Universitatea POLITEHNICA din București
Facultatea de Automatică și Calculatoare

Proiectarea algoritmilor (PA)

- seria CD -

Andrei Mogoș - Suport de curs

2020-2021

Curs 1: Informații generale

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)

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 (1)

- Scheme de algoritmi
  - Caracteristici ale problemelor şi tehnici asociate de rezolvare:
    - **divide et 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.

Planul cursului (2)

- Algoritmi pentru grafuri
  - Algoritmi pentru grafuri: parcurgeri, sortare topologică, componente tare conexe, articulaţii, punţi, arbori minimi de acoperire, drumuri de cost minim, fluxuri.
- Rezolvarea problemelor prin căutare euristică
  - Rezolvarea problemelor prin căutarea euristică A*. Completitudine şi optimalitate, caracteristici ale euristicilor.
- Algoritmi aleatorii
  - Algoritmi aleatori. Las Vegas şi Monte Carlo, aproximare probabilistică.

Notare

- Laborator: 6p (maxim 6p)
  - Activitate la curs (teste de curs): 1p
  - Activitate de laborator: 2p
  - Teme: 3p
  - Bonus Concursuri: 1p
- Examen: 4p

- Condiţie de intrare în examen: Laborator >= 3p
- Condiţie de promovare a examenului: Examen >= 2p

De la teorie la practică

- Algoritmi şi metode:
  - Înțelegere teoretică: la curs şi la laborator
  - Aplicare teoretică: calcul de mână la laborator
  - Implementare C: la laborator
  - Implementare C/C++, Java, …: la teme

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

Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare

Proiectarea algoritmilor (PA)

- seria CD -

Andrei Mogoș - Suport de curs

Curs 1: Scheme de algoritmi. Divide et impera

Curs 1 – Cuprins

- Scheme de algoritmi
- Divide et impera
- Exemplificare folosind Merge sort
- Alte exemple de algoritmi divide et impera

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 pot rezolva folosind un număr relativ mic de scheme.

- => Cunoașterea schemelor determină o rezolvare mai rapidă și mai eficientă a problemelor.

Divide et Impera (1)

- 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 (2)

- 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 (1)

- 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 (2)

- Algoritm [Cormen]

```
MERGE-SORT(A, p, r)
1 dacă p < r
2       q ← [(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)
```

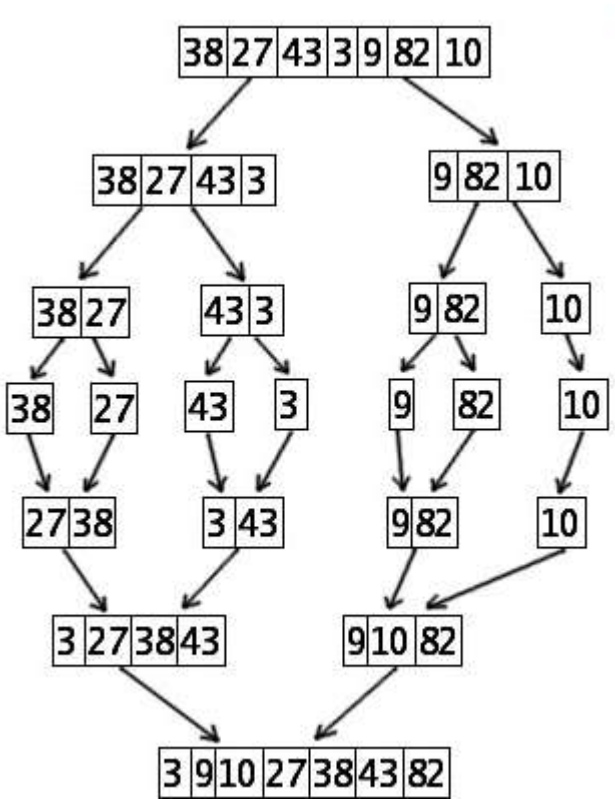
Merge Sort (3) – Algoritmul de interclasare

- Algoritm [Cormen]

```
MERGE(A, p, q, r) // p si r sunt capetele intervalului, q este "mijlocul"
 1    n1 ← q - p + 1 // numărul de elemente din partea stânga
 2    n2 ← r – q  // numărul de elemente din partea dreapta
 3    creează vectorii S[1 -> n1 + 1] și D[1 -> n2 + 1]
 4    pentru i de la 1 la n1
 5        S[i] ← A[p + i - 1] // se copiază partea stânga în S
 6    pentru j de la 1 la n2
 7        D[j] ← A[q + j] // si partea dreapta în D
 8    S[n1 + 1] ← ∞
 9    D[n2 + 1] ← ∞
10    i ← 1
11    j ← 1
12    pentru k de la p la r  // se copiază înapoi în vectorul de
13        dacă S[i] ≤ D[j]      // sortat elementul mai mic din cei
14            A[k] ← S[i]       // doi vectori sortați deja
15            i ← i + 1
16        altfel
17            A[k] ← D[j]
18            j ← j + 1
```

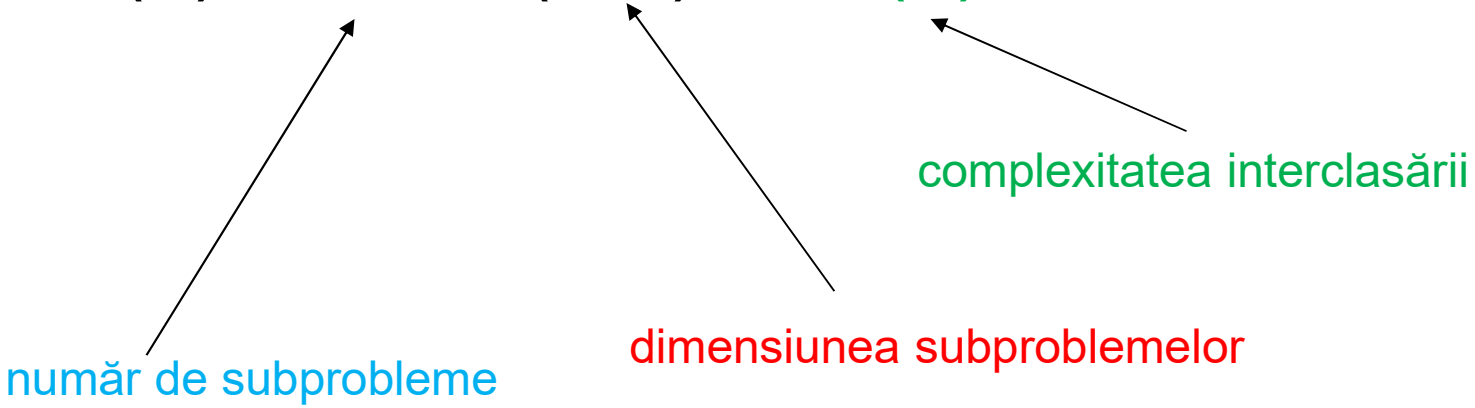
Merge Sort - Exemplu de funcționare

- 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 (1)

1) 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: $T(n) = T(n/2) + \Theta(1) \Rightarrow \Theta(\log n)$

Divide et Impera – alte exemple (2)

2) Căutare binară: se caută o valoare într-un vector sortat crescător.

```
cautBin(v, s, d, x) [Wikipedia] // este căutat x în vectorul v sortat crescător
1    if (s > d)
2        intoarce -1
3    altfel
4        m = (s + d) / 2
5        daca v[m] == x
6            intoarce m
7        altfel
8            daca (x < v[m])
9                cautBin(v, s, m - 1, x)
10           altfel
11               cautBin(v, m + 1, d, x)
```

Apel: cautBin(v, 1, n, x) // n este numărul de elemente din vector

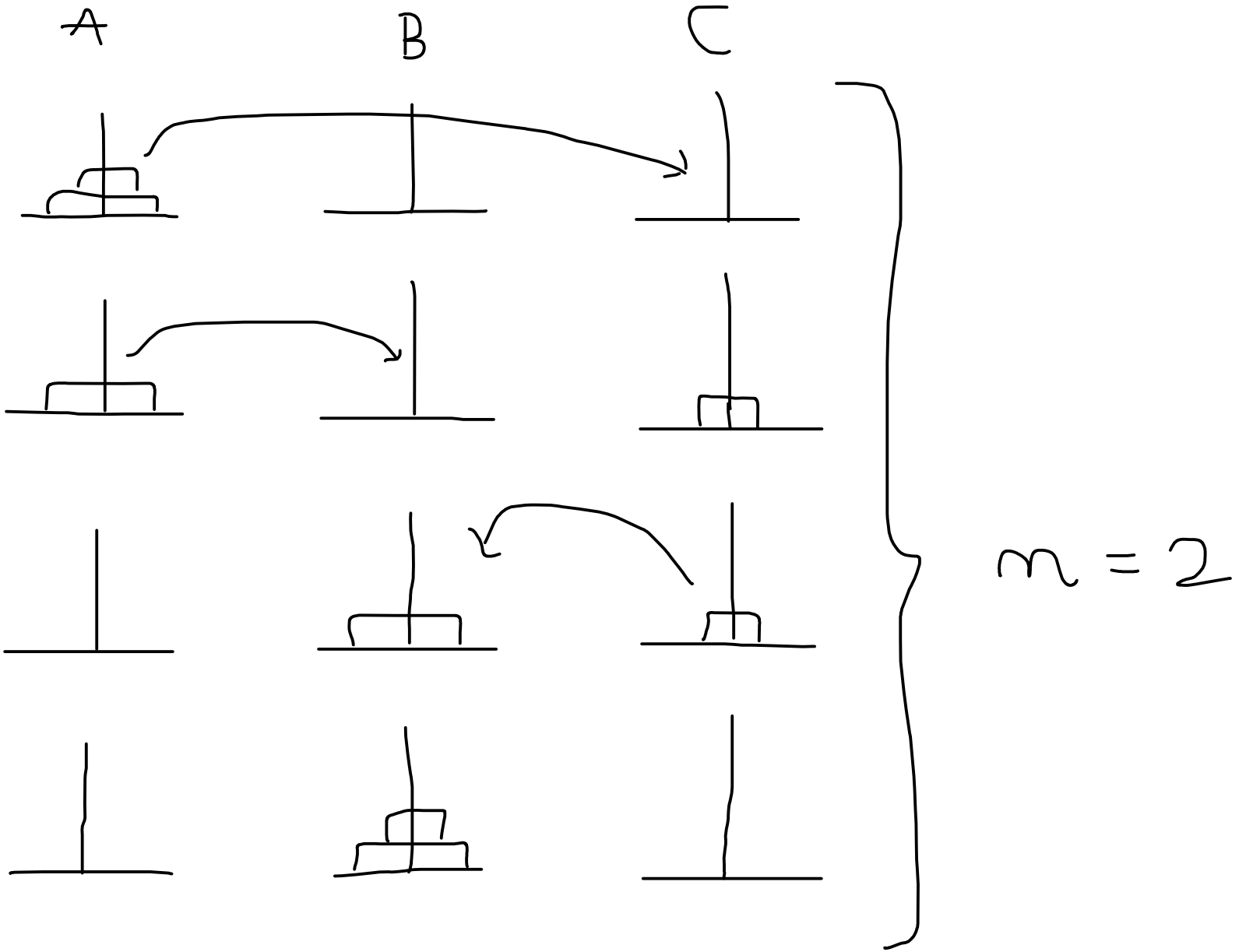
Complexitate: $T(n) = T(n/2) + \Theta(1) \Rightarrow \Theta(\log n)$

Divide et Impera – alte exemple (3)

3) Turnurile din Hanoi: descriere pe tablă

```
// mutăm n discuri de pe turnul a pe turnul b folosind turnul c
hanoi(n, a, b, c)
1     if n > 0
2         hanoi(n - 1, a, c, b)
3         mută(1, a, b)
4         hanoi(n - 1, c, b, a)
```

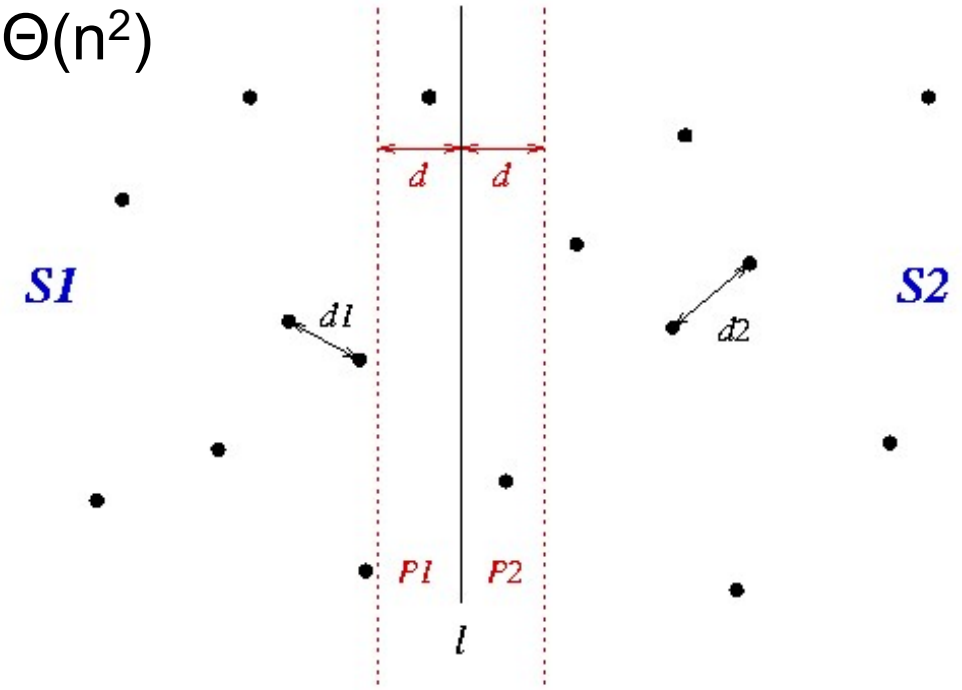
Complexitate: $T(n) = 2 * T(n - 1) + \Theta(1) \Rightarrow \Theta(2^n)$


A
B
C
m = 2

Divide et Impera – alte exemple (4)

4) 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 – Temă de gândire (1)

- 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!

Divide et Impera – Temă de gândire (2)

- Cum sortăm un vector fără interclasare sau alte operații complexe în etapele de împărțire a problemei și de combinare a soluțiilor?

- Hint: se folosesc trei apeluri recursive!
- Observație! Algoritmul este ineficient, dar este interesant!

Divide et Impera – Temă de gândire (3)

- Cum se găseste eficient elementul median al unui vector?
- Problema mai generală: cum se găsește al k-lea cel mai mic (sau cel mai mare) element dintr-un vector?
- Eficient înseamnă Θ(n)
- http://www.cs.rit.edu/~ib/Classes/CS515_Spring12-13/Slides/022-SelectMasterThm.pdf

ÎNTREBĂRI?