Proiectarea Algoritmilor

I CA: Stefan Trausan-Matu stefan.trausan@cs.pub.ro

I CB: Traian Rebedea traian.rebedea@cs.pub.ro

I CC: Costin Chiru costin.chiru@cs.pub.ro

Obiectivele cursului (I)

- Cunoașterea unui set important de algoritmi și metode de rezolvare a problemelor de algoritmică. → curs
- Dezvoltarea abilităților de adaptare a unui algoritm la o problemă din viața reală.→ laborator
- Dezvoltarea abilităților de lucru în echipă.→ proiect

Obiectivele cursului (II)

- Utilizarea teoriei predate la curs pentru proiectarea algoritmilor de rezolvare pentru probleme tipice întâlnite în practica dezvoltării sistemelor de programe.
- Discutarea relaţiei dintre caracteristicile problemelor, modul de rezolvare şi calitatea soluţiilor.
- Compararea variantelor unor algoritmi pentru rezolvarea problemelor dificile.

De ce să învăț PA?

- Exemple de utilizări ale PA-ului in diferite meserii:
  - web developer – web social, teoria grafurilor, data mining, clustering;
  - game dev – căutări, grafuri, inteligență artificială;
  - project manager – fluxuri, grafuri de activități;
  - dezvoltator de sisteme de operare – structuri de date avansate, scheme de algoritmi;
  - programator – tot ce tine de algoritmi, in special complexitate si eficiență;
  - tester – tot ce tine de algoritmi, in special complexitate, eficiență si debugging;

Planul cursului (I)

- Scheme de algoritmi
  - Caracteristici ale problemelor şi tehnici asociate de rezolvare: divide & impera (Merge Sort, puterea unui număr), rezolvare lacomă (arbori Hufmann, problema rucsacului continuă), programare dinamică (înmulțirea matricelor, AOC, problema rucsacului discretă). Backtracking cu optimizări. Propagarea restricţiilor.
- Algoritmi pentru jocuri – Minimax şi α-β.
- Algoritmi pentru grafuri
  - Algoritmi pentru grafuri: parcurgeri, sortare topologică, componente tare conexe, articulaţii, punţi, arbori minimi de acoperire, drumuri de cost minim, fluxuri.

Planul cursului (II)

- Rezolvarea problemelor prin căutare euristică
  - Rezolvarea problemelor prin căutarea euristică A*. Completitudine şi optimalitate, caracteristici ale euristicilor.
- Algoritmi aleatorii
  - Algoritmi aleatorii. Las Vegas şi Monte Carlo, aproximare probabilistică.

Evaluarea

- Citiți documentul “Regulament Proiectarea Algoritmilor 2013” de pe site! http://ocw.cs.pub.ro/courses/pa/regulament-general
- Examen 4 p
- Laborator 6 p ☺
  - 3p teme (3 teme punctate egal)
  - 2p laborator
  - 2p proiect
- Activitate științifică – maxim 0,5p bonus
- Obs. 1 - Nu se copiază in facultate!
- Obs. 2 - Prima temă copiată se punctează cu minus valoarea maximă a temei. La a doua tema copiată, se repetă materia!

Feedback 2008, 2009

- Idei preluate din feedback 2008:
  - Schimbarea modului de organizare al proiectului (etape, mod de lucru in echipă).
  - Schimbarea orientării temelor (variante mai ușoare, notare parţială).
  - Subliniată importanţa bibliografiei.
- Idei preluate din feedback 2009:
  - Eliminare restricţii echipa proiect la nivel de grupă.
  - Publicare teme pe infoarena.
  - Fără net în laboratoare.
  - Laboratoare mai bune, teme mai atent elaborate, organizare mai bună.
  - Evitată ora 8 dimineața pentru curs. ☺

Feedback 2010 (1)

- Curs:
  - Prea puține 2 ore. (nu avem ce face, așa e in programă)
  - Evitată ora 8 dimineața pentru curs. (Am reușit!!!)
  - Pseudocod uniform - română sau engleză. (Am trecut totul în română)
  - Evitarea greșelilor din cursuri. (îmi cer scuze de pe acum pentru eventualitatea in care mai apar)

- Laborator:
  - Prea grele. (am încercat să le ușurăm prin introducerea de schelete de cod)
  - Furnizare de rezolvări. (am început anul trecut – sperăm să fie ok anul acesta)
  - Laboratoare mai clare si mai uniforme din punctul de vedere al scheletului de cod. (două persoane vor scrie codul de la laboratoare – C++/Java)
  - Păreri împărțite referitoare la absenţa internetului din laborator. (niciodată nu vom putea să împăcăm pe toată lumea!)

Feedback 2010 (2)

- Teme:
  - Mai puține (3 in loc de 4) si mai utile. (vor fi 3 teme)
  - Corectate mai repede si uniform pe serii. (fiecare temă va fi corectată de către o singură persoană pe serie)
  - Temă de recuperare peste vară. (va fi disponibilă o astfel de temă)
- Proiect:
  - Schimbarea șahului. (F1- ants)
  - Interesantă competiția finală. (ne bucurăm!, am păstrat-o)
  - E bine că este pe grupe – abilități de lucru in echipă. (asta ne și dorim!)
  - Punctați si cei care nu intră în grupe. (încercăm să schimbăm punctarea – se va face un clasament general deoarece vom încerca să facem meciuri fiecare cu fiecare)
- Examen:
  - Timp prea scurt. (deja am dublat timpul fata de acum doi ani, mai mult de atât nu vi se va pune la dispoziție!)
  - Examen greu!

Feedback 2011 (1)

- Curs:
  - Demonstrații prea multe/puține în curs – să se dea mai multe demonstrații la examen! (părerile sunt împărțite!)
  - Introducere de flashuri pentru a demonstra algoritmii (anul acesta nu avem cum, dar sperăm să le introducem la anul! – Dacă veți găsi astfel de flash-uri sau tool-uri, va rog să mă anunțați și pe mine ca să le pun la bibliografia cursurilor și să le poată folosi și ceilalți colegi de-ai voștri)
  - Timp prea puțin la curs (2 ore) fiind necesare 3 ore cel puțin, prea mulți algoritmi (nu avem ce face, așa e in programă)
  - Bazat pe cărțile lui Cormen și Giumale (Așa este! Este necesară citirea capitolelor din aceste cărți! Vezi slide 15!)
- Laborator
  - Introducere de schelete de cod în Java pentru laborator (sperăm că veți fi mulțumiți de ele)
  - Toți studenții să aibă timp sa vadă din timp problemele de laborator (vor fi disponibile din timp și vă încurajăm să vă uitați peste probleme și peste laborator înainte de a face laboratorul)

Feedback 2011 (2)

- Teme
  - Teme bine alese, care reflectă probleme reale și sunt în concordanță cu materia predată (vom încerca și anul acesta să dăm teme cel puțin la fel de interesante)
- Proiect
  - Acordare premii pentru câștigătorii concursului de la proiect (încercăm să găsim sponsori pentru o astfel de inițiativă – anul trecut echipa de PA a asigurat premii din banii personali)
  - Erori in serverul de proiect (au fost eliminate astfel de erori întrucât anul acesta am schimbat proiectul și vom folosi o platformă care deja și-a demonstrat valoarea într-un concurs internațional)
- Overall
  - Entuziasm din partea echipei (ne pare bine că a fost remarcat acest aspect! Sperăm să aveți parte de același entuziasm)

Feedback 2012

- Curs
  - Complexitatea materiei este cu mult peste numarul alocat in orar (2 ore curs, 2 ore laborator).
  - Timpul dedicat cursului a fost prea putin si cateodata se mergea foarte repede, se explica rapid doar pentru a ne incadra in timp. → trebuie sa acopar materia
  - mult mai mult de 3 ore; → asa e in programa
  - Daca nu se face de 3 ore sugerez sa se predea mai succint totul, ramanand sa aprofundeze copiii acasa.. ca doar sunt la facultate nu la clasa a patra.
  - ar putea fi scoase multe din teoremele si demonstratiile din slide-urile de curs → nu se poate
  - Putine probleme (aplicatii, exemple de probleme) discutate la curs. → asta se face la laborator

Feedback 2012

- Examen
  - Timp mai mult la examene si mai putine subiecte, nu trebuie sa fim presati de timp, in plus nu trebuie ca dupa fiecare subiect sa ni se ia foile. → ne scuteste de facut politie + veti lucra mereu sub presiunea timpului, nu ar strica sa va obisnuiti
- Laborator
  - scheletul de cod ar fi bine sa fie pus pe site din timp.
  - notele de la teme si laborator apar cu intarziere (4-5 saptamani); → sper ca nu vor mai fi probleme
  - unele exercitii sa fie mai usoare pentru ca 2 ore nu sunt suficiente pentru unii sa rezolve probleme propuse la laborator. → am transmis asistentilor acest lucru
  - sa nu se mai suprapuna temele si proiectul ca deadline cu celelalte teme. → problema mai complexa

Bibliografie

- Introducere in Analiza Algoritmilor de *Cristian Giumale* – Ed. Polirom 2004
- Introducere in Algoritmi *de Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest* – Ed. Agora
- Vedeţi şi recomandările din Regulament!
- Mențiune importanta – **slide-urile reprezintă doar un suport pentru prezentare!**

Proiectarea Algoritmilor

Curs 1 – Scheme de algoritmi –
Divide et impera + Greedy

Curs 1 – Cuprins

- Scheme de algoritmi
- Divide et impera
- Exemplificare folosind Merge sort
- Alte exemple de algoritmi divide et impera
- Greedy
- Exemplificare folosind arbori Huffman
- Demonstrația corectitudinii algoritmului Huffman

Curs 1 – Bibliografie

- Giumale – Introducere in Analiza Algoritmilor cap 4.4
- Cormen – Introducere în Algoritmi cap. 17
- http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf
- 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

Scheme de algoritmi

- Prin scheme de algoritmi înțelegem tipare comune pe care le putem aplica în rezolvarea unor probleme similare.
- O gamă largă de probleme se poate rezolva folosind un număr relativ mic de scheme.
- => Cunoașterea schemelor determină o rezolvare mai rapidă și mai eficientă a problemelor.

Divide et Impera (I)

- Ideea (divide si cucerește) este atribuită lui Filip al II-lea, regele Macedoniei (382-336 i.e.n.), tatăl lui Alexandru cel Mare și se referă la politica acestuia față de statele grecești.

- In CS – Divide et impera se referă la o clasă de algoritmi care au ca principale caracteristici faptul că împart problema în subprobleme similare cu problema inițială dar mai mici ca dimensiune, rezolvă problemele recursiv și apoi combină soluțiile pentru a crea o soluție pentru problema originală.

Divide et Impera (II)

- Schema Divide et impera constă în 3 pași la fiecare nivel al recurenței:
  - Divide problema dată într-un număr de subprobleme;
  - Impera (cucerește) – subproblemele sunt rezolvate recursiv. Dacă subproblemele sunt suficient de mici ca date de intrare se rezolvă direct (ieșirea din recurență);
  - Combină – soluțiile subproblemelor sunt combinate pentru a obține soluția problemei inițiale.

Divide et Impera – Avantaje și Dezavantaje

- Avantaje:
  - Produce algoritmi eficienți.
  - Descompunerea problemei în subprobleme facilitează paralelizarea algoritmului în vederea execuției sale pe mai multe procesoare.
- Dezavantaje:
  - Se adaugă un overhead datorat recursivității (reținerea pe stivă a apelurilor funcțiilor).

Merge Sort (I)

- Algoritmul Merge Sort este un exemplu clasic de rezolvare cu D&I.
- Divide: Divide cele n elemente ce trebuie sortate în 2 secvențe de lungime n/2.
- Impera: Sortează secvențele recursiv folosind *merge sort.*
- Combină: Secvențele sortate sunt asamblate pentru a obține vectorul sortat.
- Recurența se oprește când secvența ce trebuie sortată are lungimea 1 (un vector cu un singur element este întotdeauna sortat ☺) .
- Operația cheie este: asamblarea soluțiilor parțiale.

Merge Sort (II)

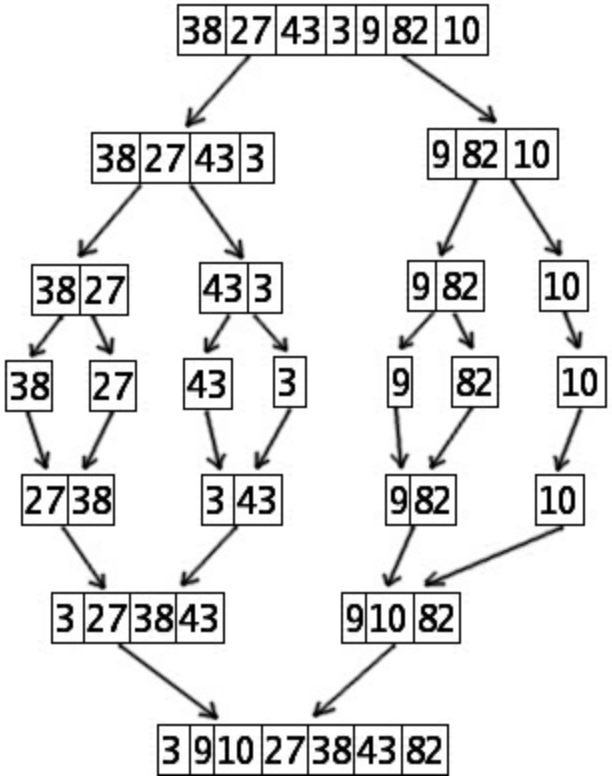
- Algoritm [Cormen]
  - MERGE-SORT(A, p, r)
  - 1 **Dacă** $p < r$
  - 2 **Atunci** $q \leftarrow [(p + r)/2]$ // divide
  - 3 MERGE-SORT(A, p, q) //impera
  - 4 MERGE-SORT(A, $q + 1$, r)
  - 5 MERGE(A, p, q, r) // combină
    // (interclasare)

Merge Sort (III) – Algoritmul de interclasare

- Algoritm [Cormen]
  - MERGE(A, p, q, r) // p si r sunt capetele intervalului, q este "mijlocul"
  - 1 $n_1 \leftarrow q - p + 1$ // numărul de elemente din partea stânga
  - 2 $n_2 \leftarrow r - q$ // numărul de elemente din partea dreapta
  - 3 **creează** vectorii $S[1 \rightarrow n_1 + 1]$ și $D[1 \rightarrow n_2 + 1]$
  - 4 **Pentru** i **de la** 1 **la** n_1
  - 5 $S[i] \leftarrow A[p + i - 1]$ // se copiază partea stânga în S
  - 6 **Pentru** j **de la** 1 **la** n_2
  - 7 $D[j] \leftarrow A[q + j]$ // si partea dreapta în D
  - 8 $S[n_1 + 1] \leftarrow \infty$
  - 9 $D[n_2 + 1] \leftarrow \infty$
  - 10 $i \leftarrow 1$
  - 11 $j \leftarrow 1$
  - 12 **Pentru** k **de la** p **la** r // se copiază înapoi în vectorul de
  - 13 **Dacă** $S[i] \leq D[j]$ // sortat elementul mai mic din cei
  - 14 **Atunci** $A[k] \leftarrow S[i]$ // doi vectori sortați deja
  - 15 $i \leftarrow i + 1$
  - 16 **Altfel** $A[k] \leftarrow D[j]$
  - 17 $j \leftarrow j + 1$

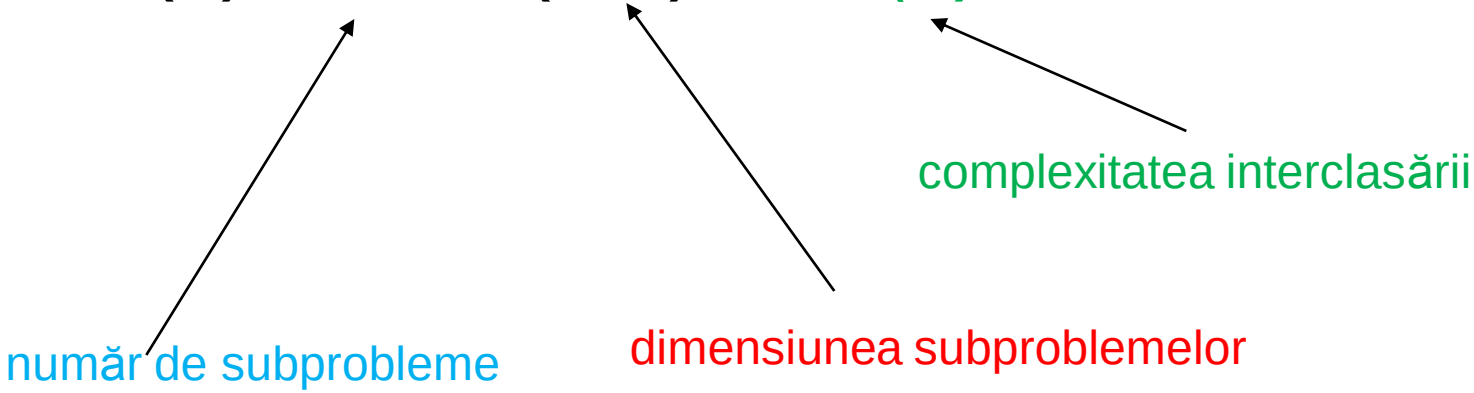
Exemplu funcționare Merge Sort

- Exemplu funcționare [Wikipedia]:



Merge Sort - Complexitate

- $T(n) = 2 * T(n/2) + \Theta(n)$



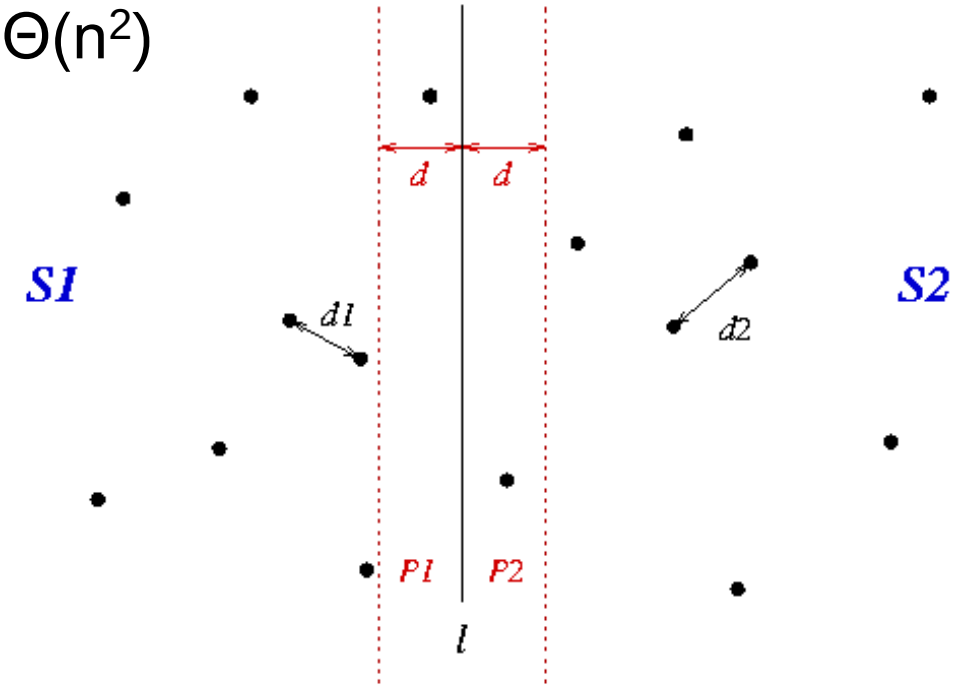
=> (din T. Master) $T(n) = \Theta(n * \log n)$

Divide et Impera – alte exemple (I)

- Calculul puterii unui număr: x^n
  - Algoritm "clasic":
    - **Pentru** *i* **de la** 1 **la** *n* rez = rez * x;
    - **Întoarce** rez.
  - Complexitate: $\Theta(n)$
  - Algoritm Divide et Impera:
    - **Dacă** n este par
      - **Atunci Întoarce** $x^{n/2} * x^{n/2}$
      - **Altfel** (n este impar) **Întoarce** $x * x^{(n-1)/2} * x^{(n-1)/2}$
  - Complexitate: $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$

Divide et Impera – alte exemple (II)

- Calculul celei mai scurte distanțe între 2 puncte din plan (http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf)
  - algoritmul naiv – $\Theta(n^2)$



Divide et Impera – alte exemple (III)

- Sortează punctele în ordinea crescătoare a coordonatei x ($\Theta(n \log n)$);
- Împărțim setul de puncte în 2 seturi de dimensiune egală și calculăm recursiv distanța minimă în fiecare set (l = linia ce împarte cele 2 seturi, d = distanța minimă calculată în cele 2 seturi);
- Elimină punctele care sunt plasate la distanță de l > d;
- Sortează punctele rămase după coordonata y;
- Calculează distanțele de la fiecare punct rămas la cei 5 vecini (nu pot fi mai mulți);
- Dacă găsește o distantă < d, atunci actualizează d.

Divide et Impera – Temă de gândire

- Se dă o mulțime M de numere întregi si un număr x. Se cere să se determine dacă există $a,b \in M$ a.î. $a + b = x$.
- Algoritmul propus trebuie să aibă complexitatea $\theta(n \log n)$.
- Temele de la curs sunt facultative!☺

Greedy (I)

- 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 (II)

- 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 // da optimizare(s, Crit_optim, Problemă)
  - 11. **Altfel** sol_parțiale = sol_parțiale \ {s}; // nu
  - 12. **Întoarce** sol_fin;

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 (I)

- K – mulțimea de simboluri ce vor fi codificate. (a, n, “ “, r, e, m)
- Arbore de codificare a cheilor K este un arbore binar ordonat cu proprietățile:
  - Doar frunzele conțin cheile din K; nu există mai mult de o cheie într-o frunză;
  - Toate nodurile interne au exact 2 copii;
  - Arcele sunt codificate cu 0 și 1 (arcul din stânga unui nod – codificat cu 0).
- 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 (II)

- 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 – algoritm de construcție (I)

- 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.

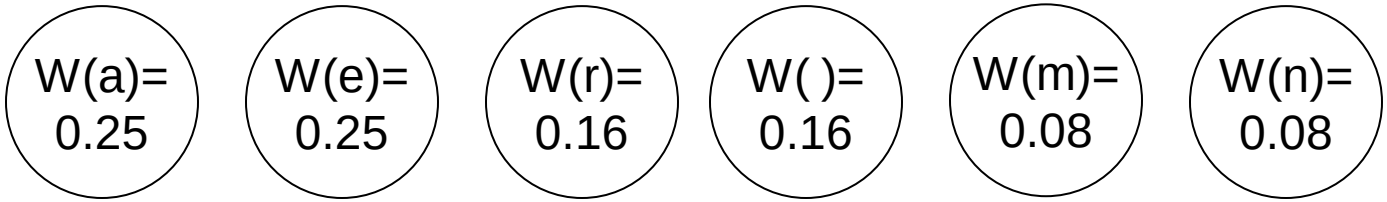
- 2. Se aleg doi subarbori a şi b din Arb astfel încât a şi b au pondere minimă.

Arbori Huffman – algoritm de construcție (II)

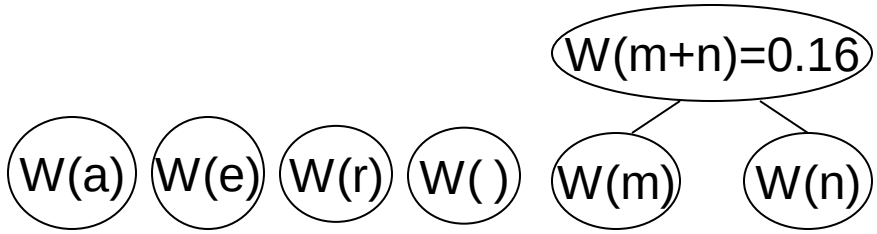
- 3. Se construiește un arbore binar cu o rădăcina *r* care nu conține nici o cheie și cu descendenții a și b. Ponderea arborelui este definită ca w(r) = w(a) + w(b).
- 4. Arborii a și b sunt eliminați din Arb iar r este inserat în Arb.
- 5. Se repetă procesul de construcție descris de pașii 2-4 până când mulțimea Arb conține un singur arbore – Arborele Huffman pentru cheile K.

Arbori Huffman – Exemplu (I)

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

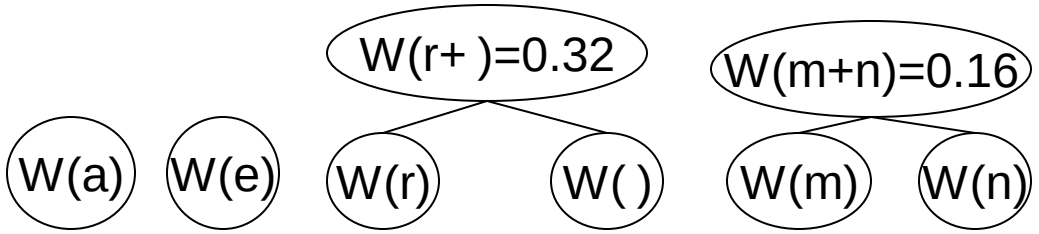


- Pasii 2-4:

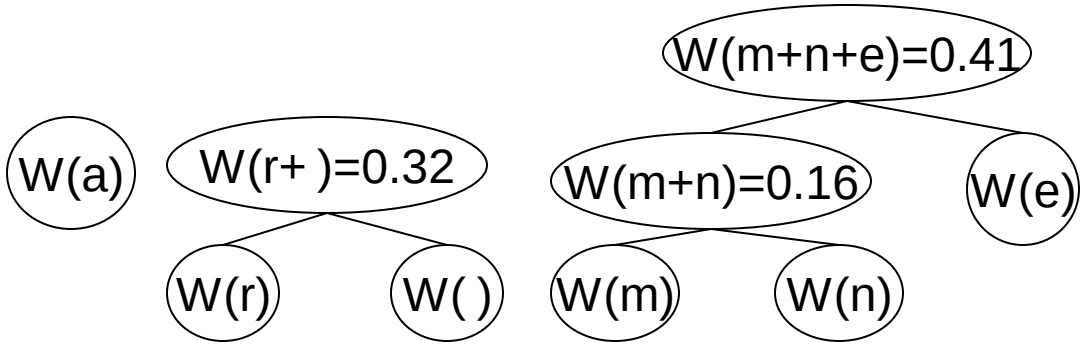


Arbori Huffman – Exemplu (II)

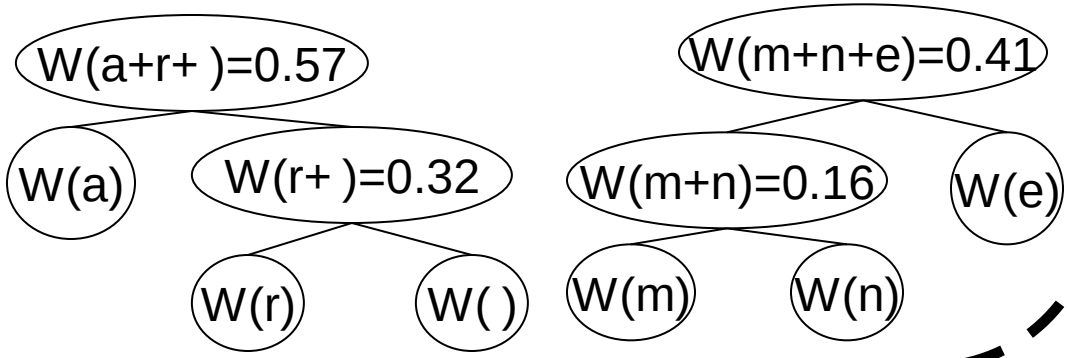
- Pasii 2-4 (II):



- Pasii 2-4 (III):

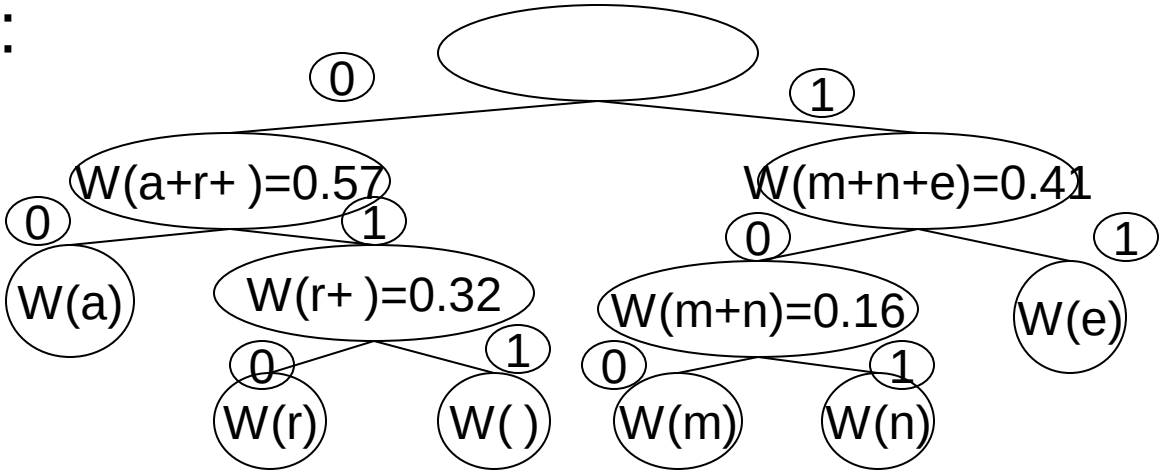


- Pasii 2-4 (IV):



Arbori Huffman – Exemplu (III)

- Pasii 2-4 (V):



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

- 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 = {k ∈ K | frunză(k, p(k))};
  - 2. **Cât timp** (card (Arb) > 1) // am mai mulți subarbori
  - 3. Fie a_1 și a_2 arbori din Arb a.i. ∀a ∈ Arb a ≠ a_1 si a ≠ a_2, avem $w(a_1) \leq w(a)$ și $w(a_2) \leq w(a)$); // practic se extrage // de două ori minimul și se salvează în a_1 și a_2
  - 4. Arb = Arb \ {a_1, a_2} U nod_intern(a_1, a_2, $w(a_1) + w(a_2)$);
  - 5. **Dacă** (Arb = Φ)
  - 6. **Întoarce** arb_vid;
  - 6. **Altfel**
  - 7. fie A singurul arbore din mulțimea Arb;
  - 8. **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)

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

ÎNTREBĂRI?