UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

CURS SDA 8:
TEHNICI DE DEZVOLTARE A ALGORITMILOR(I)
PROBLEME COMBINATORIALE. CAUTARE EXHAUSTIVA.
BACKTRACKING, BRANCH AND BOUND.

PROBLEME COMBINATORIALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Problema combinatoriala: presupune gasirea unei *grupari (submultimi), ordonari (permutari)* sau *atribuiri* a unei multimi discrete si finite de obiecte, care satisface un set de conditii (constrangeri) date
  - *Solutie candidat:* combinare de componente ale solutiei, intalnita in procesul de cautare a solutiei, dar care poate sa nu satisfaca toate conditiile date
  - *Solutie*: solutie candidat care satisface toate conditiile date (se mai numeste si *solutie fezabila*)
- Problema de optimizare: cautam cea mai buna solutie dintre cele existente (fezabile), data fiind o functie obiectiv – *solutia optima*

PROBLEME COMBINATORIALE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Asemenea probleme se pot formula astfel incat gasirea unei solutii sa presupuna de fapt o cautare pe graf/arbore

REZOLVAREA PROBLEMELOR PRIN CAUTAREA IN SPATIUL STARILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Problemele combinatoriale se pot formula astfel incat gasirea unei solutii sa presupuna de fapt o cautare pe graf/arbore
- Cum rezolvam o problema prin cautare – roadmap:
  - Formulare problema
    - model exact al solutiilor valide
    - cum arata spatiul de cautare
  - Reprezentare formala
    - *arbori sau grafuri de stari: definire stari si tranzitii intre ele*
  - Algoritmi de cautare pe arbori/grafuri
    - de regula BFS sau DFS

EXEMPLE…

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Planificare, programare (en. *scheduling, time-tabling*), alocare de resurse
- Rutare pachete de date in retele de calculatoare
- Modalitatea optima de a livra pachete (logistica)
- Ordinea optima de a "suda" punctele de contact de pe un PCB de catre un brat robotic; etc. (check this out: http://www.math.uwaterloo.ca/tsp/index.html)
- Probleme de secventiere: secventa de incarcare a containerelor pe un vas afecteaza durata incarcarii; secventa optima de incarcare
- etc…

EX. PROBLEMA: NUMARAREA RESTULUI (EN. *COUNTING CHANGE*)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- *Probleme*: Un casier are la dispozitie o colectie finita de bancnote si monede de diferite valori (formulare alternativa: colectie infinita …)

1. Se cere sa se genereze toate submultimile colectiei de bancnote si monede care au suma A.
2. Se cere sa se genereze o singura submultime a colectiei de bancnote si monede care are suma A si numar minim de elemente.

- Formulare matematica:
  - Se dau n bancnote si monede: $P=\{p_1, p_2, ..., p_n\}$
  - putem avea repetitii (2 monede de 50 bani, etc)
  - fie d_i valoarea lui p_i
  - *Pt problema 1:*
    - gasiti toate submultimile Q ale lui P astfel incat $\sum_{p_i \in Q} d_i = A$
  - *Pentru problema 2:*
    - gasiti cea mai mica submultime S a lui P ($S \subseteq P$), astfel incat $\sum_{p_i \in S} d_i = A$

NUMARAREA RESTULUI:
CUM ARATA O SOLUTIE PENTRU PROBLEMA 2

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Reprezentam *solutia* ca o tupla de *n* valori: $X=\{x_1, x_2, ..., x_n\}$

$$\begin{cases} x_i = 1 & p_i \in S \\ x_i = 0 & p_i \notin S \end{cases}$$

- Fiind data multima de *valori* pentru elementele din *P* – $\{d_1, d_2, ..., d_n\}$, obiectivul este sa minimizam suma: $\sum_{i=1}^{n} x_i$

astfel incat:

$$\sum_{p_i \in S} d_i = A$$

NUMARAREA RESTULUI: EXEMPLU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **A = 20**
- **D = {1,1,1,1,1,10,10,15}**
- Spatiul solutiilor:
  - X = {0,0,0,0,0,0,0,0}
  - X = {1,0,0,0,0,0,0,0}
  - …
  - X = {1,1,1,1,1,0,0,1}
  - …
  - X = {0,0,0,0,0,1,1,0…}
  - ….
  - X = {1,1,1,1,1,1,1,1}

NUMARAREA RESTULUI: "BRUTE FORCE"

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cautare exhaustiva, genereaza si testeaza (en. *generate and test*)
- Cautarea exhaustiva gaseste solutia *optima* prin enumerarea tuturor valorilor posibile pt. X:
  - Pentru fiecare valoare posibila a lui X, verificam constrangerea

$$\sum_{i=1}^{n} d_i x_i = A$$

  - O valoare care satisface constrangerea se numeste *solutie fezabila*
  - *Solutie optima* - solutie fezabila care minimizeaza *functia obiectiv*:

$$\sum_{i=1}^{n} x_i$$

- Cate valori posibile avem pentru X? (dimensiunea spatiului de cautare)

NUMARAREA RESTULUI: "BRUTE FORCE"

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cautarea exhaustiva nu are o forma/structura specifica; se muleaza de regula pe enunt
- Pro: simplu de implementat, aplicabilitate larga
- Contra: nu e eficienta (exploreaza tot spatiul de cautare)
- Complexitate: $\Omega(n2^n)$
  - numarul de solutii candidat - $\Omega(2^n)$
  - estimarea fezabilitatii unei solutii: $O(n)$
  - calculul valorii functiei obiectiv: $O(n)$

NUMARAREA RESTULUI: EXEMPLU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **A = 20**
- **D={1,1,1,1,1,10,10,15}**
- Spatiul solutiilor:
  - X = {0,0,0,0,0,0,0,0}
  - X = {1,0,0,0,0,0,0,0}
  - ...
  - X = {1,1,1,1,1,0,0,1} - *fezabila (Problema 1)*
  - ...
  - X = {0,0,0,0,0,1,1,0} - *fezabila, optima (Problema 2)*
  - ...
  - X = {1,1,1,1,1,1,1,1}

ALTE EXEMPLE DE ALGORITMI DE TIP FORTA BRUTA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Calculul a^n ($a>0$, n - natural)
- Calcularea $n!$
- Inmultirea a 2 matrici
- Cautarea unui element intr-o lista
- Potrivire de string-uri
- Evaluare polinom in punctul x_0
- Gasirea celei mai apropiate perechi de puncte in plan
- TSP (a.k.a. Robot Tour Optimization, Ciclu/drum Hamiltonian)
- Problema rucsacului
- etc…
  - *Care din acestea sunt probleme combinatoriale?*
  - … + *de optimizare?*

BRUTE FORCE: POTRIVIRE DE STRING-URI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
function NaiveSearch(string s[1..n], string pattern[1..m])
    for i from 1 to n-m+1
      for j from 1 to m
         if s[i+j-1] ≠ pattern[j]
            jump to next iteration of outer loop
       return i
    return not found
```

Exemplu:

AAAAAAAAAAAAAAAAAAAAAAAAAAAAH

AAAAH

1) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
   AAAAH *5 comparatii*
2) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
    AAAAH *5 comparatii*
3) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
      AAAAH *5 comparatii*

…..

n-m) AAAAAAAAAAAAAAAAAAAAAAAAAAAAH
5 comparatii AAAAH

Complexitate?

Cate incercari de potrivire? (solutii candidat)

Cat ne costa o verificare de potrivire? (verificarea fezabilitatii unei solutii)

$O(mn)$ cazul *defavorabil*

Algoritmi mai eficienti?

- salt mai mare in caz de nepotrivire in for-ul exterior(Knuth–Morris–Pratt)
- bazat pe hashing (Robin Karp)

BRUTE FORCE: EVALUARE POLINOM

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$, calculati $p(x_0)$

```
x = x0
p = 0.0
for i=n down to 0 do
    power = 1
    for  j = 1 to i do
       power =  power * x
    p = p + a[i] * power
return p
```

Complexitate: $O(n^2)$

Algoritm mai eficient

```
x = x0
p = a[0]
power = 1
for i=1 to n do
      power =  power * x
      p = p + a[i] * power
return p
```

Ideea de la programare dinamica:

- salvare rezultate, pentru a evita recalcularea lor

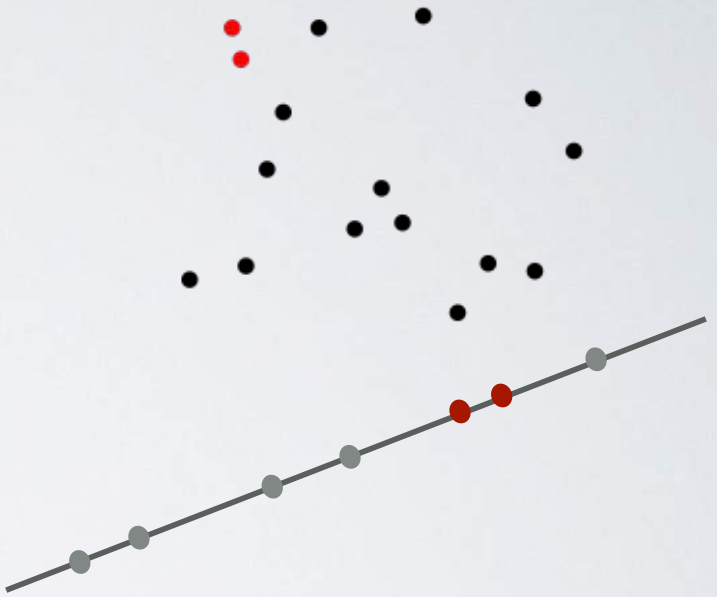
Complexitate: ?

BRUTE FORCE: GASIREA CELEI MAI APROPIATE PERECHI DE PUNCTE IN PLAN

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Se calculeaza distanta intre oricare 2 puncte din plan, si se cauta minimul:

```
min_dist = ∞
for each point i∈S
    for each point j∈S, s.t. j<>i
       if distance(i,j) < min_dist
           min_dist = distance(i,j)
           closest_pair = (i,j)
 return closest_pair
```

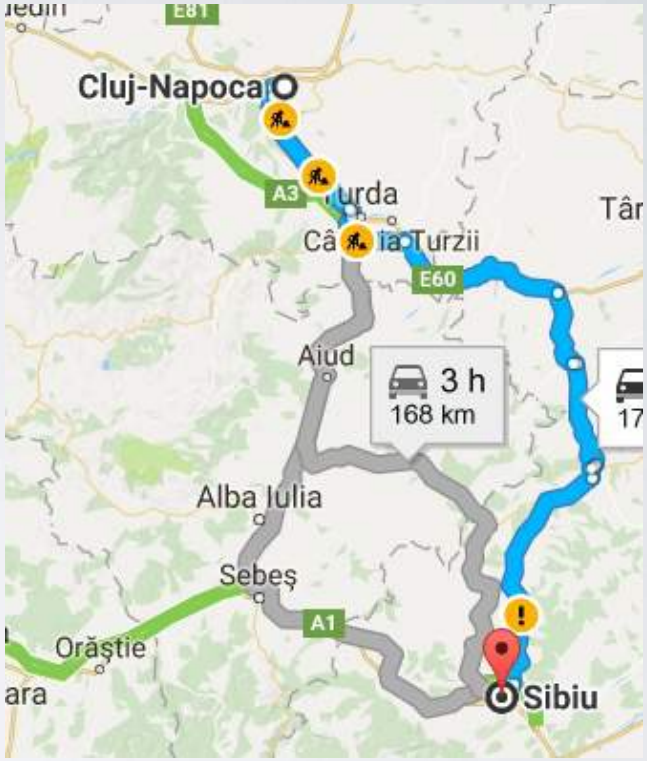


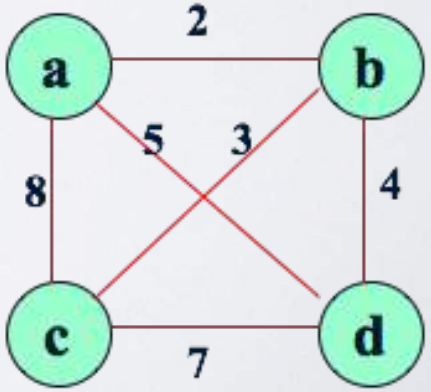
  - Complexitate: $O(n^2)$
- Dar daca punctele sunt colineare?
  - se sorteaza punctele in ordine crescatoare
  - se calculeaza diferenta intre oricare 2 pcte adiacente
  - Se cauta cea mai mica astfel de diferenta
  - Eficienta: $O(nlgn)$
- Un algoritm mai eficient pt. pcte necolineare: divide and conquer (intr-un curs viitor)

BRUTE FORCE: TSP – TRAVELING SALESMAN PROBLEM

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Se dau *n* orase, cu distante cunoscute intre fiecare 2 dintre ele (i.e. *graf neorientat, complet, etichetat* - costuri pe muchii), gasiti cel mai scurt tur care trece prin toate orasele o data, inainte de a reveni in orasul sursa.
- Formulare alternativa: Gasiti cel mai scurt *Cicuit Hamiltonian* intr-un graf complet, cu ponderi
  - Cum arata un ciclu Hamiltonian intr-un graf complet?
- Exista formulari posibile si pe grafuri orientate, grafuri neorientate care nu sunt complete, etc.



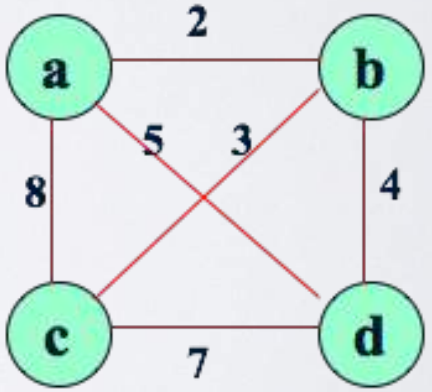


BRUTE FORCE: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
Algorithm TSP_BruteForce1(A,l,lengthSoFar)
1. n ← length[A] // number of elements in the array A
2. if ℓ = n       //finish by returning to city 1
3.    then minCost ← lengthSoFar + distance[A[n], A[1]]
4. else minCost ← ∞
5. for i ← ℓ + 1 to n
6.     do Swap A[ℓ + 1] and A[i] // select A[i] as next city
7.     newLength ← lengthSoFar + distance[A[ℓ], A[ℓ + 1]]
8.     minCost ← min(minCost, TSP_BruteForce(A,ℓ+1,newLength))
9.     Swap A[ℓ + 1] and A[i] // undo the selection
10. return minCost
```

Tur	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17



Complexitate?

Cate cicluri hamiltoniene exista intr-un graf complet avand n noduri?

BRUTE FORCE: TSP

Tur	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

Eficienta: **(n-1)!** -> ***O(n!)***

Daca un calculator ar calcula 1.000.000 cicluri/sec

n = 6, 7, 8, 9: instantaneu
n = 10: ~ 113 sec
n = 11: ~ 4 sec
n = 12: ~ 40 sec
n = 13: ~ 8 min
n = 14: ~ 2 ore
n = 15: putin peste o zi
n = 20: peste 1.000.000 ani

BACKTRACKING: STRATEGIE GENERALA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cautare in spatiul starilor posibile (en. *state space search alg.*)
- Se parcurge "arborele de stari posibile" folosind o strategie de *parcurgere in adancime (DFS)*
- O solutie este generata componenta cu componenta (*partiala*), si evaluata la fiecare pas (in timp ce este construita):
  - daca aceasta poate fi dezvoltata in continuare fara a viola constrangerile problemei, se considera urmatoarea optiune legitima pentru urmatoarea componenta a solutiei
  - daca nu mai exista optiuni legitime pentru urmatoarea componenta a solutiei, algoritmul revine si inlocuieste valoarea componentei curente a solutiei partiale, cu urmatoarea optiune (valoare) pentru aceasta

BACKTRACKING: SCHEMA GENERALA (PSEUDOCOD)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
Algorithm Backtrack-DFS(A, k)
1.  if A  = (a1, a2, ..., ak) is a solution, process it
2.  else
3.     k  = k + 1   // get to next component of a solution
4.     generate Sk   // generate set of candidate next states
5.     while Sk != Ø  do
6.        ak  = an element in Sk // set component k of sol. to ak
7.        Sk  = Sk – ak           // remove ak from options for state k
8.        Backtrack-DFS(A, k)    // explore crt. partial sol. further
```

La fiecare pas se incearca extinderea solutiei partiale prin adaugarea unui nou element, a_k

Daca la pasul curent avem o solutie, o procesam

Daca nu, vedem daca se poate extinde in continuare (generam in S_k - starile urmatoare posibile, si le exploram pe rand)

Backtracking - construieste un arbore de solutii partiale - fiecare nod - solutie partiala; avem muchie de la *x* la *y* daca in *y* am ajuns extinzand solutia din *x*; frunzele – solutii fezabile; arborele se parcurge in *adancime*

BACKTRACKING: SCHEMA GENERALA (COD)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
bool finished = FALSE; /* found all solutions yet? */
backtrack(int a[], int k, data input)
{
  int c[MAXCANDIDATES]; /* candidates for next position */
  int ncandidates; /* next position candidate count */
  int i; /* counter */
  if (is_a_solution(a,k,input))
      process_solution(a,k,input);
  else {
      k = k+1;
      construct_candidates(a,k,input,c,&ncandidates);
      for (i=0; i<ncandidates; i++){
          a[k] = c[i];
          make_move(a,k,input);
          backtrack(a,k,input);
          unmake_move(a,k,input);
          if (finished) return; /* terminate early */
      }
  }
}
```

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: SCHEMA GENERALA (COD)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- **`is_a_solution(a,k,input)`**- testeaza daca primele **`k`** elemente ale vectorului **`a`** formeaza o solutie; **`input`** ne permite sa transmitem informatie generala in functie (e.g. dimensiunea *n* a solutiei)
- **`construct_candidates(a,k,input,c,&ncandidates)`**– populeaza vectorul **`c`** cu multimea posibila de valori pentru pozitia **`k`** a lui **`a`**, cunoscandu-se primele **`k-1`** pozitii
- **`process solution(a,k,input)`** - afiseaza, numara, proceseaza o solutie completa odata de a fost construita
- **`make_move(a,k,input)`** si **`unmake_move(a,k,input)`** - ne permit sa modificam o structura in raspuns la ultima mutare efectuata (**`make_move`**), respectiv sa o curatam (**`unmake_move`**) daca decidem sa anulam mutarea; e mai eficient decat sa se reconstruiasca din *a* de fiecare data
- flag-ul **`finished`** permite terminarea prematura, si poate fi modificat in oricare din functiile de mai sus, in functie de necesitate (e.g. la gasirea unei solutii)

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: GENERAREA SUB-MULTIMILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Solutia se reprezinta ca un vector de dimensiune *n*:

$$a = \{a_0, a_1, \ldots, a_{n-1}\}, a_i \in \{0, 1\}$$

```
is_a_solution(int a[], int k, int n){
   return (k == n-1); /* is k == n? */
}

construct_candidates(int a[], int k, int n, int c[], int *ncandidates){
   c[0] = TRUE;
   c[1] = FALSE;
   *ncandidates = 2;
}

process_solution(int a[], int k){
   int i; /* counter */
   printf("{");
   for (i=0; i<k; i++)
     if (a[i] == TRUE) printf(" %d",i+1);
   printf(" }\n");
}
```

? Cum apelam functia *backtrack* in cazul curent? (presupuneti n=3)
? Cum arata o solutie de fapt? Dar spatiul de stari?
? In ce ordine se genereaza submultimile multimii {1, 2, 3} ?

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: NUMARAREA TUTUROR CAILOR INTRE 2 NODURI IN GRAF (S SI T)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Solutie: lista de noduri, pe prima pozitie: s; pentru a doua pozitie: $\{v \mid (s,v) \in E\}$; in general, la pasul *k* se adauga la lista de candidati varfurile adiacente lui a_k care nu apar in solutia partiala

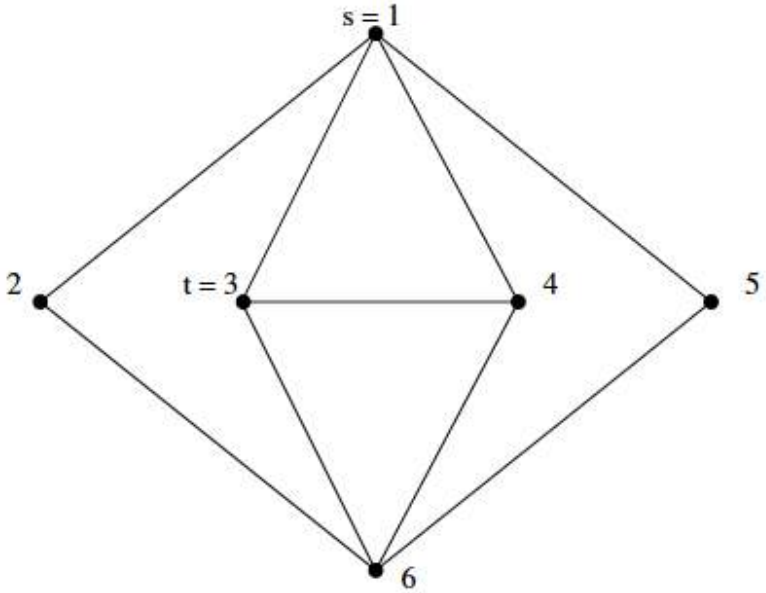
```
construct_candidates(int a[], int k, int n, int c[], int *ncandidates) {
 int i; /* counters */
 bool in_sol[NMAX]; /* what's already in the solution? */
 edgenode *p; /* temporary pointer */
 int last; /* last vertex on current path */
 for (i=1; i<NMAX; i++) in_sol[i] = FALSE;
 for (i=1; i<k; i++) in_sol[ a[i] ] = TRUE;
 if (k==1) { /* always start from vertex 1 */
   c[0] = 1;
   *ncandidates = 1;
 } else {
   *ncandidates = 0;
   last = a[k-1];
   p = g.edges[last];
   while (p != NULL) {
     if (!in_sol[ p->y ]) {
       c[*ncandidates] = p->y;
       *ncandidates = *ncandidates + 1;
     }
     p = p->next;
   }
 }
}
```

```
is_a_solution(int a[], int k, int t){
    return (a[k] == t);
}
process_solution(int a[], int k){
    solution_count ++; /* count all paths */
}
```

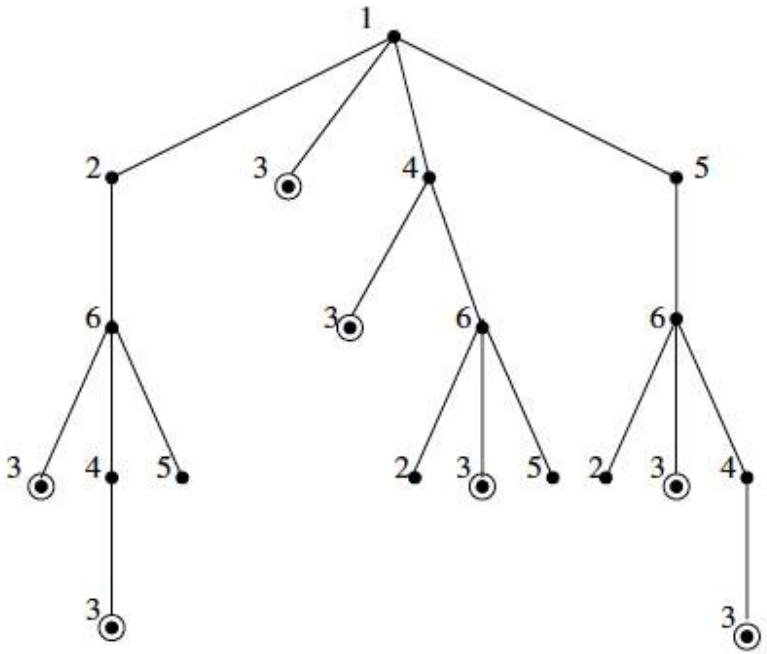
BACKTRACKING: NUMARAREA TUTUROR CAILOR INTRE 2 NODURI IN GRAF (S SI T)

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Graful, s si t



Arborele de stari

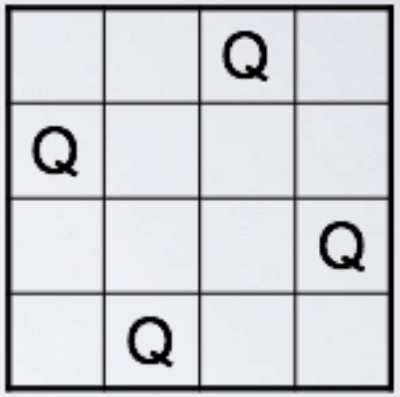


S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: PROBLEMA REGINELOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Cum asezam *n* regine pe o table de sah de dimensiune *nxn* astfel incat sa nu se atace, i.e. nu avem 2 regine pe aceeasi *linie*, *coloana* sau *diagonala*

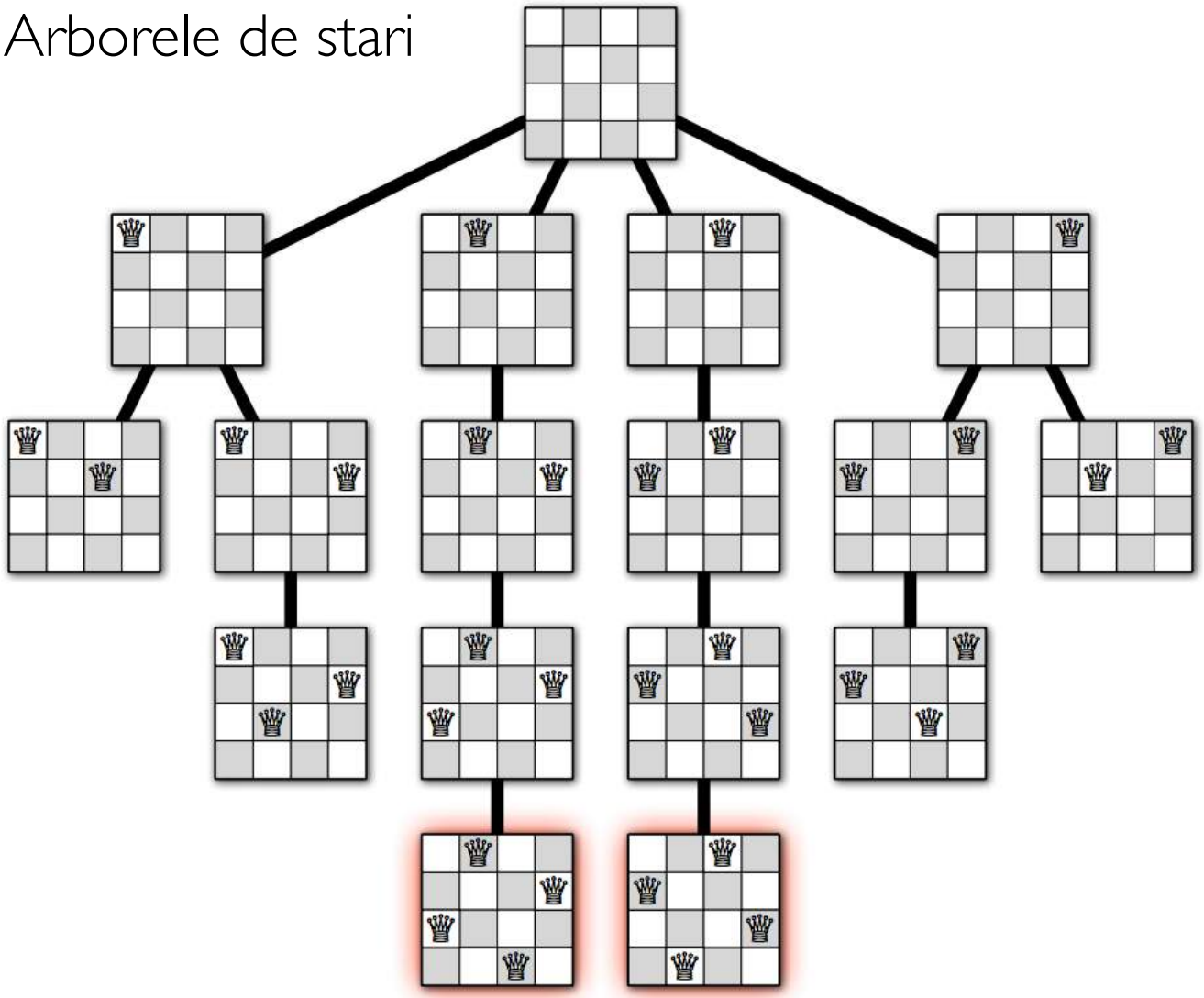


- Strategie:
  - se incearca plasarea cate unei regine pe fiecare linie (coloana)
  - la fiecare linie (coloana) noua se plaseaza regina pe coloana (linia) care nu este in conflict cu configuratia de pana atunci
  - daca nu este posibila plasarea reginei pe linia (coloana) curenta, se revine la cea anterioara

BACKTRACKING: PROBLEMA REGINELOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Arborele de stari



Plasarea reginei pe prima linie

Plasarea reginei pe a doua linie

...

BACKTRACKING: PROBLEMA REGINELOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
Algorithm Backtrack_NQueens(Q[1 .. n],r)
1. if r = n + 1
2.    print Q
3. else
4.    for j ← 1 to n
5.       legal ← True
6.       for i ← 1 to r - 1
7.          if (Q[i]=j) or (Q[i]=j+r-i) or (Q[i]=j-r+i)
8.             legal ← False
9.       if legal
10.         Q[r] ← j
11.         Backtrack_NQueens(Q[1 .. n],r + 1)
```

Incercati sa identificati componentele schemei generale pe aceasta implementare

BACKTRACKING: SUDOKU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

					1	2		
			3	5				
		6				7		
7					3			
			4		8			
1								
			1	2				
		8				4		
		5			6			

6	7	3	8	9	4	5	1	2
9	1	2	7	3	5	4	8	6
8	4	5	6	1	2	9	7	3
7	9	8	2	6	1	3	5	4
5	2	6	4	7	3	8	9	1
1	3	4	5	8	9	2	6	7
4	6	9	1	2	8	7	3	5
2	8	7	3	5	6	1	4	9
3	5	1	9	4	7	6	2	8

Candidatii pentru celula (i,j):

- intregii intre 1 si 9, care nu au aparut inca in randul i, coloana j si patratul de dimensiune 3x3 care contine celula (i,j)
- revenirea se face de indata ce nu mai exista candidati viabili pentru o celula

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: SUDOKU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
#define DIMENSION 9 /* 9*9 board */
#define NCELLS DIMENSION*DIMENSION /* 81 cells in a 9*9 problem */

typedef struct {
  int x, y;
} point;
/* x and y coordinates of point */

typedef struct {
  int m[DIMENSION+1][DIMENSION+1]; /* matrix of board contents */
  int freecount; /* how many open squares remain? */
  point move[NCELLS+1]; /* how did we fill the squares? */
} boardtype;

 is_a_solution(int a[], int k, boardtype *board) {
    if (board->freecount == 0)
      return (TRUE);
    else
      return(FALSE);
}
```

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: SUDOKU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
construct_candidates(int a[], int k, boardtype *board, int c[],
int *ncandidates){
    int x,y; /* position of next move */
    int i; /* counter */
    bool possible[DIMENSION+1]; /* what is possible for the square */

    next_square(&x,&y,board); /* which square should we fill next? */
    board->move[k].x = x; /* store our choice of next position */
    board->move[k].y = y;
    *ncandidates = 0;
    if ((x<0) && (y<0)) return; /* error, no moves possible */
    possible_values(x,y,board,possible);
    for (i=0; i<=DIMENSION; i++)
        if (possible[i] == TRUE) {
          c[*ncandidates] = i;
          *ncandidates = *ncandidates + 1;
        }
}
```

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: SUDOKU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
make_move(int a[], int k, boardtype *board){
    fill_square(board->move[k].x,board->move[k].y,a[k],board);
}

unmake_move(int a[], int k, boardtype *board){
    free_square(board->move[k].x,board->move[k].y,board);
}

 process_solution(int a[], int k, boardtype *board){
    print_board(board);
    finished = TRUE;
}
```

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING: SUDOKU

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

`next_square(&x,&y,board)`

- *selectie arbitrara* - primul/ultimul/aleator din celulele necompletate
- *selectia celei mai constranse celule* - celula cu cele mai putine valori candidat disponibile (dar > 0)

`possible_values(x,y,board,possible)`

- *numarare locala* - constrangerea le rand, coloana si sector
- *"look ahead"* - daca constrangerea provine de la o alta celula, astfel incat nu exista nici o valoare posibila pentru celula curenta

Pruning Condition		Puzzle Complexity		
`next_square`	`possible_values`	Easy	Medium	Hard
arbitrary	local count	1,904,832	863,305	never finished
arbitrary	look ahead	127	142	12,507,212
most constrained	local count	48	84	1,243,838
most constrained	look ahead	48	65	10,374

S. Skiena: *The Algorithm Design Manual*

BACKTRACKING IN PROBLEME COMBINATORIALE DE **OPTIMIZARE**

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

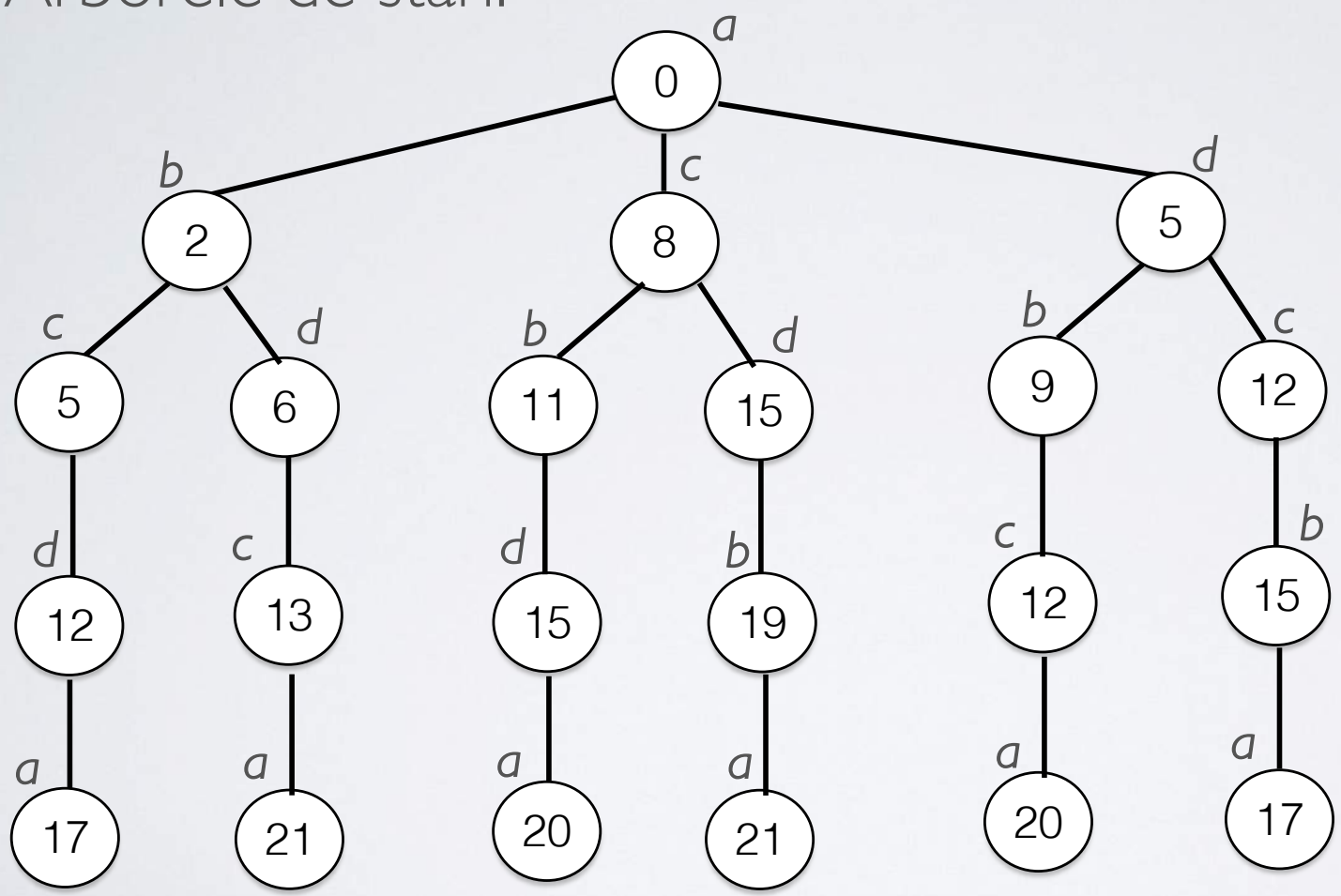
!!! **Remember**: vrem cea mai buna solutie dintre cele existente !!!

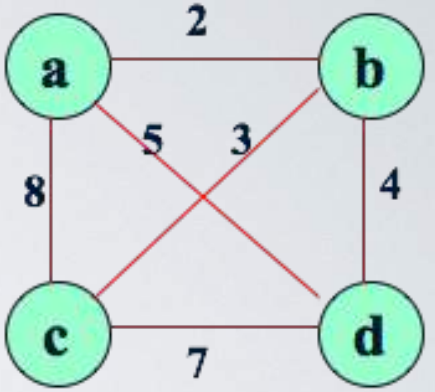
- Cea mai buna solutie gasita pana in momentul curent e utilizata pentru a elimina (en. *prune*) solutii partiale ne-promitatoare, i.e. nu pot conduce la:
  - o solutie *fezabila* (backtracking in general)
  - o solutie *mai buna* decat cea gasita pana la acel moment (a.k.a ***Branch and bound***)
- Scopul este de a elimina suficiente stari din spatiul de cautare (*exponential,* de regula, in marime), astfel incat solutia optima sa poata fi gasita intr-un timp rezonabil
- In cazul defavorabil, strategia da tot algoritmi exponentiali

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

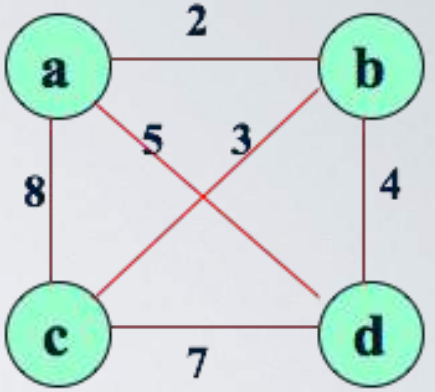
- Arborele de stari:



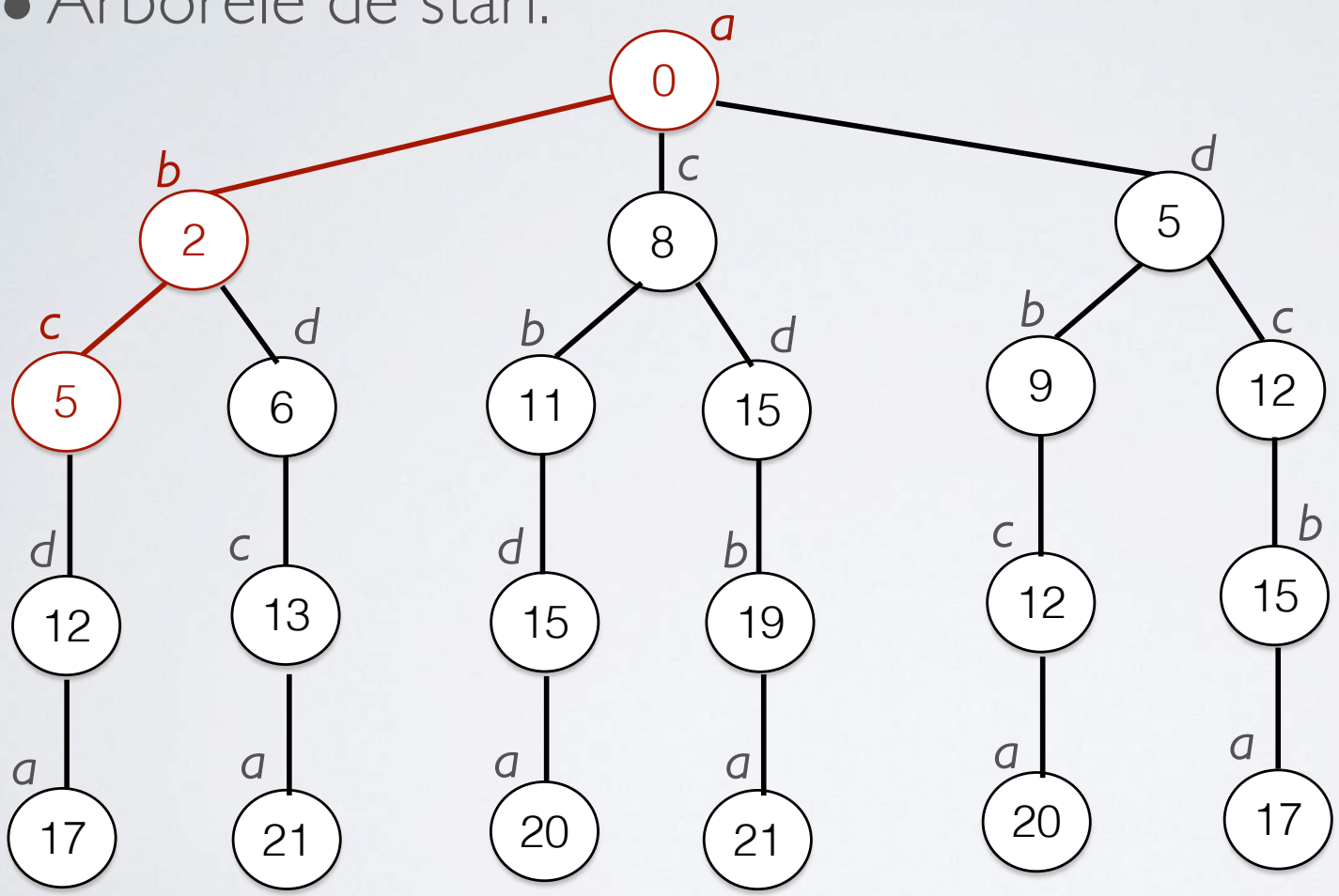


BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA



- Arborele de stari:

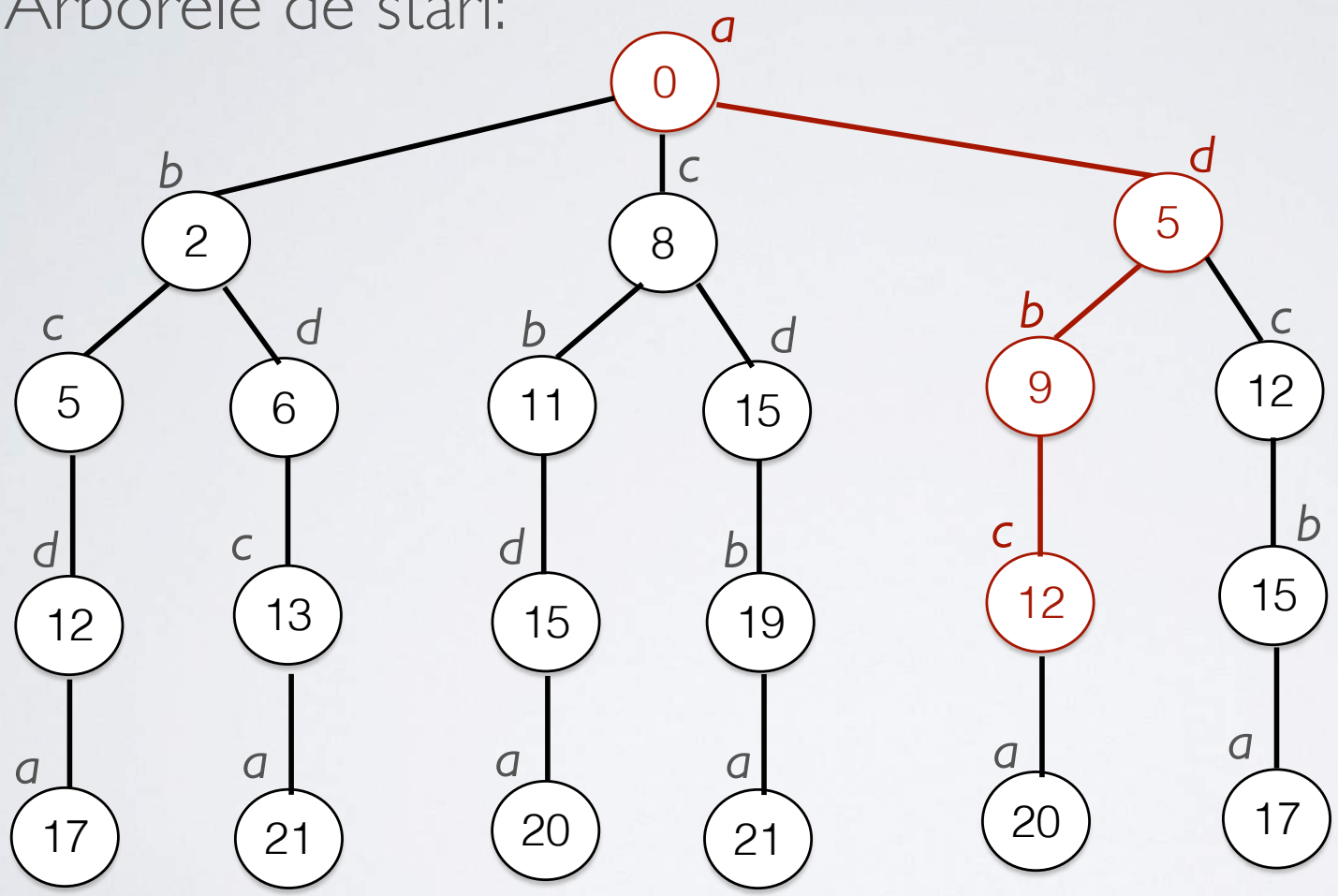


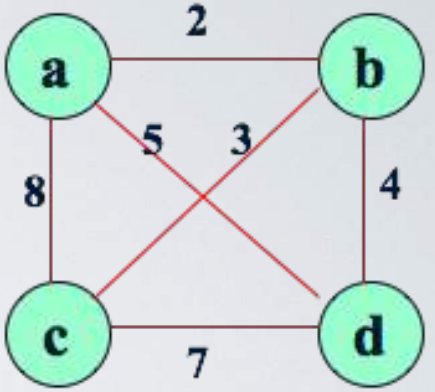
a-b-c este o solutie partiala

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Arborele de stari:



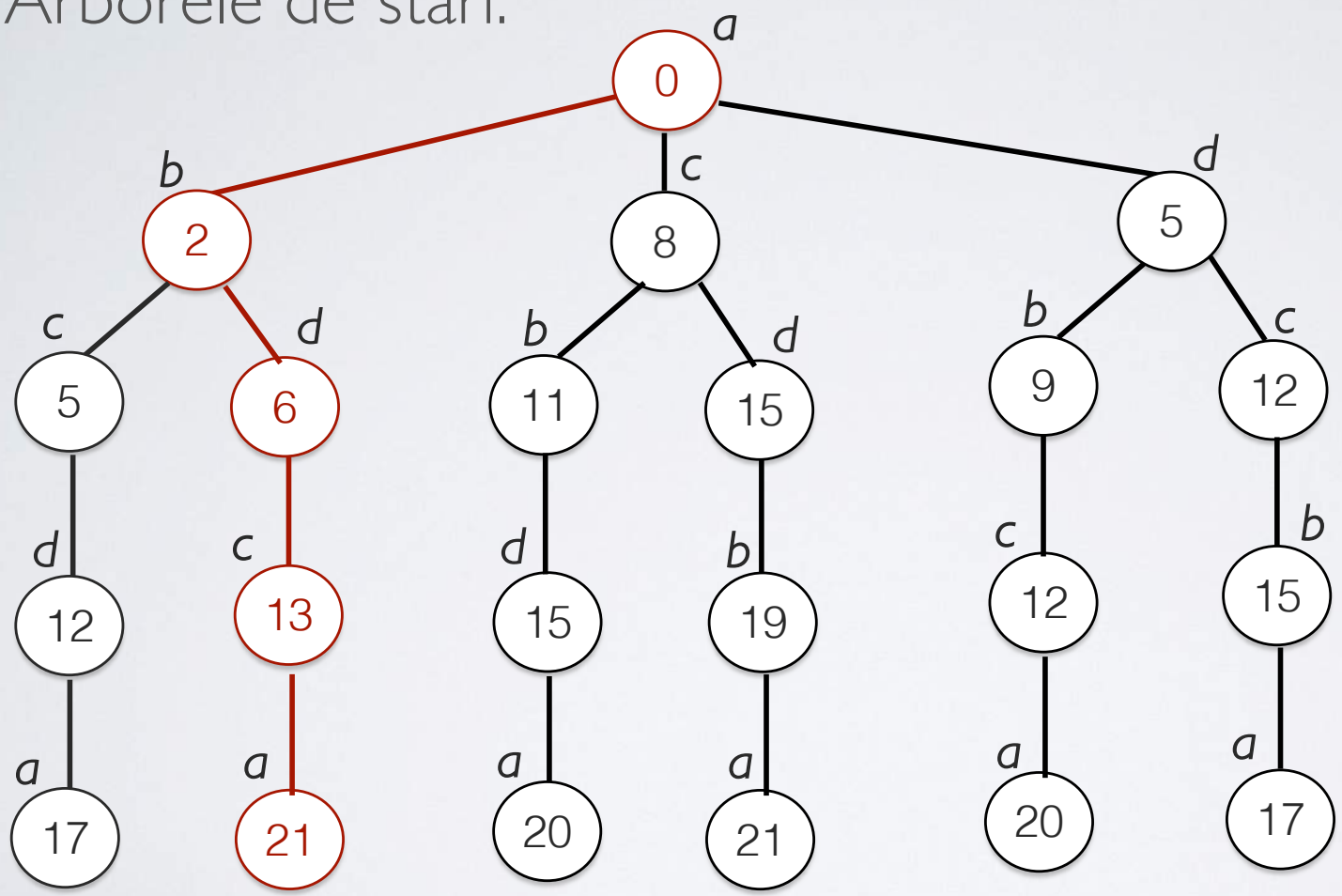


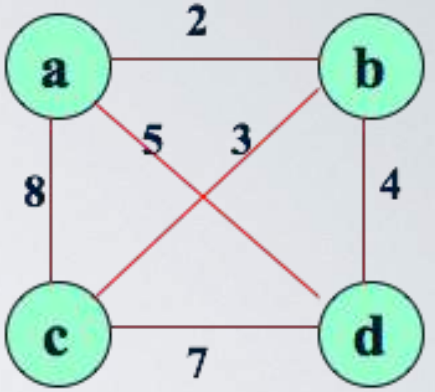
a-d-b-c este o solutie partiala

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Arborele de stari:



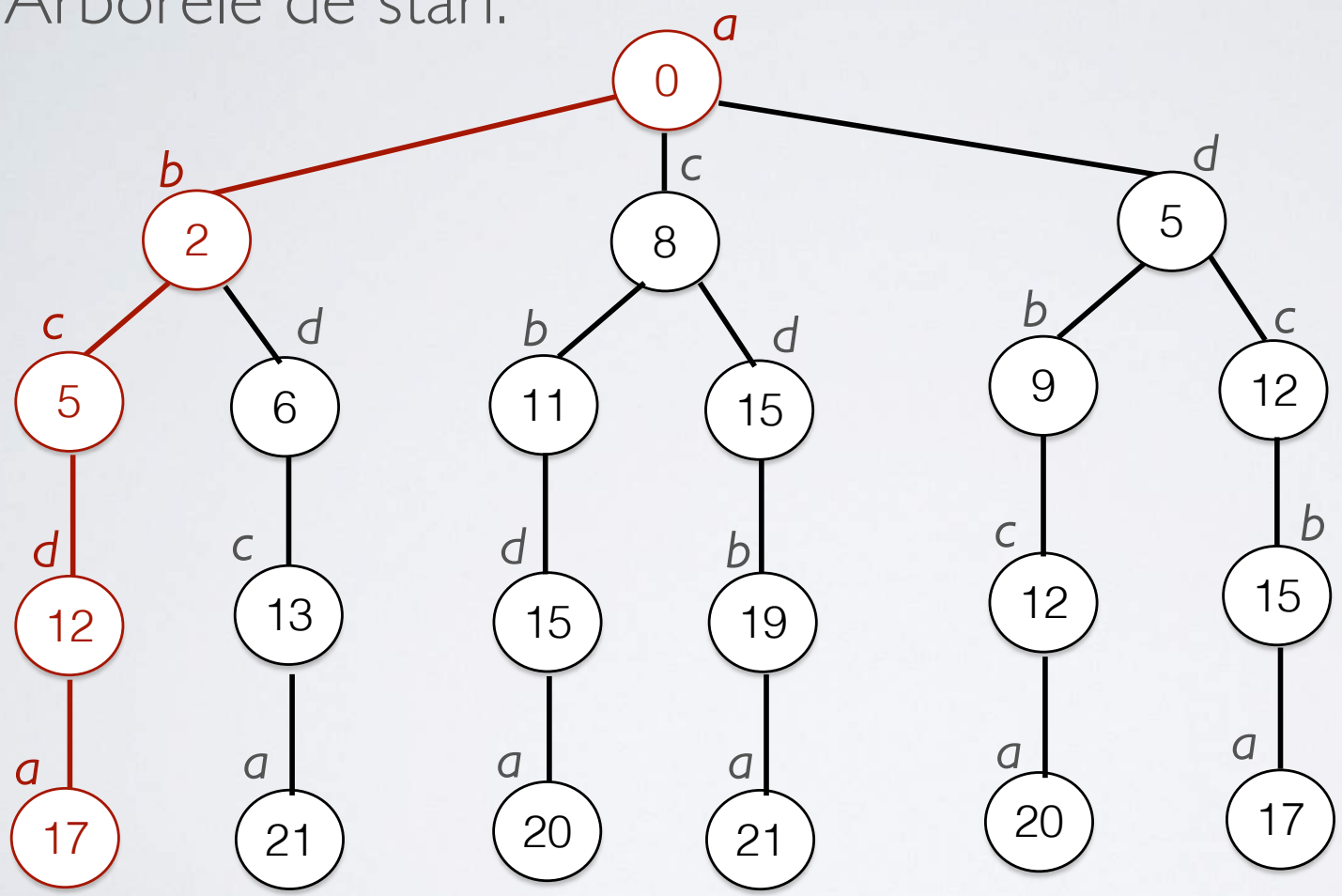


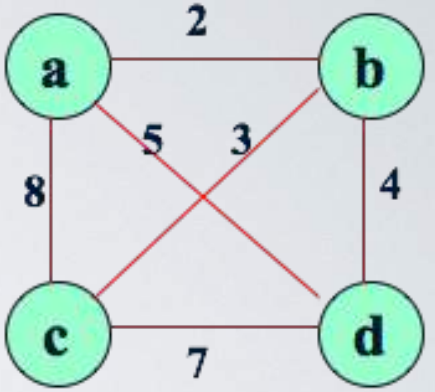
a-b-d-c-a este o solutie (fezabila)

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Arborele de stari:





a-b-c-d-a este solutia optima

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

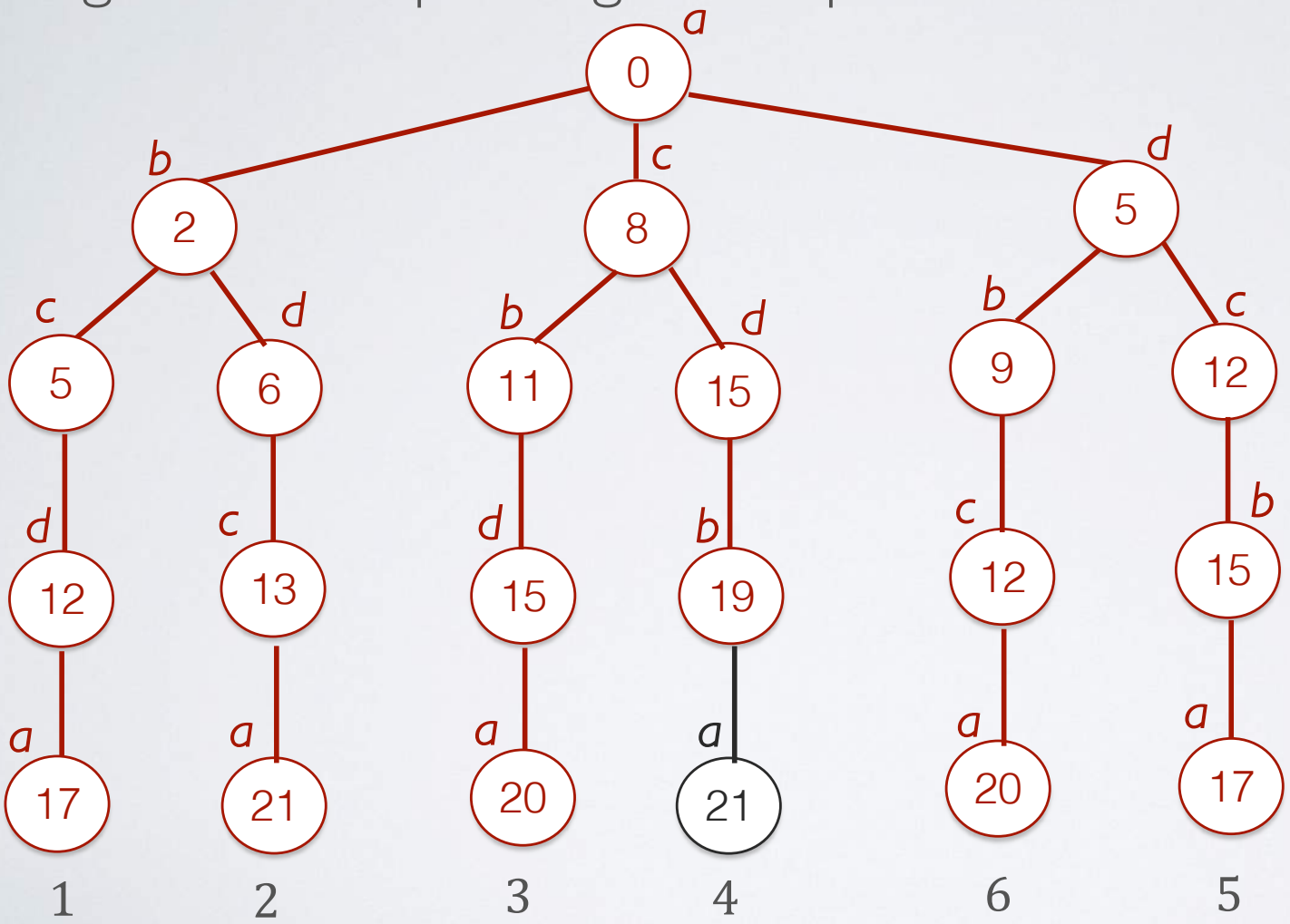
```
Algorithm TSP_Backtrack(A, ℓ, lengthSoFar , minCost)
1. n ← length[A] // number of elements in the array A
2. if ℓ = n  //found a new solution, update best so far
3.    then minCost ← min(minCost,lengthSoFar+distance[A[n],A[1]])
4. else
5.    for i ← ℓ + 1 to n
6.       do Swap A[ℓ + 1] and A[i] // select A[i] as the next city
7.       newL ← lengthSoFar + distance[A[ℓ], A[ℓ + 1]]
8.       if newL > minCost    //this won't be a better solution
9.          then skip         //prune
10.         else minCost ←    //develop crt. solution further
11.             min(minCost, TSP_Backtrack(A,ℓ + 1,newL,minCost))
12.      Swap A[ℓ + 1] and A[i] // undo the selection
13. return minCost
```

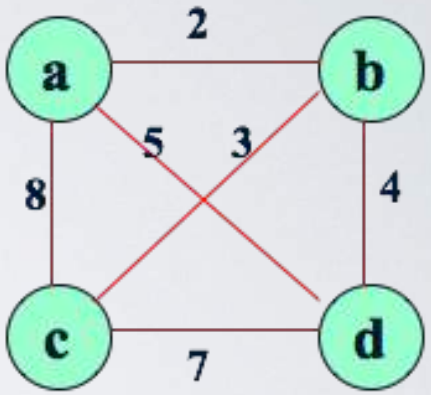
Branch and Bound

BACKTRACKING: TSP

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Algoritmul va parcurge doar portiunea marcata cu visiniu:





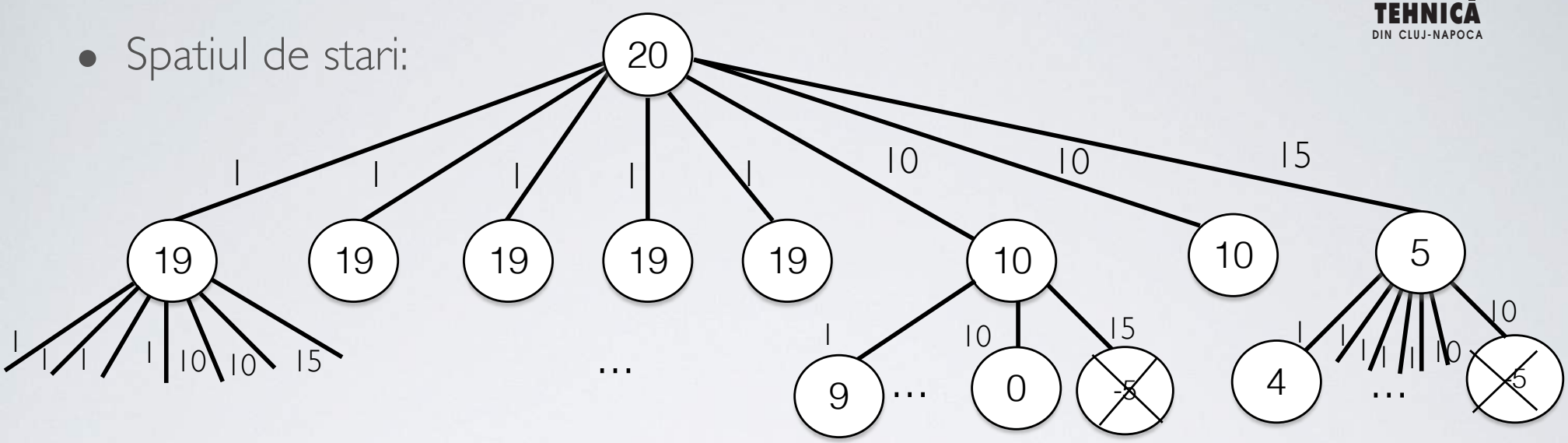
Ordinea de parcurgere

Tema: modificati algoritmul a.i. incat sa nu se exploreze ciclurile duplicate

BACKTRACKING: NUMARAREA RESTULUI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Spatiul de stari:



- Care sunt criteriile de eliminare a starilor neinteresante?
  - cantitate negativa a restului de returnat
  - la o singura moneda pana la cea mai buna solutie de pana atunci, mai avem rest de returnat (branch and bound)
- Cum ordonam nodurile in arbore (pe acelasi nivel?)
  - nicicum (in ordinea in care apar pt parcurgere)
  - *Ar ajuta ordonarea?*

- *TEMA*: Implementati solutia bazata pe backtracking pentru problema numararii restului(ambele formulari), utilizand schema de cod propusa in curs

BACKTRACKING: UNDE E UTILIZAT

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- CSP – Constraint Satisfaction Problems
  - Puzzles
    - Sudoku
    - Crosswords
    - Aranajarea reginelor pe tabla de sah
    - Colorarea grafurilor
    - Etc.
  - Optimizare combinatoriala
    - pruning in functie de "best so far" (bound): Branch and bound
    - TSP
    - Numararea restului
    - Problema rucsacului (cursul urmator – sol. greedy)
    - Etc.
- Programare logica – mecanismul de executie al unui program
  - Prolog
- Etc.

BACKTRACKING VS BRANCH AND BOUND

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Backtracking
  - DFS pentru a parcurge spatiul de stari
  - Se poate opri la prima solutie (daca se cere doar o solutie), sau exploreaza tot spatiul (daca se cer toate/cea mai buna solutie)
    - Se poate utiliza in probleme de optimizare
  - Strategia de pruning/revenire – functie de *fezabilitate* (a solutiei)
    - Solutia candidat nu este fezabila
- B&B
  - Poate utiliza in principiu orice alg de cautare (BFS, DFS)
  - Cauta dupa cea mai buna solutie, in tot spatiul de cautare
    - Se utilizeaza *exclusiv* in probleme de optimizare
  - Strategia de pruning/revenire – functie de *marginire* (bounding) a sol
    - Solutia candidat nu e fezabila
    - Solutia candidat nu poate conduce la o solutie mai buna decat cea gasita pana la acel punct

OPTIMIZARE COMBINATORIALA
DISCUTIE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Spatiul de stari poate fi explorat in alta ordine (nu neaparat DFS)
- Ordonam nodurile de pe un nivel
  - Exploram intai nodurile "mai promitatoare"
  - E.g. TSP:
    1. Cost minim al caii partiale pana in acel moment
    2. Estimam costul asteptat:

$$\mathbf{ExpectedCost}\,(\mathbf{PPath}) = \mathbf{Cost}(\mathbf{PPath}) + \mathbf{EstimatedDistance}(\mathbf{last}(\mathbf{PPath}), \mathbf{DEST})$$

  - *Coada de prioritati* care sa mentina ordinea nodurilor candidat
- *Strategii neinformate*
  - Bazate pe BFS, DFS – ordinea de cautare a nodurilor "aleatoare"
- *Strategii informate* (*mai multe in anul III la IA…*)
  - Estimam cat de aproape de solutie ne aflam la un moment dat
  - Expandeaza cele mai "promitatoare" noduri intai
    - Cautari *euristice*: Best-first search, A*, etc…

BIBLIOGRAFIE

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- S. Skiena: *The Algorithm Design Manual*, cap 7
- https://courses.cs.washington.edu/courses/cse143/12su/lectures.shtml#today

PROBLEME PROPUSE I

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- *Problema numararii restului*: Un casier are la dispozitie o colectie finita de bancnote si monede de diferite valori.

1. Se cere sa se genereze toate submultimile colectiei de bancnote si monede care au suma A.
2. Se cere sa se genereze o singura submultime a colectiei de bancnote si monede care are suma A si numar minim de elemente.

PROBLEME PROPUSE II

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

1. Scrieti o functie *aruncaZaruri(int n)* care are ca parametru un numar intreg n, reprezentand nu numar de zaruri. Functia va afisa toate combinatiile posibile de valori care pot apare la aruncarea celor n zaruri.
2. Scrieti o functie `zaruriSuma(int n, int sum)` care are un parametru in plus, sum si afiseaza doar combinatiile de zaruri care au suma egala cu sum.
3. Permutari: scrieti o functie *permuta(string a)* care are ca si parametru un sir de caractere si afiseaza toate permutarile posibile ale literelor din cuvantul dat. Permutarile pot fi in orice ordine.
4. Combinari: scrieti o functie `combinari(string s, int k)` care afiseaza toate cuvintele ce se pot forma din literele cuvantului s si au exact k litere.
5. Baktracking in plan: un soricel se gaseste intr-un labirint de forma dreptunghiulara cu m linii si n coloane. Peretii sunt marcati cu 1 si culoarele cu 0. Se cunosc coordonatele initiale ale soricelului: Li, Ci. Sa se determine toate posibilitatile pe care le are soricelul pentru a iesi din labirint. Soricelul poate avansa pe 4 directii cate o celula (sus, dreapta , jos, stanga). Traseul soricelului va fi retinut de un vector cu doua campuri: coordonatele x si y.
6. Romeo si Julieta se gasesc intr-un labirint (se cunosc culoarele si peretii si coordonatele celor doi indragostiti). (a)Exista posibilitatea ca Romeo sa ajunga la Julieta? (b)in cazul in care cei doi se indreapta simultan unul catre celalat pentru fiecare solutie se va afisa locul intalnirii (coordonatele celulelor alaturate sau celulei comune de intalnire)

DISCUTIE: ARUNCAREA ZARURILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Scrieti o functie *aruncaZaruri(int n)* care are ca parametru un numar intreg n, reprezentand nu numar de zaruri. Functia va afisa toate combinatiile posibile de valori care pot apare la aruncarea celor n zaruri.

`aruncaZaruri(2):`

[1, 1]	[3, 1]	[5, 1]
[1, 2]	[3, 2]	[5, 2]
[1, 3]	[3, 3]	[5, 3]
[1, 4]	[3, 4]	[5, 4]
[1, 5]	[3, 5]	[5, 5]
[1, 6]	[3, 6]	[5, 6]
[2, 1]	[4, 1]	[6, 1]
[2, 2]	[4, 2]	[6, 2]
[2, 3]	[4, 3]	[6, 3]
[2, 4]	[4, 4]	[6, 4]
[2, 5]	[4, 5]	[6, 5]
[2, 6]	[4, 6]	[6, 6]

`aruncaZaruri(3):`

[1, 1, 1]
[1, 1, 2]
[1, 1, 3]
[1, 1, 4]
[1, 1, 5]
[1, 1, 6]
[1, 2, 1]
[1, 2, 2]
...
[6, 6, 4]
[6, 6, 5]
[6, 6, 6]

EXEMPLU: ARUNCAREA ZARURILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

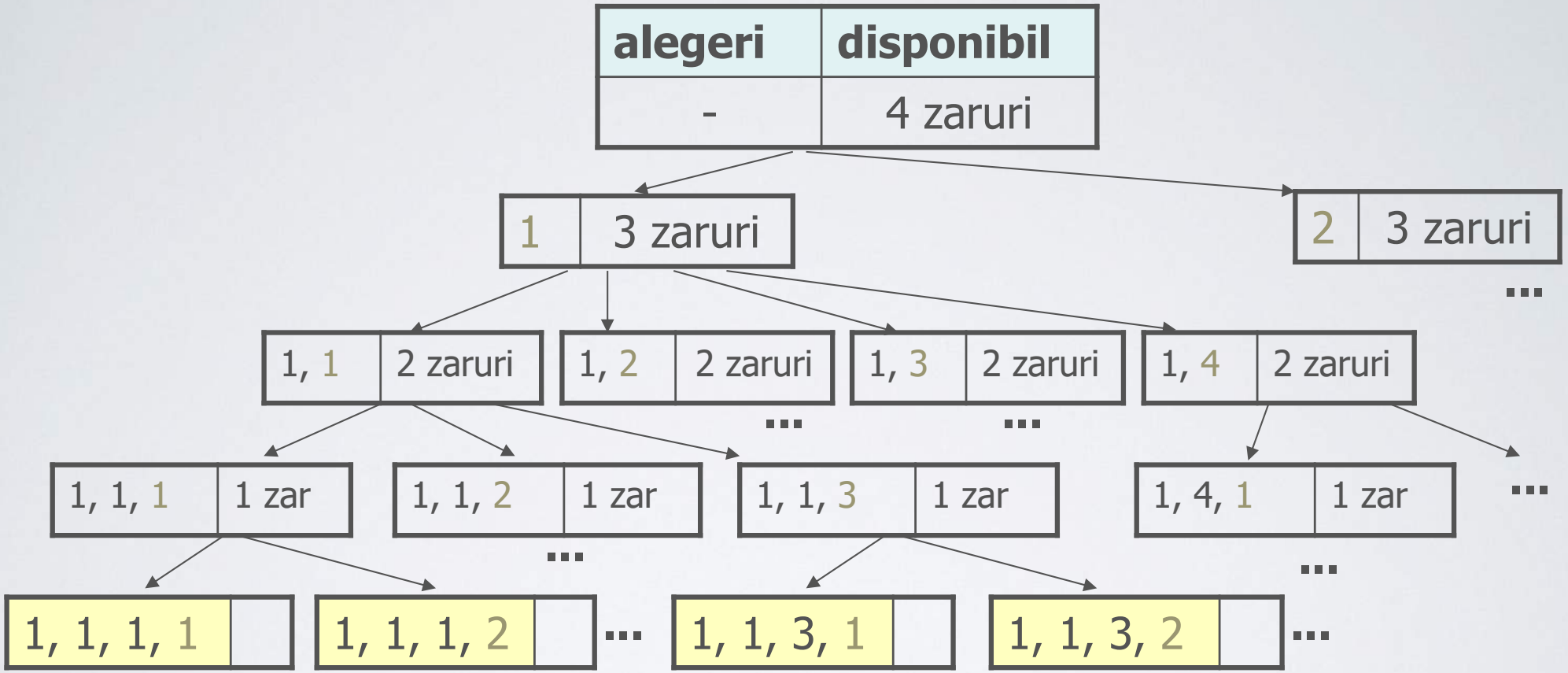
Sa examinam problema:

- Dorim sa generam toate secventele posibile de valori.
  - Pentru (fiecare valoare de la 1 la 6 pentru primul zar):
    - Pentru (fiecare valoare posibila de la 1 la 6 pentru al doilea zar):
      - Pentru (fiecare valoare posibila de la 1 la 6 pentru al 3-lea zar):
        - …..
          - Afiseaza !

- Aceasta este cautare in adancime in spatial solutiilor !
- Cum putem explora asa un spatiu mare de cautare?

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

ARBORE DE DECIZIE



FUNCTII AJUTOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Deseori functia nu accepta parametrii doriti
  - Construim functii ajutor care accepta mai multi parametrii
  - Parametrii pot reprezenta starea curenta, alegerile facute, etc

```
int backtracking(params):
    ...
    return ajutor(params, moreParams);


private int ajutor(params, moreParams):
    ...
    (se folosesc moreParams pentru a rezolva problema)
```

SOLUTIE LA PROBLEMA ZARURILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

```
void aruncaZaruri_ajutor(int n, node *alegeri){
    if (n == 0) afiseazaLista(alegeri);
    else { for (int i = 1; i <= 6; i++){
            insertLast(&alegeri, i);
            aruncaZaruri_ajutor(n-1, alegeri);
            deleteLast(&alegeri);
        }
    }
}
void aruncaZaruri(int n){
    node *alegeri = NULL;
    aruncaZaruri_ajutor(n, alegeri);
}
int main() {
    aruncaZaruri(2);
    return 0;
}
```

DISCUTIE: ARUNCARE ZARURI CU O SUMA

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Scrieti o functie `zaruriSuma(int n, int sum)` care are un parametru in plus, sum si afiseaza doar combinatiile de zaruri care au suma egala cu sum.

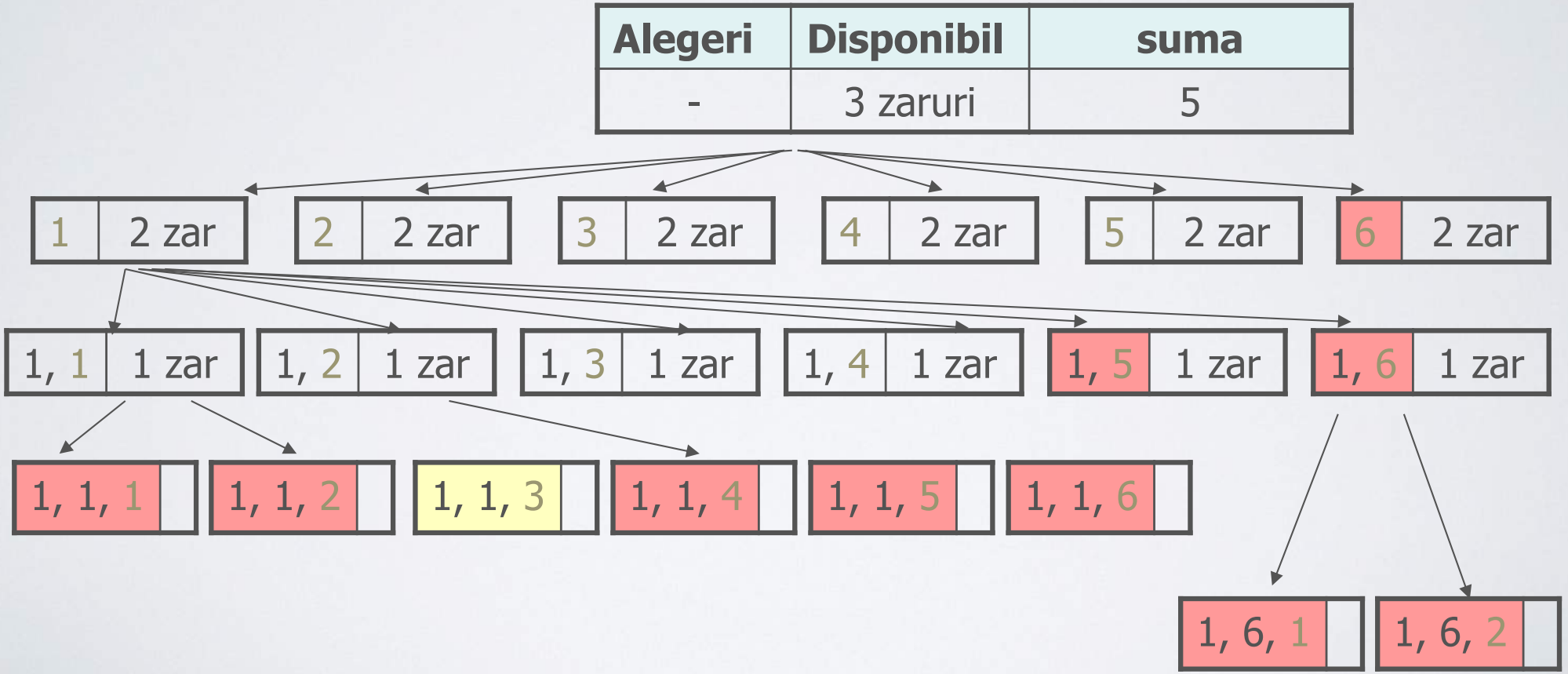
- zaruriSuma(2, 7):

```
[1, 6]
[2, 5]
[3, 4]
[4, 3]
[5, 2]
[6, 1]
```

zaruriSuma(3, 7):

```
[1, 1, 5]
[1, 2, 4]
[1, 3, 3]
[1, 4, 2]
[1, 5, 1]
[2, 1, 4]
[2, 2, 3]
[2, 3, 2]
[2, 4, 1]
[3, 1, 3]
[3, 2, 2]
[3, 3, 1]
[4, 1, 2]
[4, 2, 1]
[5, 1, 1]
```

UN NOU ARBORE DE DECIZIE:



OPTIMIZARI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Nu e nevoie sa se viziteze fiecare ramura a arborelui de decizie.
  - Unele ramuri nu vor conduce la o solutie.
  - Se poate sa ne oprim sau sa taiem (prune) aceste ramuri.

- Discutie – ineficiente in functia care calculeaza suma zarurilor.
  - Suma curenta este deja prea mare.
    - (Chiar daca apare 1 pentru toate zarurile ramase, suma va depasi valoarea sumei cerute.
  - Uneori suma curenta este prea mica.
    - (Chiar daca apare 6 pentru toate zarurile ramase suma totala nu va depasi suma ceruta)
  - Cand am terminat – codul trebuie sa calculeze suma de fiecare data ! (1+1+1 = ..., 1+1+2 = ..., 1+1+3 = ..., 1+1+4 = ..., ...)

DISCUTIE: PROBLEMA PERMUTARILOR

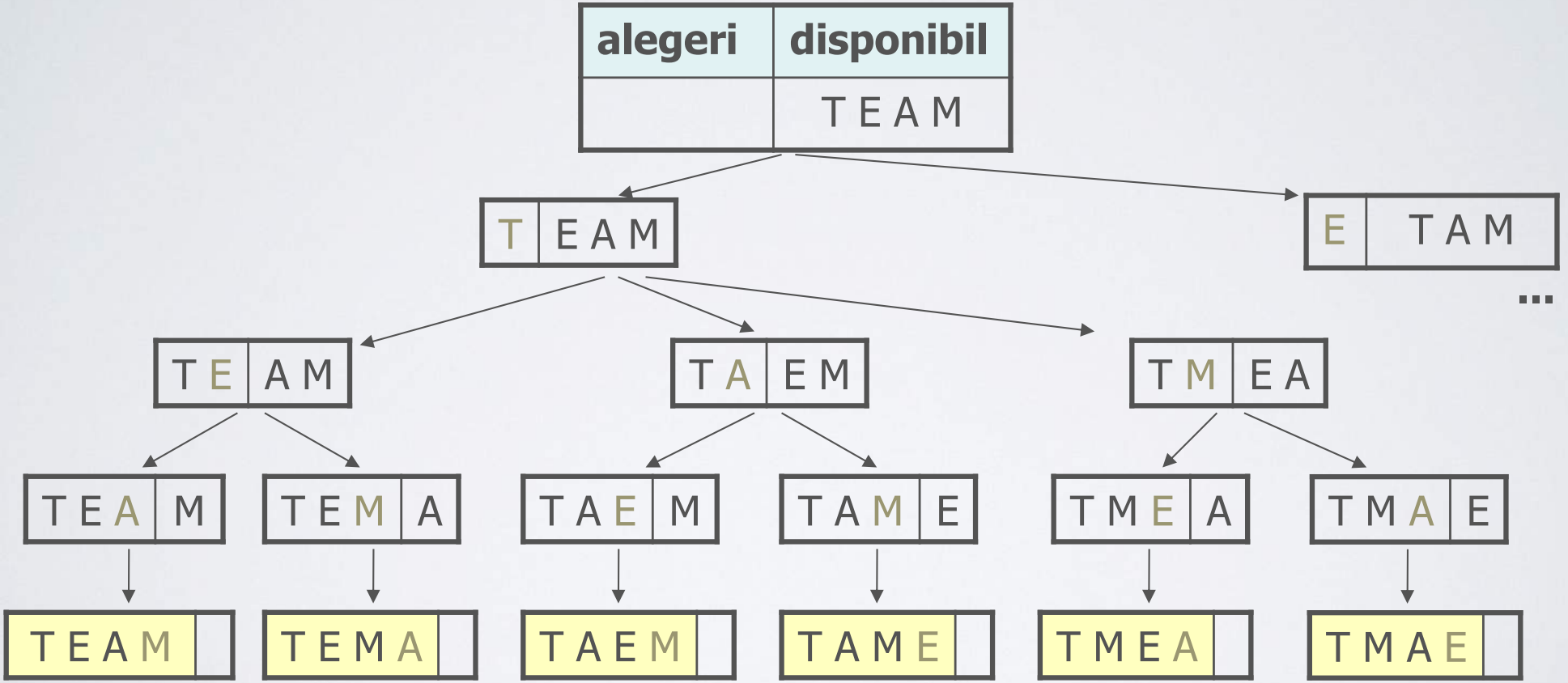
- Pentru sirul de caractere: TEAM se obtin urmatoarele permutari:

TEAM	ATEM
TEMA	ATME
TAEM	AETM
TAME	AEMT
TMEA	AMTE
TMAE	AMET
ETAM	MTEA
ETMA	MTAE
EATM	META
EAMT	MEAT
EMTA	MATE
EMAT	MAET

Examinarea problemei:

ARBORELE DE DECIZIE PENTRU PROBLEMA PERMUTARILOR

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA



DISCUTIE: COMBINARI

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

- Exemplu:
  `combinari ("GOOGLE", 3)`
  va afisa urmatoarele:

EGL	LEG
EGO	LEO
ELG	LGE
ELO	LGO
EOG	LOE
EOL	LOG
GEL	OEG
GEO	OEL
GLE	OGE
GLO	OGL
GOE	OLE
GOL	OLG