



# Inteligența Artificială

**Universitatea Politehnică București**  
**Anul universitar 2010-2011**

Adina Magda Florea

[http://turing.cs.pub.ro/ia\\_10](http://turing.cs.pub.ro/ia_10) și  
[curs.cs.pub.ro](http://curs.cs.pub.ro)



## Curs nr. 2

---

### Strategii de rezolvare a problemelor

- Reprezentarea solutiei problemei
- Strategii de cautare de baza: nivel, adancime, nivel iterativ (ID), bidirectionala
- Strategii de cautare informate: best first, cost uniform, A\*, cautare euristica cu memorie limitata



# 1. Reprezentarea solutiei problemei

---

- Reprezentare prin spatiul starilor
- Reprezentare prin grafuri SI/SAU
- Echivalenta reprezentarilor
- Caracteristicile mediului de rezolvare
  
- Pentru a reprezenta si gasi o solutie:
  - Structura simbolica
  - Instrumente computationale
  - Metoda de planificare

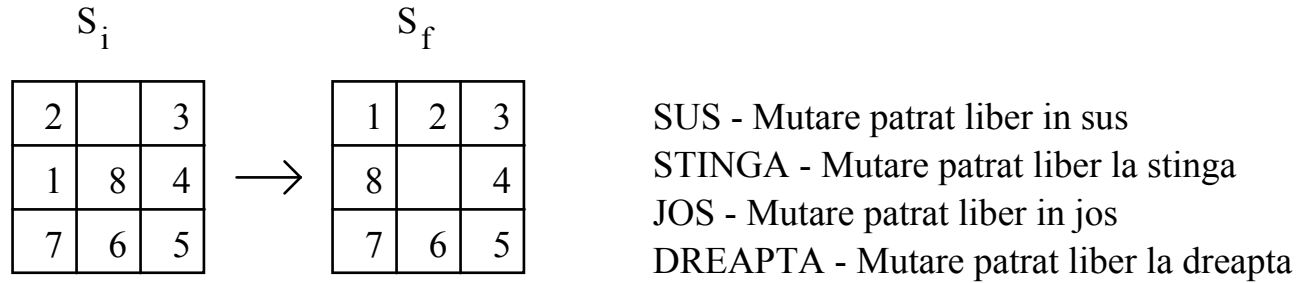


## 1.1. Rezolvarea problemei reprezentata prin spatiul starilor

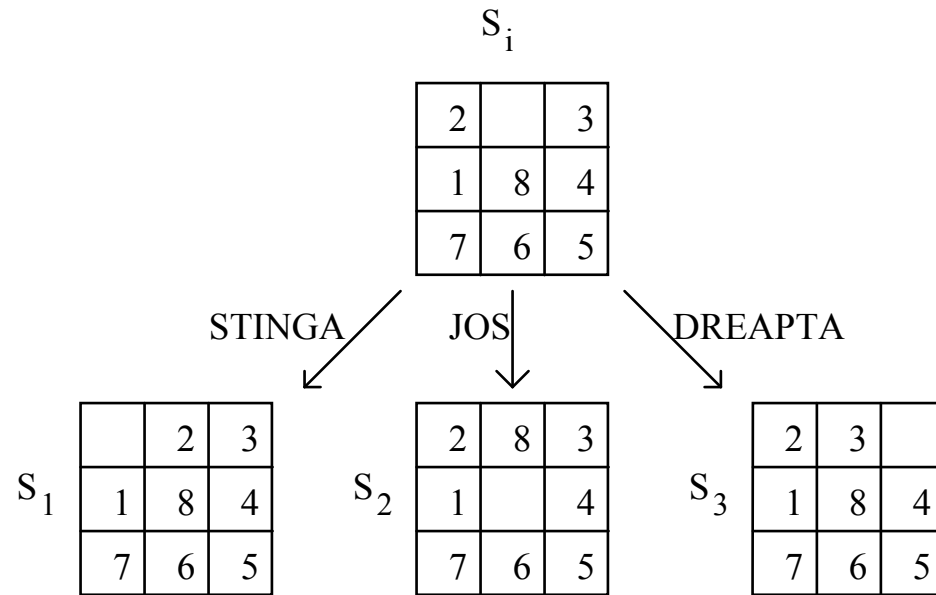
---

- Stare
- Spatiu de stari
- Stare initiala
- Stare/stari finala/finale
- $(S_i, O, S_f)$
- Solutia problemei
- Caracteristicile mediului

# 8-puzzle



(a) Stare initiala      (b) Stare finala      (c) Operatori



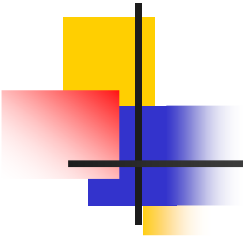
(d) Tranzitii posibile din starea  $S_1$



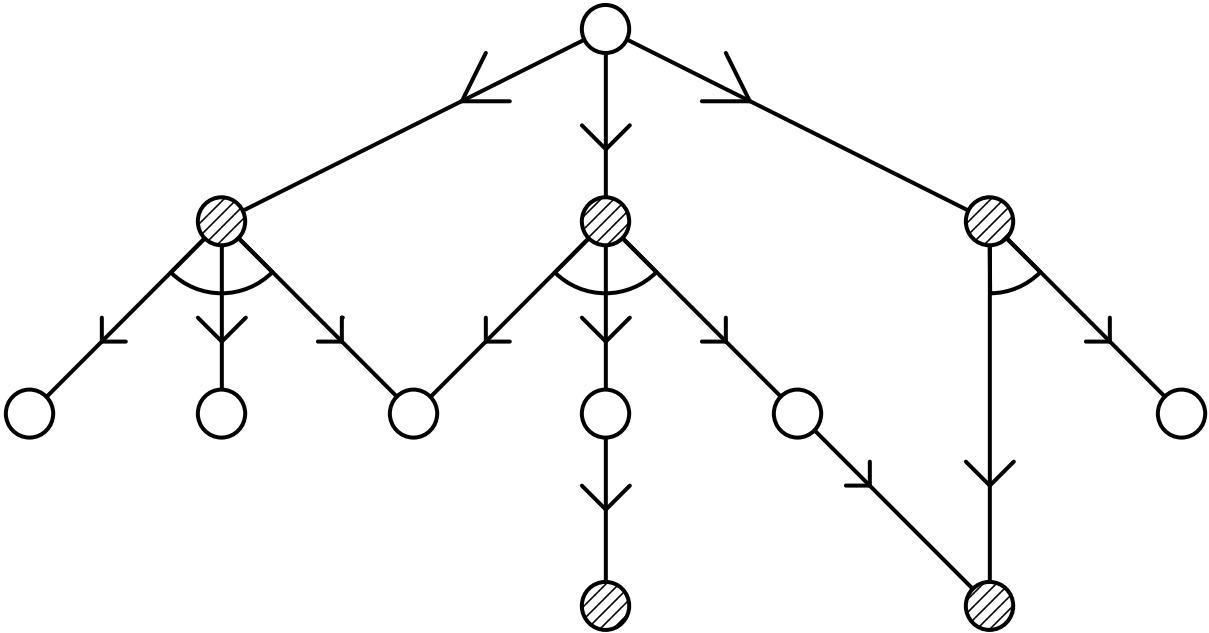
## 1.2 Rezolvarea problemei reprezentata prin grafuri SI/SAU

---

- $(P_i, O, P_e)$
- Semnificatie graf SI/SAU
- Nod rezolvat
- Nod nerezolvabil
- Solutia problemei



# Graf SI/SAU

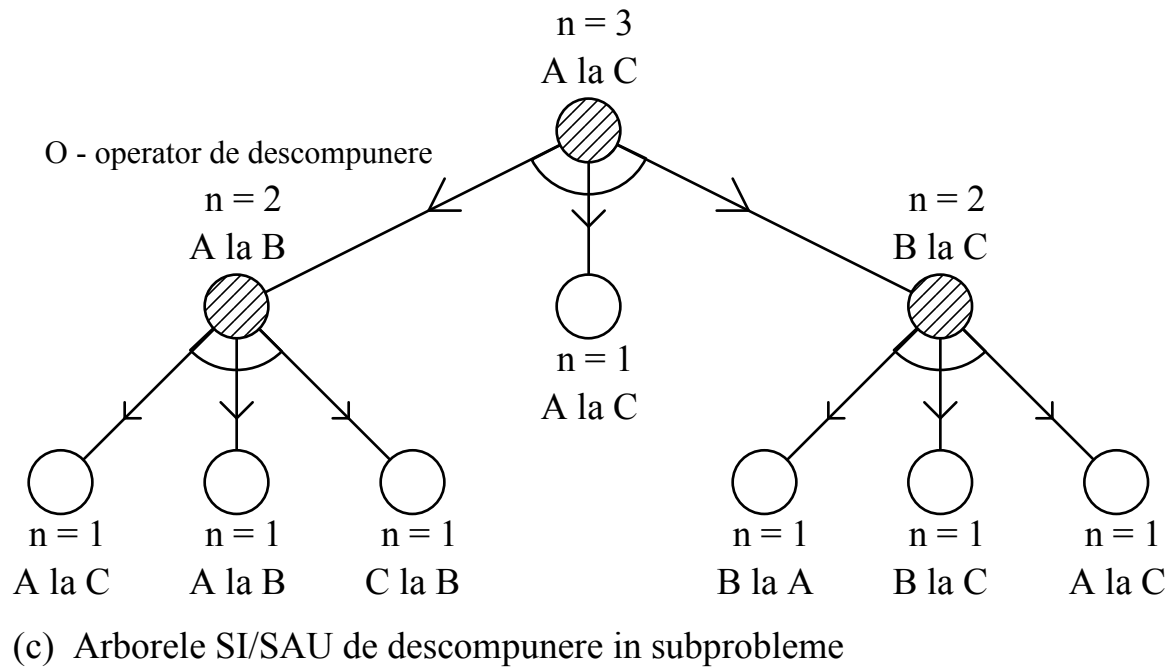
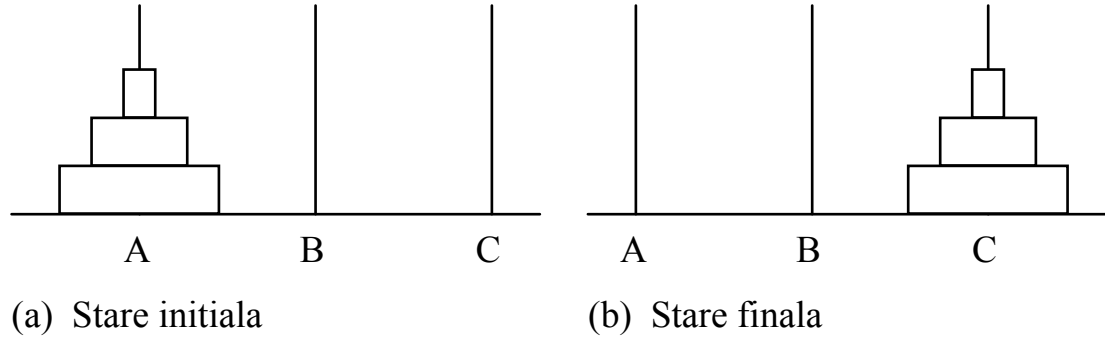


Nod SAU

Noduri SI

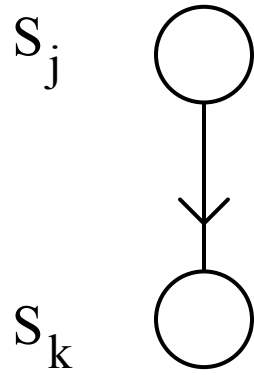
Noduri SAU

# Turnurile din Hanoi



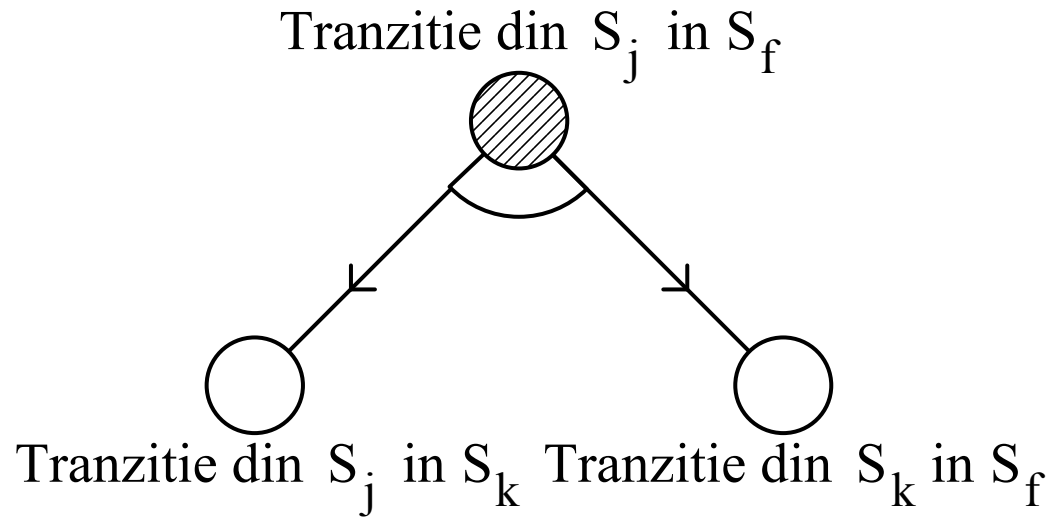


# 1.3 Echivalenta reprezentarilor



$S_j, S_k$  - stari intermediare  $S_f$  - stare finala

(a) Spatiul starilor



(b) Descompunerea problemei in subprobleme



## 1.4 Caracteristicile mediului

---

- Observabil / neobservabil
- Discret / continuu
- Finit / infinit
- Determinist / nedeterminist



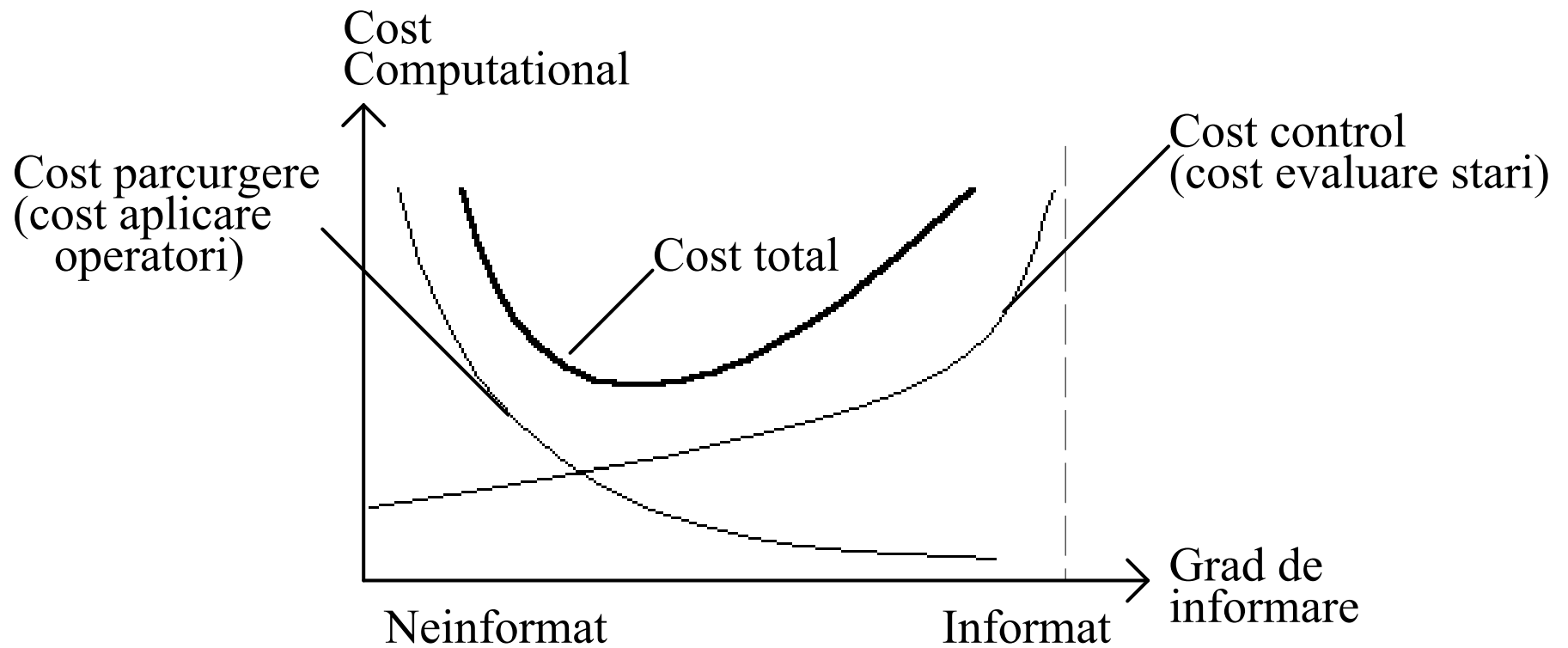
## 2. Strategii de cautare de baza

---

### Criteria de caracterizare

- Completitudine
  - Optimalitate
  - Complexitate
  - Capacitatea de revenire
  - Informare
- 
- Conventii: nod necunoscut, evaluat, expandat, FRONTIERA, TERITORIU

# Costuri ale cautarii



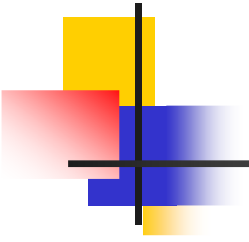


## 2.1. Cautari neinformate in spatiul starilor

---

**Algoritm NIV: Strategia cautarii pe nivel in spatiul starilor**

1. Initializeaza listele FRONTIERA  $\leftarrow \{S_i\}$ , TERITORIU  $\leftarrow \{\}$
2. **daca** FRONTIERA =  $\{\}$   
**atunci intoarce** INSUCCES
3. Elimina primul nod S din FRONTIERA si insereaza-l in TERITORIU
4. Expandeaza nodul S
  - 4.1. Genereaza toti succesorii directi  $S_j$  ai nodului S
  - 4.2. **pentru** fiecare succesor  $S_j$  al lui S **executa**
    - 4.2.1. Stabileste legatura  $S_j \rightarrow S$
    - 4.2.2. **daca**  $S_j$  este stare finala  
**atunci**
      - i. Solutia este  $(S_j, S, \dots, S_i)$
      - ii. **intoarce** SUCCES
    - 4.2.3. Insereaza  $S_j$  in FRONTIERA, *la sfarsit*
5. **repete de la 2 sfarsit.**

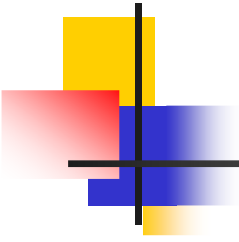
- 
- 
- Caracteristici cautare pe nivel
  - Algoritmul presupune spatiul de cautare arbore si nu graf
  - Pentru un spatiu de cautare graf se insereaza pasul 3'  
3'. **daca**  $S \in \text{FRONTIERA} \cup \text{TERITORIU}$  **atunci repeta de la 2**

### **Strategia cautarii in adancime in spatiul starilor**

- Intr-o reprezentare a solutiei problemei prin spatiul starilor *adancimea unui nod* se defineste astfel:
- $Ad(S_i) = 0$ , unde  $S_i$  este nodul stare initiala,
- $Ad(S) = Ad(S_p) + 1$ , unde  $S_p$  este nodul predecesor nodului  $S$ .

**Algoritm ADANC(AdMax): Strategia cautarii in adancime in spatiul starilor**

1. Initializeaza listele FRONTIERA  $\leftarrow \{S_i\}$ , TERITORIU  $\leftarrow \{\}$
2. **daca** FRONTIERA =  $\{\}$   
**atunci intoarce** INSUCCES
3. Elimina primul nod S din FRONTIERA si insereaza-l in TERITORIU
- 3'. **daca** Ad(S) = AdMax **atunci repeta de al 2**
4. Expandeaza nodul S
  - 4.1. Genereaza toti succesorii directi  $S_j$  ai nodului S
  - 4.2. **pentru** fiecare succesor  $S_j$  al lui S **executa**
    - 4.2.1. Stabileste legatura  $S_j \rightarrow S$
    - 4.2.2. **daca**  $S_j$  este stare finala  
**atunci**
      - i. Solutia este  $(S_j, \dots, S_i)$
      - ii. **intoarce** SUCCES
    - 4.2.3. Insereaza  $S_j$  in FRONTIERA, *la inceput*
5. **repete de la 2**  
**sfarsit.**

- 
- 
- Caracteristici cautare in adancime
  - **Cautare in adancime cu nivel iterativ (ID)**  
pentru  $AdMax=1$ , m **executa**  
 $ADANC(AdMax)$

Caracteristici cautare in adancime **cu nivel iterativ**

- **Cautare de tip backtracking**
- **Cautare bidirectionala**
- **Care strategie este mai buna ?**





## 2.2. Cautari neinformate in grafuri SI/SAU

---

### Adancimea unui nod

- $Ad(S_i) = 0$ , unde  $S_i$  este nodul problema initiala,
- $Ad(S) = Ad(S_p) + 1$  daca  $S_p$  este nod SAU predecesor al nodului S,
- $Ad(S) = Ad(S_p)$  daca  $S_p$  este nod SI predecesor al nodului S.

**Algoritm NIV-SI-SAU: Strategia cautarii pe nivel in arbori SI/SAU.**

1. Initializeaza listele FRONTIERA  $\leftarrow \{S_i\}$ , TERITORIU  $\leftarrow \{\}$
2. Elimina primul nod S din FRONTIERA si insereaza-l in TERITORIU
3. Expandeaza nodul S
  - 3.1. Genereaza toti succesorii directi  $S_j$  ai nodului S
  - 3.2. **pentru** fiecare succesor  $S_j$  al lui S **executa**
    - 3.2.1. Stabileste legatura  $S_j \rightarrow S$
    - 3.2.2. **daca**  $S_j$  reprezinta o multime de cel putin 2 subprobleme **atunci** /\* este nod SI \*/
      - i. Genereaza toti succesorii subprobleme  $S_j^k$  ai lui  $S_j$
      - ii. Stabileste legaturile intre nodurile  $S_j^k \rightarrow S_j$
      - iii. Insereaza nodurile  $S_j^k$  in FRONTIERA, *la sfirsit*
    - 3.2.3. **altfel** insereaza  $S_j$  in FRONTIERA, *la sfirsit*

4. **daca** nu s-a generat nici un succesori al lui S in pasul precedent  
(3)

**atunci**

4.1. **daca** S este nod terminal etichetat cu o problema  
neelementara

**atunci**

4.1.1. Eticheteaza S nerezolvabil

4.1.2. Eticheteaza cu nerezolvabil toate nodurile  
predecesoare lui S care devin nerezolvabile  
datorita lui S

4.1.3. **daca** nodul  $S_i$  este nerezolvabil

**atunci intoarce** INSUCCES /\* problema nu are solutie \*/

4.1.4. Elimina din FRONTIERA toate nodurile care au  
predecesori nerezolvabili

4.2. **altfel** /\* S este nod terminal etichetat cu o problema elementara \*/

4.2.1. Eticheteaza S rezolvat

4.2.2. Eticheteaza cu rezolvat toate nodurile predecesoare lui S care devin rezolvate datorita lui S

4.2.3. **daca** nodul  $S_i$  este rezolvat

**atunci**

i. Construiește arborele soluție urmărind legăturile

ii. **intoarce** SUCCES /\* s-a găsit soluția \*/

4.2.4. Elimina din FRONTIERA toate nodurile rezolvate și toate nodurile care au predecesori rezolvați

5. **repetă de la 2**

**sfârșit.**



## 2.3. Complexitatea strategiilor de cautare

---

- *B* - *factorul de ramificare* al unui spatiu de cautare

### 8-puzzle

- Numar de miscari:

- 2 m pt colt = 8

- 3 m centru lat = 12

- 4m centru  $\Rightarrow$  24 miscari

- **$B = \text{nr. misc.} / \text{nr. poz. p. liber} = 2.67$**

- Numar de miscari:

- 1 m pt colt = 4

- 2 m centru lat = 8

- 3m centru  $\Rightarrow$  15 miscari  $\Rightarrow$   **$B = 1.67$**



## Complexitatea strategiilor de cautare

---

- **B** - *factorul de ramificare*
- **d**- adancimea celui mai apropiat nod solutie
- **m** – lungimea maxima a oricarei cai din spatiul de cautare

Rad – B noduri,  $B^2$  pe niv 2, etc.

- Numarul de stari posibil de generat pe un nivel de cautare  $d$  este  $B^d$
- T - numarul total de stari generate intr-un proces de cautare,  $d$  – adancime nod solutie

$$\mathbf{T = B + B^2 + \dots + B^d = O(B^d)}$$



# Complexitatea strategiilor de cautare

---

- **Cautare pe nivel**

Numar de noduri generate

$$\mathbf{B + B^2 + \dots + B^d = O(B^{d+1})}$$

Complexitate timp, spatiu

- **Cautare in adancime**

Numar de noduri generate

**B\*m** – daca nodurile expandate se sterg din  
TERITORIU

Complexitate timp, spatiu



# Complexitatea strategiilor de cautare

---

- **Cautare backtracking**

Numar de noduri generate **m** – daca se elimina  
TERITORIU

Complexitate timp, spatiu

- **Cautare cu nivel iterativ**

Numar de noduri generate

$$d*B+(d-1)*B^2+ \dots + (1)*B^d = O(B^d)$$

Complexitate timp, spatiu

B=10, d=5

N(BFS)=111110, N(IDS) = 123 450



b = 10  
1 mil. noduri/sec  
1000 bytes/nod

Adancime	Nr noduri	Timp	Memorie
2	110	.11 milisec	107 KB
4	11 100	11 milisec	10.6 MB
6	$10^6$	1.1 sec	1 GB
8	$10^8$	2 min	103 GB
10	$10^{10}$	3 h	10 TB
12	$10^{12}$	13 zile	1 petabytes
14	$10^{14}$	3.5 ani	99 petabytes

## Complexitatea strategiilor de cautare

Criteriu	Nivel	Adanci me	Adanc. limita	Nivel iterativ	Bidirec tionala
Timp	$B^d$	$B^d$	$B^m$	$B^d$	$B^{d/2}$
Spatiu	$B^d$	$B^*d$	$B^*m$	$B^d$	$B^{d/2}$
Optima litate?	Da	Nu	Nu	Da	Da
Comple ta?	Da	Nu	Da daca $m \geq d$	Da	Da

**B** – factor de ramificare, **d** – adancimea solutiei,  
**m** – adancimea maxima de cautare (AdMax)



## 3. Strategii de cautare informate

---

Cunostintele euristice pot fi folosite pentru a creste eficienta cautarii in trei moduri:

- **Selectarea nodului urmator de expandat in cursul cautarii.**
- In cursul expandarii unui nod al spatiului de cautare se poate decide pe baza informatiilor euristice care dintre succesorii lui vor fi generati si care nu
- Eliminarea din spatiul de cautare a anumitor noduri generate



## 3.1 Cautare informata de tip "best-first"

---

- Evaluarea cantitatii de informatie
- Calitatea unui nod este estimata de *functia de evaluare euristica*, notata  $w(n)$  pentru nodul  $n$
- Presupuneri pentru functia  $w(n)$
- Strategia de cautare a alpinistului
- Strategia de cautare "best-first"

**Algoritm BFS: Strategia de cautare "best-first" in spatiul starilor**

1. Initializeaza listele  $FRONTIERA \leftarrow \{S_i\}$ ,  $TERITORIU \leftarrow \{\}$
2. Calculeaza  $w(S_i)$  si asociaza aceasta valoare nodului  $S_i$
3. **daca**  $FRONTIERA = \{\}$   
**atunci intoarce** INSUCCES
4. Elimina nodul  $S$  cu  $w(S)$  minim din  $FRONTIERA$  si insereaza-l in  $TERITORIU$
5. **daca**  $S$  este stare finala  
**atunci**
  - i. Solutia este  $(S, \dots, S_i)$
  - ii. **intoarce** SUCCES
6. Expandeaza nodul  $S$ 
  - 6.1. Genereaza toti succesorii directi  $S_j$  ai nodului  $S$
  - 6.2. **pentru** fiecare succesor  $S_j$  al lui  $S$  **executa**
    - 6.2.1 Calculeaza  $w(S_j)$  si asociaza-l lui  $S_j$
    - 6.2.2. Stabileste legatura  $S_j \rightarrow S$

6.2.3. **daca**  $S_j \notin \text{FRONTIERA} \cup \text{TERITORIU}$   
**atunci** introduce  $S_j$  in **FRONTIERA** cu  $w(S_j)$   
asociat

6.2.5. **altfel**

i. Fie  $S'_j$  copia lui  $S_j$  din **FRONTIERA** sau **TERITORIU**

ii. **daca**  $w(S_j) < w(S'_j)$

**atunci**

**atunci**

- Elimina  $S'_j$  din **FRONTIERA** sau  
**TERITORIU** (de unde apare copia)

- Insereaza  $S_j$  cu  $w(S_j)$  asociat in  
**FRONTIERA**

iii. **altfel** ignora nodul  $S_j$

7. **repetă de la 3**

**sfarsit.**

## *Varianta alternativa pentru pasul 6.2.5*

### 6.2.5. **altfel**

i. Fie  $S'_j$  copia lui  $S_j$  din FRONTIERA sau TERITORIU

ii. **daca**  $w(S_j) < w(S'_j)$

**atunci**

- Distruge legatura  $S'_j \rightarrow S_p$ , cu  $S_p$  pred. lui  $S'_j$

- Stabileste legatura  $S'_j \rightarrow S$ , si actualizeaza costul lui  $S'_j$  la  $w(S_j)$

- **daca**  $S'_j$  este in TERITORIU

- **atunci** elimina  $S'_j$  din TERITORIU si insereaza  $S'_j$  in FRONTIERA

iii. **altfel** ignora nodul  $S_j$

7. **repetă de la 3**

**sfarsit.**



# Cazuri particulare

---

- Strategia de cautare "best-first" este o generalizare a strategiilor de cautare neinformate
  - strategia de cautare pe nivel  $w(S) = Ad(S)$
  - strategia de cautare in adincime  $w(S) = -Ad(S)$

- **Strategia de cautare de cost uniform**

$$w(S_j) = \sum_{k=i}^{j-1} \text{cost\_arc}(S_k, S_{k+1})$$

- **Minimizarea efortului de cautare – cautare euristica**

$$w(S) = \text{functie euristica}$$



## 3.2 Cautarea solutiei optime in spatiul starilor.

### Algoritmul A\*

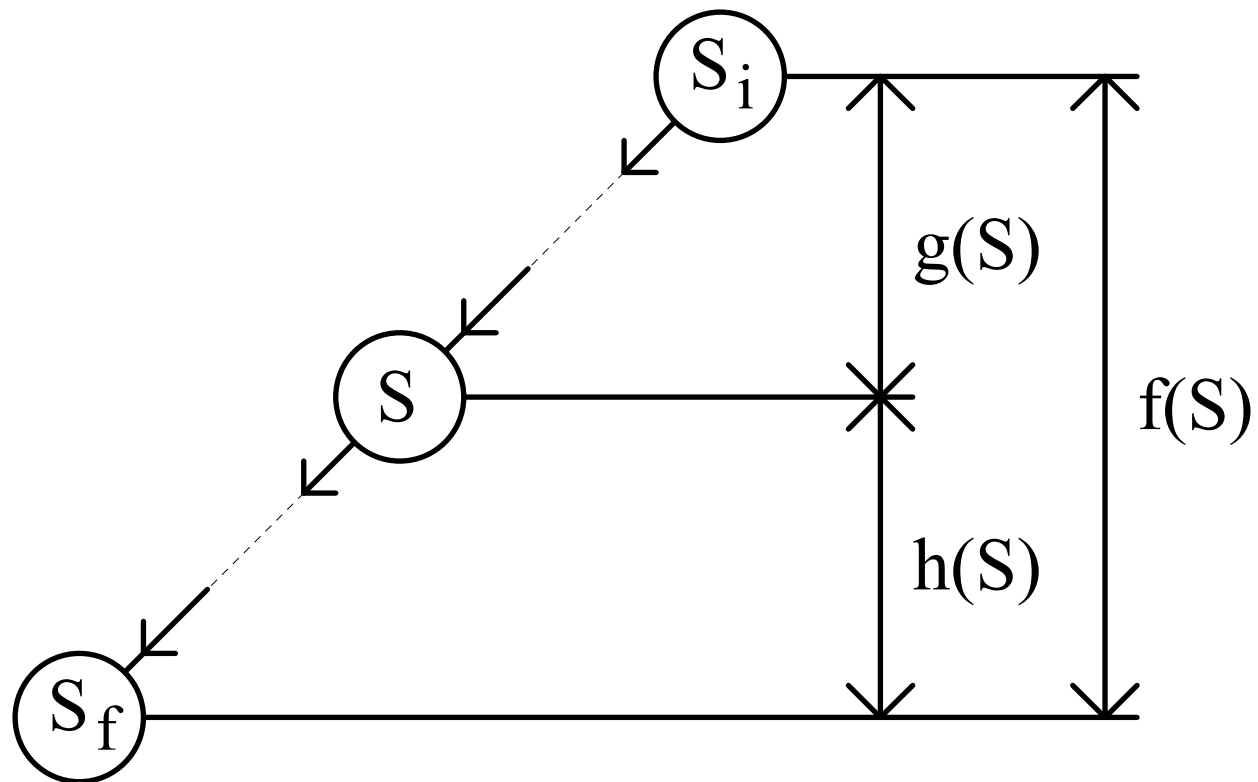
---

$w(S)$  devine  **$f(S)$**  cu 2 comp:

- **$g(S)$** , o functie care estimeaza costul real  $g^*(S)$  al caii de cautare intre starea initiala  $S_i$  si starea  $S$ ,
- **$h(S)$** , o functie care estimeaza costul real  $h^*(S)$  al caii de cautare intre starea curenta  $S$  si starea finala  $S_f$ .
- **$f(S) = g(S) + h(S)$**
- **$f^*(S) = g^*(S) + h^*(S)$**



## Componentele functiei euristice din algoritmul A\*





# Calculul lui $f(S)$

---

- Calculul lui  $g(S)$

$$g(S) = \sum_{k=i}^n \text{cost\_arc}(S_k, S_{k+1})$$

- Calculul lui  $h(S)$
- Trebuie sa fie admisibila
- O functie euristica  $h$  se numeste *admisibila* daca pentru orice stare  $S$ ,  $h(S) \leq h^*(S)$ .
- Definitia stabileste ***conditia de admisibilitate*** a functiei  $h$  si este folosita pentru a defini ***proprietatea de admisibilitate*** a unui algoritm  $A^*$ .



# A\* admisibil

---

Fie un algoritm  $A^*$  care utilizeaza cele doua componente  $g$  si  $h$  ale functiei de evaluare  $f$ . Daca

- (1) functia  $h$  satisface conditia de admisibilitate
- (2)  $\text{cost\_arc}(S, S') \geq c$

pentru orice doua stari  $S, S'$ , unde  $c > 0$  este o constanta si costul  $c$  este finit

- atunci **algoritmul  $A^*$  este admisibil**, adica este garantat sa gaseasca calea de cost minim spre solutie.
- Completitudine



# Implementare A\*

---

Strategia de cautare "best-first" se modifica:

...

2. Calculeaza  $w(S_i) = g(S_i) + h(S_i)$  si asociaza aceasta valoare nodului  $S_i$

3. **daca** FRONTIERA = {}

**atunci intoarce** INSUCCES - *nemodificat*

4. Elimina nodul S cu  $w(S)$  minim din FRONTIERA si insereaza-l in TERITORIU - *nemodificat*

.....

6.2.5. **altfel**

i. Fie  $S'_j$  copia lui  $S_j$  din FRONTIERA sau  
TERITORIU

ii. **daca**  $g(S_j) < g(S'_j)$

**atunci ...**



## Caracteristicile euristicii algoritmului A\*

- Fie doi algoritmi A\*, A1 si A2, cu functiile de evaluare  $h_1$  si  $h_2$  admisibile,  $g_1 = g_2$

$$f_1(S) = g_1(S) + h_1(S) \quad f_2(S) = g_2(S) + h_2(S)$$

- Se spune ca algoritmul **A2 este mai informat decat** algoritmul **A1** daca pentru orice stare S

$$h_2(S) > h_1(S)$$

cu  $S \neq S_f$

- Monotonie

$$h(S) \leq h(S') + \text{cost\_arc}(op, S, S') \quad \text{pentru orice } op, S \text{ si } S' \\ S \text{ diferit de } S_f$$



# Determinarea functiei de evaluare f

---

- **8-puzzle**

$$h_1(S) = \sum_{i=1}^8 t_i(S)$$

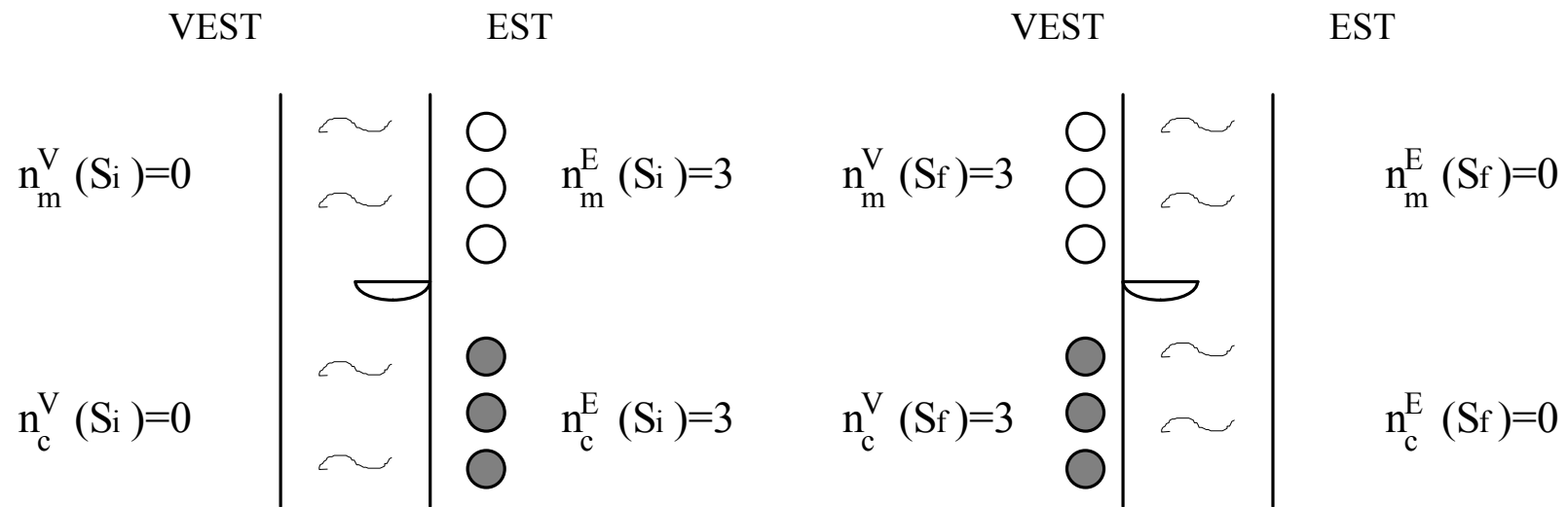
$$h_2(S) = \sum_{i=1}^8 \text{Distanța}(t_i)$$

- **Problema comis-voiajorului**

$$h_1(S) = \text{cost\_arc}(S_i, S)$$

$h_2(S)$  = costul arborelui de acoperire de cost minim al oraselor neaparcurse pana in starea S

# Problema misionarilor si canibalilor



(a) Stare initiala

(b) Stare finala





## Problema misionarilor si canibalilor

---

$$f_1(S) = g(S) + h_1(S)$$

$$h_1(S) = n^E(S)$$

$$f_2(S) = g(S) + h_2(S)$$

$$h_2(S) = n^E(S) / 2$$

$$f_3(S) = g(S) + h_3(S)$$

$$h_3(S) = \begin{cases} n^E(S) + 1 & \text{daca barca este pe malul de VEST si } n^E(S) \neq 0 \\ n^E(S) - 1 & \text{daca barca este pe malul de EST si } n^E(S) \neq 0 \\ 0 & \text{daca } n^E(S) = 0 \end{cases}$$



## Relaxarea conditiei de optimalitate a algoritmului A\*

---

- O functie euristica  $h$  se numeste  **$\varepsilon$ -admisibila** daca
$$h(S) \leq h^*(S) + \varepsilon \quad \text{cu } \varepsilon > 0$$
- Algoritmul A\* care utilizeaza o functie de evaluare  $f$  cu o componenta  $h$   $\varepsilon$ -admisibila gaseste intotdeauna o solutie al carei cost depaseste costul solutiei optime cu cel mult  $\varepsilon$ .
- Un astfel de algoritm se numeste *algoritm A\*  $\varepsilon$ -admisibil* iar solutia gasita se numeste *solutie  $\varepsilon$ -optimala*.



## Relaxarea conditiei de optimalitate a algoritmului A\*

---

- 8-puzzle

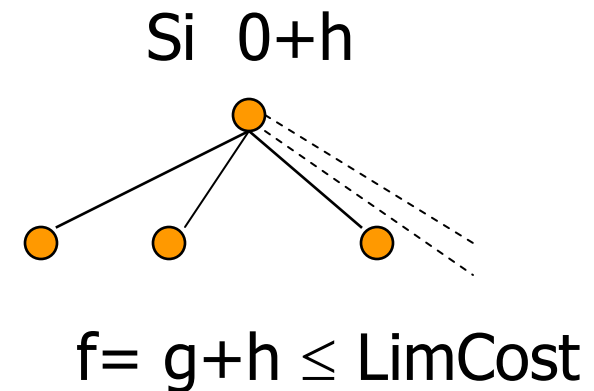
$$f_3(S) = g(S) + h_3(S) \quad h_3(S) = h_2(S) + 3 \cdot T(S)$$

$$T(S) = \sum_{i=1}^8 \text{Scor}[t_i(S)]$$

$$\text{Scor}[t_i(S)] = \begin{cases} 2 & \text{daca patratul } t_i \text{ in starea } S \text{ nu este urmat de} \\ & \text{succesorul corect din starea finala} \\ 0 & \text{pentru orice pozitie a lui } t_i \text{ diferita de centru} \\ 1 & \text{pentru } t_i \text{ aflat la centrul mozaicului} \end{cases}$$

# Cautare euristica cu memorie limitata - IDA\*

- Avantaje si dezavantaje A\*
- IDA\*
- Cautarea in adancime este modificata a.i. sa utilizeze o **limita de cost** (LimCost) in loc de o limita a adancimii (AdMax)
- Fiecare iteratie expandeaza nodurile din interiorul unui **contur de cost LimCost** pentru a vedea care sunt nodurile de pe urmatorul contur



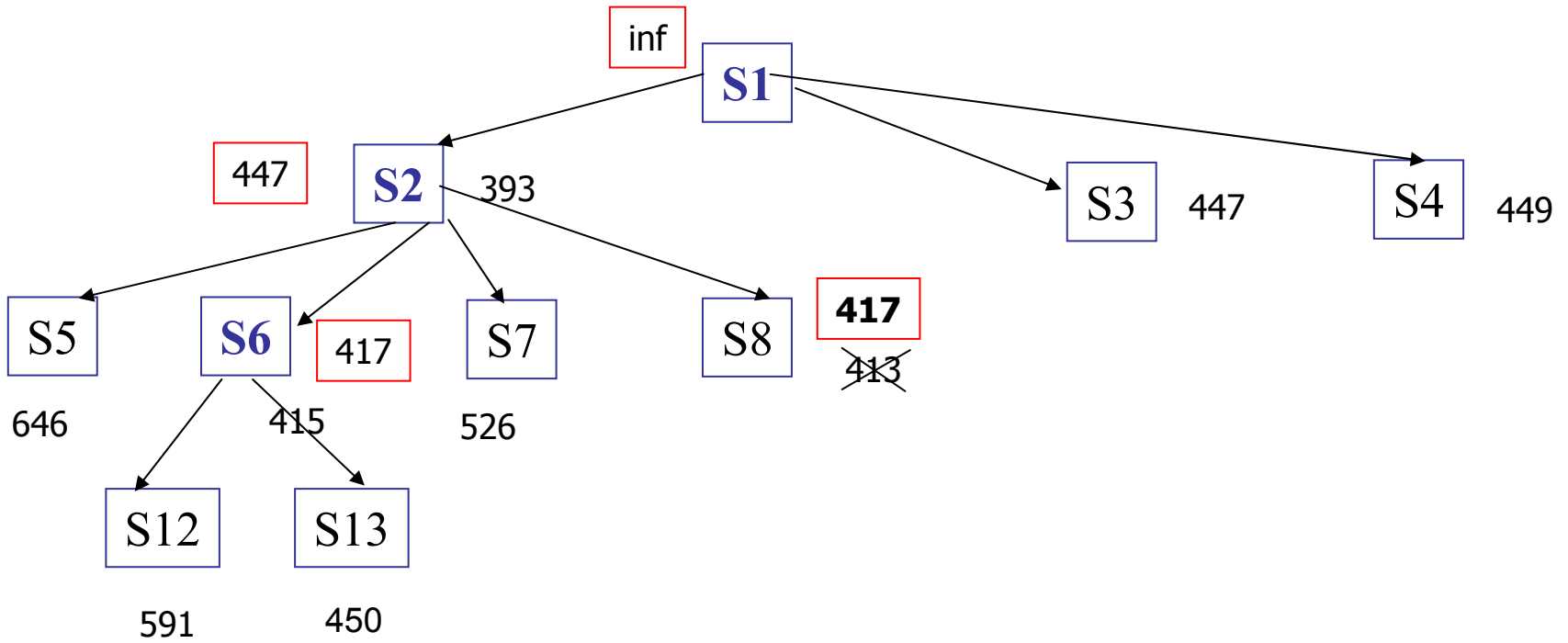
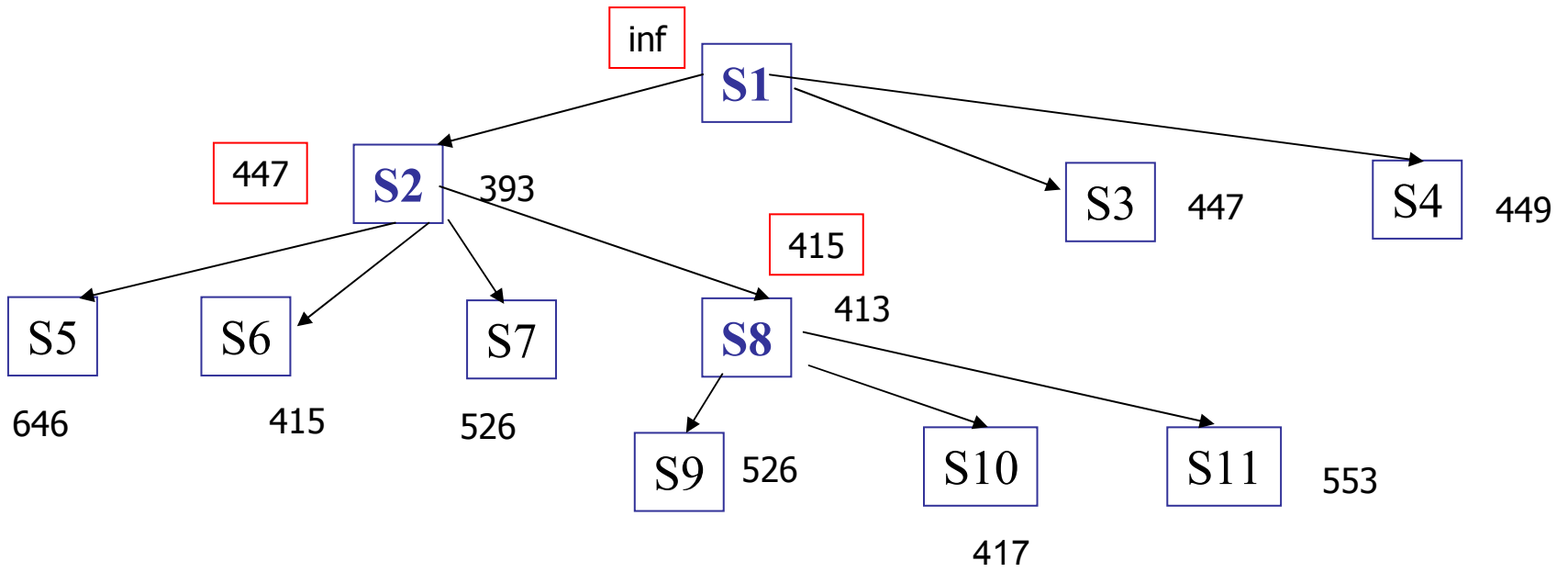
- Daca esueaza (nenatural) se actualizeaza LimCost la **min f** al nodurilor din cele neexpandate anterior

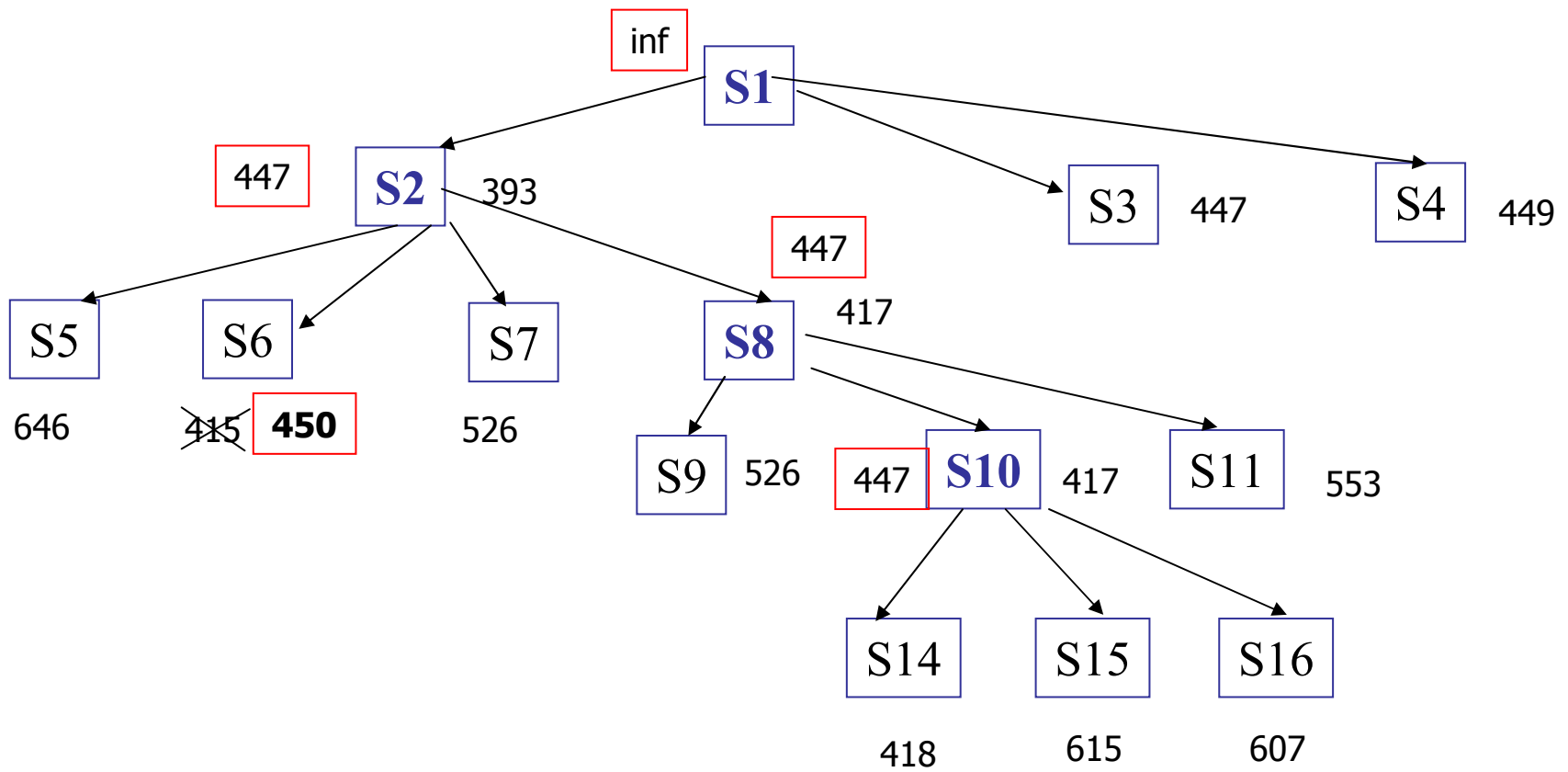


## Cautare euristica cu memorie limitata - Best First Recursiv

---

- Best first cu spatiu liniar
- Implementare recursiva
- Tine minte valoarea  $f$  a celei mai bune **cai alternative** care porneste din *orice nod anterior* nodulului curent
- Gaseste solutia de cost minim daca  $h$  este admisibila dar are complexitatea spatiu  $O(B*d)$





**BestFR(S)**      **Strategia Best First recursiv**

/\* intoarce solutie sau INSUCCES \*/

BFR(Si, inf)

**Algoritm BFR(S, f\_lim): Strategia Best First recursiv**

/\* Intoarce o solutie (nod) sau INSUCCES si o noua limita de cost f\_limit \*/

1. **daca** S stare finala **atunci intoarce** S
  2. Genereaza toti succesorii  $S_j$  ai lui S
  3. **daca** nu exista nici un succesori  
**atunci intoarce** INSUCCES, inf
  4. **pentru** fiecare succesori  $S_j$  **repeta**  
 $f(S_j) \leftarrow \max(g(S_j) + h(S_j), f(S))$
  5.  $Best \leftarrow S_j^{\min}$ , nodul cu valoarea  $f(S_j)$  minima dintre succesori
  6. **daca**  $f(Best) > f\_lim$   
**atunci intoarce** INSUCCES,  $f(Best)$
  7.  $Alternat \leftarrow f(S_j^{\min2})$ , a 2-a val  $f(S_j)$  cea mai mica
  8.  $Rez, f(Best) \leftarrow BFR(Best, \min(f\_lim, Alternat))$
  9. **daca**  $Rez \neq INSUCCES$  **atunci intoarce** Rez,  $f(Best)$
  10. **repeta de la 5**
- sfarsit.**