240

Problema rucsacului (3)

- Varianta 2: Algoritmul Greedy nu funcţionează => contraexemplu:
  - Exemplu: M = 50; m_1 = 10 kg, v_1 = 60, m_2 = 30 kg, v_2 = 120, m_3 = 20 kg, v_3 = 100
  - Obiectele sortate: m_1, m_3, m_2 (v_i/m_i: 6, 5, 4)
  - Rezultat corect: (m_2,v_2), (m_3,v_3)
    - valoarea totală: 220
  - Rezultat algoritm Greedy: (m_1,v_1), (m_3,v_3)
    - valoarea totală: 160

Arbori Huffman

- Metodă de codificare folosită la compresia fișierelor.
- Construcția unui astfel de arbore se realizează printr-un algoritm Greedy.
- Considerăm un text, de exemplu:
  - **ana are mere**
- Vom exemplifica pas cu pas construcția arborelui de codificare pentru acest text și vom defini pe parcurs conceptele de care avem nevoie.

Arbori Huffman – Definiții (1)

- K: mulțimea de simboluri ce vor fi codificate. (a, n, “ “, r, e, m)
- Un arbore de codificare a cheilor K este un arbore binar cu proprietățile:
  - Doar frunzele conțin cheile din K; nu există mai mult de o cheie într-o frunză;
  - Orice nod intern are exact 2 copii (arbore binar complet);
  - Arcele sunt etichetate cu 0 și 1
    - etichetă(u,v) = 0, dacă v este succesorul stâng al lui u;
    - etichetă(u,v) = 1, dacă v este succesorul drept al lui u.
- k: Codul unei chei – este șirul etichetelor de pe calea de la rădăcina arborelui la frunza care conține cheia k (k este din K).
- p(k): frecvența de apariție a cheii k în textul ce trebuie comprimat.
- Ex pentru “ana are mere”:
  - p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p( ) = 0.166

Arbori Huffman – Definiții (2)

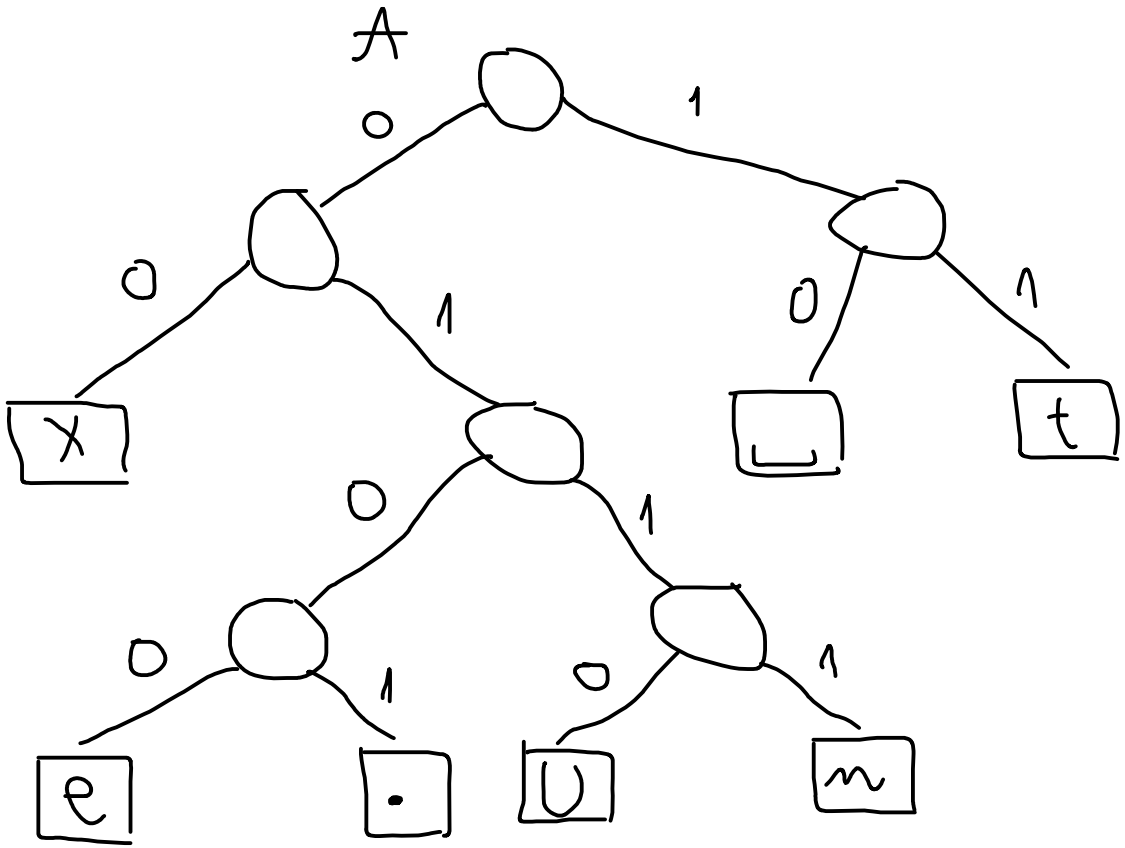
- A – arborele de codificare a cheilor.
- lg_cod(k) – lungimea codului cheii k conform A.
- nivel(k,A) – nivelul pe care apare în A frunza ce conține cheia K.
- Costul unui arbore de codificare A al unor chei K relativ la o frecventa p este:

$$Cost\ (A) = \sum_{k \in K} lg\_cod\ (k) * p(k) = \sum_{k \in K} nivel\ (k, A) * p(k)$$

- Un arbore de codificare cu cost minim al unor chei K, relativ la o frecvență p este un arbore Huffman, iar codurile cheilor sunt coduri Huffman.

Arbori Huffman – Exemplu (1)

Textul: “Un text x.” K = {U, n, t, e, x, ., spatiu}



Codificare: 0110 0111 10 11 0100 00 11 10 00 0101

Arbori Huffman – Exemplu (2)

28 biţi → 4 octeţi: text codificat
10 octeţi: textul iniţial
=> Factor de compresie: 60% (doar pentru textul comprimat)

U, n, e, .: o singură apariţie
x, t, spaţiu: două apariţii
=> p(x) = p(t) = p(spaţiu) = 0.2
p(U) = p(n) = p(e) = p(.) = 0.1

Cost(A) = 2.8 biţi (lungimea medie a codului unei chei din K)
Codificarea ASCII: 8 biţi pentru o cheie

Observaţie: Arborele prezentat în acest exemplu este un arbore Huffman.

Arbori Huffman – Algoritm de construcție (1)

1. Pentru fiecare k din K se construiește un arbore cu un singur nod care conține cheia k si este caracterizat de ponderea w = p(k). Subarborii construiți formează o mulțime numită Arb.

2. Se aleg doi subarbori a şi b din Arb astfel încât a şi b au pondere minimă.

Arbori Huffman – Algoritm de construcție (2)

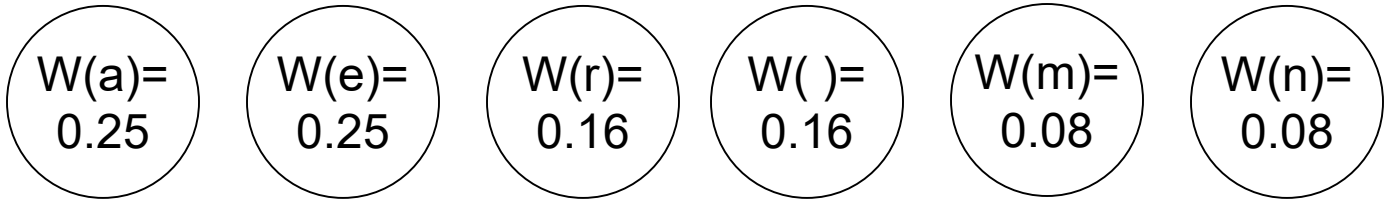
3. Se construiește un arbore binar complet cu o rădăcina *r* care nu conține nici o cheie și cu descendenții a și b. Ponderea arborelui este definită ca w(r) = w(a) + w(b).

4. Arborii a și b sunt eliminați din Arb iar r este inserat în Arb.

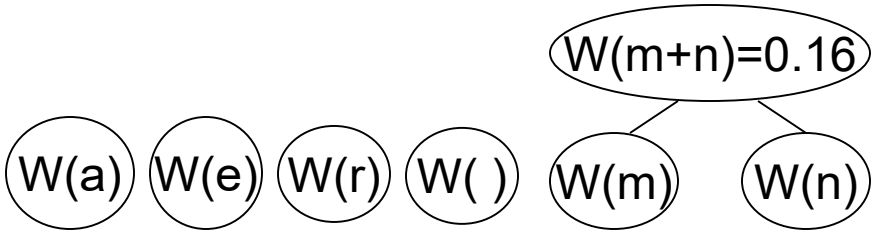
5. Se repetă procesul de construcție descris de pașii 2-4 până când mulțimea Arb conține un singur arbore – Arborele Huffman pentru cheile K.

Arbori Huffman – Exemplu (1)

- Text: ana are mere
- p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p( ) = 0.166
- Pasul 1:

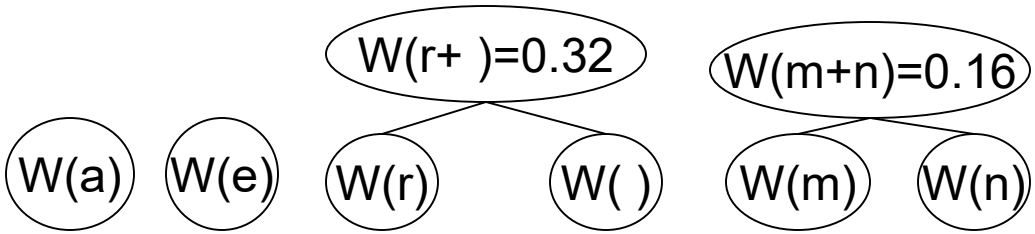


- Pasii 2-4:

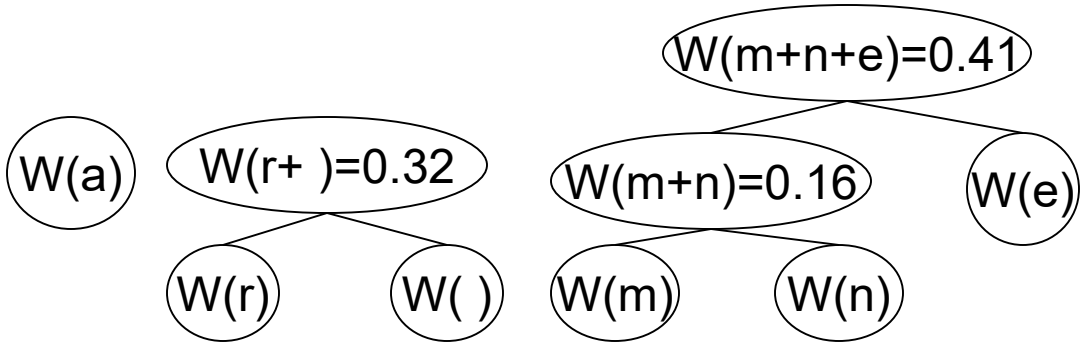


Arbori Huffman – Exemplu (2)

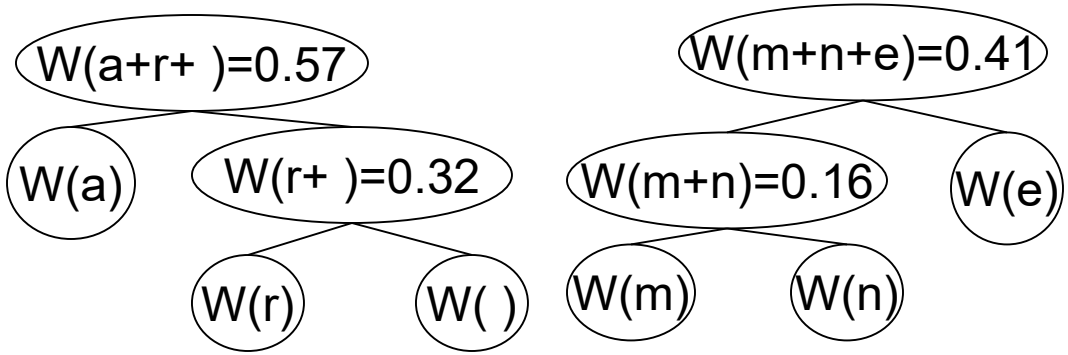
- Pasii 2-4 (II):



- Pasii 2-4 (III):

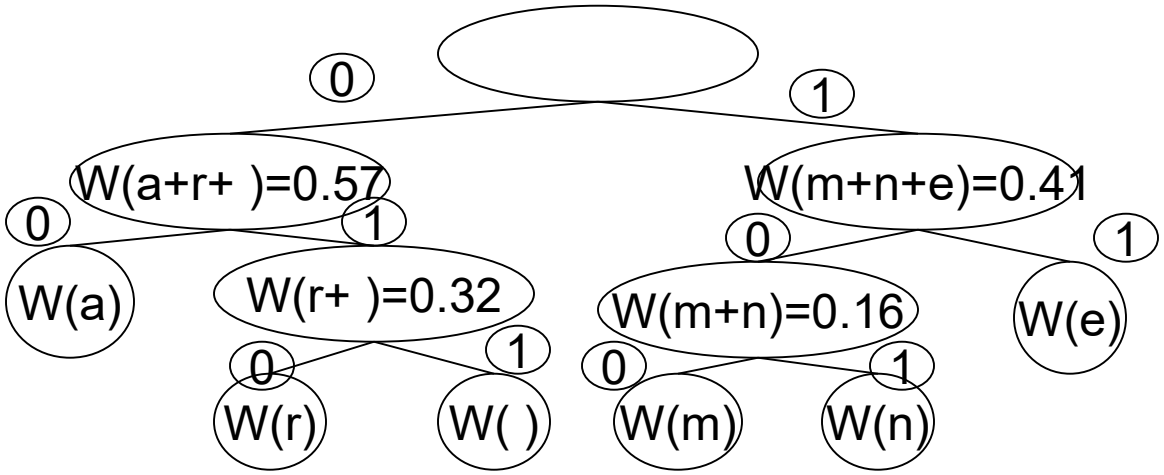


- Pasii 2-4 (IV):



Arbori Huffman – Exemplu (3)

- Pasii 2-4 (V):



- Codificare: a - 00; e -11; r - 010; ’ ‘ - 011; m - 100; n - 101;

$$Cost(A) = \sum_{k \in K} lg\_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$

- p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p( ) = 0.166
- Cost(A) = 2 * 0.25 + 2 * 0.25 + 3 * 0.083 + 3 * 0.083 + 3 * 0.166 + 3 * 0.166 = 1 + 1.2 = 2.2 biti.

Arbori Huffman - Pseudocod

- Huffman(K,p){

```
1  Arb = {frunză(k, p(k)) | k ∈ K};
2  Cât timp (card (Arb) > 1) // am mai mulți subarbori
3      fie a1 și a2 arbori din Arb a.i. ∀a ∈ Arb a ≠ a1 si a ≠ a2, avem
                  w(a1) ≤ w(a) și w(a2) ≤ w(a));     // practic se extrage
                  // de două ori minimul și se salvează în a1 și a2
4      Arb = Arb \ {a1, a2} U nod_intern(a1, a2, w(a1) + w(a2));
5  Dacă (Arb = Φ)
6      Întoarce arb_vid;
7  Altfel
8      fie A singurul arbore din mulțimea Arb;
9      Întoarce A;
```

- Notații folosite:
  - a = frunză (k, p(k)) – subarbore cu un singur nod care conține cheia k, iar w(a) = p(k);
  - a = nod_intern(a_1, a_2, x) – subarbore format dintr-un nod intern cu descendenții a_1 și a_2 și w(a) = x.

Arbori Huffman - Decodificare

- Se încarcă arborele și se decodifică textul din fișier conform algoritmului:

- Decodificare (in, out) // in: fișierul comprimat; out: fișierul decomprimat

```
1   A = restaurare_arbore(in)      // reconstruiesc arborele
2   Cât timp (! terminare_cod(in))          // mai am caractere de citit
3       nod = A                             // pornesc din rădăcină
4       Cât timp (! frunză(nod)) // cât timp nu am determinat caracterul
5           Dacă (bit(in) = 1) nod = dreapta(nod) // avansez în arbore
6           Altfel nod = stânga(nod)
7       Scrie (out, cheie(nod)) // am determinat caracterul și îl scriu la ieșire
```

Demonstrație (1)

- Arborele de codificare construit trebuie să aibă cost minim pentru a fi arbore Huffman.

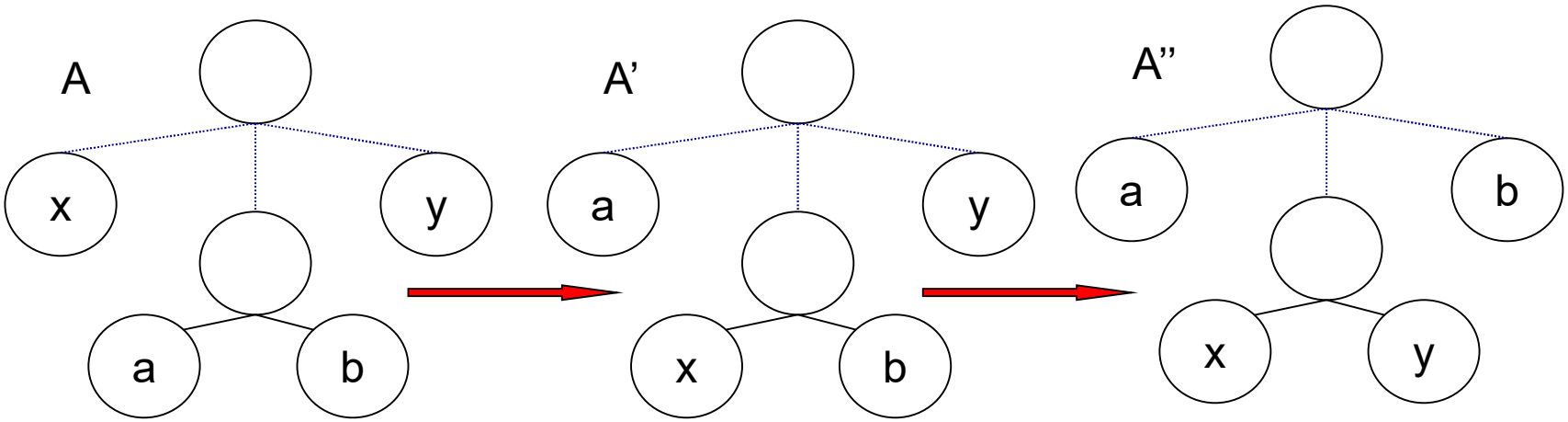
Lema 1. Fie K mulțimea cheilor dintr-un arbore de codificare, card(K) ≥ 2, x, y două chei cu pondere minimă. Există un arbore Huffman de înălțime h în care cheile x şi y apar pe nivelul h fiind descendente ale aceluiași nod intern.

Observație. Numărăm nivelurile unui arbore (de sus in jos) începand cu nivelul 0.

Demonstrație (2)

Demonstrație Lema 1:

$$Cost(A) = \sum_{k \in K} lg\_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$



- Se interschimbă a cu x şi b cu y şi din definiţia costului arborelui => cost(A'') ≤ cost(A') ≤ cost(A) => A'' arbore Huffman

Demonstrație (3)

Lema 2. Fie A un arbore Huffman cu cheile K, iar x şi y două chei direct descendente ale aceluiaşi nod intern *a*. Fie K' = K \ {x,y} U {z} unde z este o cheie fictivă cu ponderea w(z) = w(x) + w(y). Atunci arborele A' rezultat din A prin înlocuirea subarborelui cu rădăcina *a* și frunzele x, y cu subarborele cu un singur nod care conţine frunza z, este un arbore Huffman cu cheile K'.

Demonstrație Lema 2:

- 1) analog Cost(A') ≤ Cost(A) (Cost(A) = Cost(A') + w(x) + w(y))
- 2) pp există A'' a.i. Cost(A'') < Cost(A') =>
  - Cost(A'') < Cost(A) - (w(x) + w(y));
  - Cost(A'') + w(x) + w(y) < Cost(A); => A nu este Huffman (contradicţie)

Demonstrație (4)

Teorema 1 – Algoritmul Huffman construiește un arbore Huffman.

Demonstrație Teorema 1: prin inducție după numărul de chei din mulțimea K.

- $n \leq 2$ => evident
- $n > 2$
  - Ip. Inductivă: algoritmul Huffman construiește arbori Huffman pentru orice mulțime cu n-1 chei
  - Fie $K = \{k_1, k_2, \ldots, k_n\}$ a.i. $w(k_1) \leq w(k_2) \leq \ldots \leq w(k_n)$

Demonstrație (5)

- Cf. Lema 1, ∃ Un arbore Huffman unde cheile k_1, k_2 sunt pe același nivel şi descendente ale aceluiași nod.
- A_{n-1} – arborele cu n-1 chei K' = K - $\{k_1,k_2\} \cup z$ unde w(z) = $w(k_1) + w(k_2)$.
- A_{n-1} rezultă din A_n prin modificările prezentate in Lema 2 => A_{n-1} este Huffman, şi cf. ipotezei inductive e construit prin algoritmul Huffman(K',p').
- => Algoritmul Huffman(K, p) construiește arborele format din k_1 si k_2 si apoi lucrează ca şi algoritmul Huffman(K', p') ce construiește A_{n-1} => Algoritmul Huffman(K, p) construiește un arbore Huffman.

Când funcţionează algoritmii Greedy? (1)

1) Problema are proprietatea alegerii locale

- Alegând soluţia optimă local se ajunge la soluţia optimă global.

2) Problema are proprietatea de substructură optimă

- O soluţie optimă a problemei conţine soluţiile optime ale subproblemelor.

1)+2): facem o alegere locală (lacomă) => rămâne o subproblemă. Soluţia optimă a subproblemei + alegerea locală deja facută => soluţia optimă pentru problemă.

ÎNTREBĂRI?