Curs 14:

Recapitulare

Tematica examen

- Algoritmi iterativi: proiectare și analiza complexitate
- Algoritmi recursivi: descriere și analiza complexitate

- Tehnici de reducere și divizare: descriere și analiza complexitate; aplicații

- Algoritmi de sortare:
  - algoritmi elementari (inserție, selecție, interschimbare)
  - algoritmi eficienți (sortare prin interclasare, sortare rapidă)

- Stive și cozi – implementarea operațiilor, complexitatea operațiilor + aplicații simple
- Liste simplu și dublu înlănțuite – implementarea operațiilor, complexitatea operațiilor + aplicații simple

- Căutare local optimală (greedy) – aplicații
- Backtracking – exemple simple

Exemple

- Tehnica reducerii: Problema turnurilor din Hanoi
- Tehnica divizării: Problema selecției
- Tehnica greedy: construirea unui arbore minim de acoperire
- Tehnica backtracking: generare submultimi

Problema turnurilor din Hanoi

Istoric: problemă propusă de matematicianul Eduard Lucas în 1883

Ipoteze:

- Considerăm 3 vergele etichetate cu S (sursă), D (destinație) și I (intermediar).
- Inițial pe vergeaua S sunt plasate n discuri de dimensiuni diferite în ordine descrescătoare a dimensiunilor (cel mai mare disc este la baza vergelei iar cel mai mic în varf)

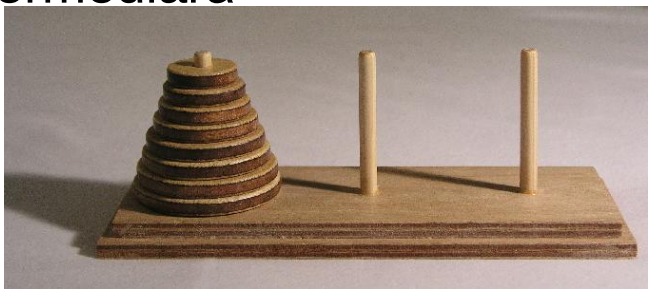
Scop:

- Să se mute toate discurile de pe S pe D (la final sunt tot în ordine descrescătoare) utilizând vergeaua I ca intermediară

Restricție:

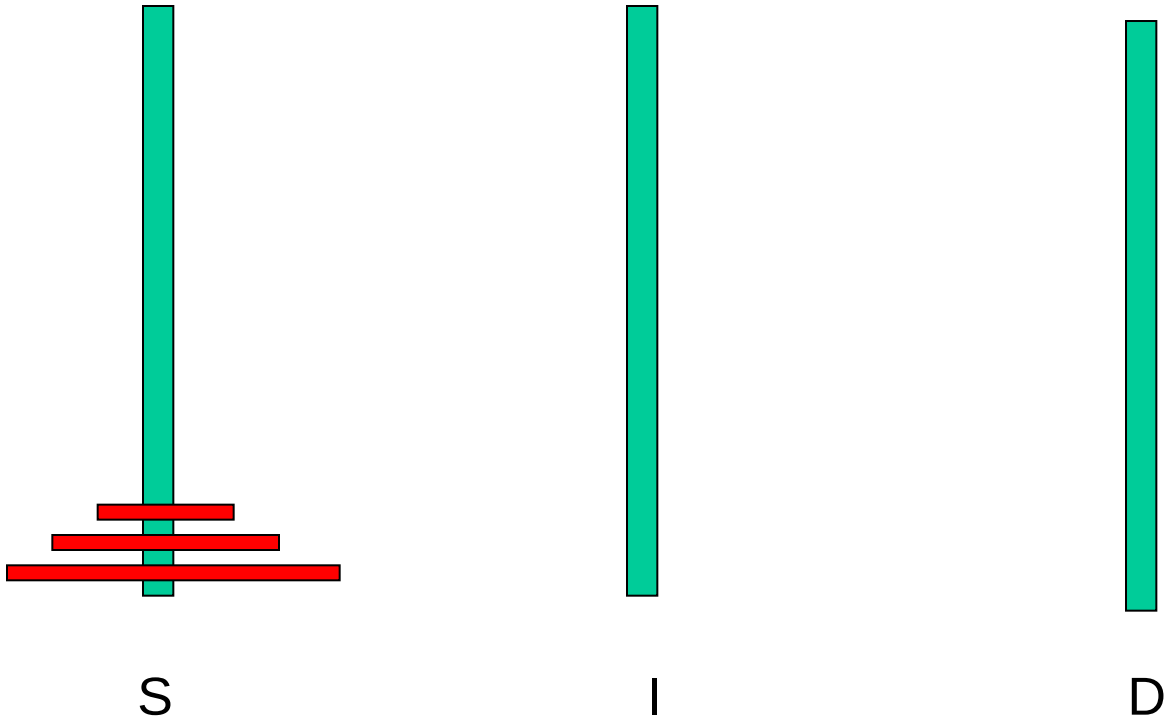
- La o etapă se poate muta un singur disc

și este interzisă plasarea unui disc mai mare peste un disc mai mic.



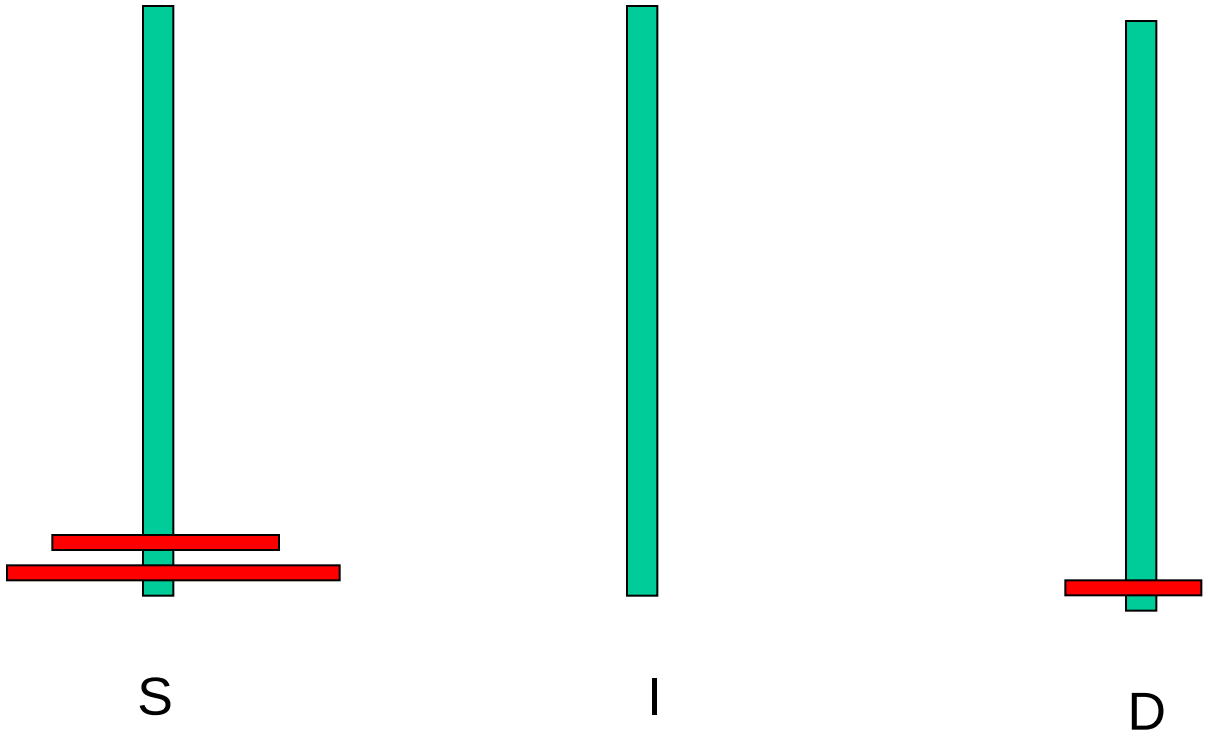
Problema turnurilor din Hanoi

Prima mutare: S->D



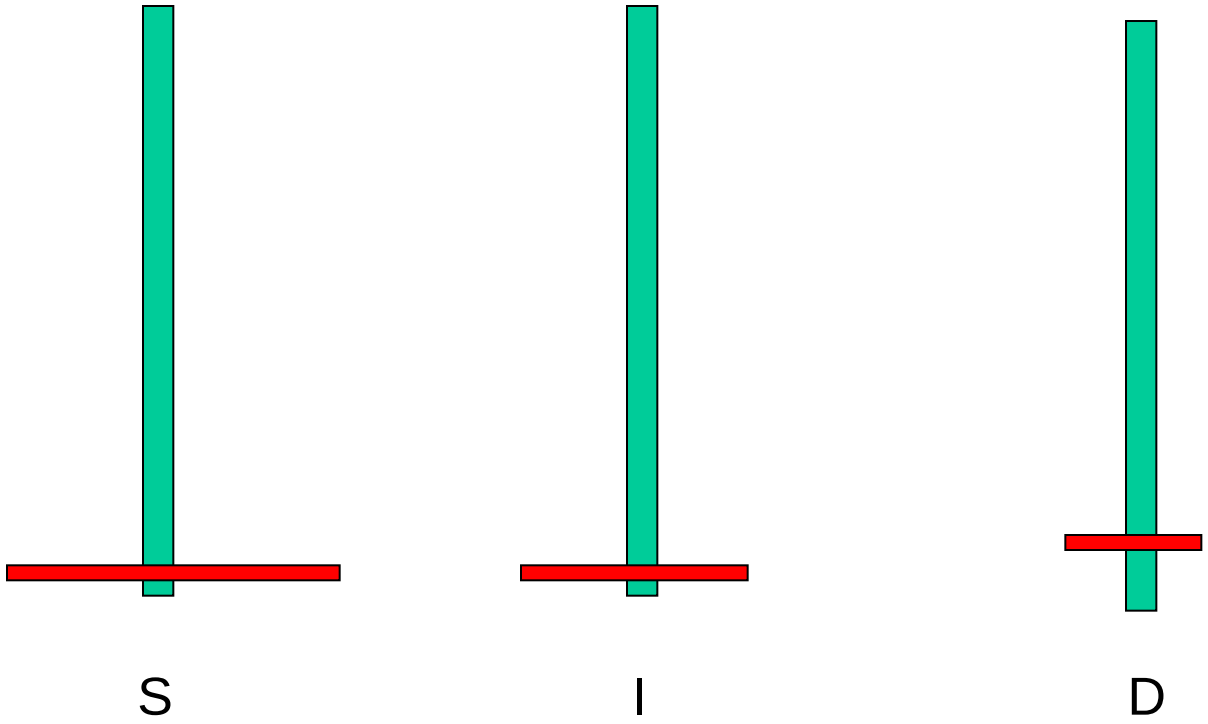
Problema turnurilor din Hanoi

A doua mutare: S->I



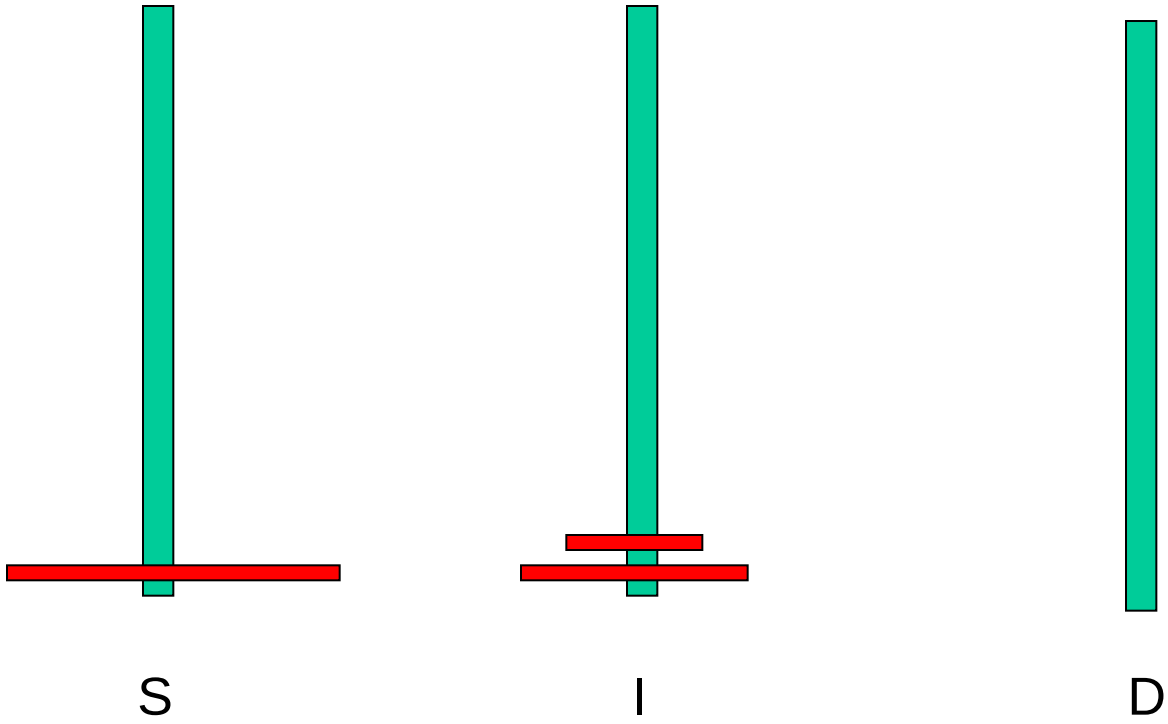
Problema turnurilor din Hanoi

A treia mutare: D->i



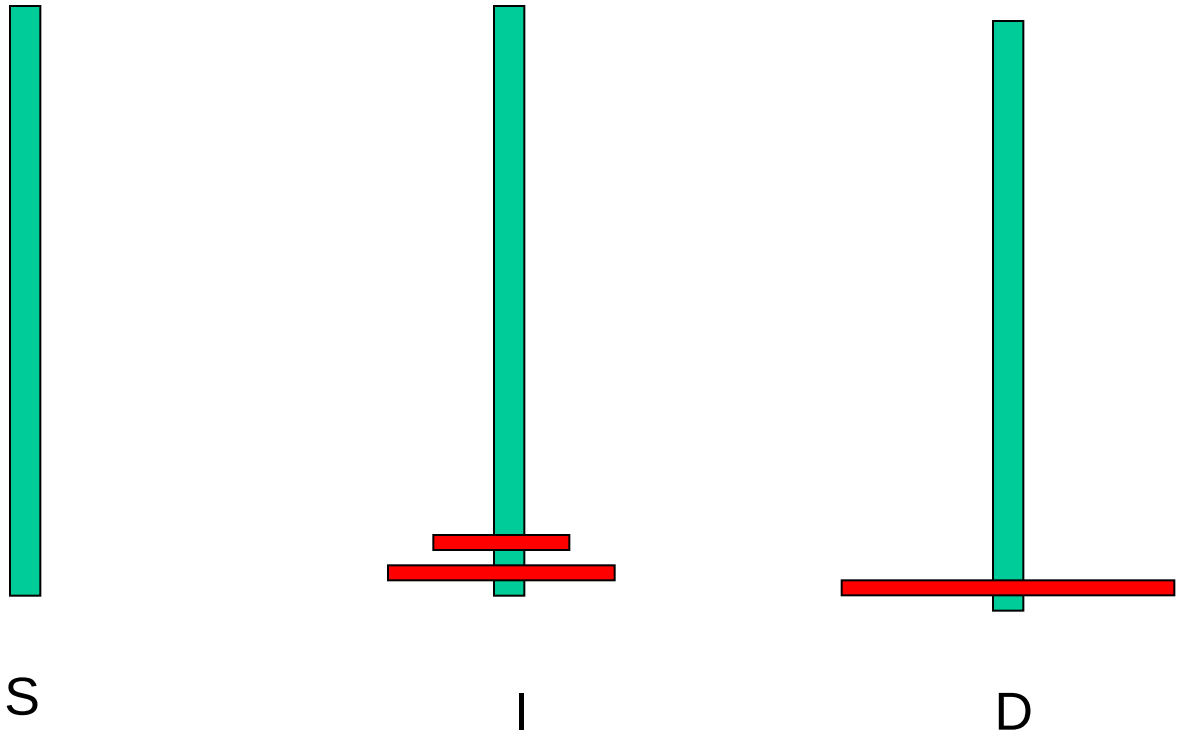
Problema turnurilor din Hanoi

A patra mutare: S->D



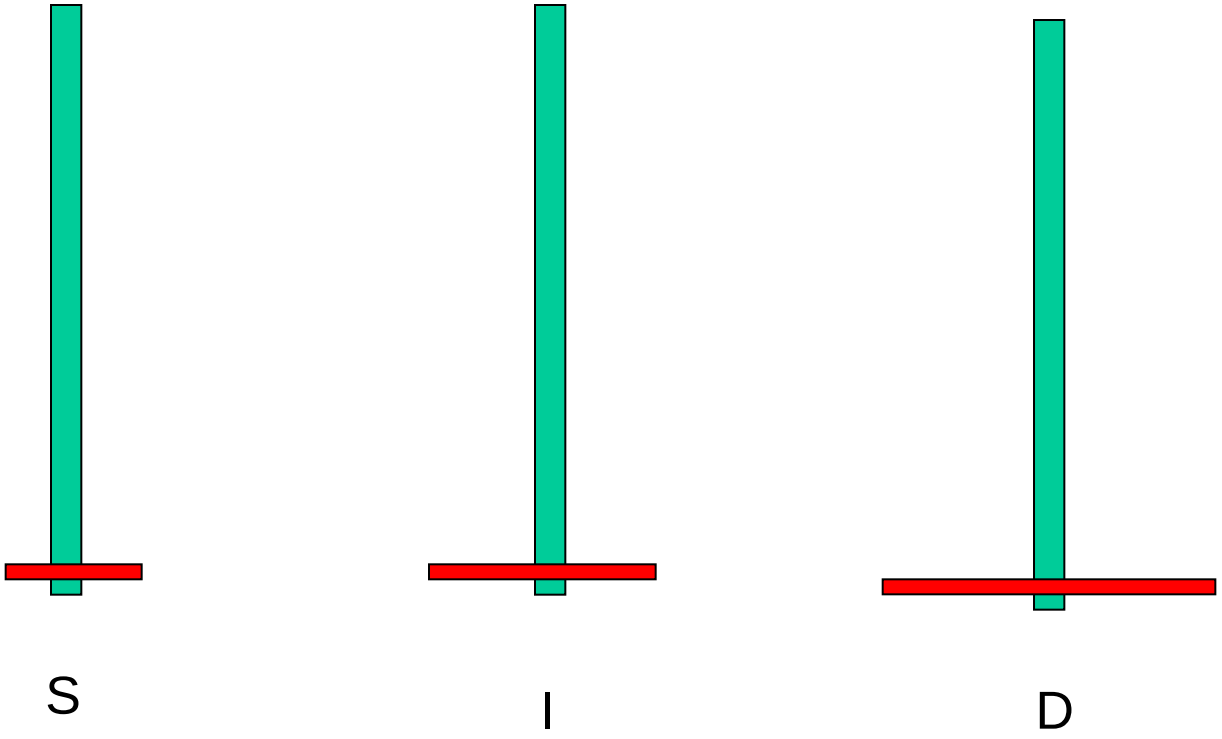
Problema turnurilor din Hanoi

A cincea mutare: I->S



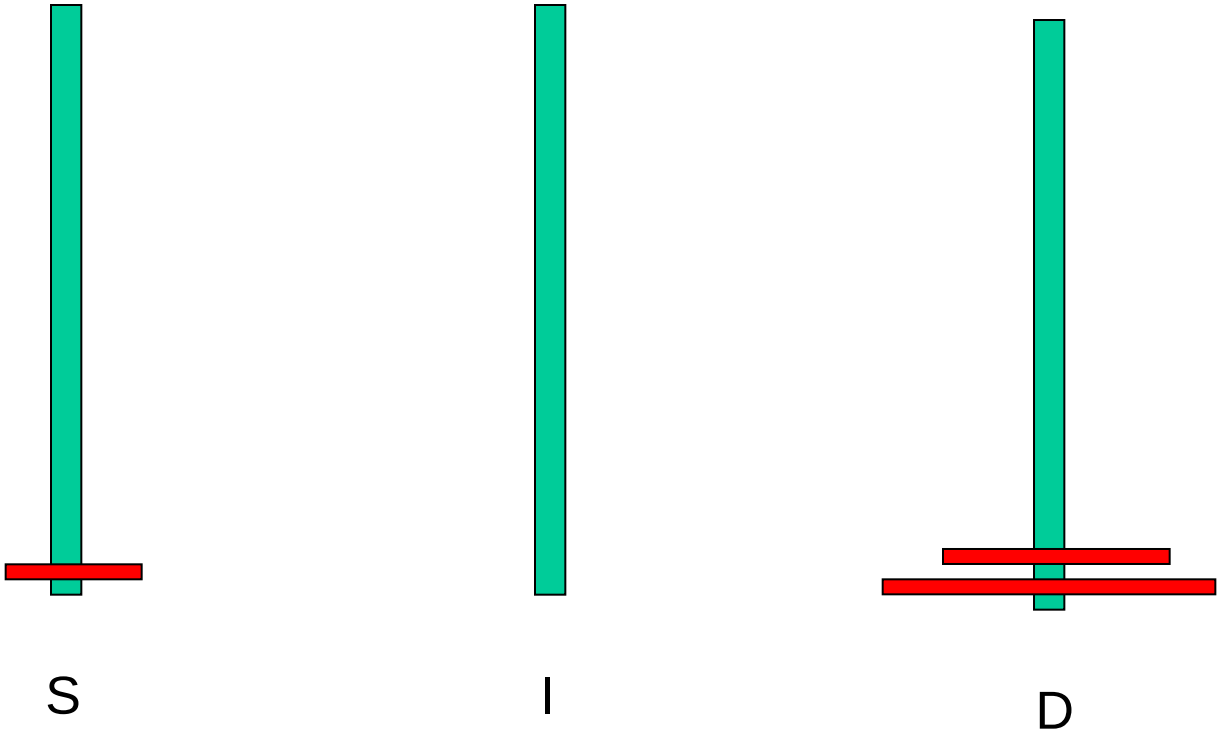
Problema turnurilor din Hanoi

A sasea mutare: I->D



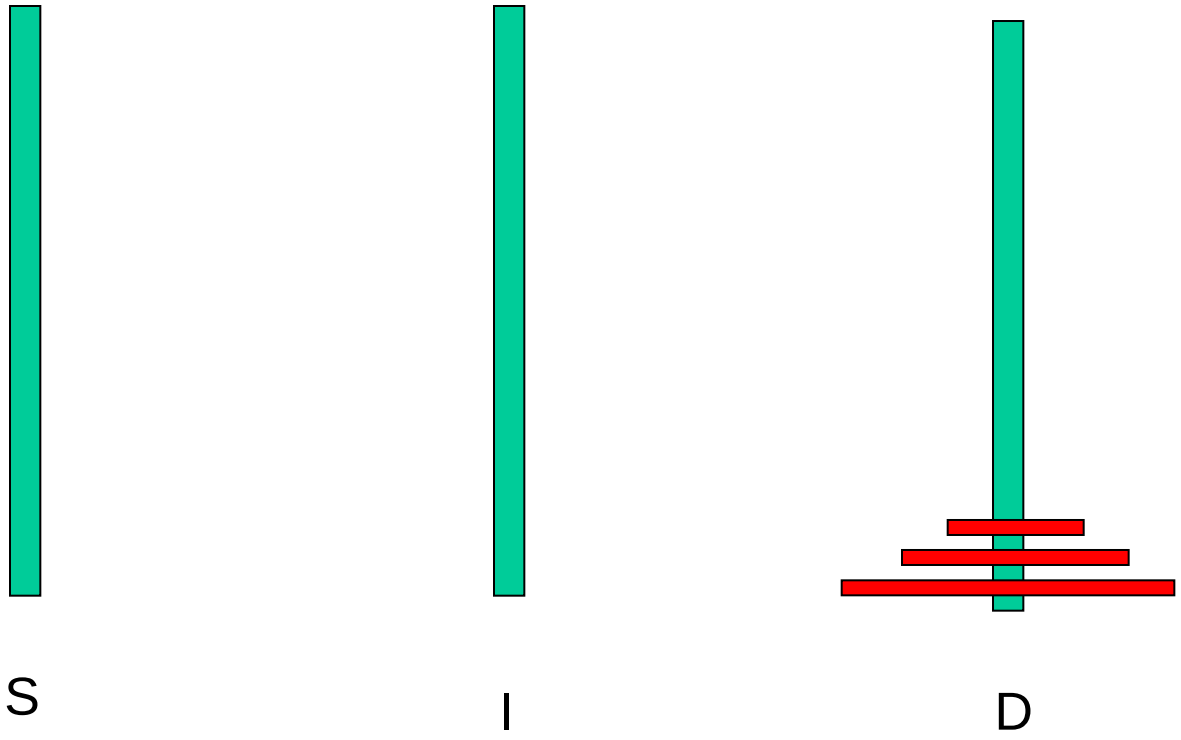
Problema turnurilor din Hanoi

A saptea mutare: S->D



Problema turnurilor din Hanoi

Rezultat



Problema turnurilor din Hanoi

Idee:

- Se muta (n-1) discuri de pe S pe I (utilizând D ca vergea auxiliară)
- Se muta discul rămas pe S direct pe D
- Se muta (n-1) discuri de pe I pe D (utilizând S ca vergea auxiliară)

Algoritm:

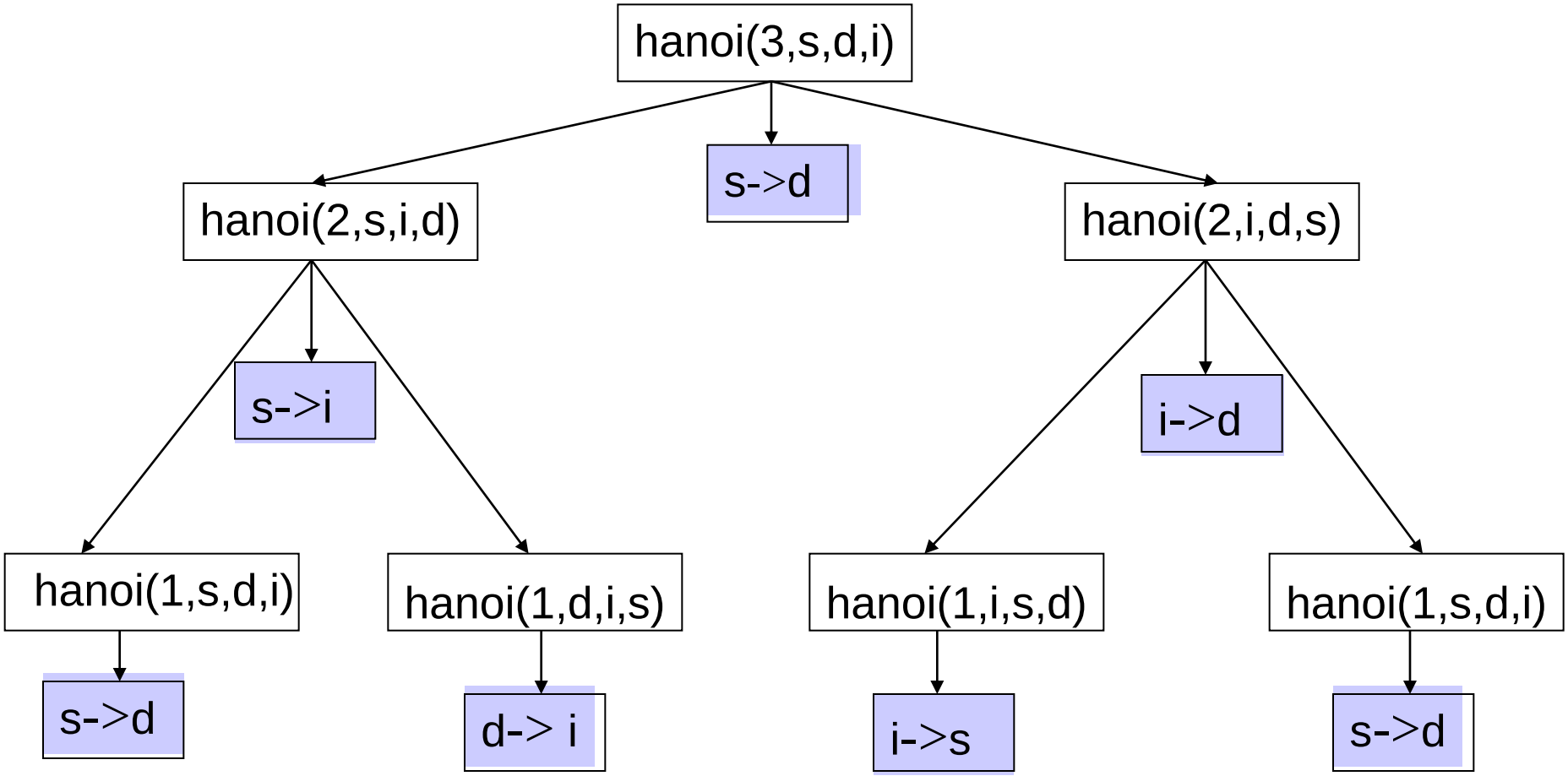
```
hanoi(n,S,D,I)
    IF n=1 THEN   "move from
    S to D"
    ELSE  hanoi(n-1,S,I,D)
              "move from S to D"
              hanoi(n-1,I,D,S)
    ENDIF
```

Semnificația parametrilor funcției hanoi:

- Primul parametru: numărul discurilor
- Al doilea parametru: vergea sursă
- Al treilea parametru: vergea destinație
- Al patrulea parametru: vergea intermediară

Problema turnurilor din Hanoi

Ilustrare apeluri recursive pentru n=3.



Problema turnurilor din Hanoi

```
hanoi(n,S,D,I)
    IF n=1 THEN   "move from S to D"
    ELSE  hanoi(n-1,S,I,D)
          "move from S to D"
          hanoi(n-1,I,D,S)
    ENDIF
```

Dim pb: n

Operație dominanta: move

Relație de recurență:

$$T(n)=\begin{cases} 1 & n=1 \\ 2T(n-1)+1 & n>1 \end{cases}$$

$T(n) = 2T(n-1)+1$

$T(n-1)=2T(n-2)+1 \quad |*2$

$T(n-2)=2T(n-3)+1 \quad |*2^2$

…

$T(2) = 2T(1)+1 \quad |*2^{n-2}$

$T(1) = 1 \quad |*2^{n-1}$

$T(n)=1+2+\ldots+2^{n-1} = 2^n -1$

$T(n)\in\Theta(2^n)$

Tehnica divizării: selecția celui de al k-lea element

Fiind dat un tablou x[1..n] (neordonat), se consideră problema determinării celui de al k-lea element în ordine crescătoare

Exemplu: x=[4,1,5,3,8,6]

k=1 => 1 (cel mai mic element)

k=3 => 4

k=6 => 8 (cel mai mare element)

Variante de rezolvare:

- Sortare x[1..n] => O(nlogn)
- Sortare parțială prin selecție => O(kn) – eficient doar pentru k mic (sau apropiat de n)

Există variantă mai eficientă?

Tehnica divizării : selecția celui de al k-lea element

Idee: se folosește strategia de partiționare de la quicksort și, în funcție de relația dintre valoarea curentă a lui k și numărul de elemente din x[s..q] se continuă căutarea în prima parte (x[s..q]) sau în a doua parte (x[q+1..d])

Obs: dacă pentru apelul inițial k este între 1și n, la fiecare dintre apeluri, k va fi între 1 și d-s+1 (k e rangul unui element dar nu indicele elementului)

```
Selectie(x[s..d],k)
  if s==d then return x[s]
  else
     q←partitie(x[s..d])
     r ← q-s+1
     if k<=r then return selectie(x[s..q],k)
                 else return selectie(x[q+1..d],k-r)
     endif
  endif
```

Tehnica divizării: selecția celui de al k-lea element

```
Selectie(x[s..d],k)
  if s==d then return x[s]
  else
     q←partitie(x[s..d])
     r ← q-s+1
     if k<=r then return selectie(x[s..q],k)
              else return selectie(x[q+1..d],k-r)
     endif
  endif
```

Cazul cel mai favorabil (partiționare echilibrată):

$$T(n)=\begin{cases} 0 & n=1 \\ T(n/2)+n & n>1 \end{cases}$$

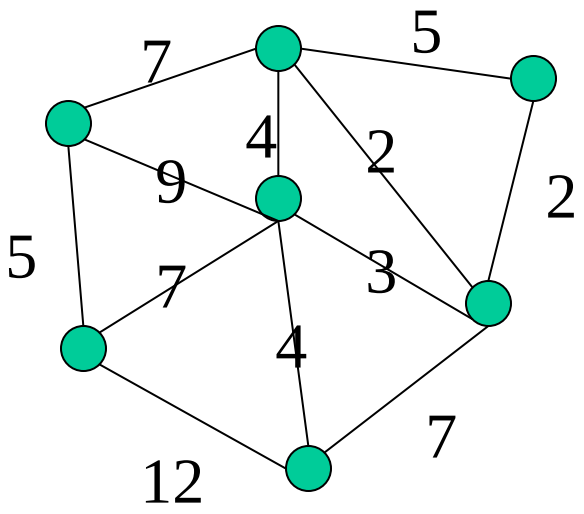
$\Rightarrow$ (t. Master, caz 1: m=2,k=1,d=1)

T(n) aparține lui O(n)

Tehnica greedy: arbore minim de acoperire

Se consideră un set de n locaţii intre care există conexiuni cu diferite costuri.

Se pune problema selectării a n-1 conexiuni astfel încât între oricare două locaţii există o cale de acces şi costul total al conexiunilor selectate să fie minim. Conexiunile selectate formează o structura arborescentă numită arbore minim de acoperire (minimum spanning tree)



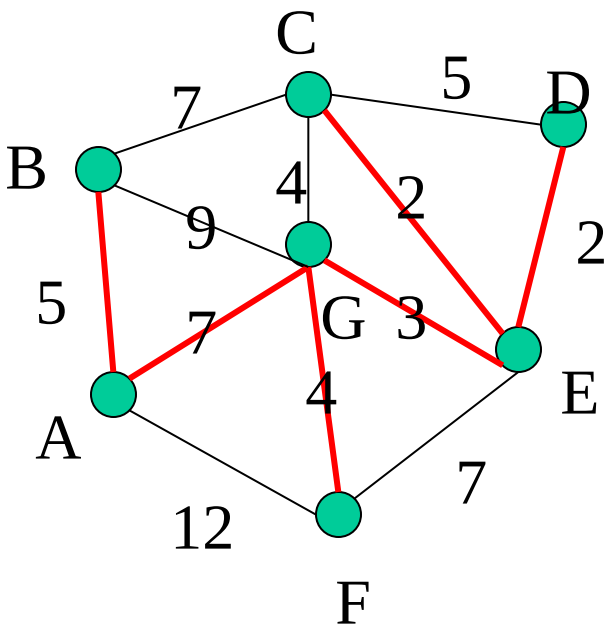
Idee: se porneşte de la unul dintre noduri si se adaugă succesiv noduri care conectează un nod deja selectat cu un nod încă neselectat.

De fiecare dată se alege conexiunea disponbiliă care are costul cel mai mic

Tehnica greedy: arbore minim de acoperire

Se consideră un set de n locaţii intre care există conexiuni cu diferite costuri.

Se pune problema selectării a n-1 conexiuni astfel încât între oricare două locaţii există o cale de acces şi costul total al conexiunilor selectate să fie minim. Conexiunile selectate formează o structura arborescentă numită arbore minim de acoperire (minimum spanning tree)



Idee: se porneşte de la unul dintre noduri si se adaugă succesiv noduri care conectează un nod deja selectat cu un nod încă neselectat.

De fiecare dată se alege conexiunea disponibilă care are costul cel mai mic

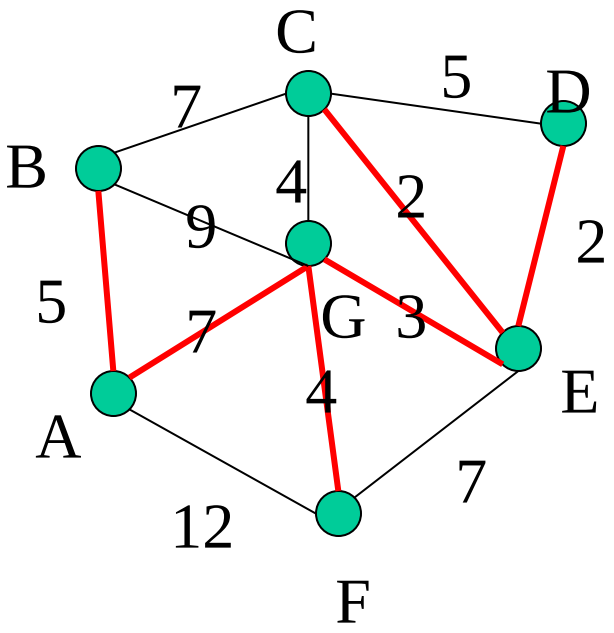
Obs: corespunde algoritmului lui Prim

Tehnica greedy: arbore minim de acoperire

Specificare conexiuni: lista cu triplete (nod1, nod2, cost)

Exemplu: (C,E,2), (D,E,2), (E,G,3), (C,G,4), (F,G,4), (A,B,5), (C,D,5), (A,G,7), (B,C,7)

Idee rezolvare: se ordonează crescător dupa cost lista cu conexiuni si se selectează conexiunea cu cel mai mic cost



Celelalte n-2 muchii se selectează în ordinea crescătoare a costului dar astfel încât se selectează muchia cu cel mai mic cost dintre cele care conectează un nod deja selectat cu un nod neselectat încă

Tehnica greedy: arbore minim de acoperire

Notaţii: n = nr noduri, m = nr conexiuni

c[1..m] = lista cu triplete asociate conexiunilor
(c[j,1]=indice nod start conexiune j; c[j,2]=indice nod destinatie conexiune j
c[j,3]=cost conexiune j)

v[1..n]= lista cu indicatori de selectie nod (v[i]=1 daca nodul a fost selectat,
v[i]=0 daca nodul i nu a fost selectat)

Tehnica greedy: arbore minim de acoperire

```
algPrim(n, c[1..m,1..3])  // c ordonat  crescator dupa a treia componenta (cost)
  v[1..n]←0
  rez[1] ←c [1]; v[c[1,1]] ←1; v[c[1,2]] ←1
  for i ← 2,n-1 do
      imin ←m
// caut muchia de cost minim care uneste un nod selectat cu unul neselectat
// (suma valorilor corespunzatoare in v trebuie sa fie 1)
      for j ← 1,m-1 do
        if (v[c[j,1]]+v[c[j,2]]=1) and (c[j,3]<c[imin,3]) then imin ← j endif
      endfor
      rez[i] ←c[imin]
      v[c[imin,1]] ← 1  //marchez nodurile ca fiind selectate (unul dintre ele era
      v[c[imin,2]] ← 1  // deja selectat insa nu stim care)
   endfor
   return rez
```

Tehnica backtracking: generare submultimi

Problema. Se consideră o mulţime finită cu n elemente $A=\{a_1,a_2,\ldots,a_n\}$ şi se pune problema generării tuturor submulţimilor lui A care satisfac:

a. Numărul de elemente al submulţimii este m ($1<m<n$)

b. Suma elementelor din submulţime este S

Tehnica backtracking: generare submultimi

Formalizarea problemei. Intrucât ordinea elementelor dintr-o mulţime nu contează vom considera că elementele lui A şi ale submulţimilor generate sunt în ordine strict crescătoare

1. Reprezentarea soluției

   $S=(s_1,\ldots,s_m)$ cu s_k element din A şi $s_1< s_1 < \ldots<s_m$

2. Mulțimile $A_1,\ldots,A_n$ sunt toate egale cu A

Tehnica backtracking: generare submultimi

3. Condiții de continuare: o soluție parțială $(s_1,s_2,\ldots,s_k)$ trebuie să satisfacă:

   fie k=1 fie k>1 şi $s_{k-1} < s_k$

4. Condiția ca o soluție parțială $(s_1,\ldots,s_k)$ să fie soluție finală dacă are exact m elemente (k=m)

Generare submultimi

a. Submultimi cu m elemente

```
subm(k)
  if k=m then print s[1..m]
  else
     for i←1,n do
        s[k] ← a[i]
        if valid(s[1..k])=True then
            subm(k+1)
        endif
      endfor
endif
Valid(s[1..k])
  if k>1 and s[k]<=s[k-1]
  then return False
  else return True
  endif
```

b. Submultimi cu suma = val

```
submsum(k)
  suma ← calculSuma (s[1..k-1])
  if suma =val  then print s[1..m]
  else
     for i←1,n do
        s[k] ← a[i]; suma ←suma+s[k]
        if validSum(s[1..k],suma)=True
        then subm(k+1)  endif
      endfor
endif
ValidSum(s[1..k], suma)
  if k>1 and(s[k]<=s[k-1]  or suma>val)
    then return False
    else return True
  endif
```

Tematica examen

- Algoritmi iterativi: proiectare și analiza complexitate
- Algoritmi recursivi: descriere și analiza complexitate

- Tehnici de reducere și divizare: descriere și analiza complexitate; aplicații

- Algoritmi de sortare:
  - algoritmi elementari (inserție, selecție, interschimbare)
  - algoritmi eficienți (sortare prin interclasare, sortare rapidă)

- Stive și cozi – concepte de baza + aplicații simple
- Liste simplu și dublu înlănțuite – tipuri de operatii+ aplicații simple

- Căutare local optimală (greedy) – aplicații
- Backtracking – exemple simple