



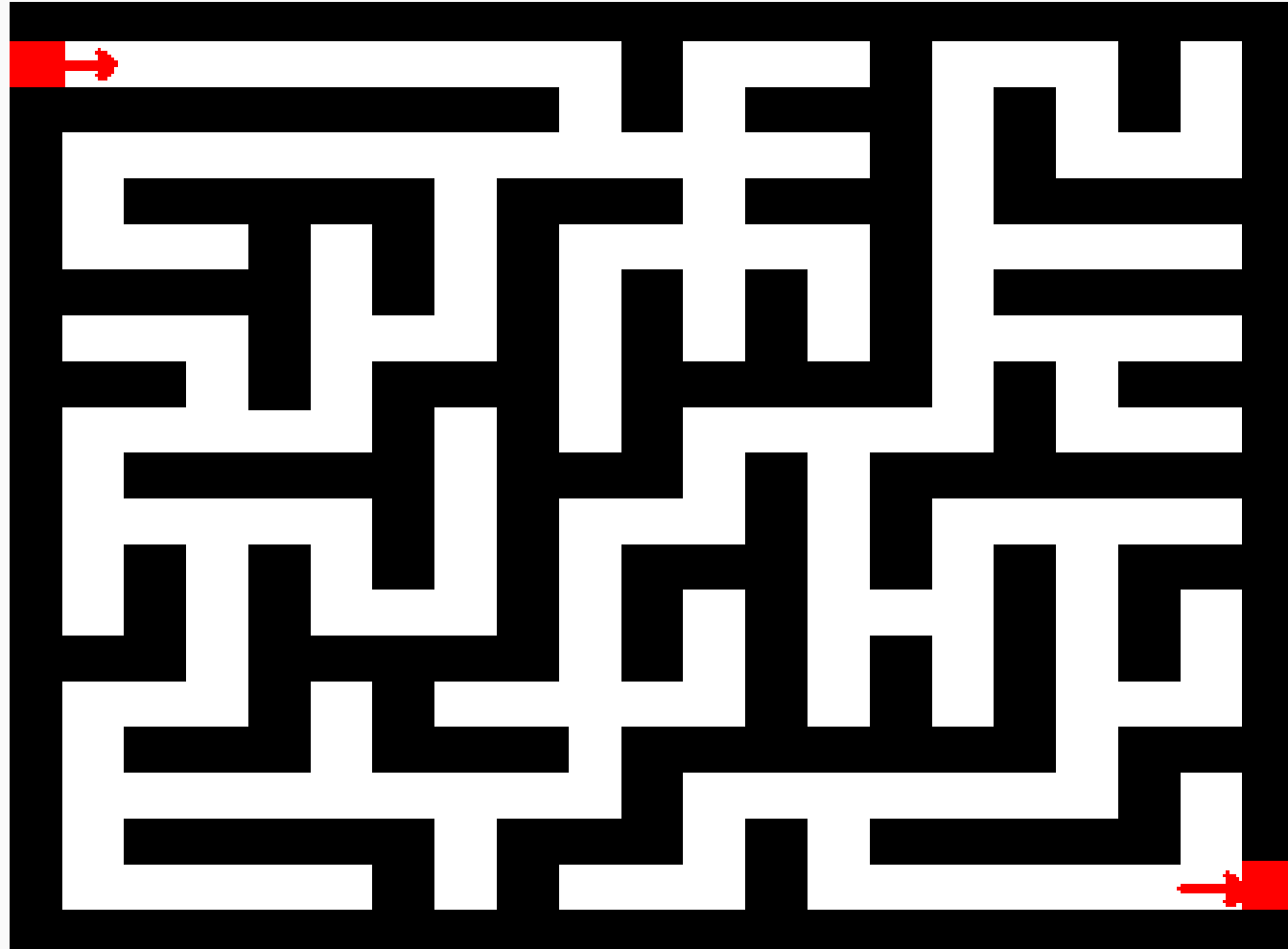
# Structuri de date și algoritmi Backtracking

Lect. Dr. Ing. Cristian Chilipirea – [cristian.chilipirea@mta.ro](mailto:cristian.chilipirea@mta.ro)





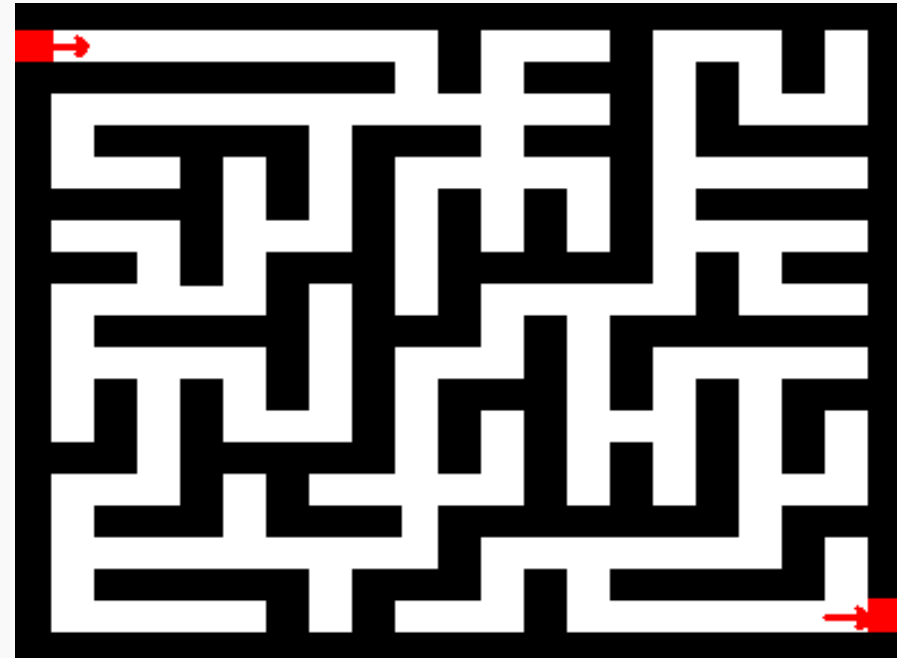
# Rezolvarea unui labirint





# Rezolvarea unui labirint

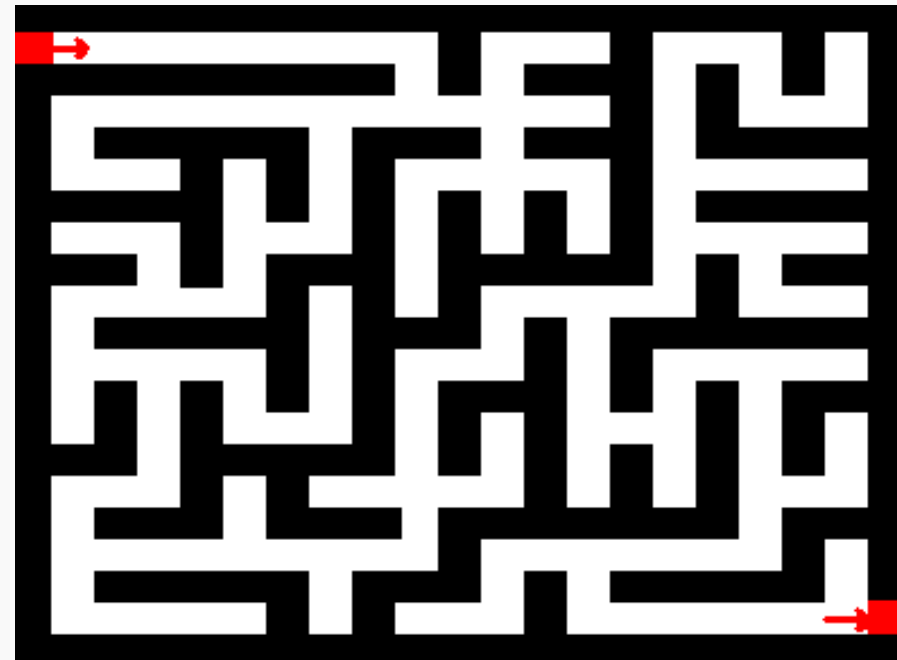
- Din orice poziție avem maxim 4 mutări:
  - SUS
  - JOS
  - STÂNGA
  - DREAPTA





# Rezolvarea unui labirint

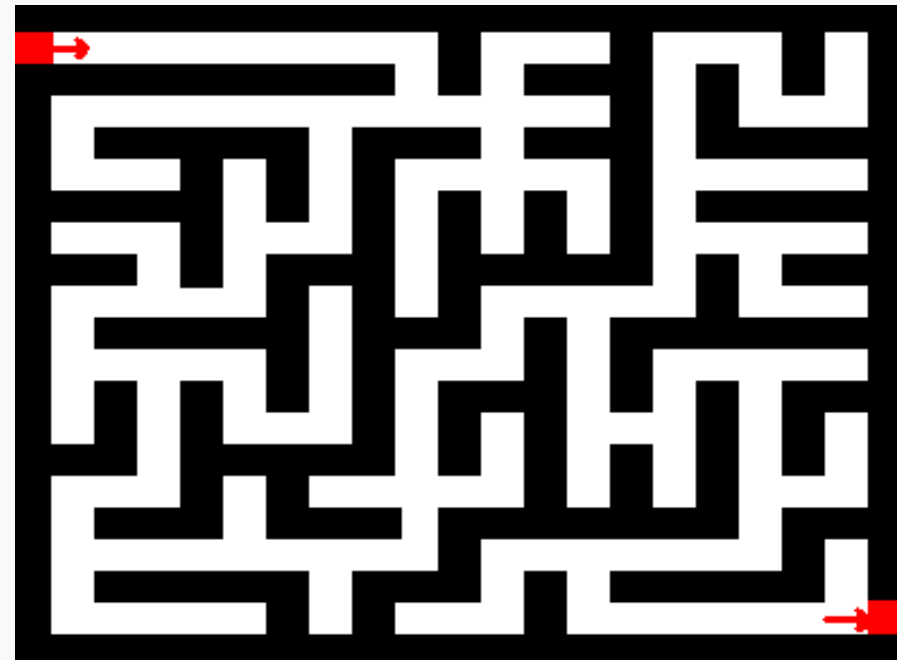
- Unele mutări evident nu pot fi făcute. Nu putem intra într-un zid.
- Nu dorim să trecem printr-o celulă de 2 ori.





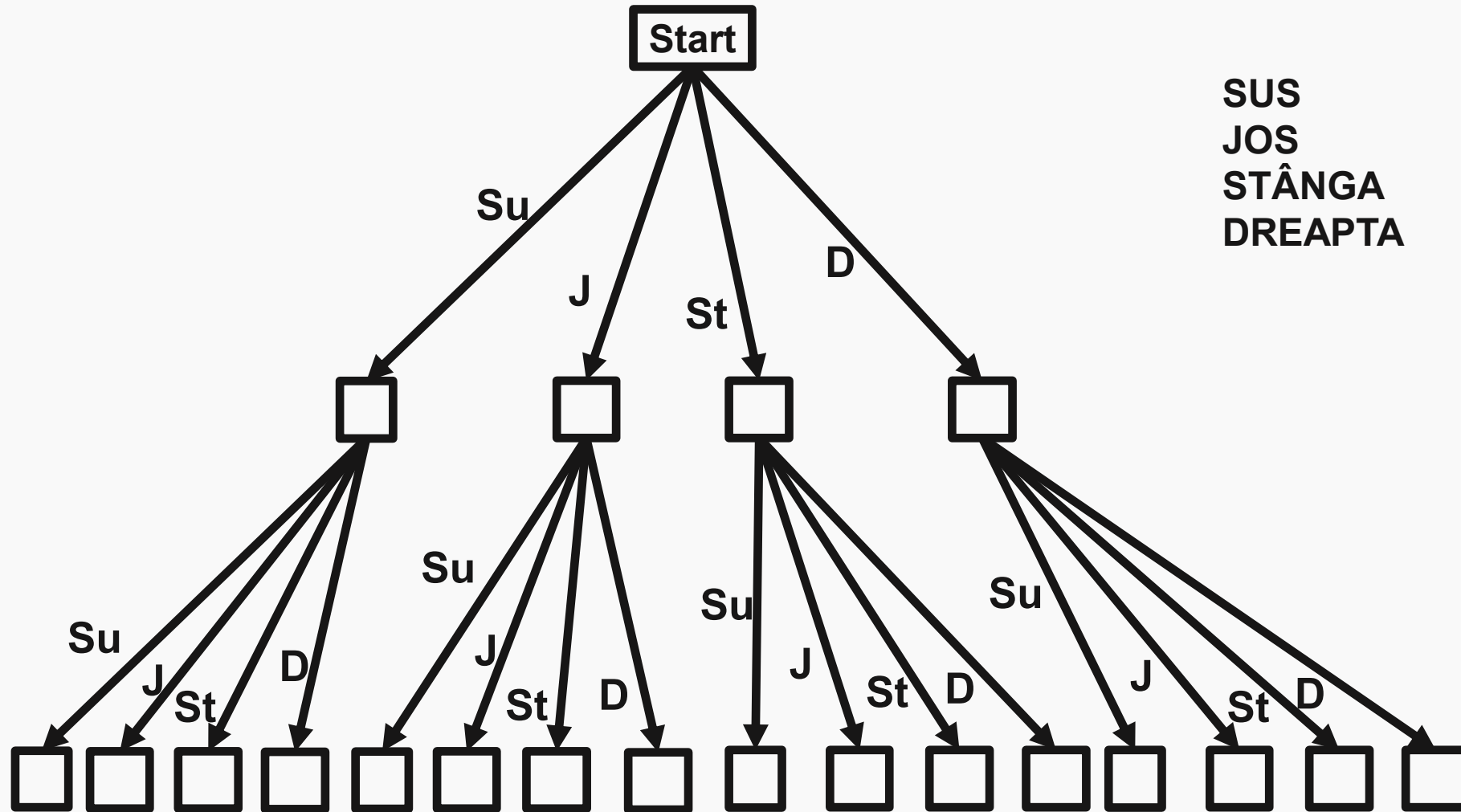
# Rezolvarea unui labirint

- La fiecare pas putem lua maxim 4 decizii.
  - Oricare ar fi decizia repetăm procesul până nu mai avem posibilitate de mișcare sau am ajuns la sfârșit.
- 
- Dacă reprezentăm toate deciziile avem un arbore.





# Rezolvarea unui labirint



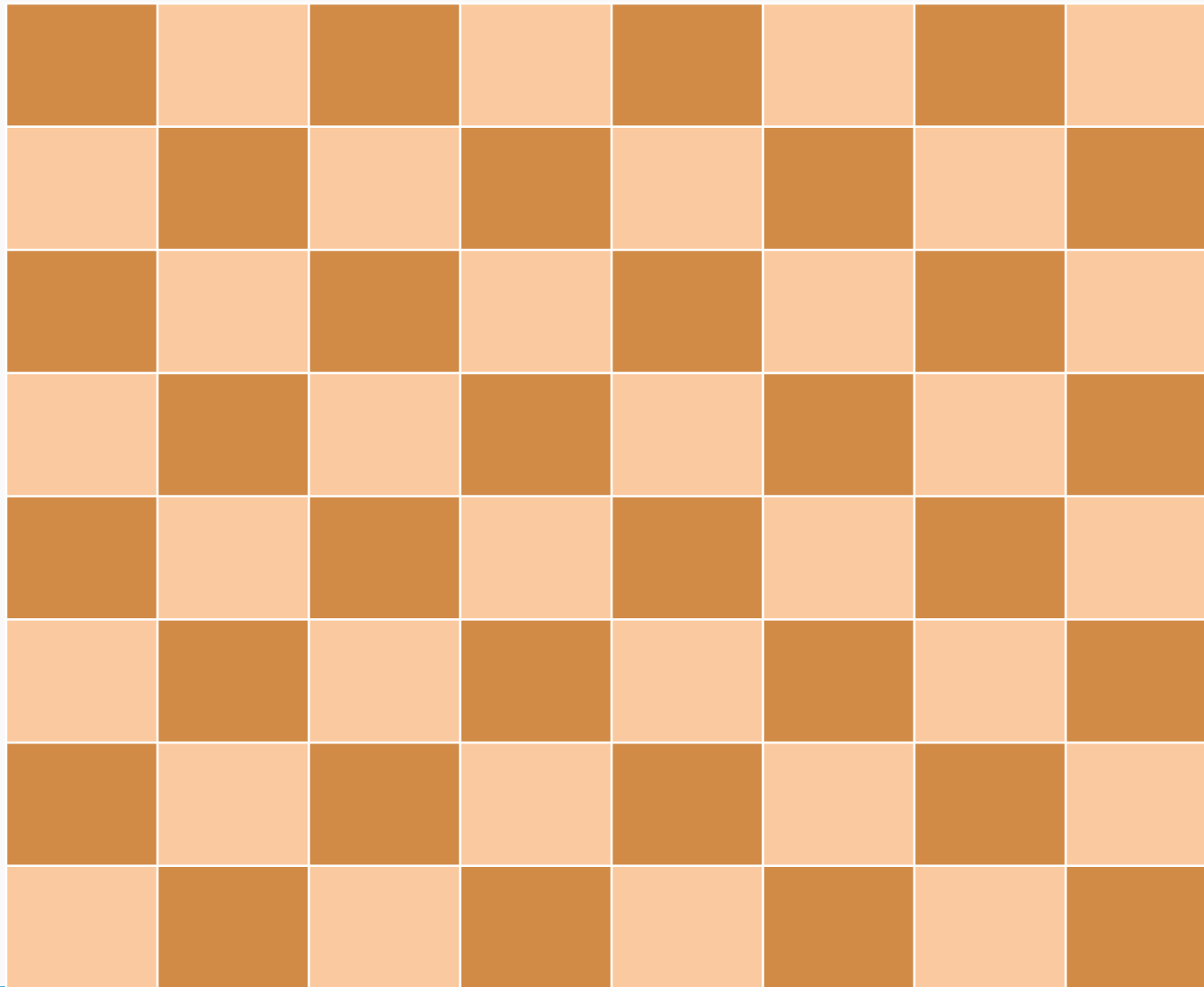






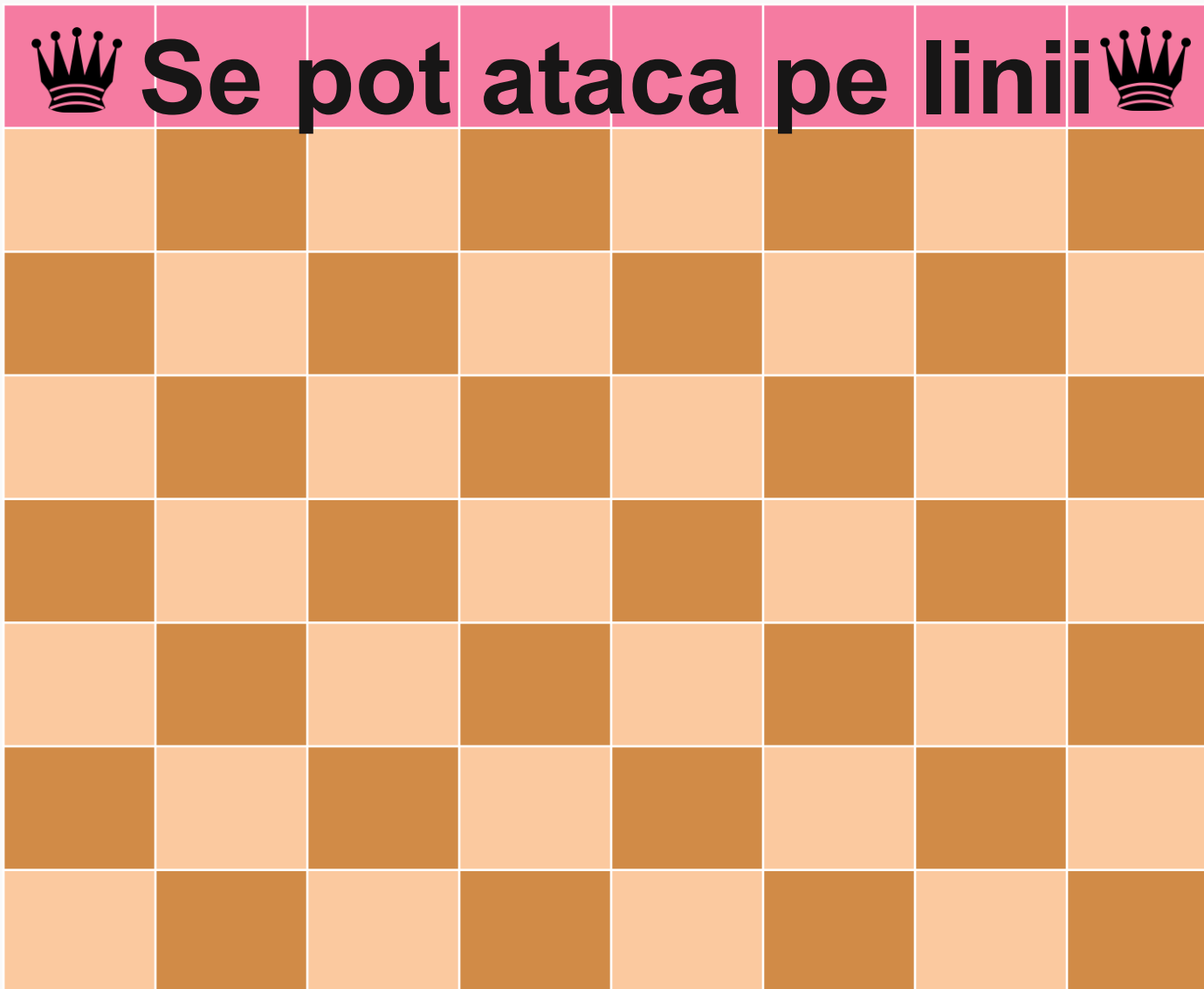


# Problema reginelor



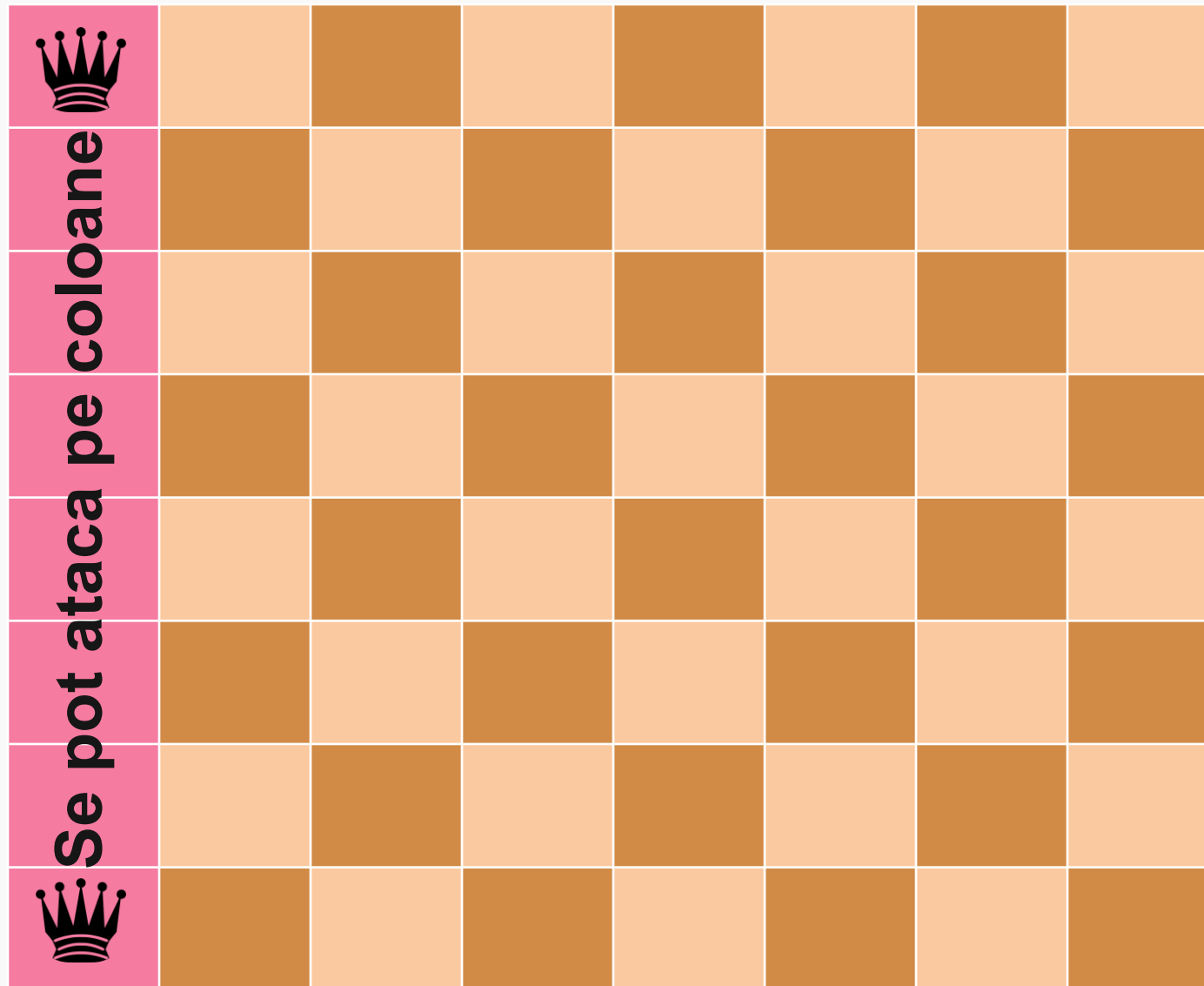


# Problema reginelor



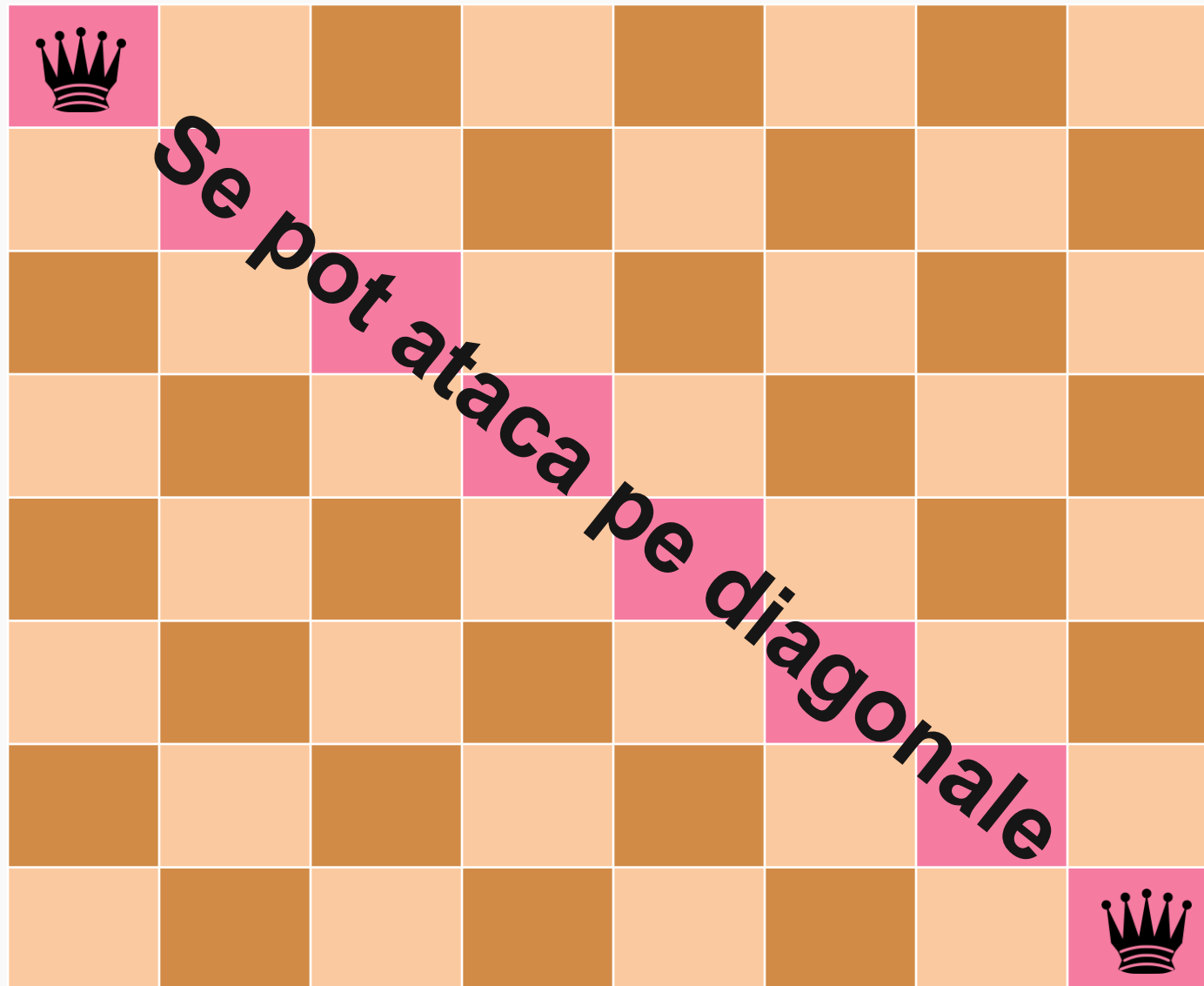


# Problema reginelor



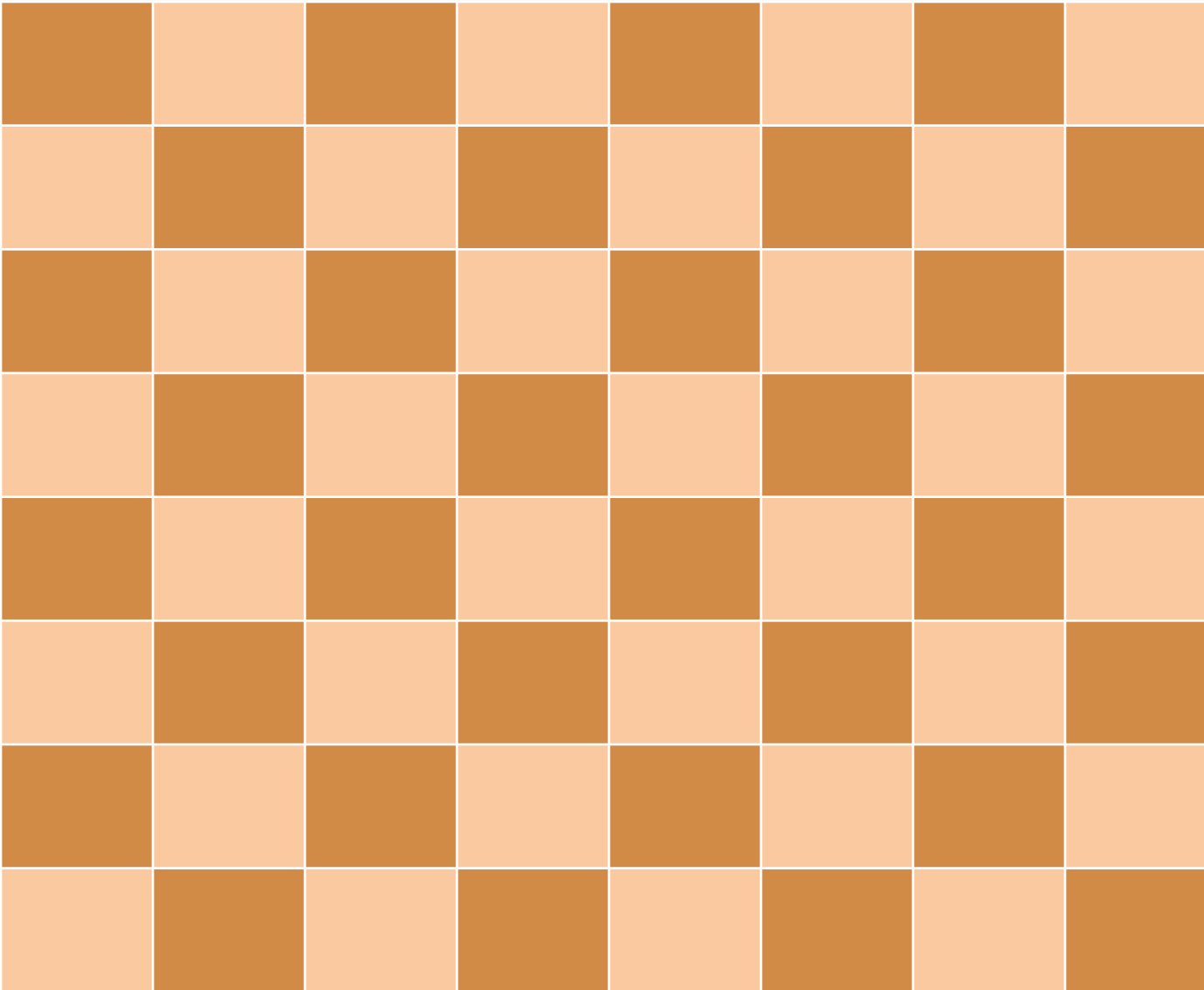


# Problema reginelor





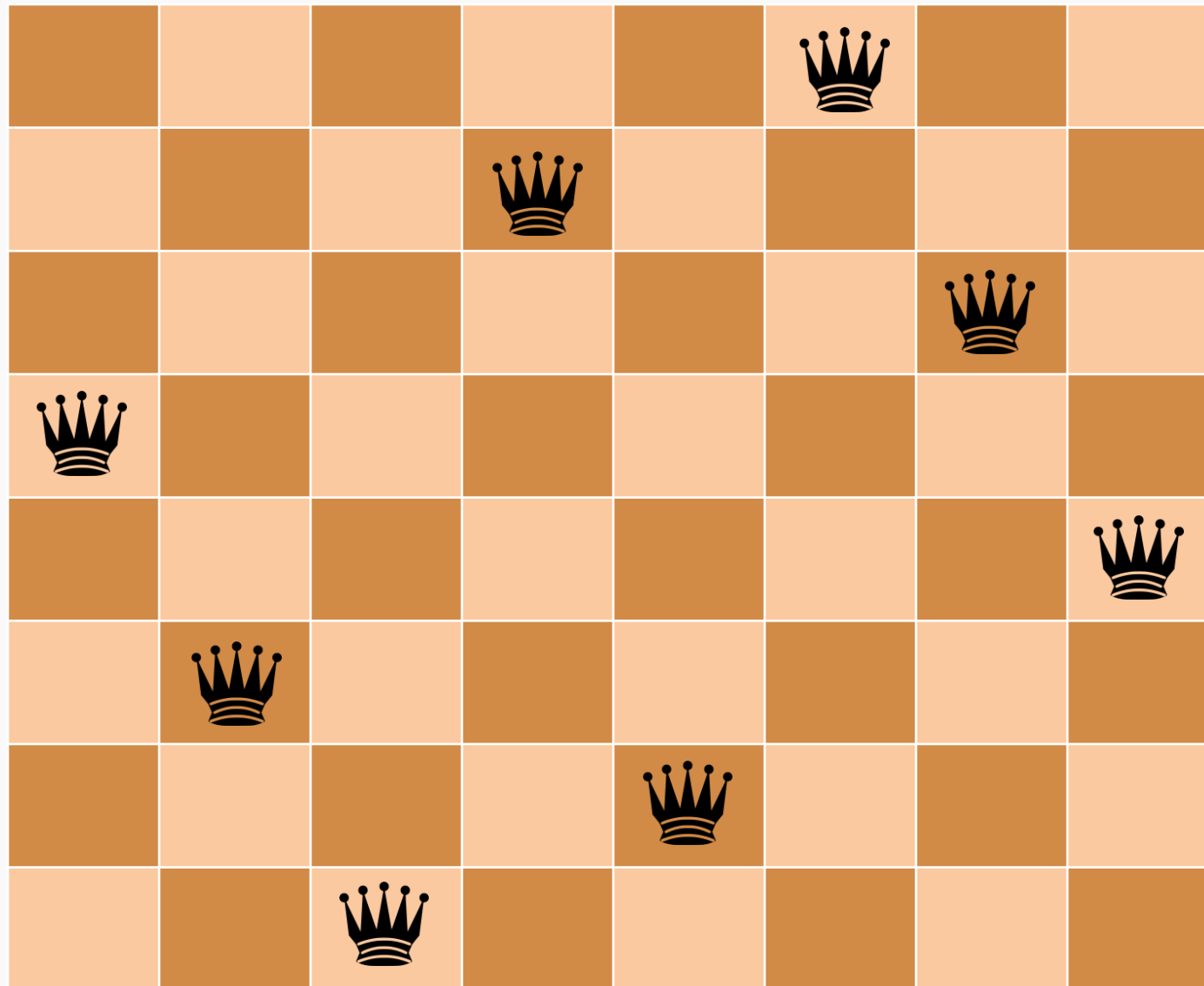
# Problema reginelor



**Vrem să  
așezăm 8  
regine  
care NU  
se pot  
ataca**

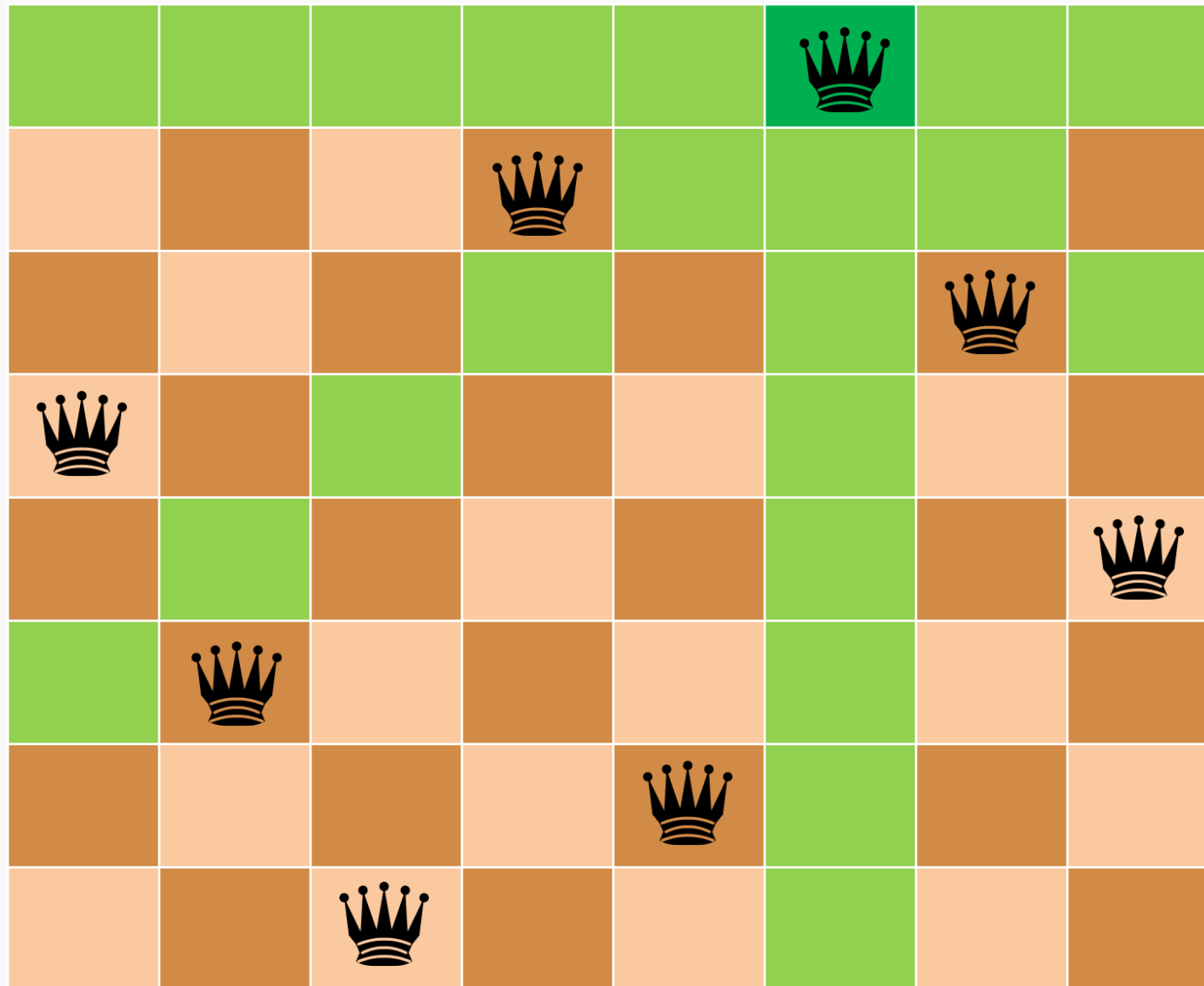


# Problema reginelor – soluție



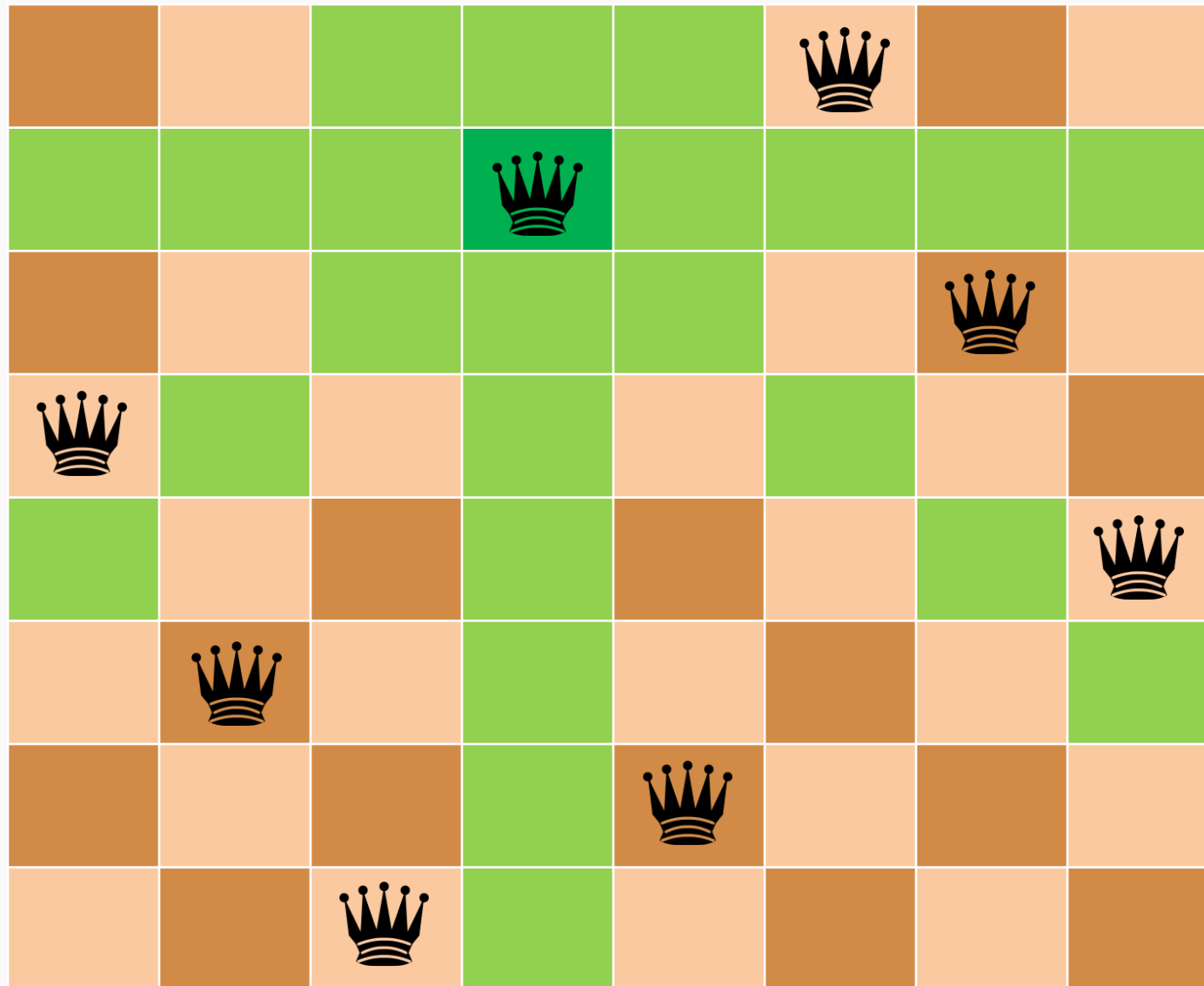


# Problema reginelor – soluție





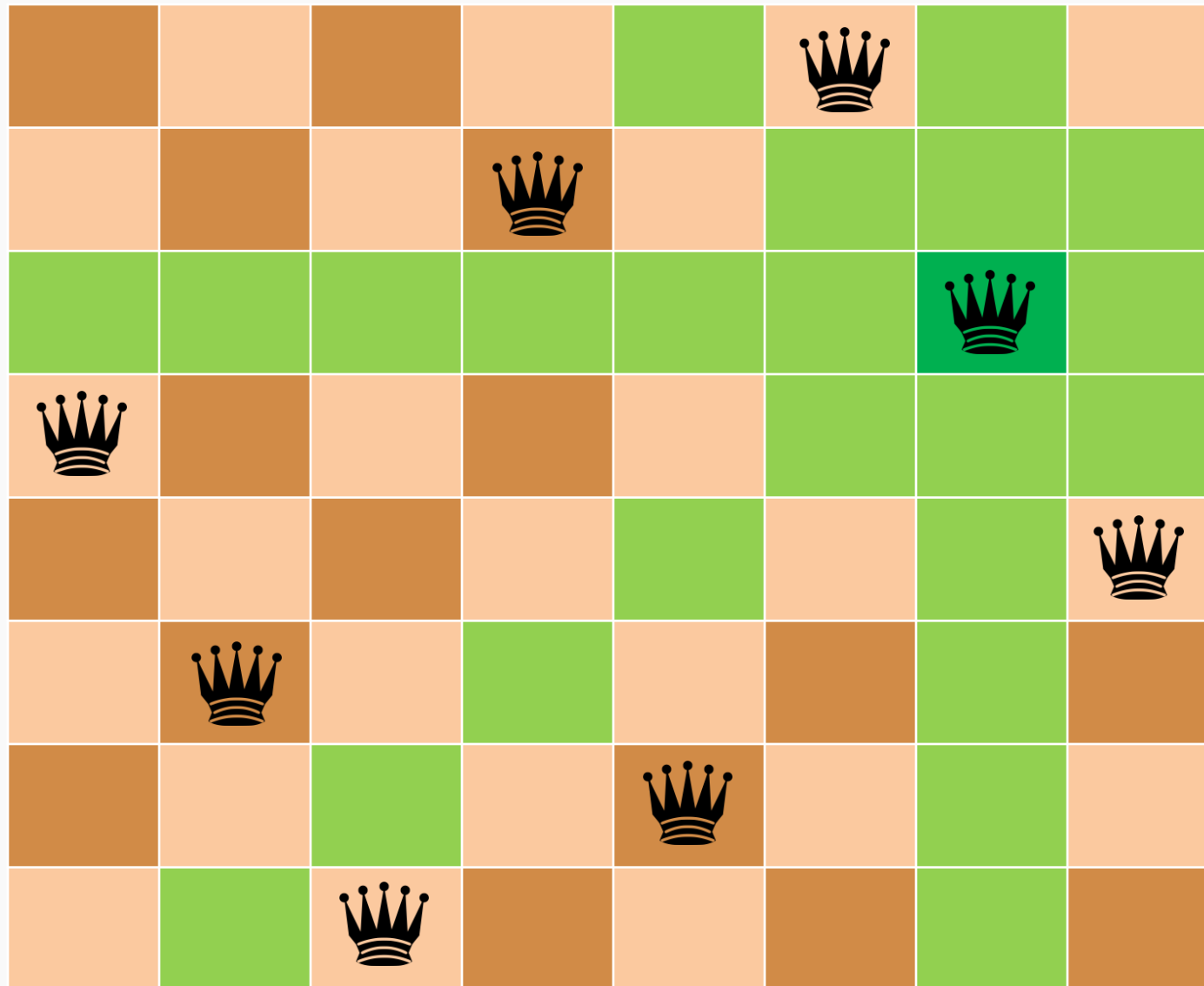
# Problema reginelor – soluție





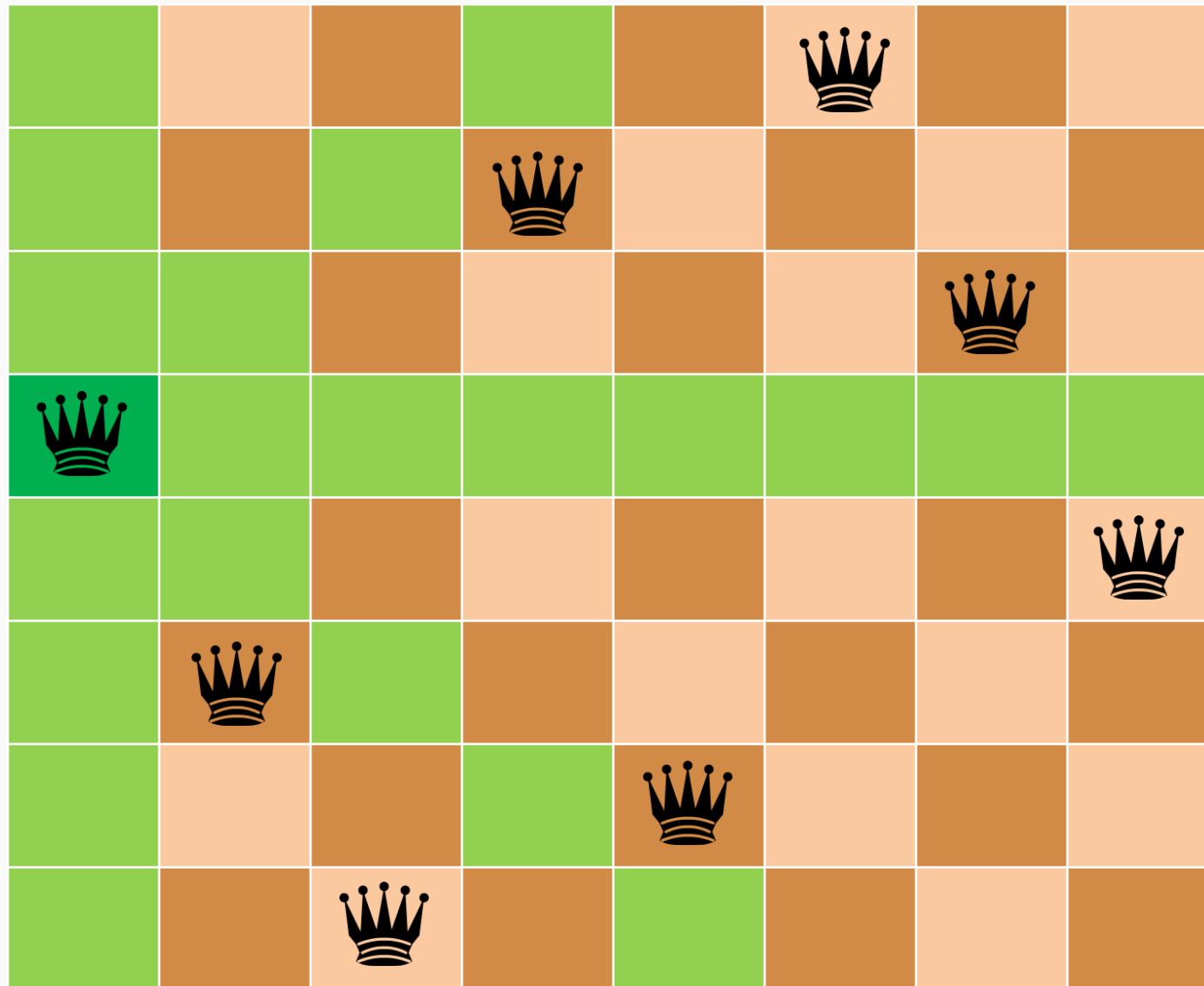


# Problema reginelor – soluție



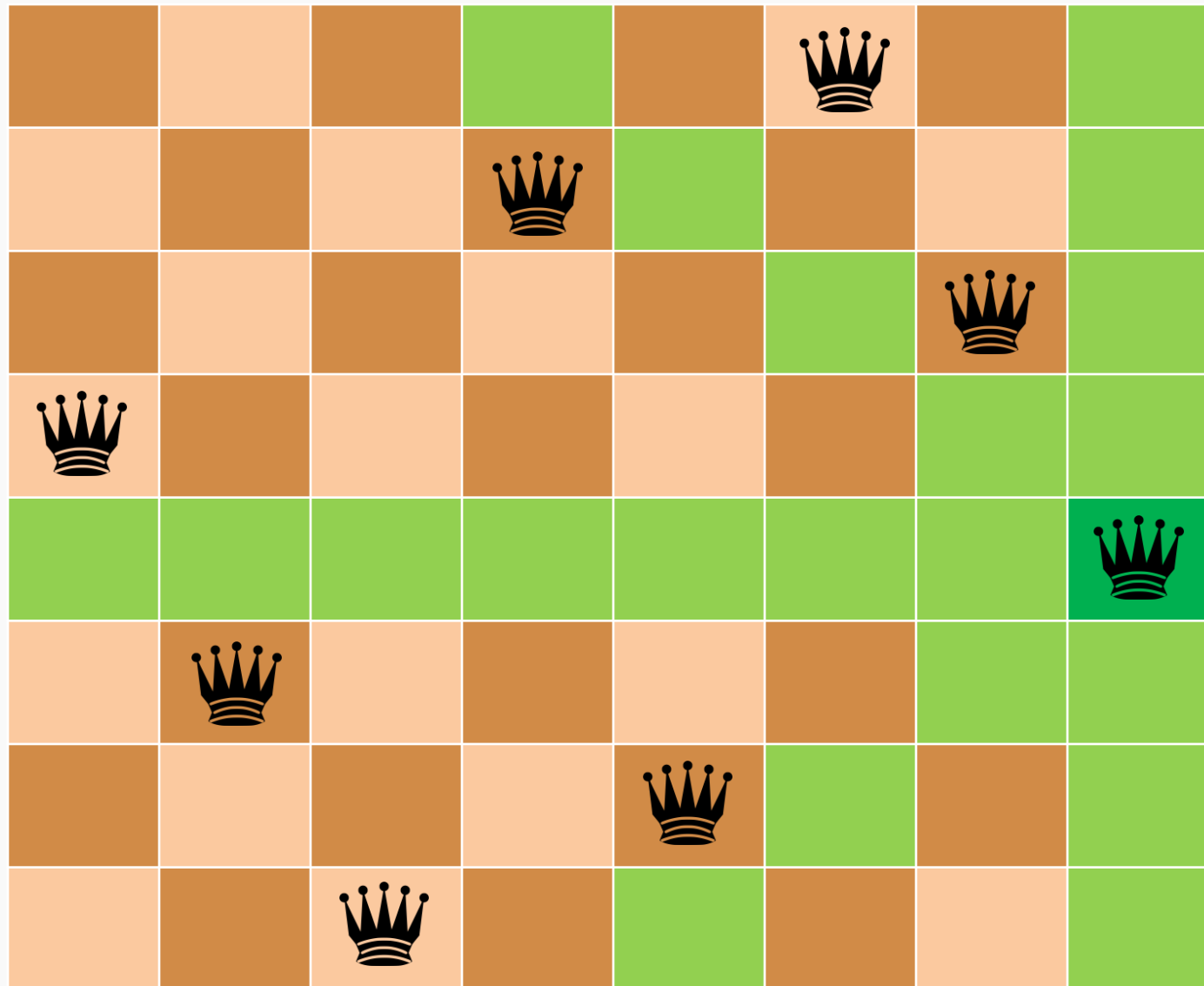


# Problema reginelor – soluție



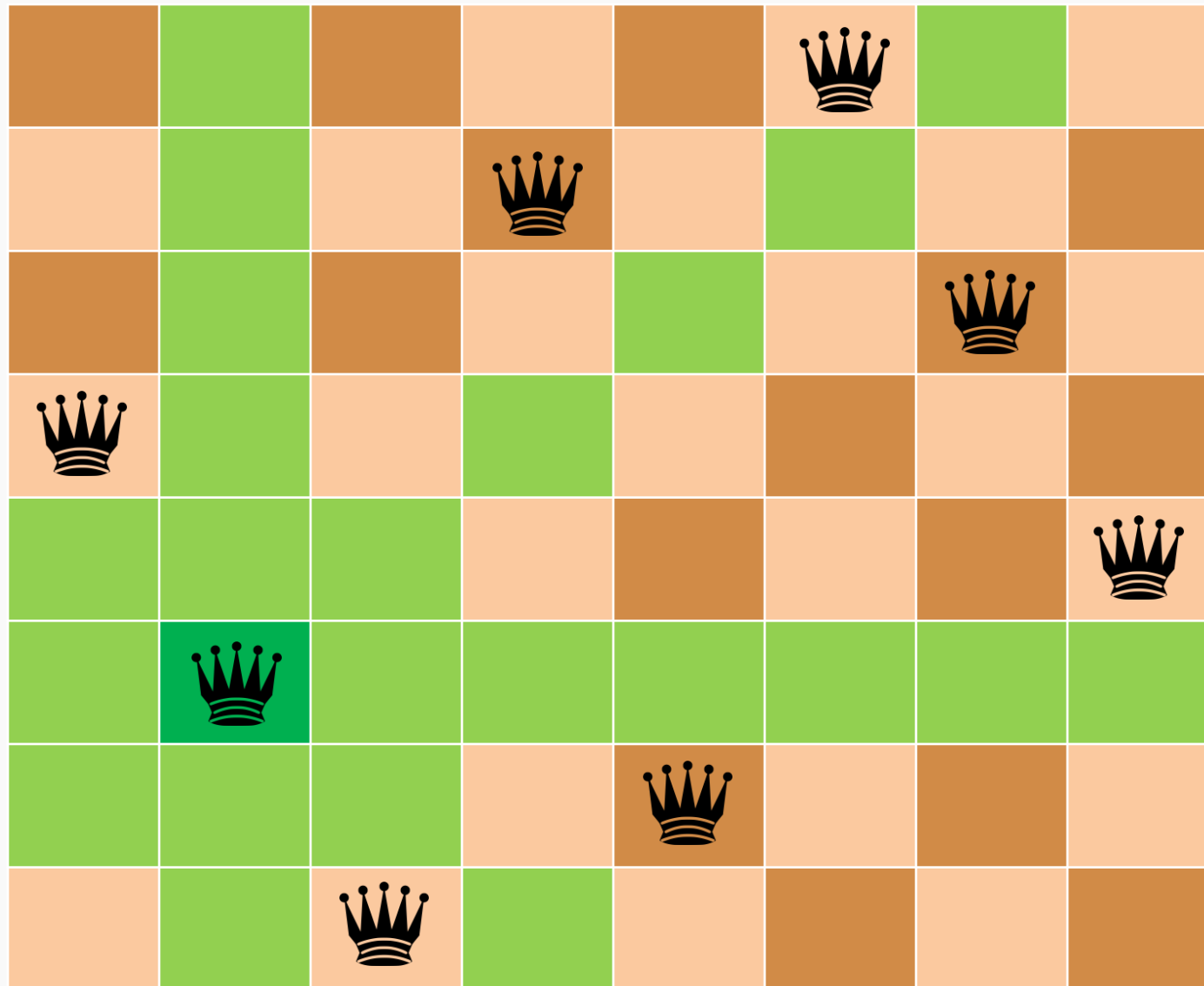


# Problema reginelor – soluție



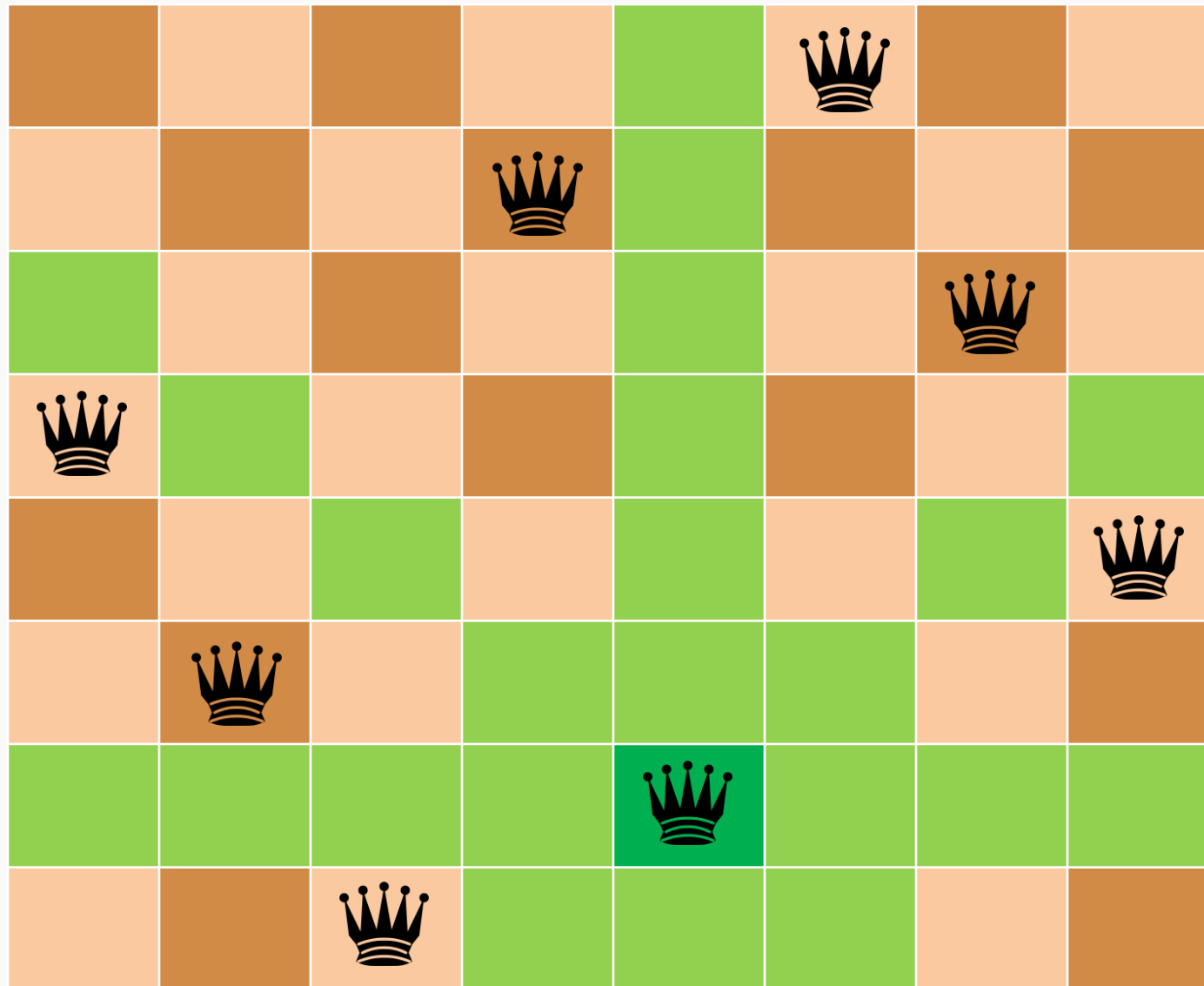


# Problema reginelor – soluție



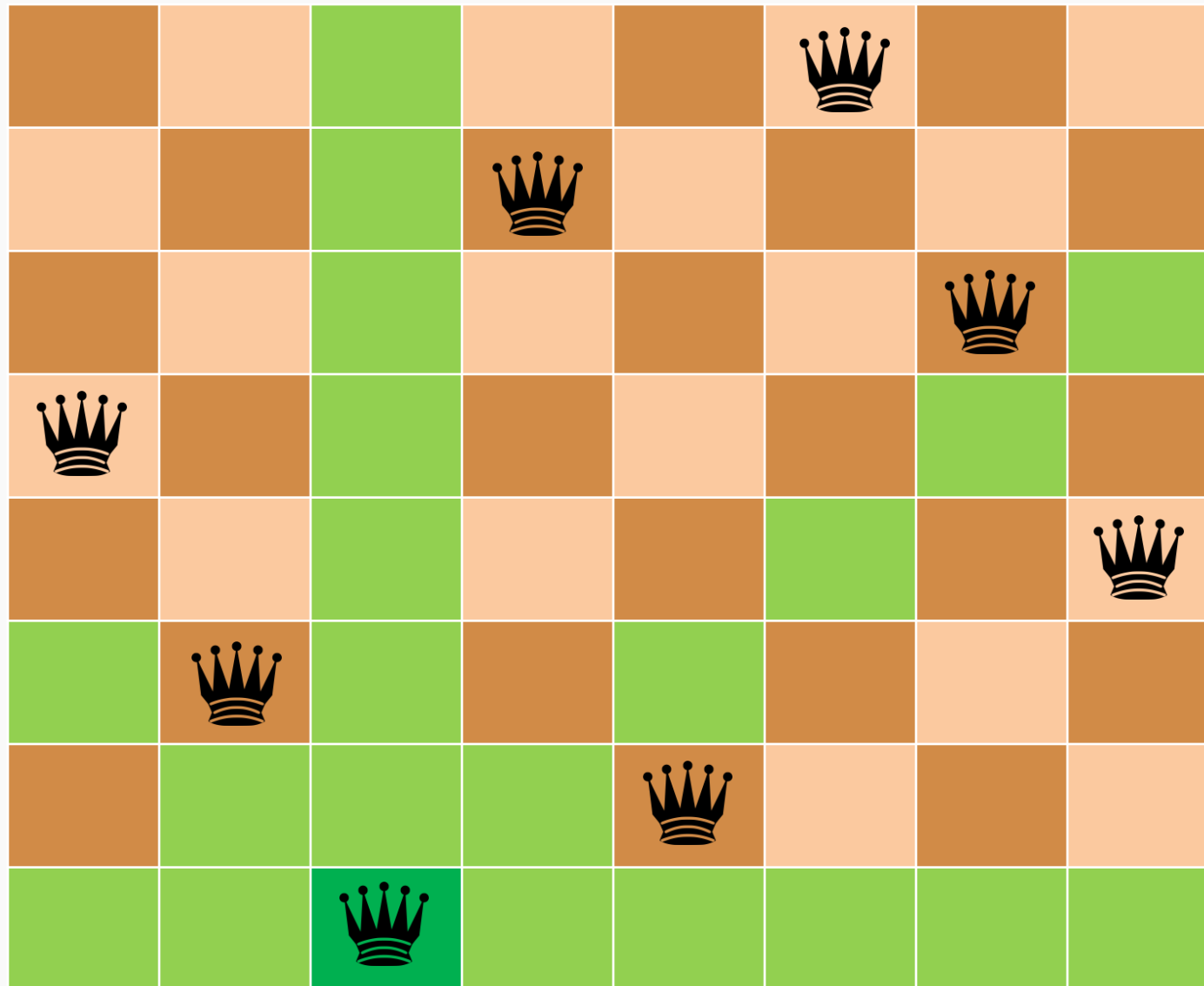


# Problema reginelor – soluție



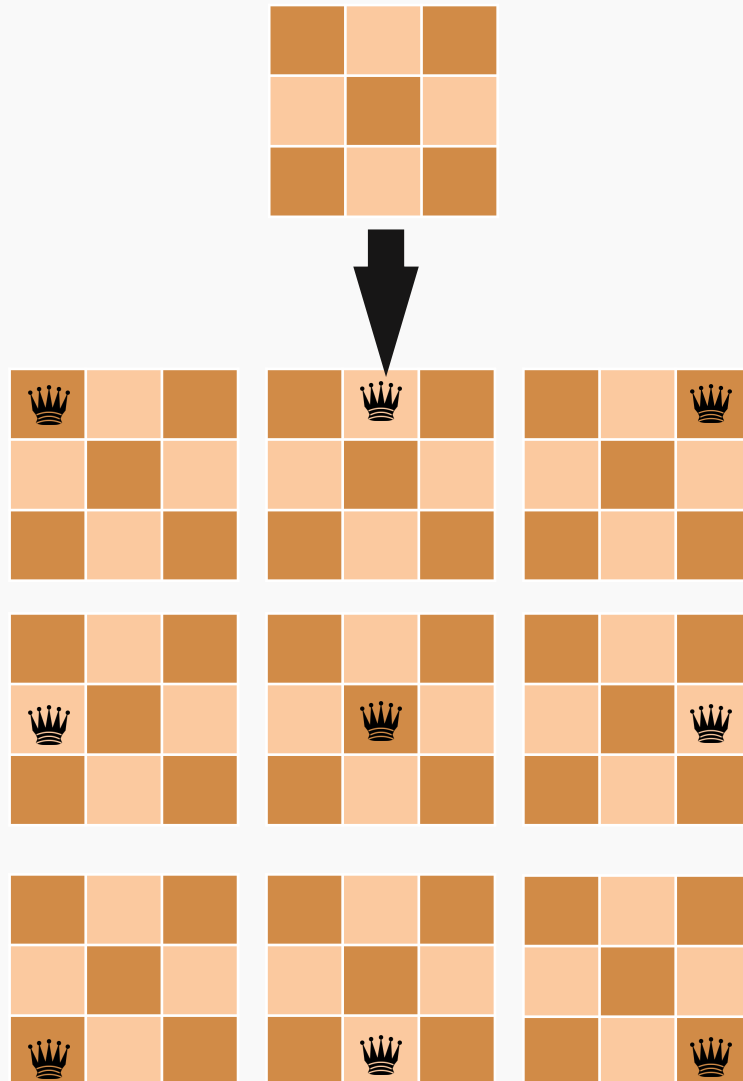


# Problema reginelor – soluție



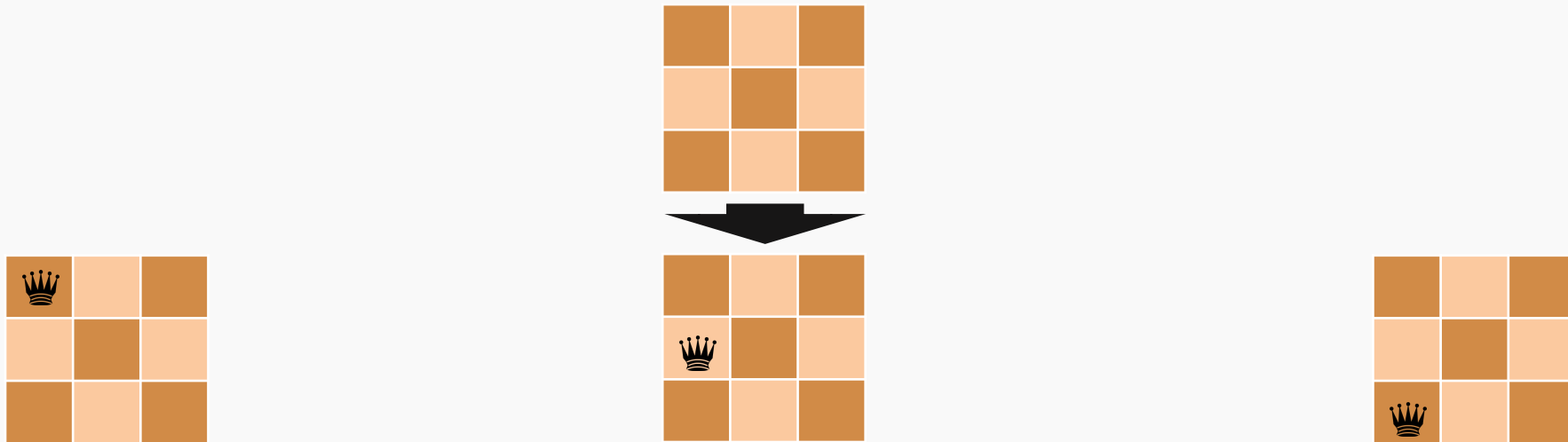


# Problema reginelor





# Problema reginelor

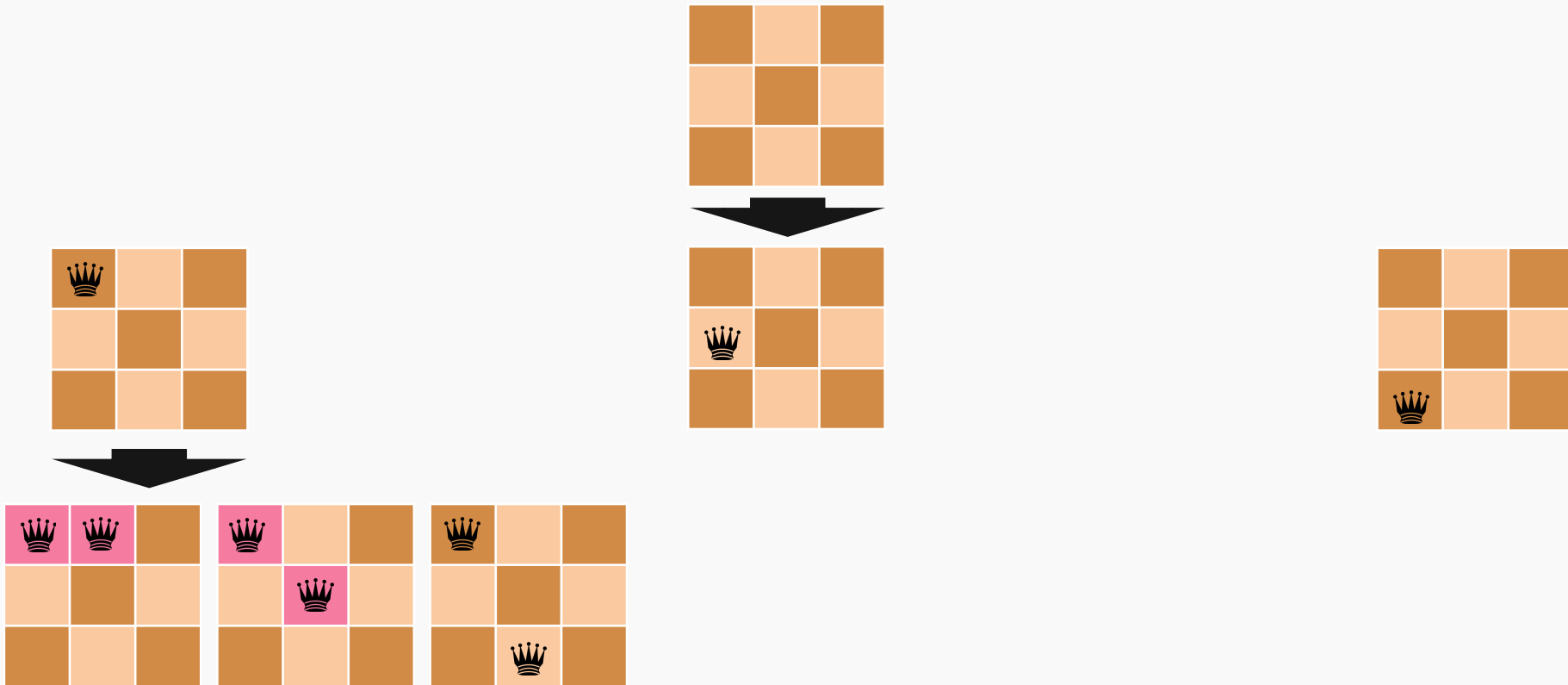


**Simplificare:** Știm că putem avea maxim o regină pe coloană și atâtea regine cât coloane. Putem astfel să completăm reginele coloană cu coloană.



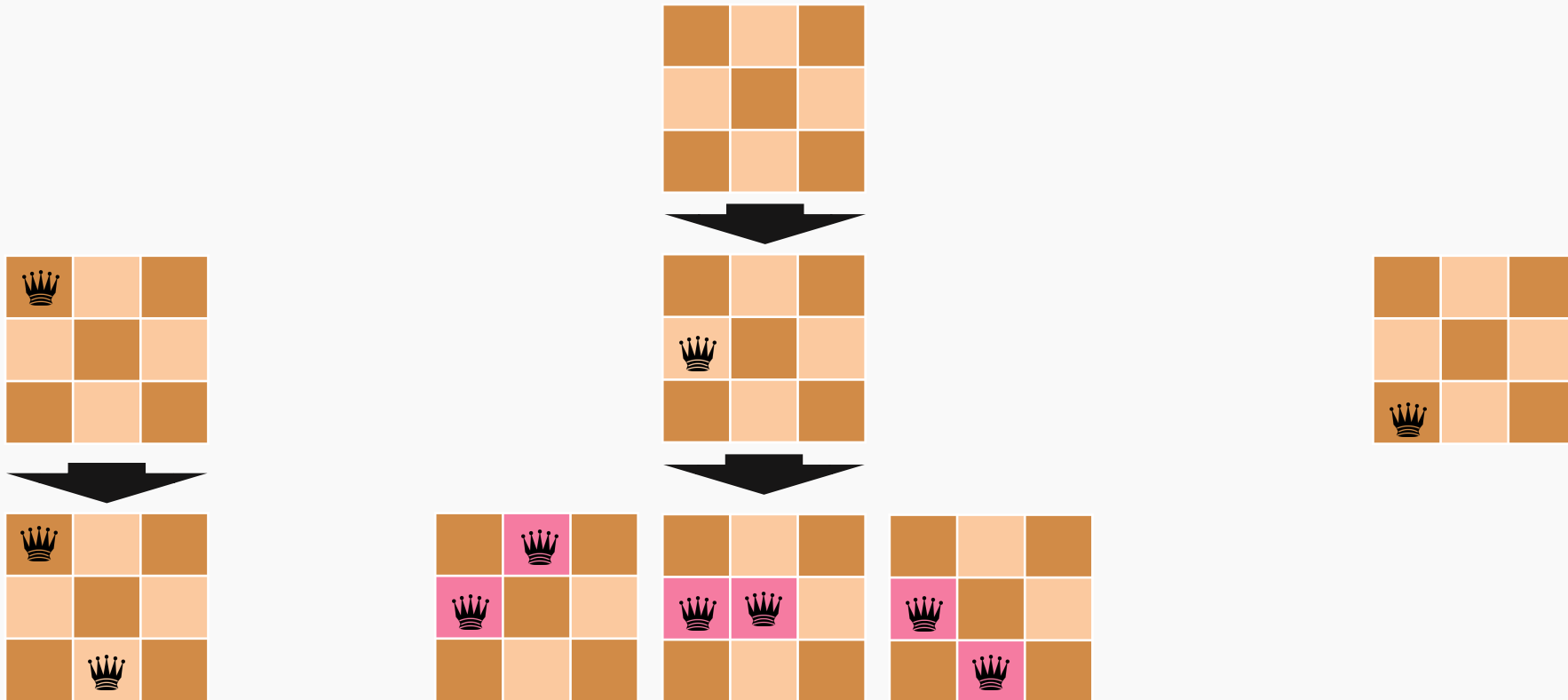


# Problema reginelor



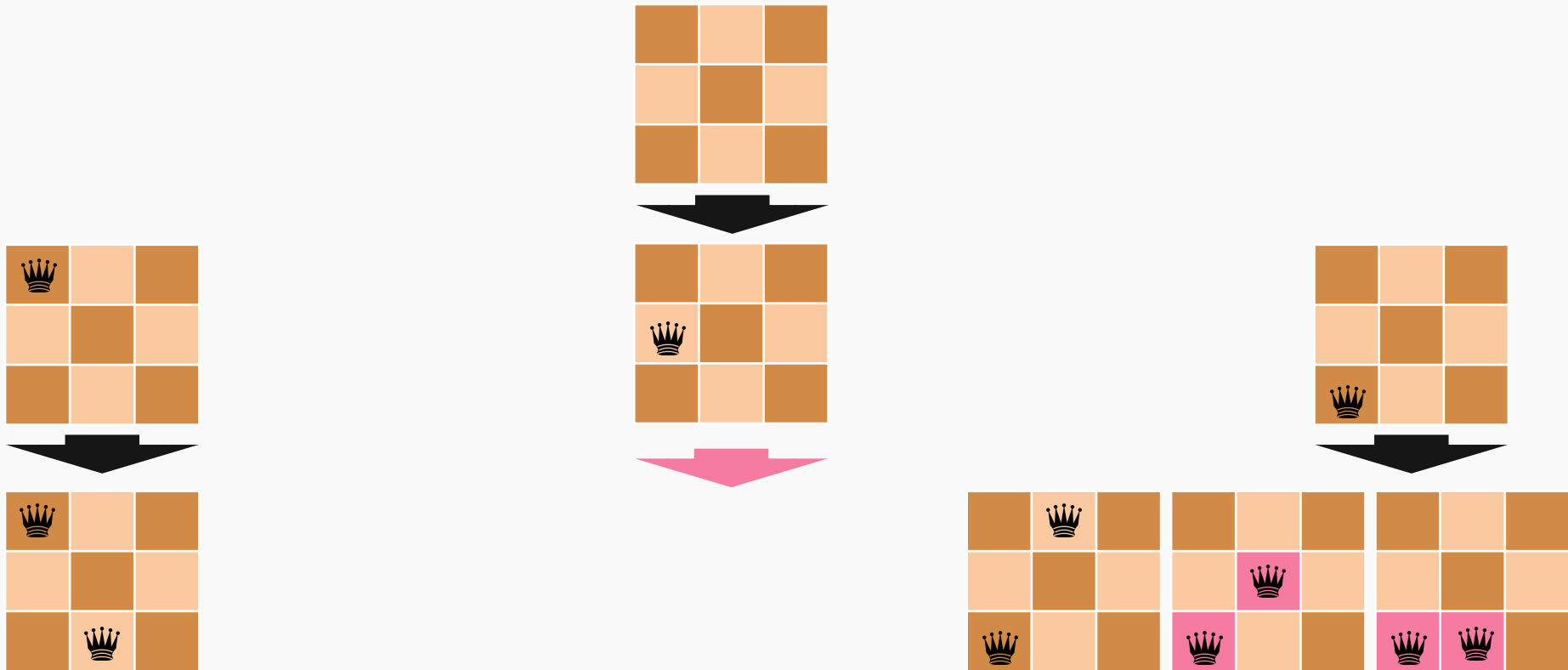


# Problema reginelor



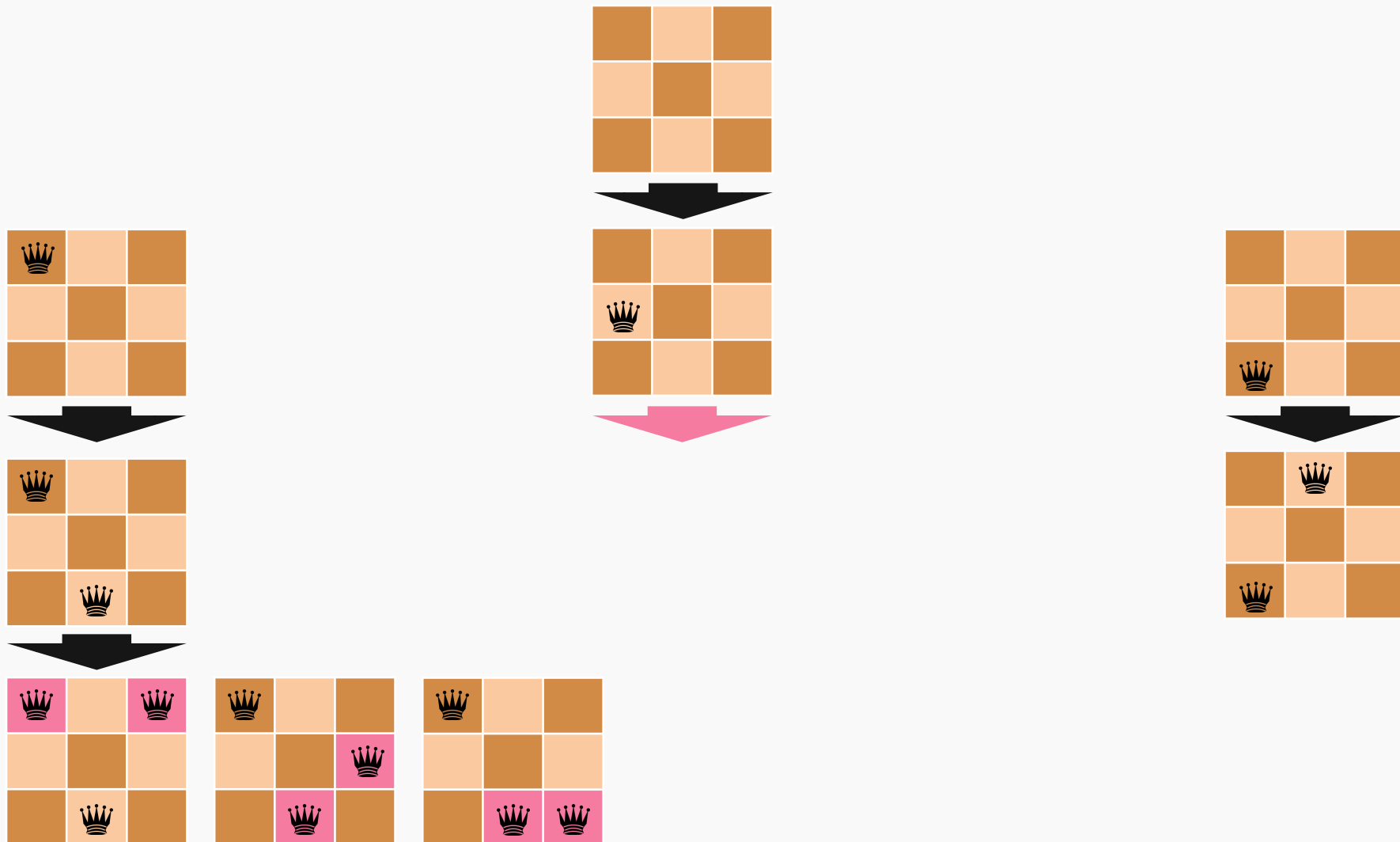


# Problema reginelor



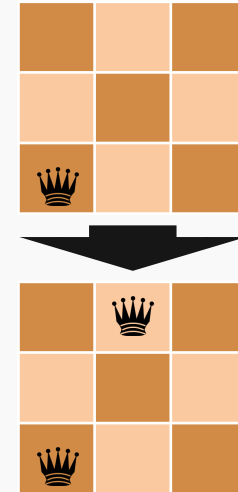
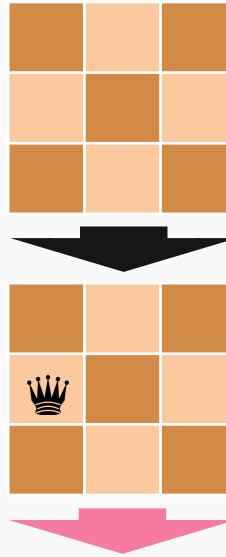
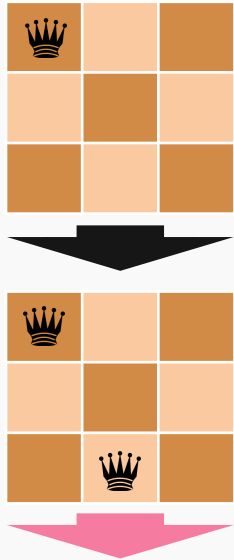


# Problema reginelor





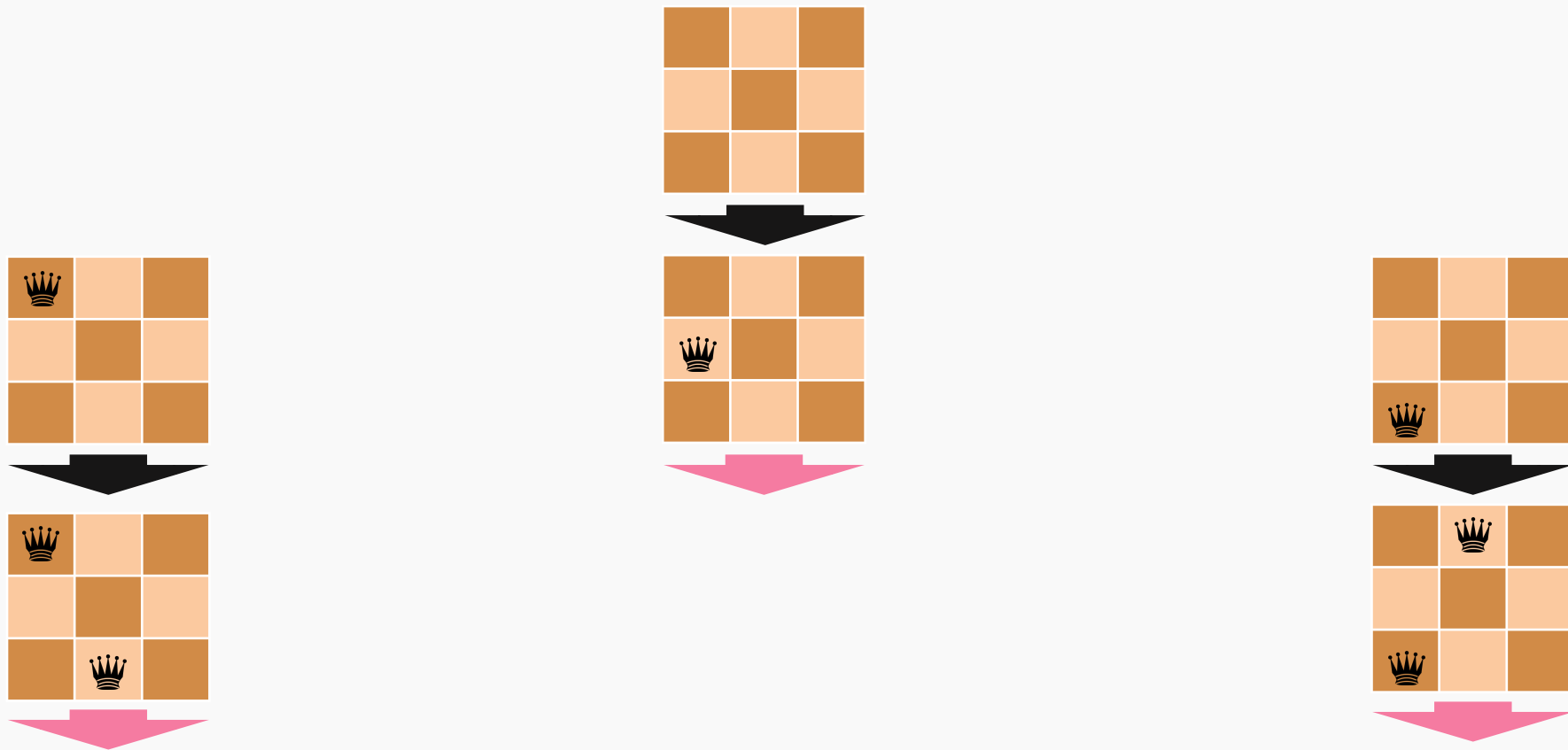
# Problema reginelor







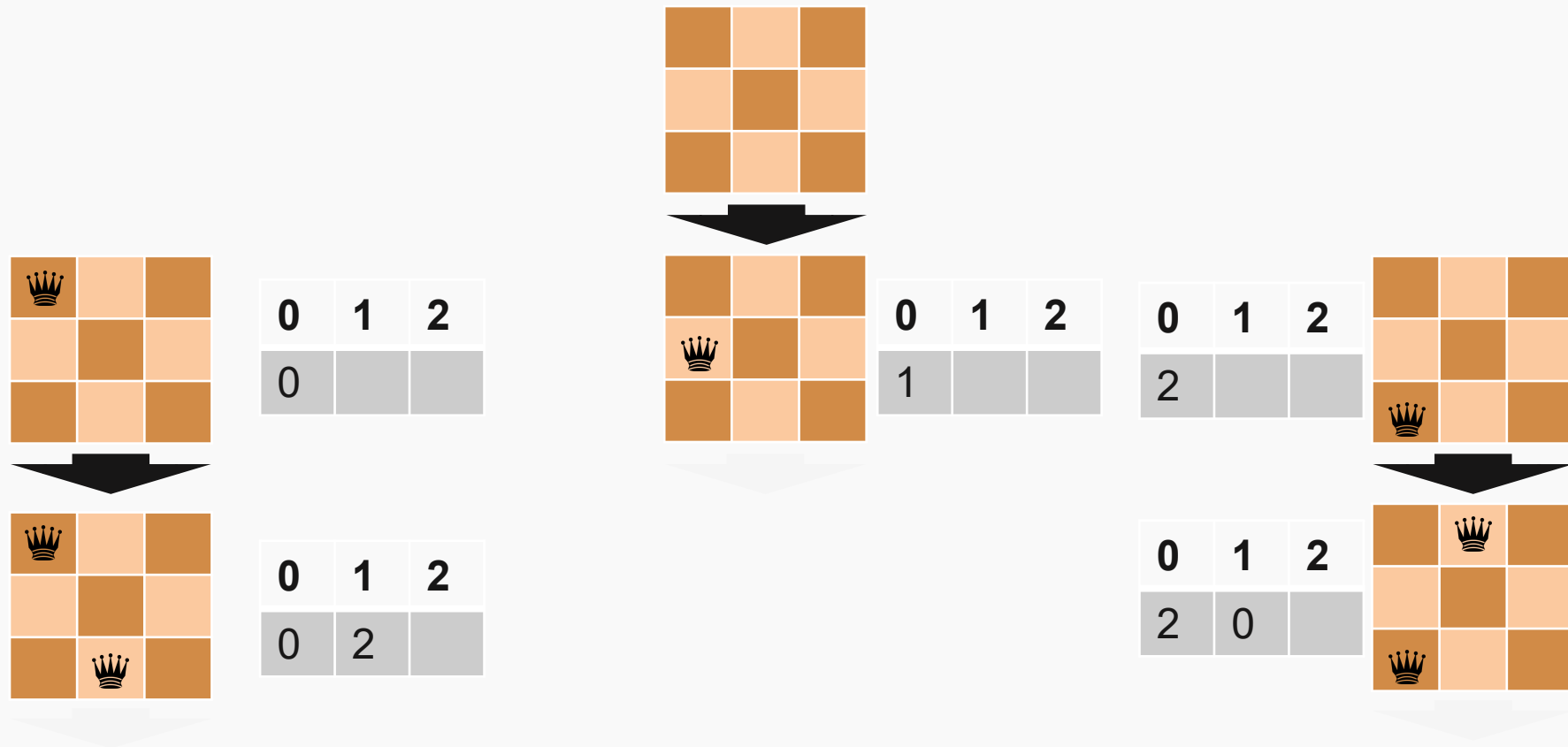
# Problema reginelor



**Am verificat tot arborele. Pentru 3 regine nu există soluție.**



# Problema reginelor

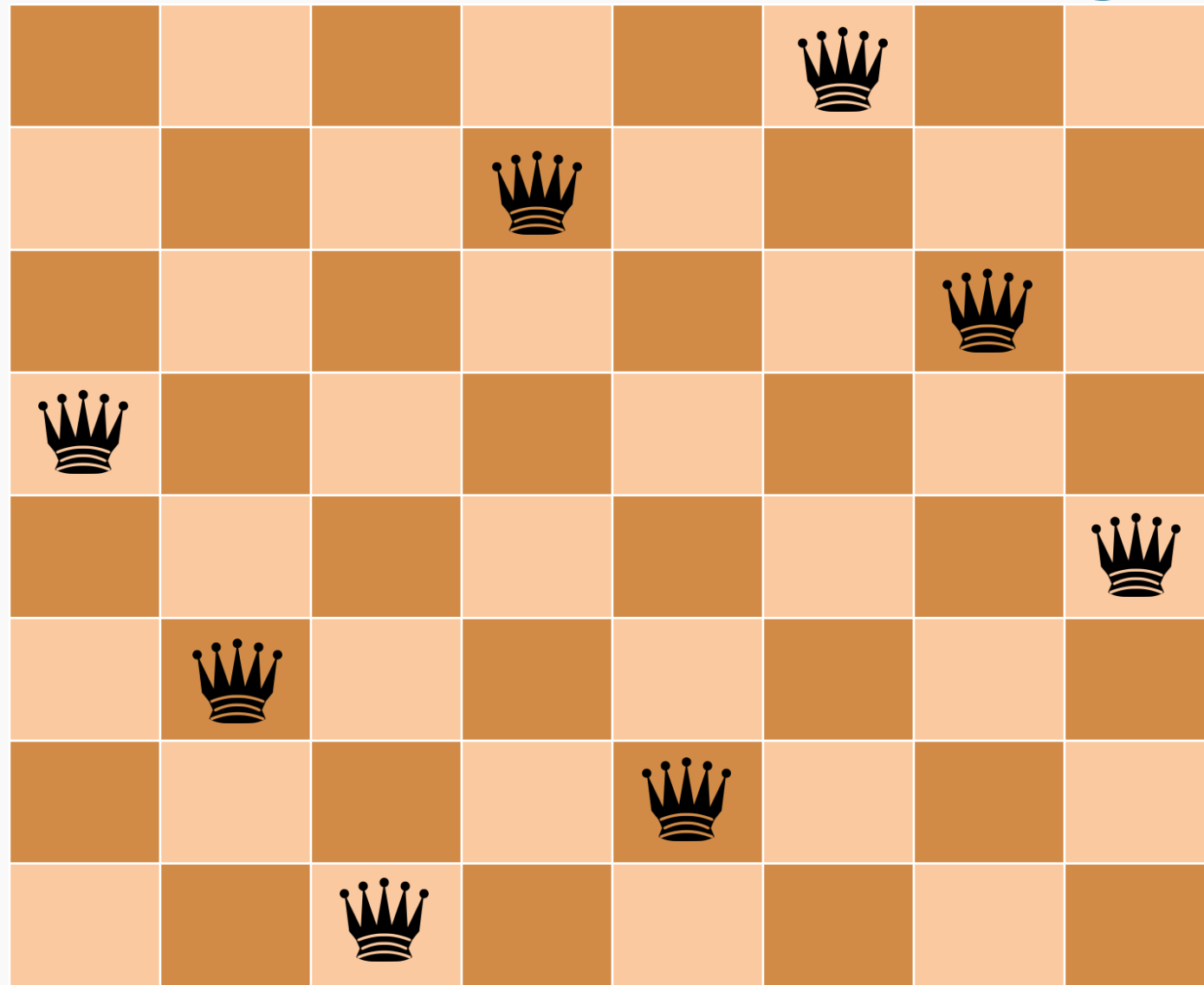


**Dacă avem doar o regină pe coloană putem să nu folosim o matrice ci un vector.  
Index-ul vectorului reprezintă coloana iar valoarea reprezintă linia.**





# Problema reginelor – soluție



0	1	2	3	4	5	6	7
3	5	7	1	6	0	2	4



# Modelarea problemelor

- Foarte multe soluții pentru probleme pot fi modelate ca un arbore.
  - Fiecare nod al arborelui reprezintă o soluție parțială.
  - Cu cât mergem mai adânc, mai jos, în arbore soluția crește.
  - Unele frunze pot fi soluții.
- 
- E important ca verificarea soluției să se poate face mai rapid decât găsirea ei.
- 
- Dacă putem construi un astfel de arbore putem găsi soluții printr-o simplă parcurgere a sa.



# Reminder DFS

```
int visited[N] = { 0 };  
void DFS(vertex currentNode) {  
    visited[currentNode.name] = 1;  
    for (int i = 0; i < currentNode.numNeighbors; i++) {  
        if (!visited[currentNode.neighbors[i]->name])  
            DFS(*(currentNode.neighbors[i]));  
    }  
}
```



# Backtracking

```
void backtracking(partialSolution PS) {  
    if (canReject(PS))  
        return;  
    if (isSolution(PS))  
        printSolution(PS);  
    PS = increaseStep(PS);  
    while (hasChoiceAtStep(PS)) {  
        PS = getNextChoiceAtStep(PS);  
        backtracking(PS);  
    }  
}
```



# De ce backtracking și nu DFS?

- Nu este necesar să ținem avem tot arborele, știind structura sa putem să lucrăm cu un singur nod la un moment dat.





# Problemă: generarea aranjamentelor

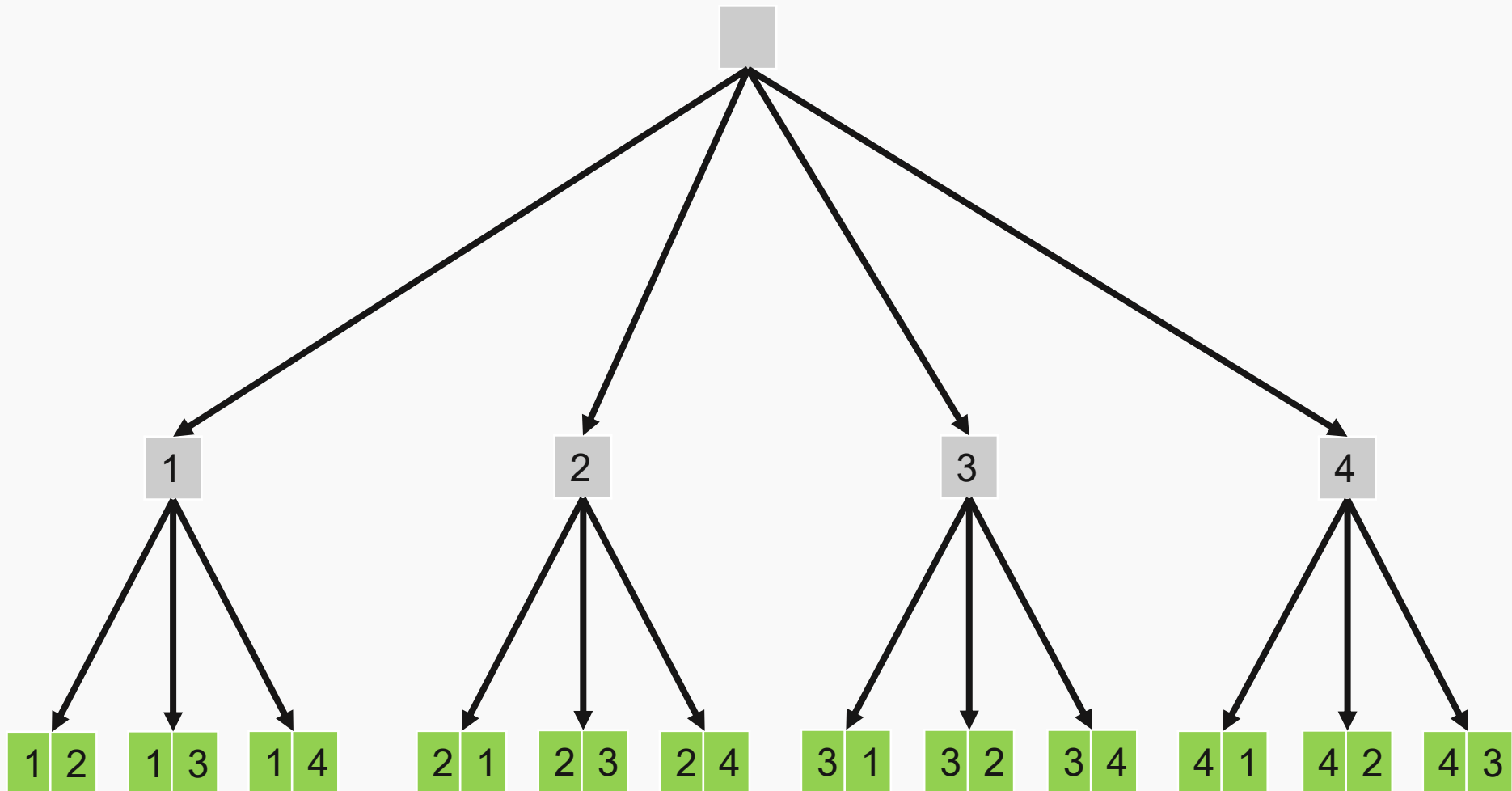
- Aranjamente de 4 luate câte 2:

$\{1,2\}, \{1,3\}, \{1,4\}, \{2,1\}, \{2,3\}, \{2,4\}, \{3,1\}, \{3,2\}, \{3,4\}, \{4,1\}, \{4,2\}, \{4,3\}$

- Aranjamente de  $n$  luate câte  $k$  sunt toate submulțimile de  $k$  elemente :  $\{x_1, x_2, \dots, x_k\}, x_i \neq x_j, x_i \in \{1, 2 \dots n\}$



# Problemă: generarea aranjamentelor







# Backtracking algorithm

```
void backtracking(partialSolution PS) {  
    if (canReject(PS))  
        return;  
    if (isSolution(PS))  
        printSolution(PS);  
    PS = increaseStep(PS);  
    while (hasChoiceAtStep(PS)) {  
        PS = getNextChoiceAtStep(PS);  
        backtracking(PS);  
    }  
}
```



# Problemă: generarea aranjamentelor

```
#define N 4
#define k 2

//ARANGEMENTS
typedef struct partialSolution {
    int originalVector[N] = { 0, 2, 4, 8 };
    int arrangement[k] = { -1,-1 };
    int step = -1;
    int choice = -1;
}partialSolution;

void printSolution(partialSolution PS) {
    for (int i = 0; i < k; i++)
        printf("%3i", PS.arrangement[i]);
    printf("\n");
}
```



# Backtracking algorithm

```
void backtracking(partialSolution PS) {  
    if (canReject(PS))  
        return;  
    if (isSolution(PS))  
        printSolution(PS);  
    PS = increaseStep(PS);  
    while (hasChoiceAtStep(PS)) {  
        PS = getNextChoiceAtStep(PS);  
        backtracking(PS);  
    }  
}
```



# Problemă: generarea aranjamentelor

```
partialSolution increaseStep(partialSolution PS) {  
    PS.step++;  
    PS.choice = 0;  
    return PS;  
}  
  
int hasChoiceAtStep(partialSolution PS) {  
    return PS.step < k && PS.choice < N;  
}  
  
partialSolution getNextChoiceAtStep(partialSolution PS) {  
    PS.arrangement[PS.step] = PS.originalVector[PS.choice];  
    PS.choice++;  
    return PS;  
}
```



# Problemă: generarea aranjamentelor

```
int canReject(partialSolution PS) {  
    for (int i = 0; i < PS.step; i++) {  
        if (PS.arrangement[i] == PS.arrangement[PS.step])  
            return 1;  
    }  
    return 0;  
}  
  
int isSolution(partialSolution PS) {  
    return !canReject(PS) && PS.step == k-1;  
}
```



# Întrebări

- Cum scoatem o singură soluție? (Nu ne interesează care)
- Cum scoatem cea mai bună soluție după un criteriu?



# Problemă: generarea combinărilor

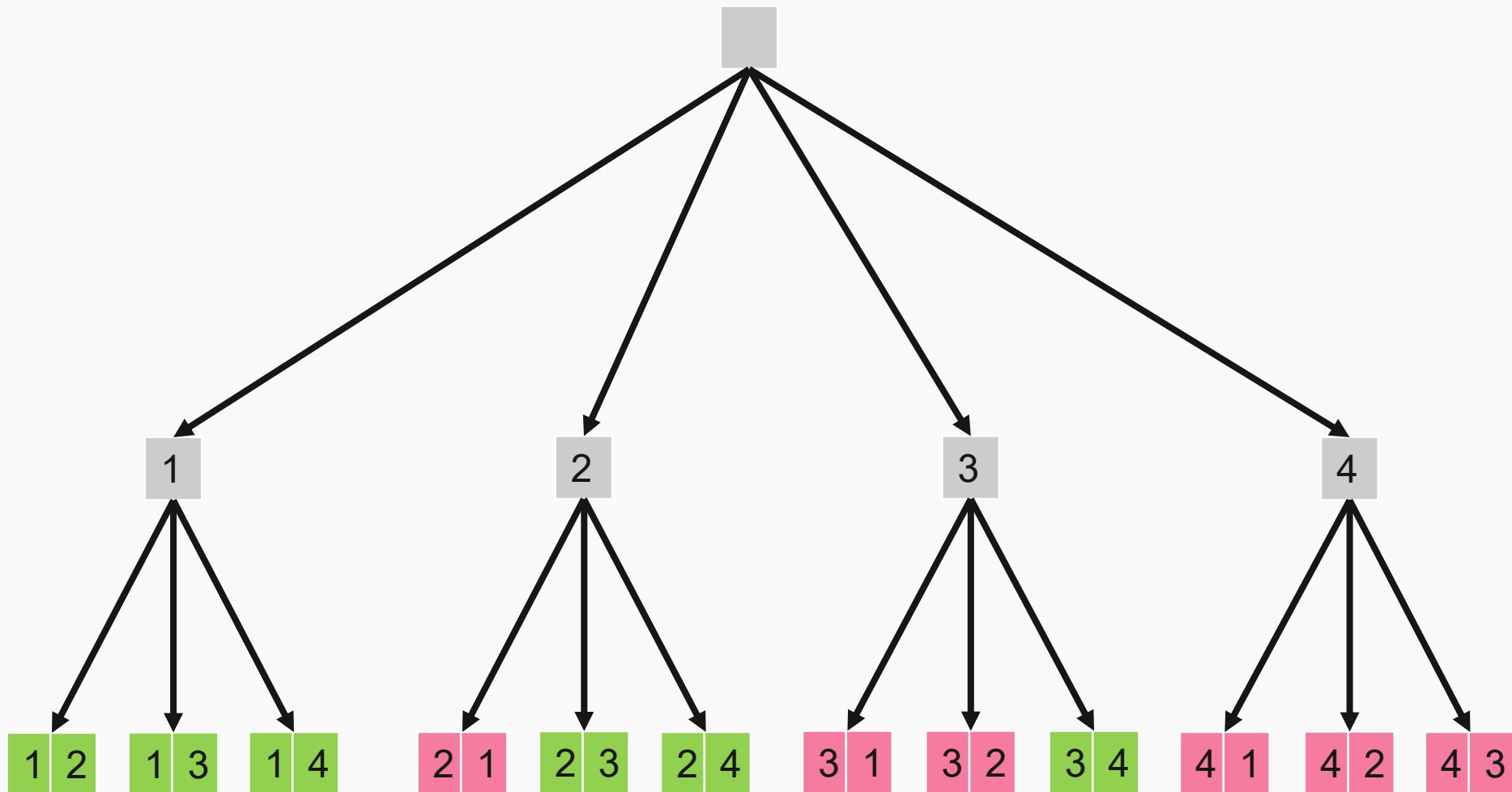
- Combinări de 4 luate câte 2:

$\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}$

- Combinări de  $n$  luate câte  $k$  sunt toate submulțimile de  $k$  elemente :  $\{x_1, x_2, \dots, x_k\}, x_i < x_j, x_i \in \{1, 2 \dots n\}$



# Problemă: generarea combinărilor







# Problemă: generarea permutărilor

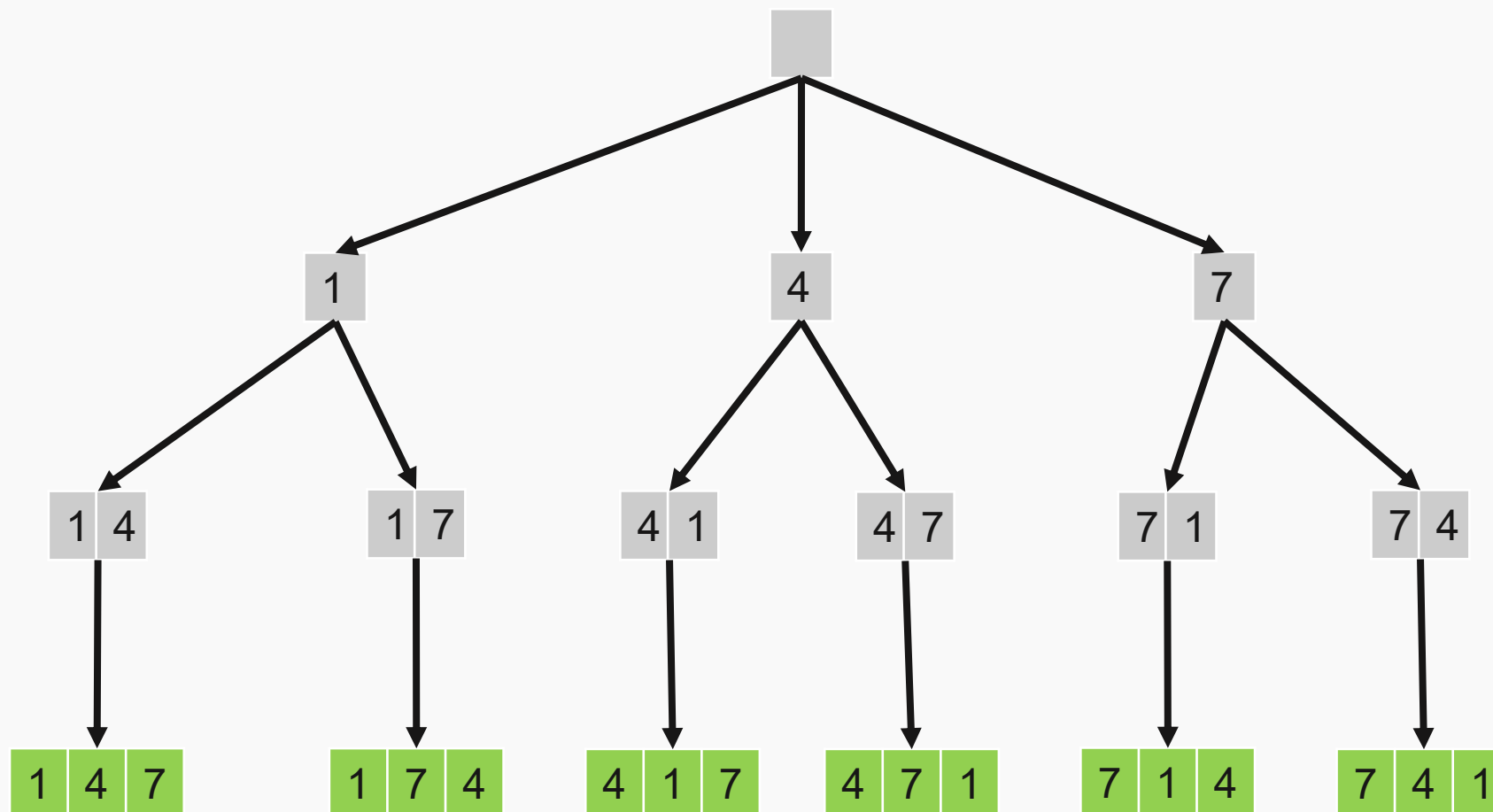
- Permutări de 3 elemente. Fie acestea  $X = \{1, 4, 7\}$ :

$\{1, 4, 7\}, \{1, 7, 4\}, \{4, 1, 7\}, \{4, 7, 1\}, \{7, 1, 4\}, \{7, 4, 1\}$

- Permutările unei mulțimi de  $n$  elemente sunt toate mulțimile:  
 $\{x_1, x_2, \dots, x_k\}, x_i \neq x_j, x_i \in X$



# Problemă: generarea permutărilor



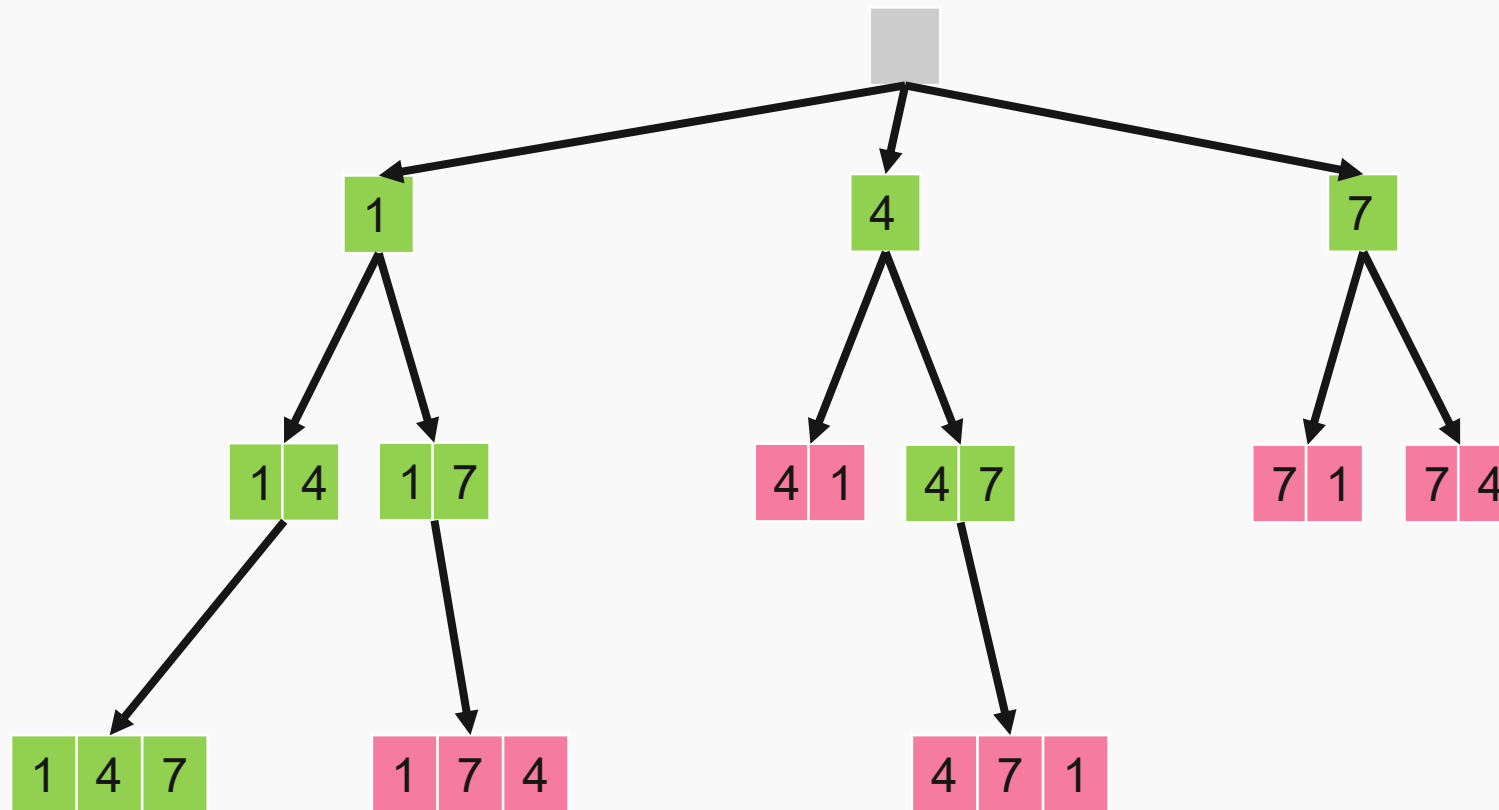


# Problemă: generarea tuturor submulțimiilor

- Submulțimile mulțimii de 3 elemente. Fie acestea  $X = \{1, 4, 7\}$ :  
 $\emptyset, \{1\}, \{4\}, \{7\}, \{1, 4\}, \{1, 7\}, \{4, 7\}, \{1, 4, 7\}$
- Submulțimiile unei mulțimi de  $n$  elemente sunt toate mulțimile:  
 $\{x_1, x_2, \dots, x_k\}, x_i \neq x_j, x_i \in X; 0 \leq k \leq n$



# Problemă: generarea permutărilor





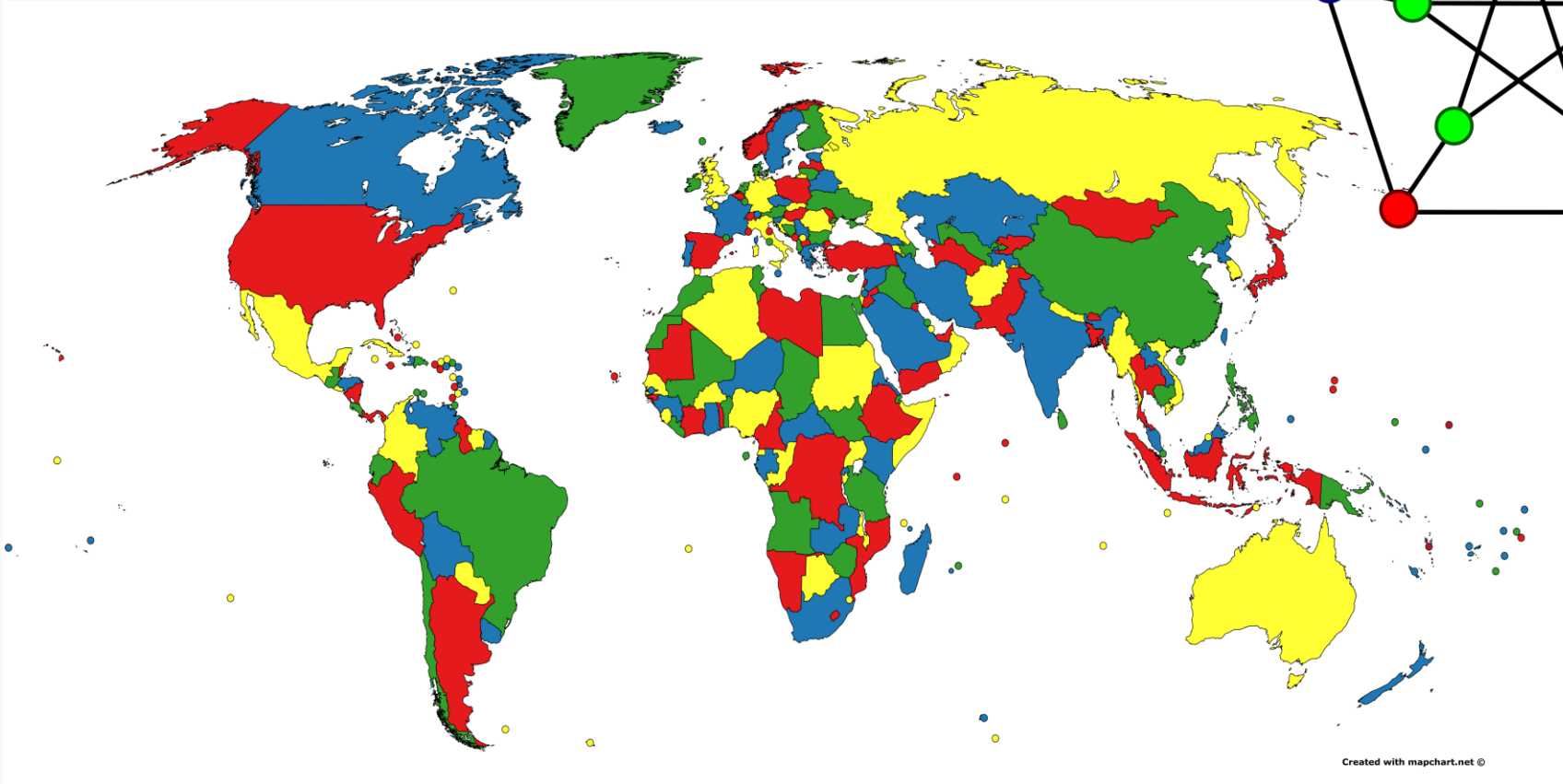
# Problemă: 15 – sliding puzzle



<http://lorecioni.github.io/fifteen-puzzle-game/>



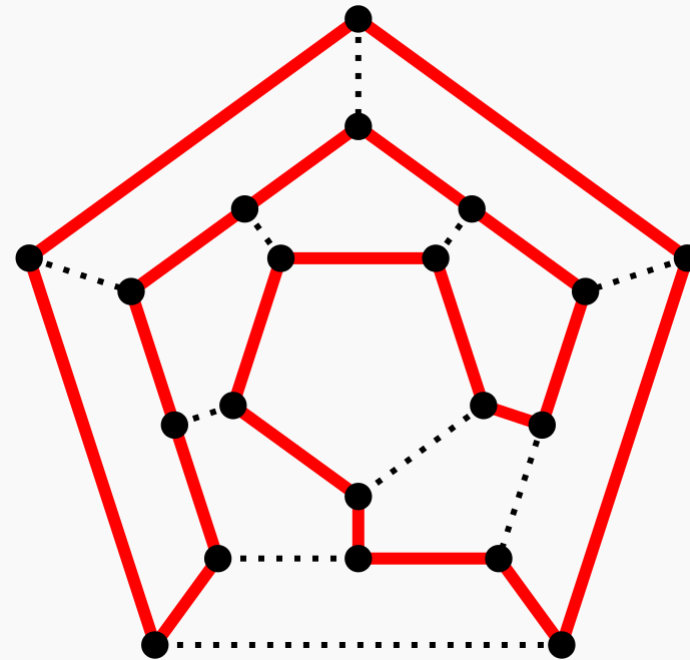
# Problemă: Graph coloring



Două noduri adiacente trebuie să aibă culori diferite.



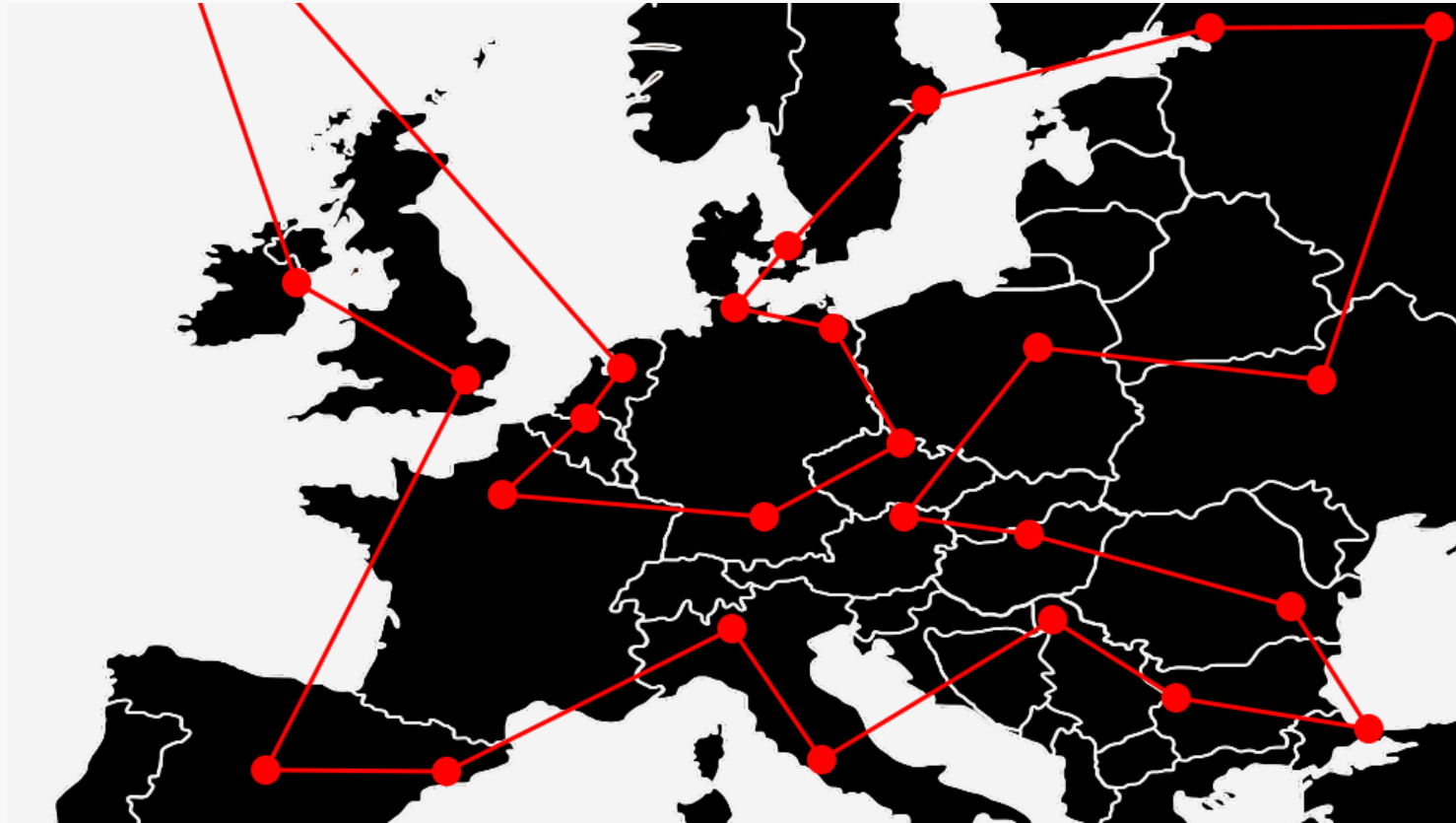
# Problemă: Ciclu hamiltonian



Ciclu care trece prin fiecare nod exact o dată.



# Problemă: comis voiajor (travelling salesman)

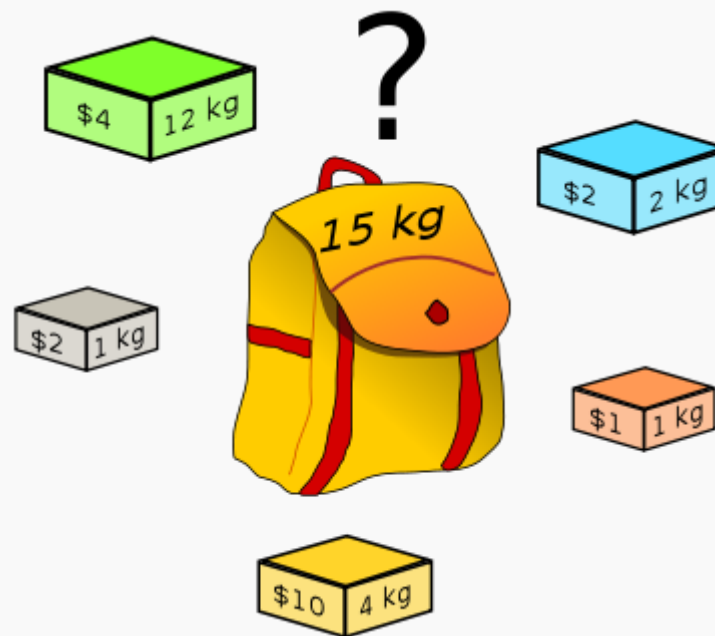


Cel mai scurt drum care trece prin toate orașele.





# Problemă: Rucsacului (Knapsack)



Ce obiecte aleg să fac cât mai valoros ghiozdanul?



## Problemă: subset de sume (subset sub)

Fiind dată o mulțime  $S$  de valori naturale  $x_1, x_2 \dots x_n$  și un număr  $T$ , să se găsească submulțimea de sumă  $T$ .

Exemplu : dacă  $S=\{8, 11, 26, 29, 37\}$  și  $T=37$ .

Soluții posibile sunt:  $\{8, 29\}$ ,  $\{11, 26\}$ ,  $\{37\}$ .

La fiecare pas, set de alegeri cunoscute (unul dintre elementele rămase) !

**Constrângerile sunt date de elemente diferite în cadrul mulțimii (posibil ordonate) și nedepășirea sumei date!**



# Complexitate backtracking?

alegeri pe nivel 0 \*

alegeri pe nivel 1 \*

alegeri pe nivel 2 \*

...

**În general:**

$$O(N!)$$



# Backtracking – discuție finală

- Nu putem folosi dacă pe un nivel avem extrem de multe sau o infinitate de alegeri disponibile.
  - ▣ Gen: care două numere adunate dau 15?
- Este **extraordinar de lent**, în general dacă avem altă opțiune e bine să alegem pe aceea.
  - ▣ Gândiți-vă cât ar dura o sortare cu backtracking?
- Aproape orice problemă poate fi abordată prin backtracking. Avantajul este că o astfel de implementare este ușor de scris și poate fi folosită pentru a genera soluții complexe corecte cu care putem să verificăm corectitudinea unui algoritm mai eficient.