Força Bruta, Recursão e Backtracking

Laboratório de Programação Competitiva I

Toki Yoshida
Nick Barbosa Gomes
Xelson Henrique Morelli

Paradigmas de Projeto de Algoritmos

- Determinados problemas requerem abordagens adequadas em suas resoluções;

- Dependendo da estratégia adotada, o desempenho do algoritmo pode ser ineficiente, resultando em TLE;

- Os paradigmas são como estratégias de como abordar e modelar uma solução de maneira eficiente quando aplicados corretamente.

Paradigmas de Projeto de Algoritmos

- Indução Matemática
- Força Bruta
- Recursividade
- Backtracking (Tentativa e Erro)
- Algoritmos Gananciosos
- Divisão e Conquista
- Programação Dinâmica
- Algoritmos Aproximados (Heurísticas)

Força Bruta

- Algoritmos simples;

- Percorrem o espaço de busca do problema, procurando todas as possíveis soluções candidatas, verificando quais satisfazem a questão inicial;

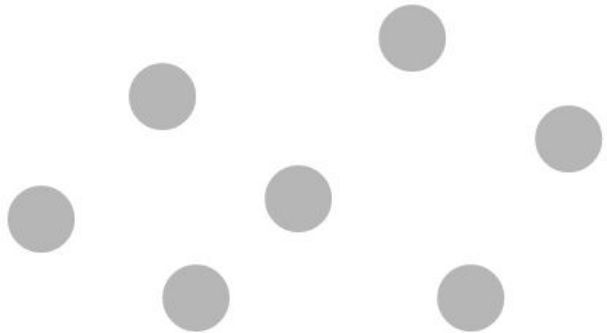
- Varredura “cega”;

Força Bruta

- Útil para pequenos problemas;
- Fácil implementação;
- Sempre que existir uma solução, o algoritmo irá encontrá-la.
- Exigem grande esforço computacional, seu custo tende a crescer exponencialmente.

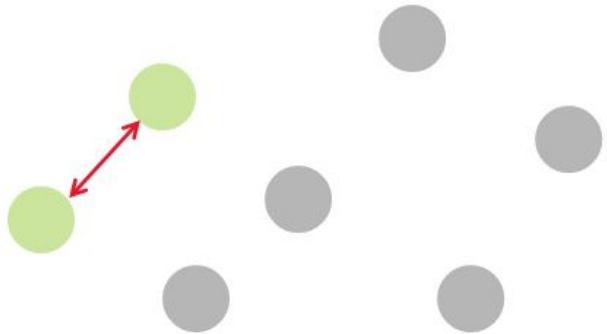
The Closest Pair Problem

- Problema: dados N pontos no plano, determinar a distância mínima entre qualquer par de pontos.



$$\text{Distância} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

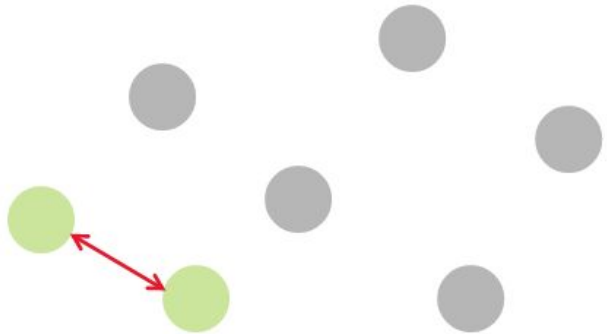
The Closest Pair Problem



Distância mínima = ∞
Distância calculada = 7.65

Calculada < Mínima ? SIM!
Mínima = Calculada

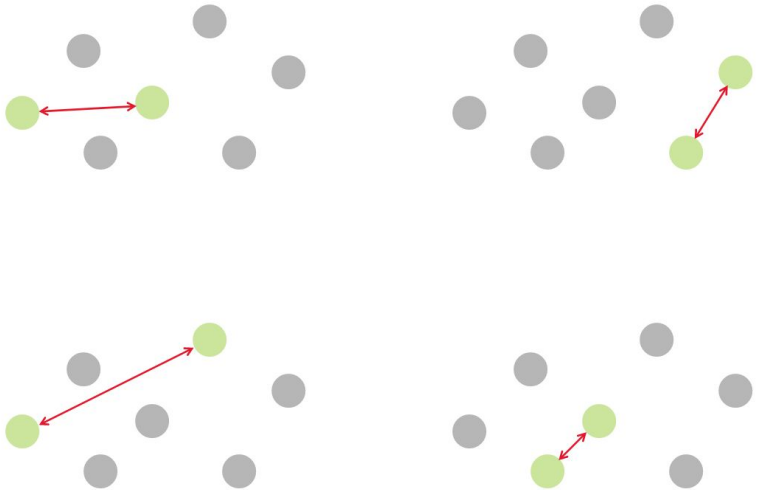
The Closest Pair Problem



Distância mínima =7.65
Distância calculada = 8.94

Calculada < Mínima ? NÃO!
Manter a mínima atual

The Closest Pair Problem



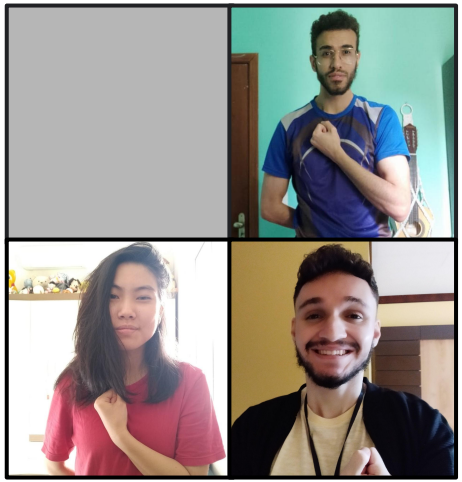
Repetir o processo para todos os pares de pontos pertencentes ao conjunto.

```cpp
double euclidean(double x1, double y1, double x2, double y2) {
    return (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
}

int main() {
    int n;
    cin >> n;

    vector<double> x(n), y(n);

    for (int i = 0; i < n; i++)
        cin >> x[i] >> y[i];

    double mi = INF;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            double dist = sqrt(euclidean(x[i], y[i], x[j], y[j]));
            mi = min(mi, dist);
        }
    }

    cout << dist << "\n";

    return 0;
}
```

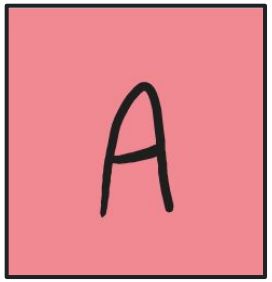
Complexidade: $O(N^2)$

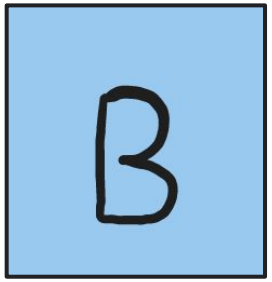
Amity Assessment

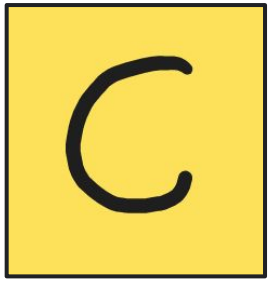
- Problema: dados 2 quebra-cabeças com disposições de peças distintas, determinar se é possível que ambos cheguem na mesma configuração.



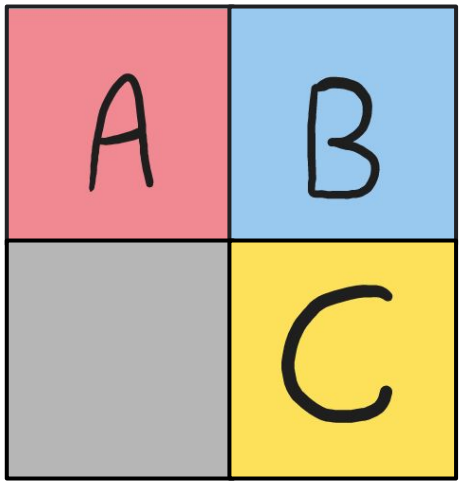
Amity Assessment

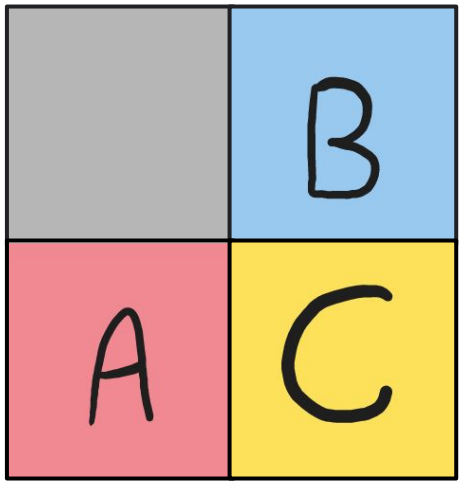






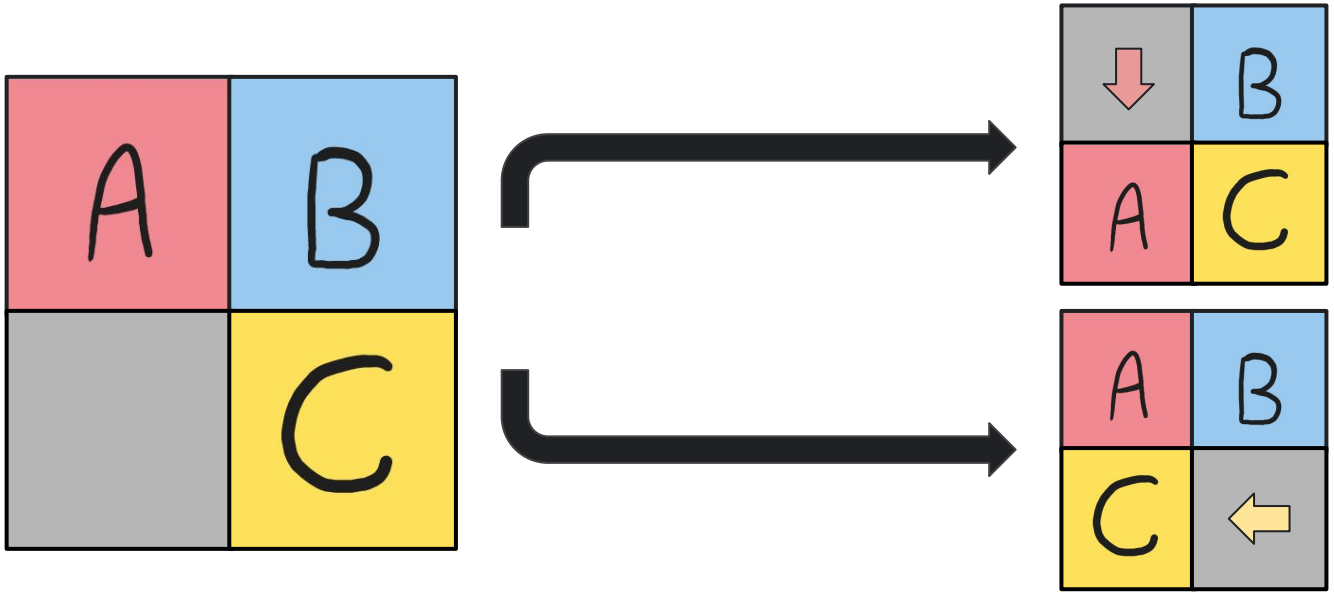
Amity Assessment





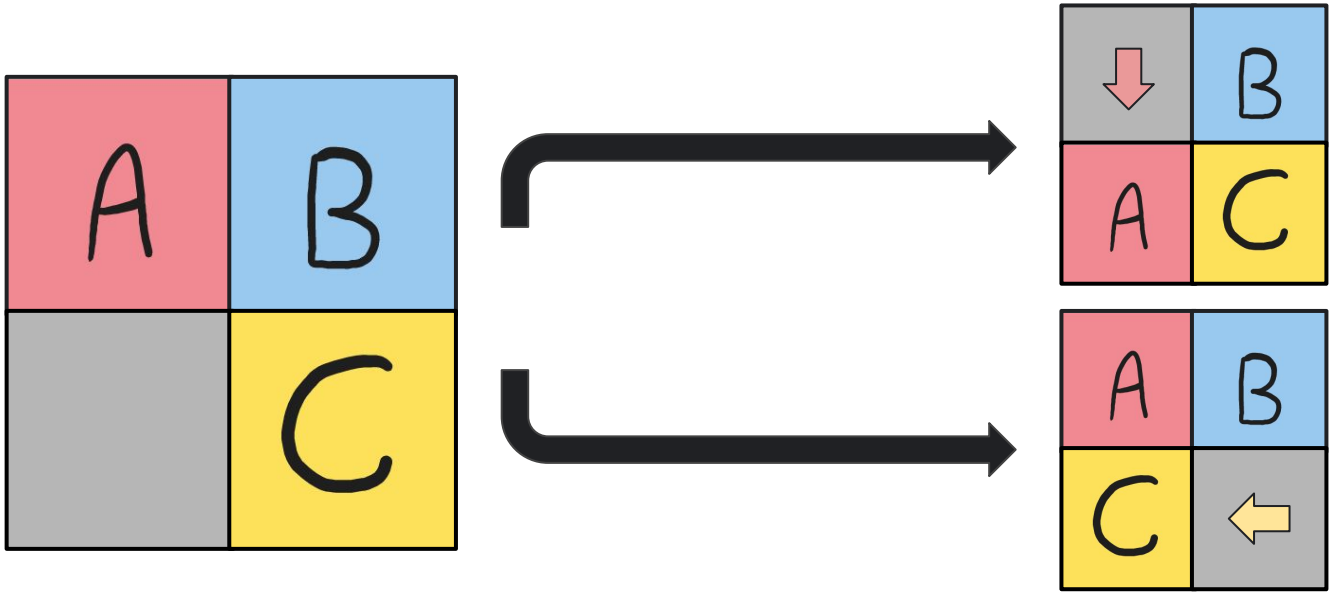
Amity Assessment

- Regra: só podemos trocar de posição as peças adjacentes à peça cinza.



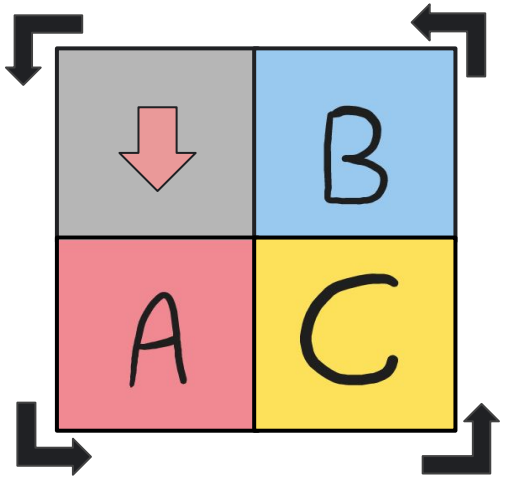
Amity Assessment

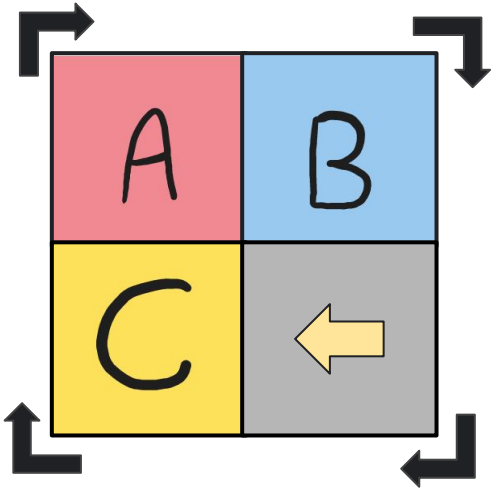
- Regra: só podemos trocar de posição as peças adjacentes à peça cinza.



Amity Assessment

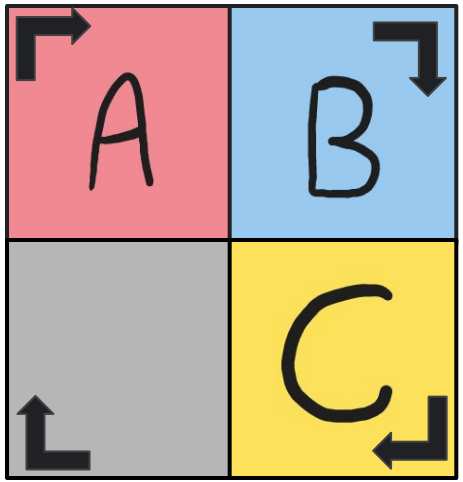
- Podemos perceber que o movimento das peças ocorre de forma circular, nos sentidos horário e anti-horário, retornando a composição inicial eventualmente.





Amity Assessment

- Assim, podemos escrever o quebra-cabeça como uma string, colocando um indicador para o quadrado cinza e mudando sua posição com os adjacentes para formar todas as configurações possíveis.

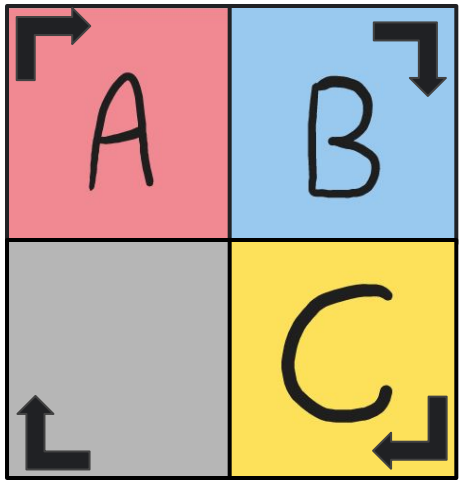


Configuração atual: ABCX

Outras configurações:
ABXC
AXBC
XBCA
ABCX
…

Amity Assessment

- Assim, podemos escrever o quebra-cabeça como uma string, colocando um indicador para o quadrado cinza e mudando sua posição com os adjacentes para formar todas as configurações possíveis.



Configuração atual: ABCX

Outras configurações:
ABXC
AXBC
XBCA
ABCX
…

Amity Assessment

- Podemos montar apenas todas as configurações de um dos quebra-cabeças e compará-las com o estado atual do segundo quebra-cabeças para definir se existe solução

Possíveis configurações quebra-cabeças 1:	Configuração quebra-cabeças 2:
ABCX	XBAC
AXBC	
XBCA	
…	
XBAC	
…	

```cpp
int main() {
    string str1, str2, aux;
    cin >> str1 >> aux;
    str1 = str1 + aux[1] + aux[0];
    cin >> str2 >> aux;
    str2 = str2 + aux[1] + aux[0];

    int pos;
    vector<string> possible;
    aux = str2;

    for (int i = 0; i < 4; i++) {
        if (aux[i] == 'X')
            pos = i;
    }

    do {
        int i = (pos + 1) % 4;
        swap(aux[pos], aux[i]);
        possible.push_back(aux);
        pos = i;
    } while (aux != str2);

    for (int i = 0; i < possible.size(); i++) {
        if (possible[i] == str1) {
            cout << "YES\n";
            return 0;
        }
    }

    cout << "NO\n";

    return 0;
}
```

Complexidade: O(N!)

Sugestão de Problemas

- The Closest Pair Problem - UVa
- Amity Assessment - Codeforces

Recursividade

- Mecanismo no qual a definição de uma função ou de um objeto se refere ao próprio objeto definido;

- Permite descrever problemas que utilizam estruturas recursivas de forma clara e concisa;

- Um procedimento que opera em termos de si mesmo é dito ser recursivo.

Gerando Subconjuntos

_ _ _

- Nossa primeira aplicação da recursão será gerar todos os subconjuntos de um conjunto de N elementos;

Gerando Permutações

_ _ _

- A segunda aplicação de recursão que veremos será gerar todas as permutações de um conjunto de N elementos;

Desafio

- Amity Assessment - Codeforces
  - Generalizar a solução do problema para 2 ≤ N ≤ 10, elencando todas as possíveis soluções do quebra-cabeças com um algoritmo recursivo.

Backtracking

___

- Aperfeiçoamento do algoritmo de força bruta;
- Várias soluções da força bruta são eliminadas, visto que não são testadas;
- Inicialmente se tem uma solução vazia e vamos estendendo ela passo a passo, incrementalmente.

Problema das Rainhas

___

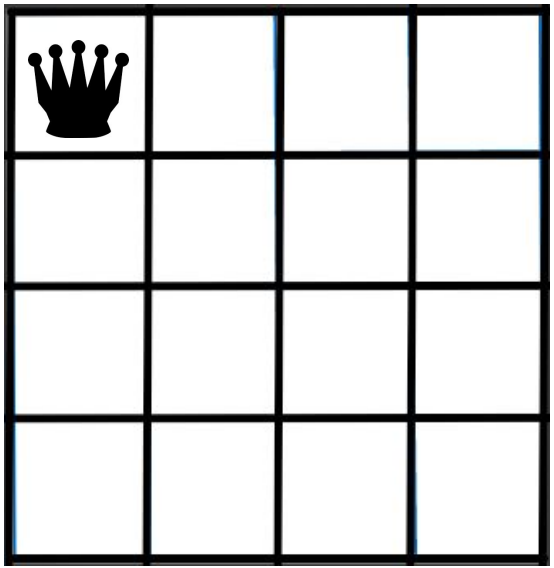
- Calcular o número de maneiras de se colocar N rainhas em um tabuleiro NxN.

- Em um problema de backtracking inicia-se com uma solução vazia, nesse caso com o tabuleiro vazio.

- A partir do tabuleiro vazio a solução é estendida passo a passo até que se consiga colocar todas as N rainhas.

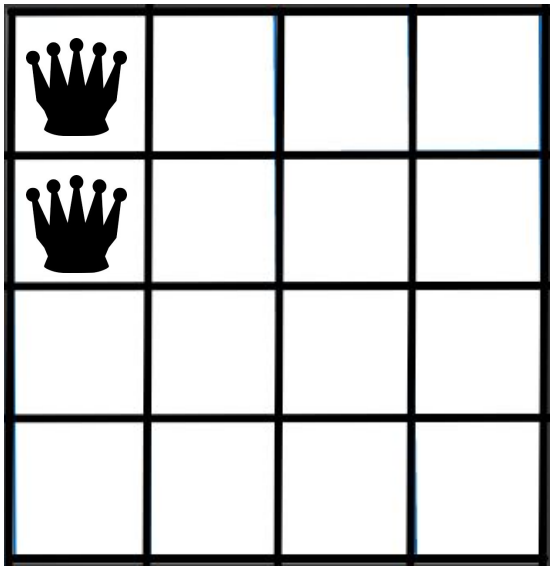
Problema das Rainhas

_ _ _ _



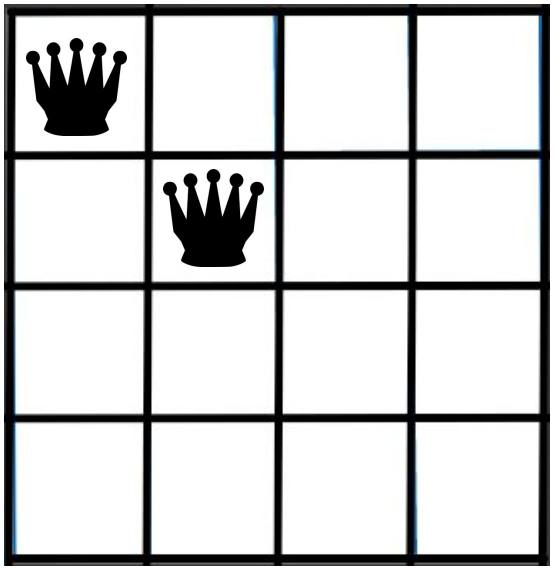
Problema das Rainhas

_ _ _ _



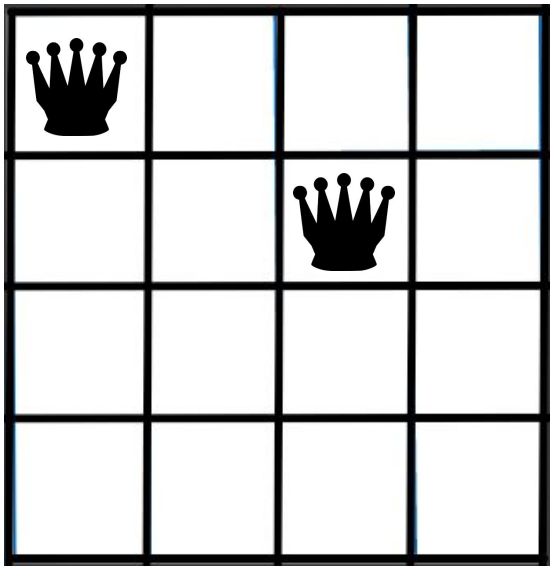
Problema das Rainhas

_ _ _ _



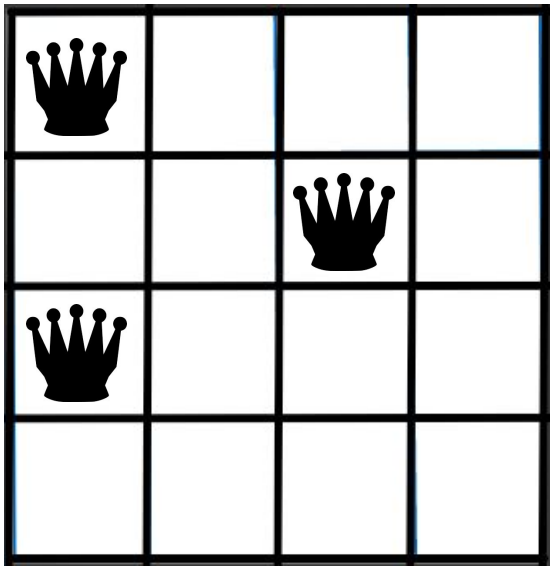
Problema das Rainhas

_ _ _ _



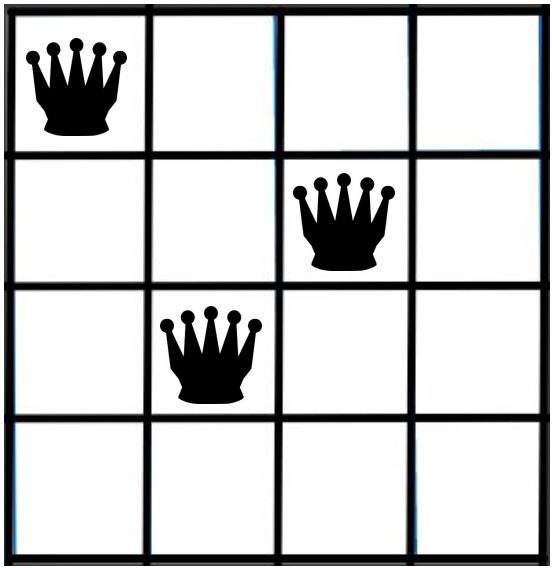
Problema das Rainhas

_ _ _ _



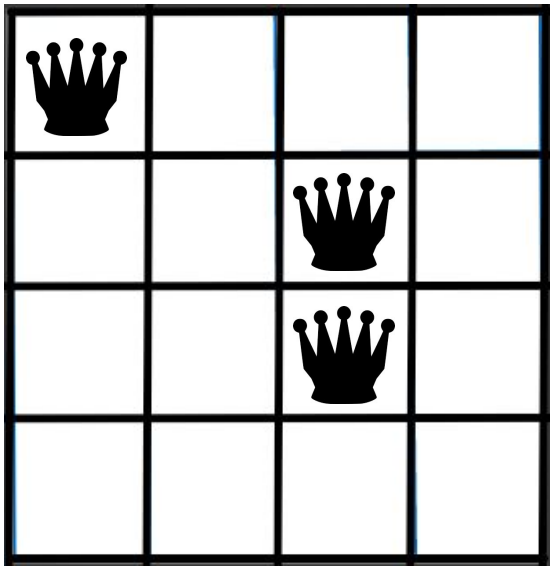
Problema das Rainhas

_ _ _ _



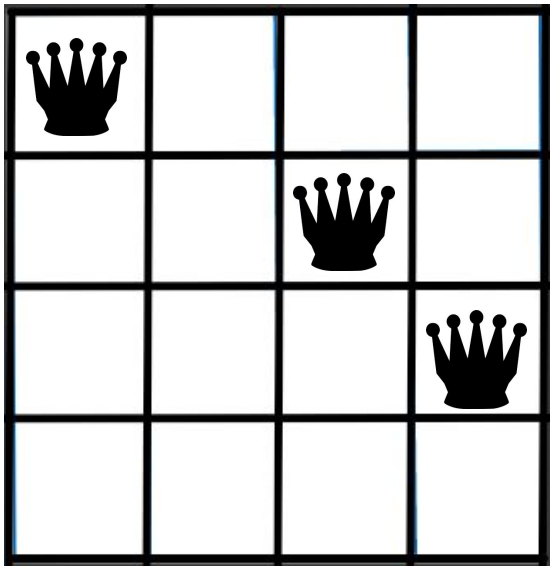
Problema das Rainhas

_ _ _ _



Problema das Rainhas

_ _ _ _

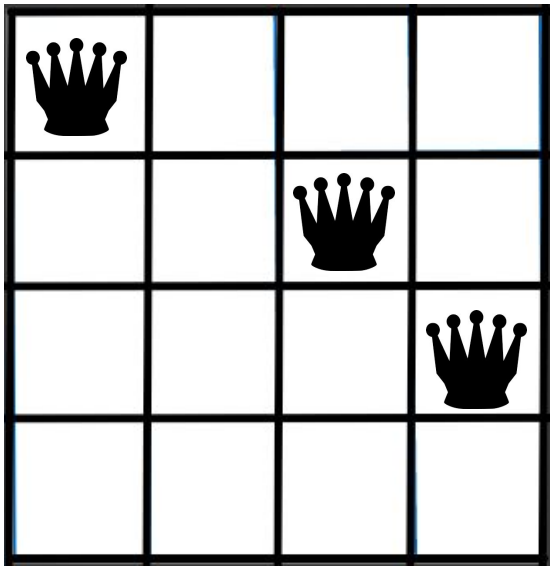


Problema das Rainhas

_ _ _ _

Não existe nenhuma posição válida para a rainha da 3ª linha.

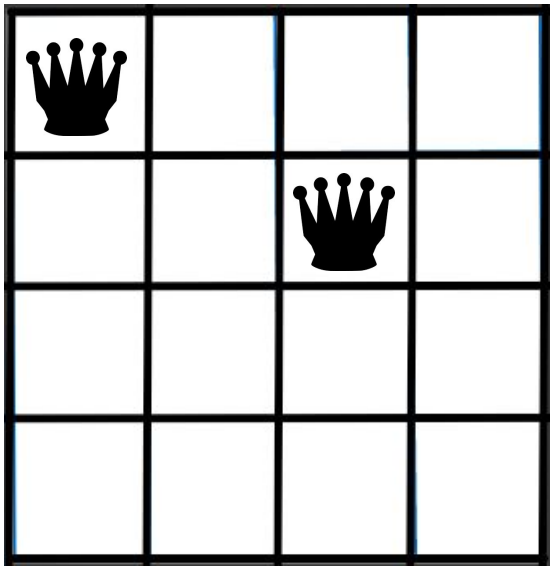
O que o backtracking faz?



Problema das Rainhas

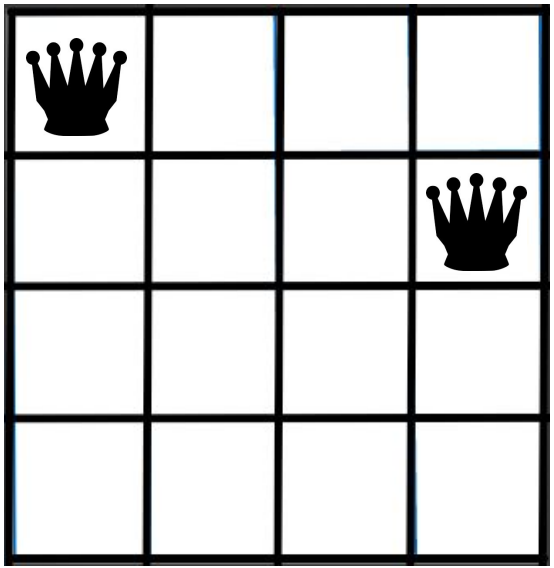
_ _ _

Volta na rainha da segunda linha que é um passo anterior e tenta achar outra posição válida para ela.



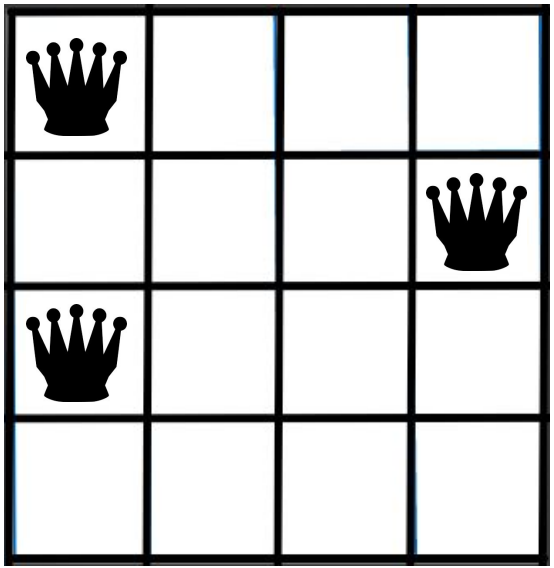
Problema das Rainhas

_ _ _ _



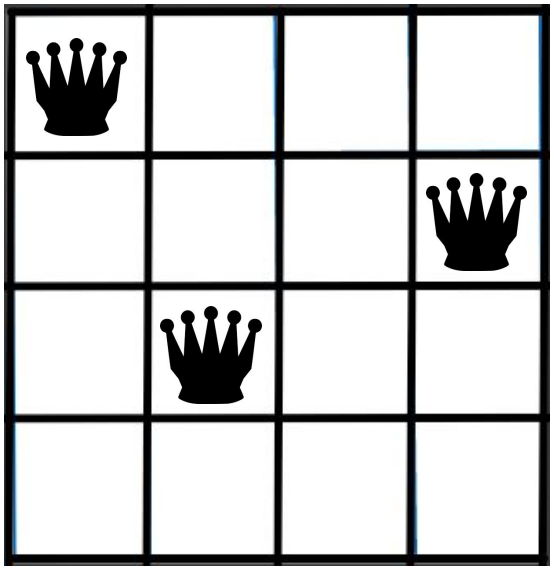
Problema das Rainhas

_ _ _ _



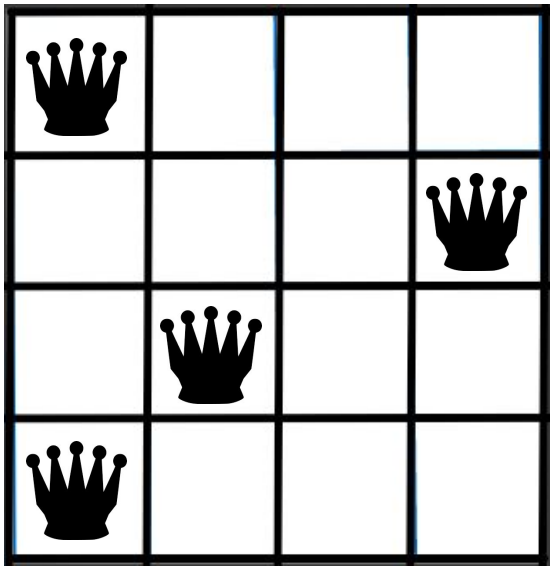
Problema das Rainhas

_ _ _ _



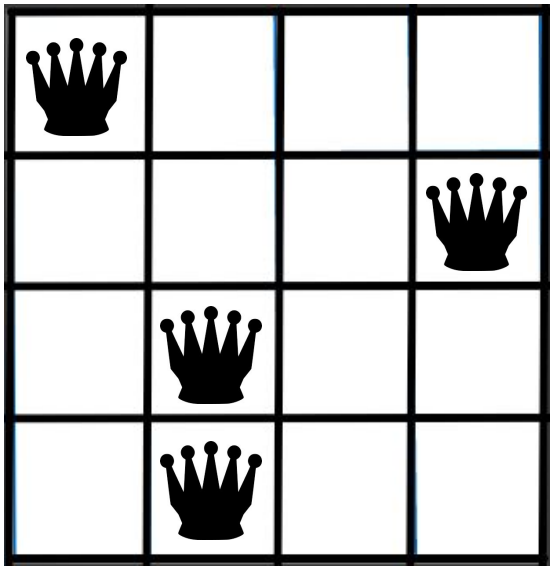
Problema das Rainhas

_ _ _



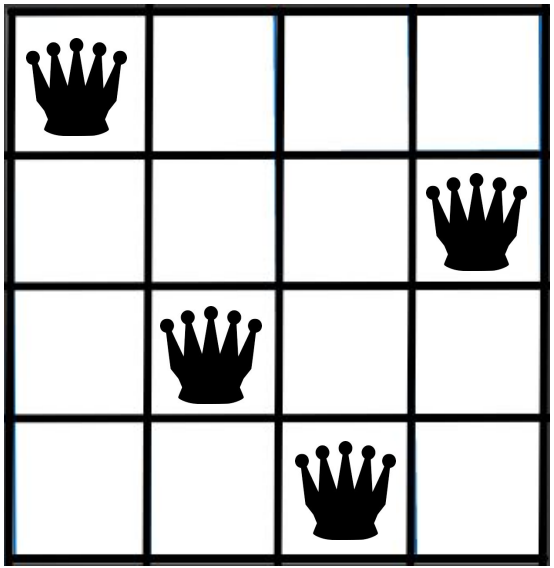
Problema das Rainhas

_ _ _ _



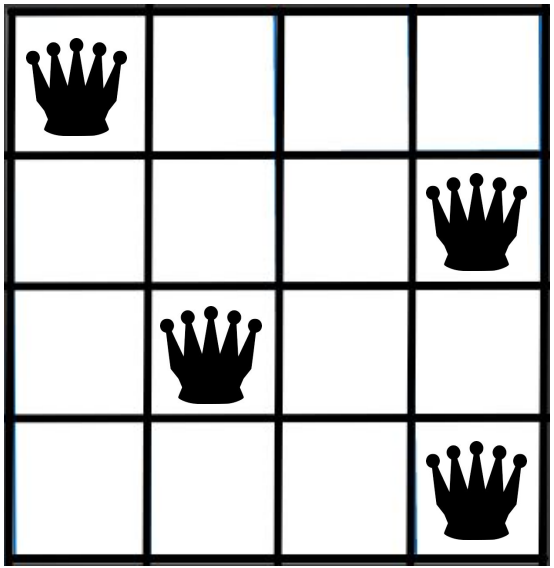
Problema das Rainhas

_ _ _ _



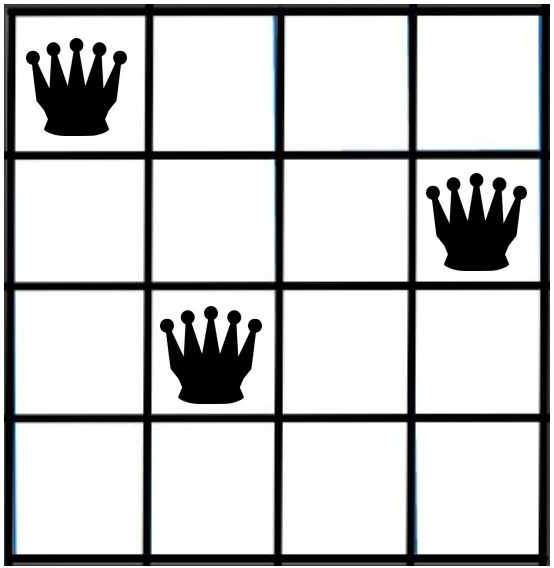
Problema das Rainhas

_ _ _ _



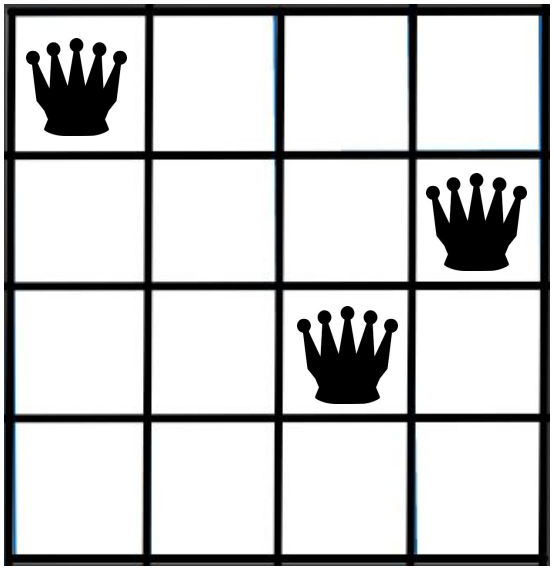
Problema das Rainhas

_ _ _ _



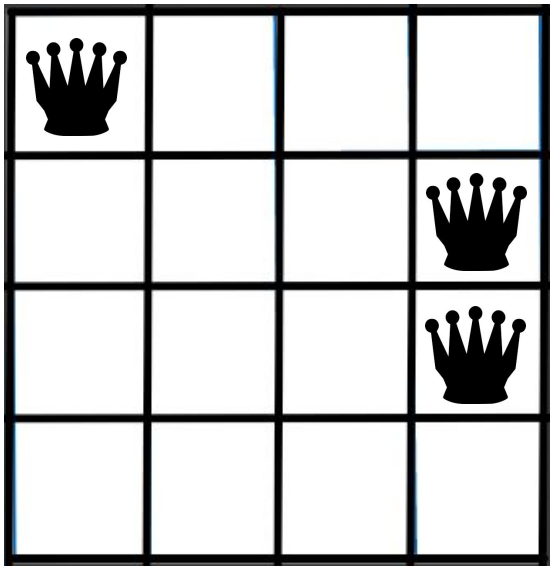
Problema das Rainhas

_ _ _ _



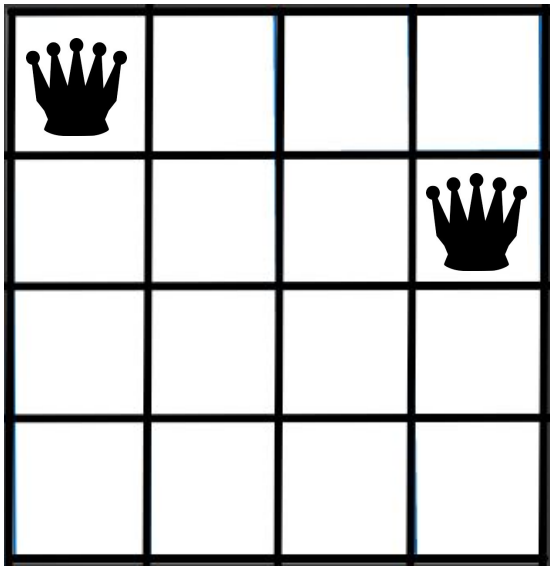
Problema das Rainhas

_ _ _ _



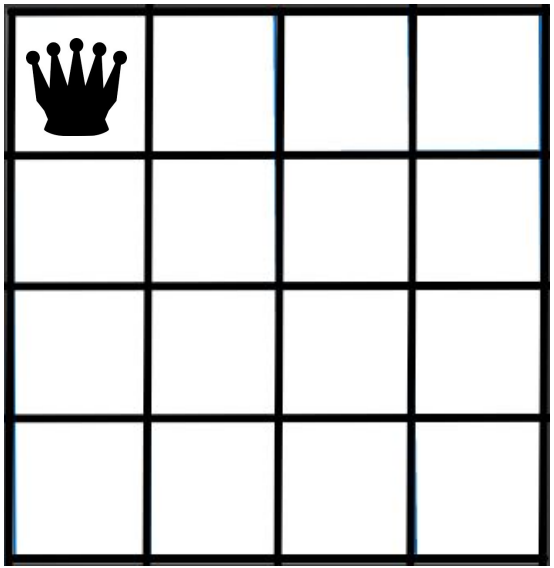
Problema das Rainhas

_ _ _ _



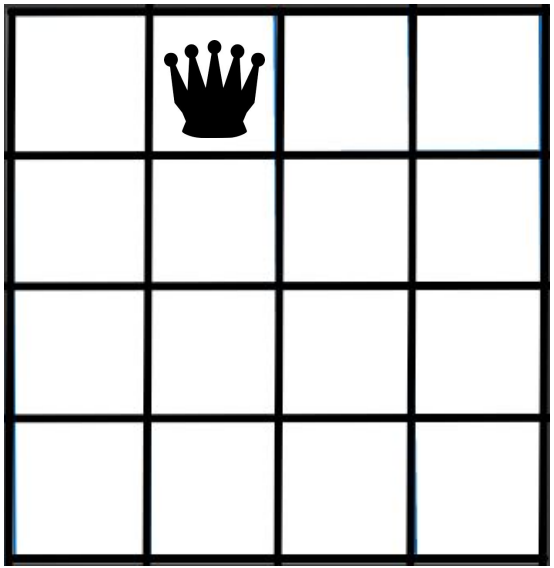
Problema das Rainhas

_ _ _ _



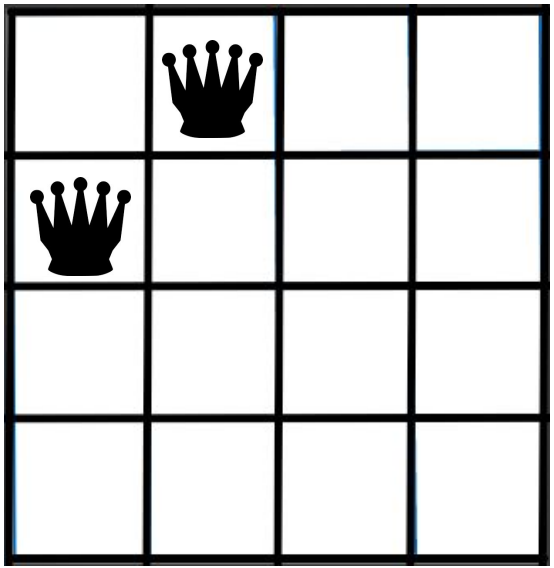
Problema das Rainhas

_ _ _ _



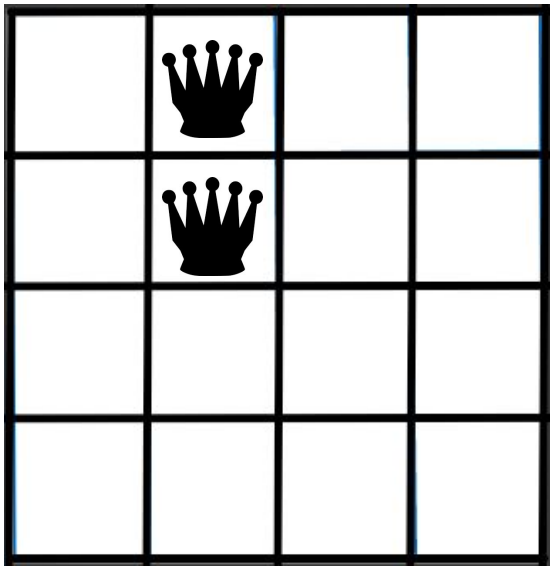
Problema das Rainhas

_ _ _ _



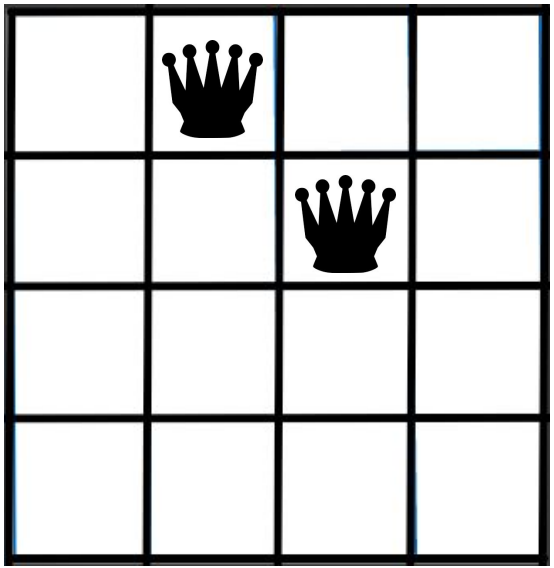
Problema das Rainhas

_ _ _ _



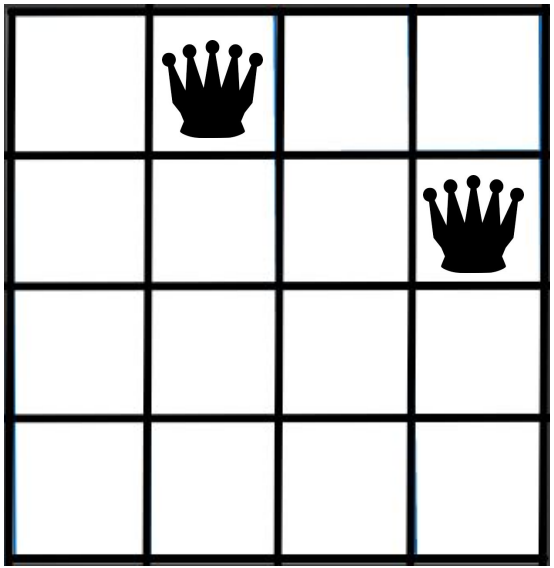
Problema das Rainhas

_ _ _ _



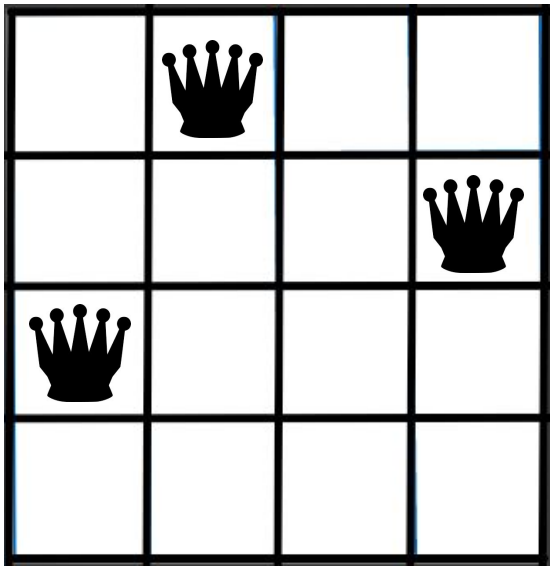
Problema das Rainhas

_ _ _ _



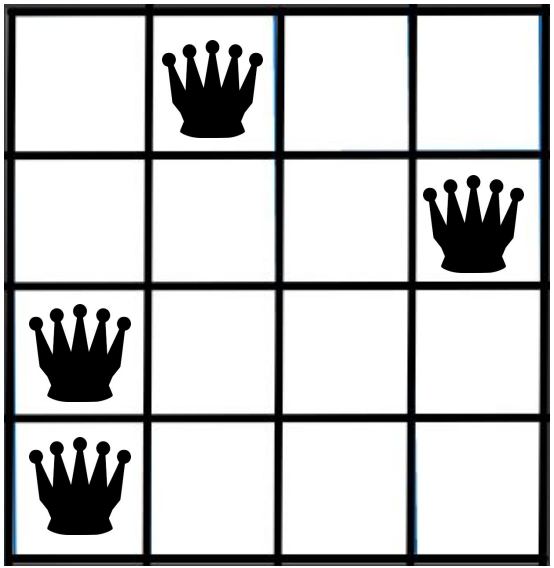
Problema das Rainhas

_ _ _ _



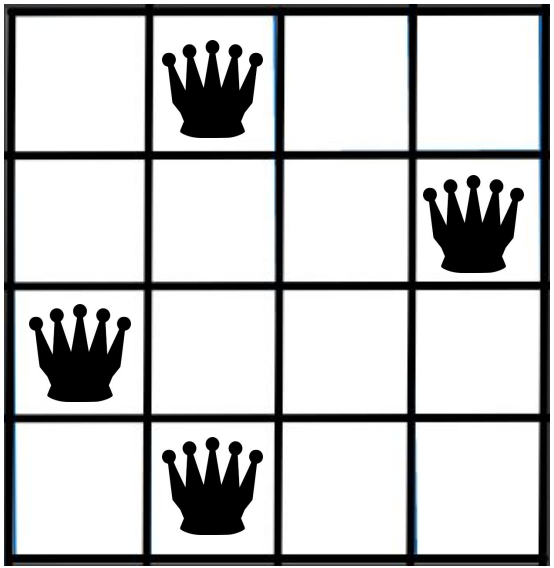
Problema das Rainhas

_ _ _ _



Problema das Rainhas

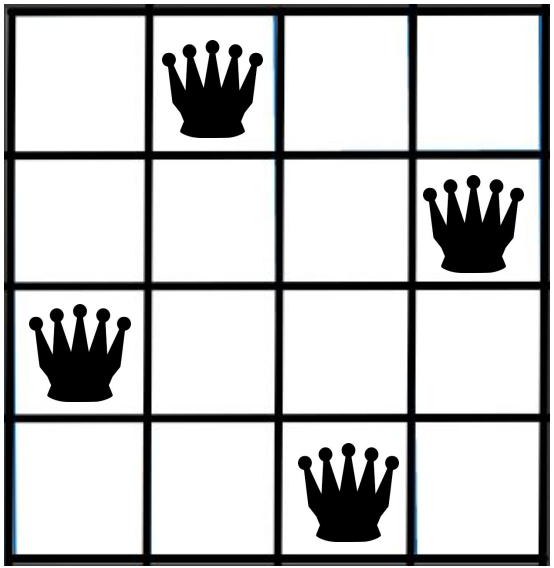
_ _ _ _



Problema das Rainhas

_ _ _ _

Wiiiiii,
conseguimos
achar uma das
respostas :)



```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

vector<bool> diagonal1, diagonal2, coluna, linha;

ll n; // tamanho do tabuleiro

ll cont = 0; // contar as maneiras diferentes que eu posso colocar n rainhas no tabuleiro

void colocar_rainha(ll lin, ll  col){

    if(lin == n+1){
        cont++;
        return;
    }
    if(col == n+1)return;

    if(diagonal1[n + (lin - col)] || diagonal2[lin+col] || coluna[col] || linha[lin]){
        colocar_rainha(lin, col+1);
    }
    else{
        colocar_rainha(lin+1, 0);
    }
    return;
}
```

Sugestão de Problemas

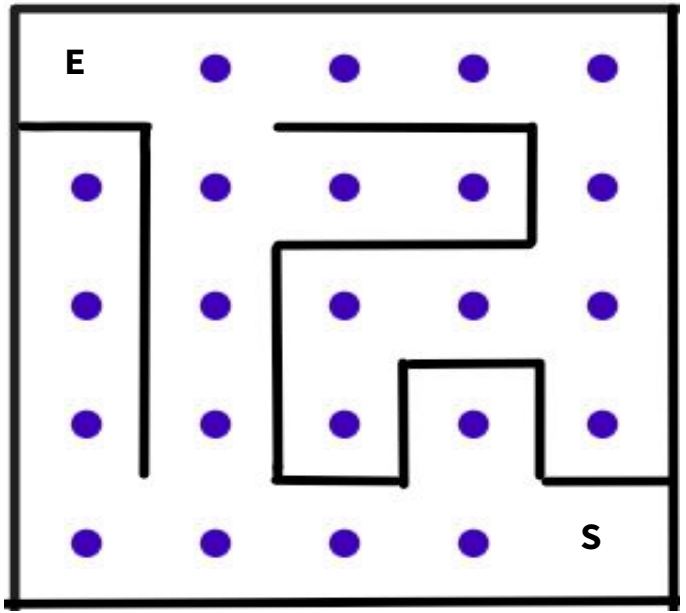
- Chessboard and Queens - CSES
- MKJUMPS - SPOJ

Problema do Labirinto

- Encontrar um caminho da entrada do labirinto até a saída.

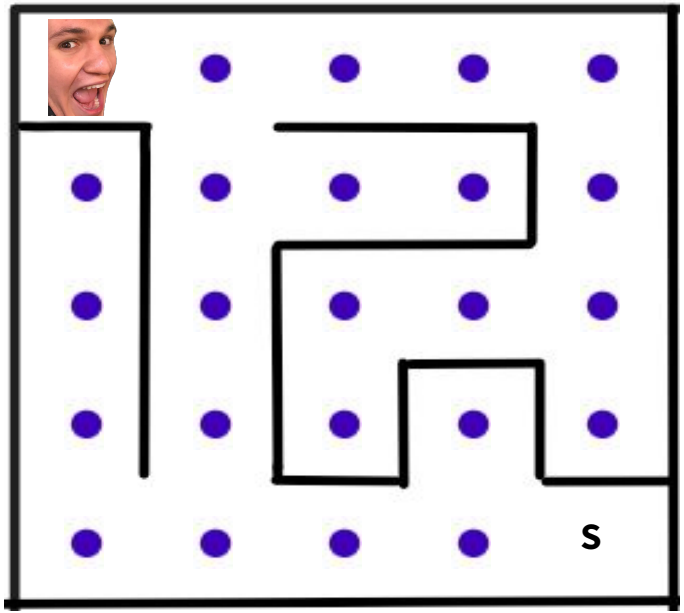
- Gerar a solução por partes, e quando encontrarmos em um beco sem saída, retornar os passos até que se possa ter acesso à outro caminho.

Problema do Labirinto



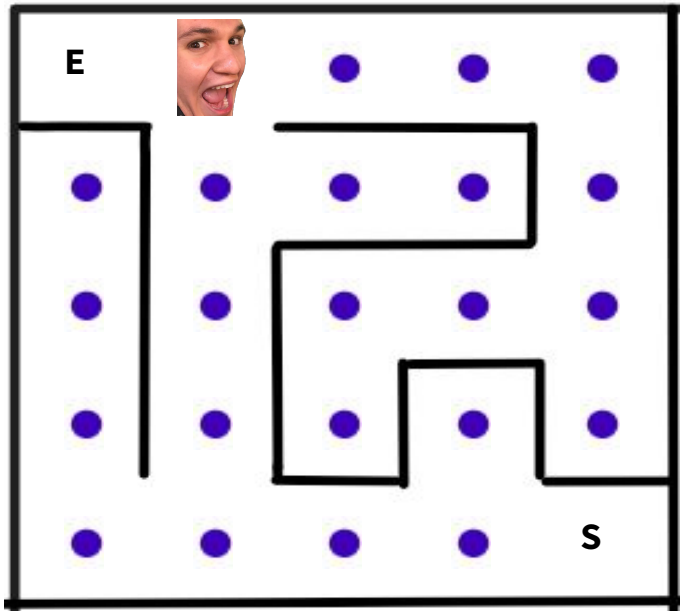
Problema do Labirinto

_ _ _



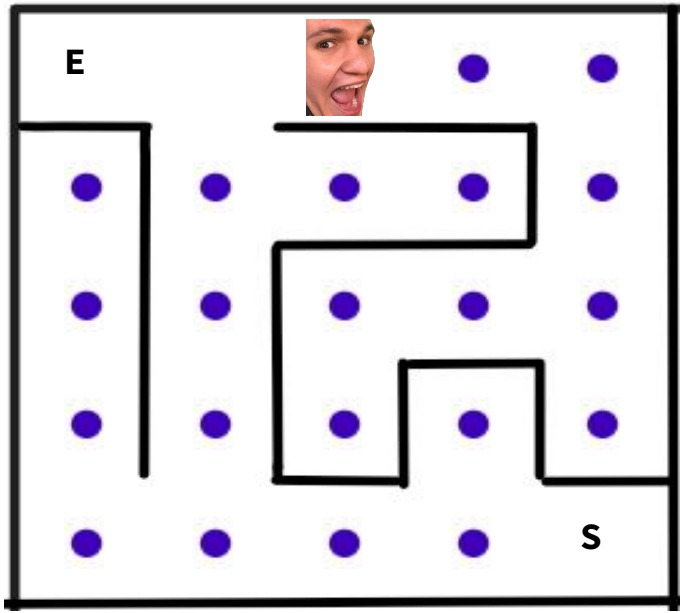
Problema do Labirinto

_ _ _

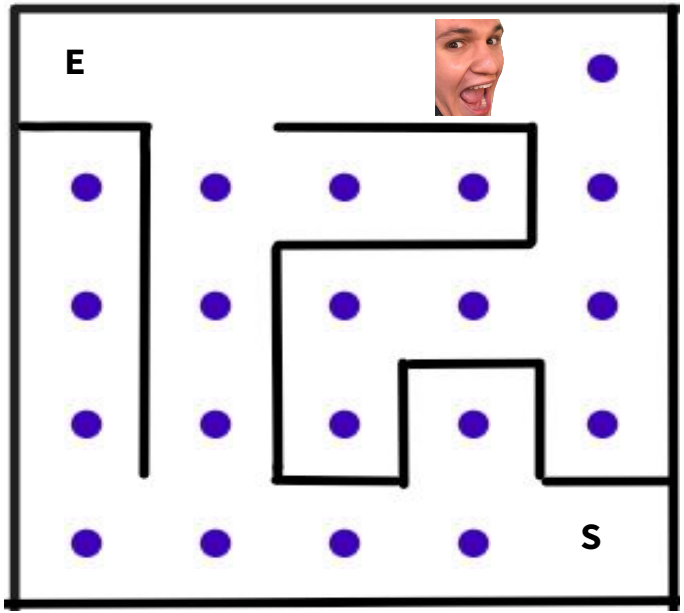


Problema do Labirinto

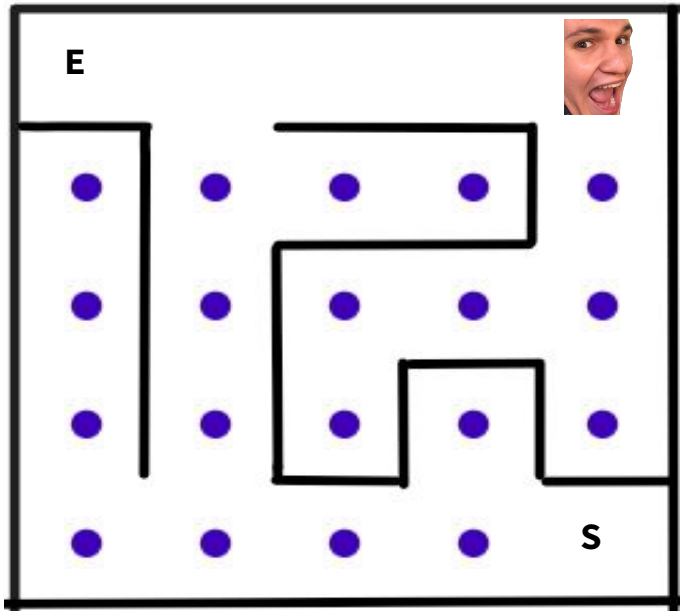
_ _ _



Problema do Labirinto

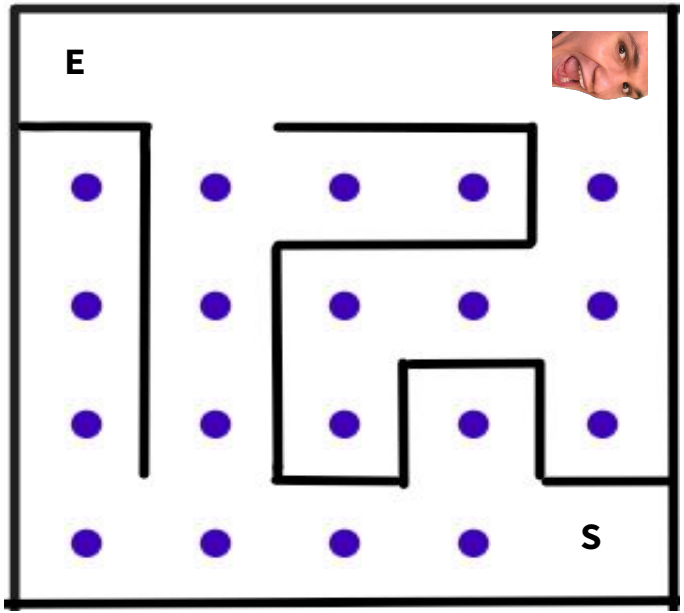


Problema do Labirinto

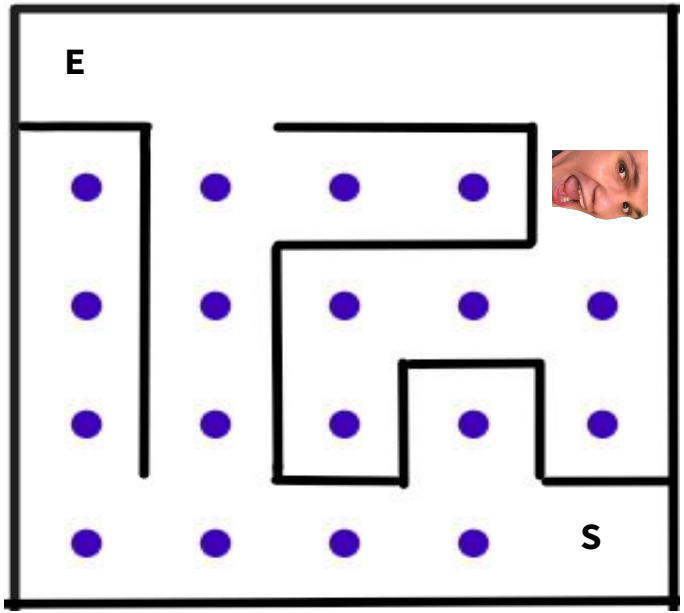


Problema do Labirinto

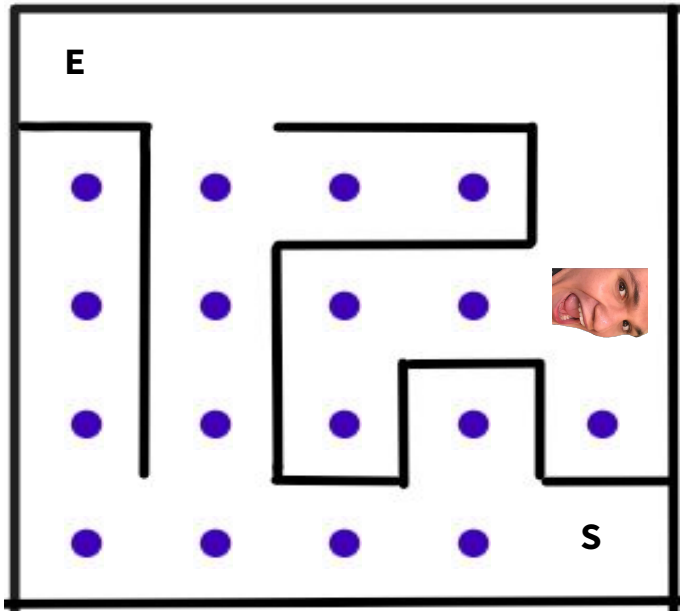
_ _ _



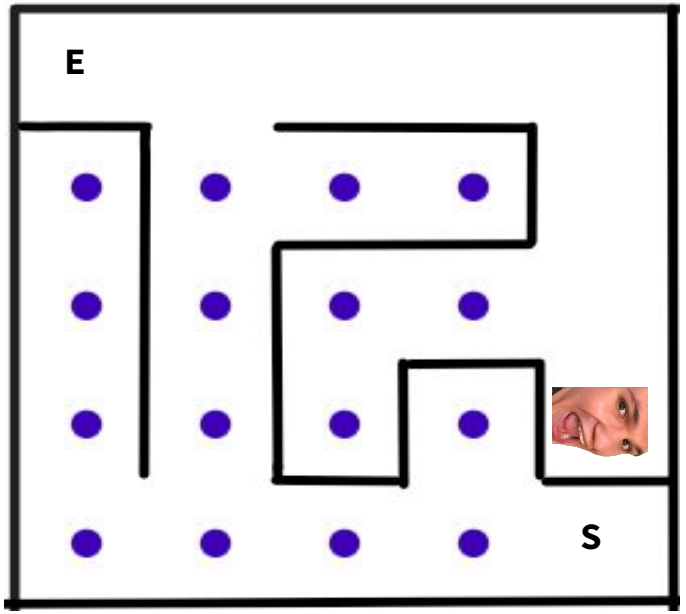
Problema do Labirinto



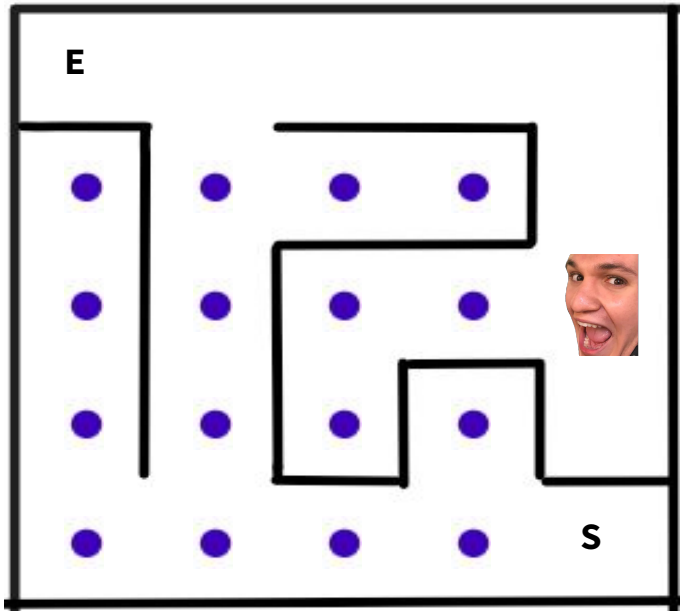
Problema do Labirinto



Problema do Labirinto

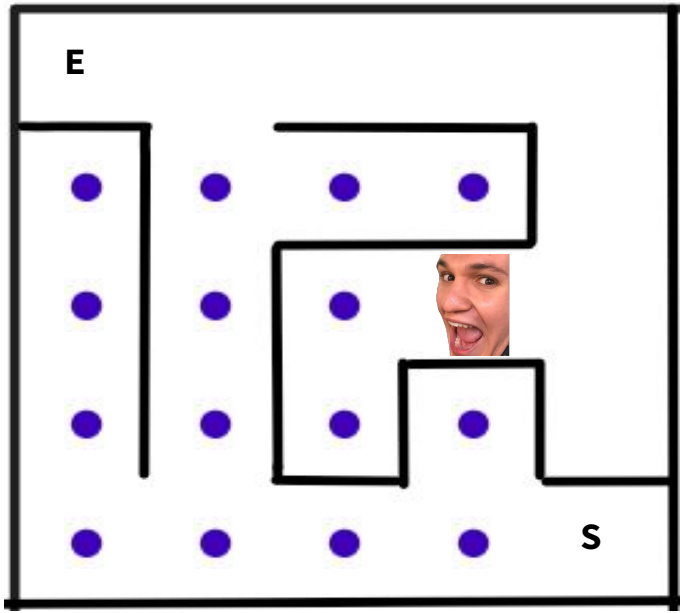


Problema do Labirinto



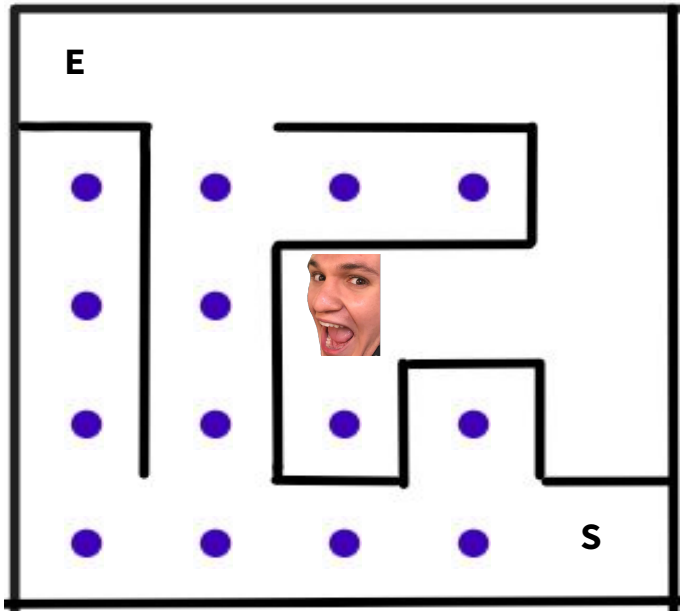
Problema do Labirinto

_ _ _



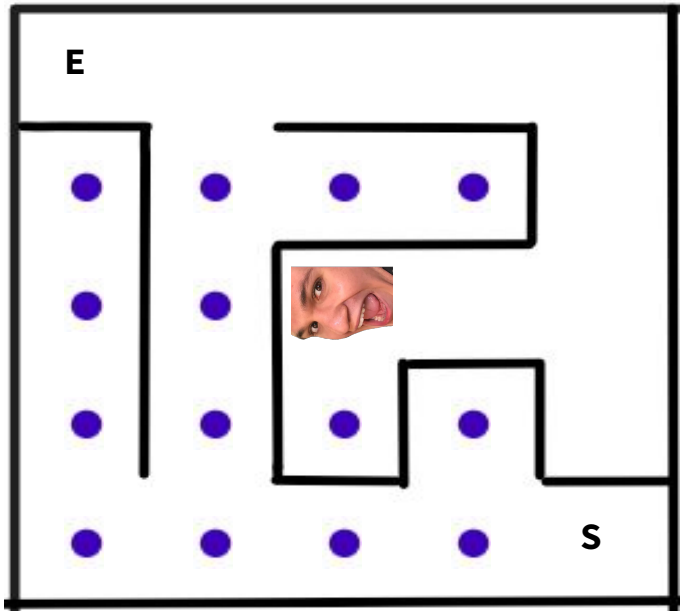
Problema do Labirinto

_ _ _



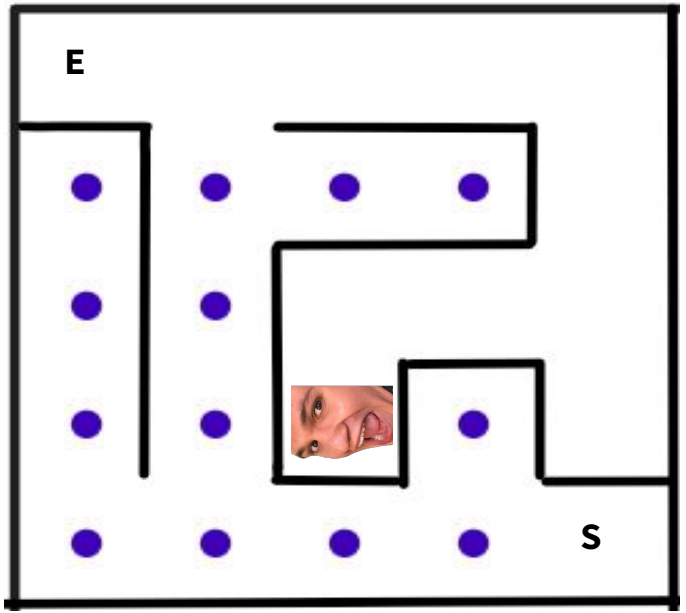
Problema do Labirinto

_ _ _

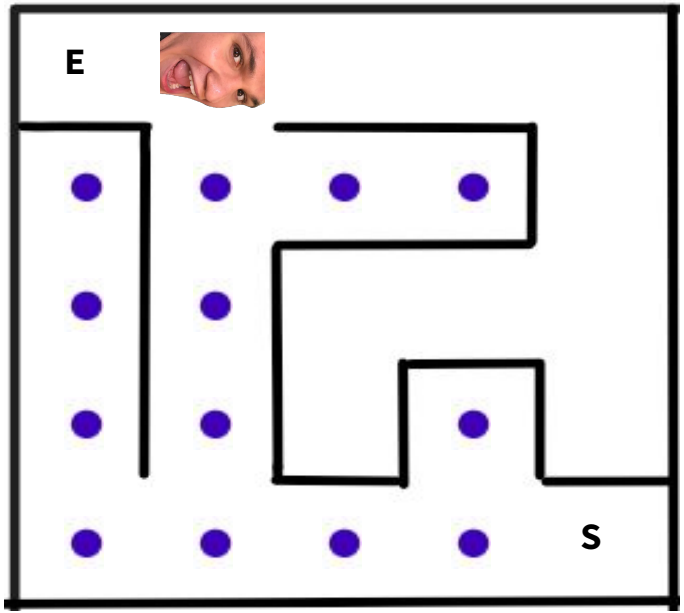


Problema do Labirinto

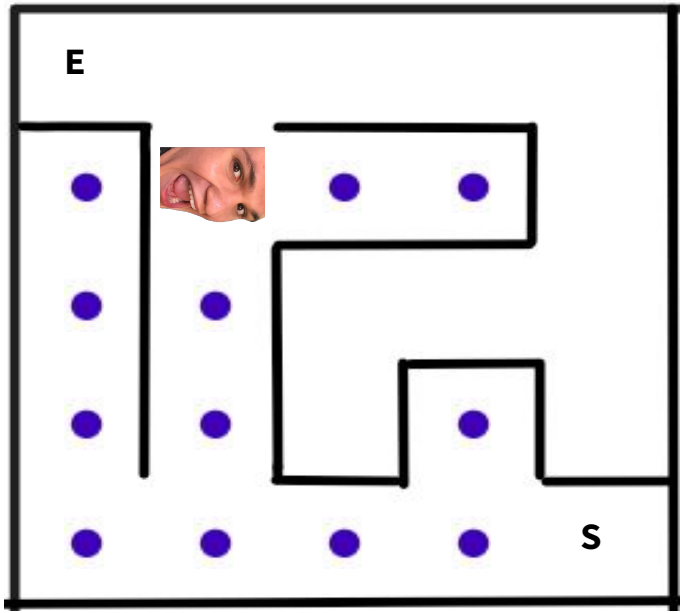
_ _ _



Problema do Labirinto

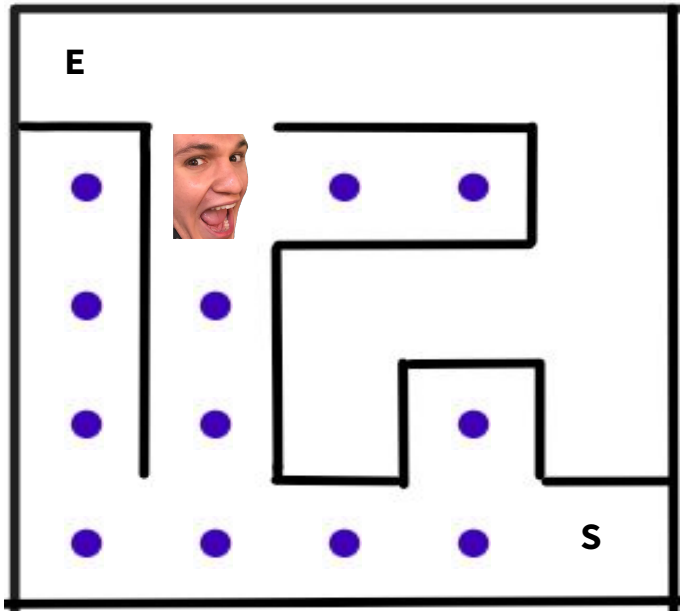


Problema do Labirinto



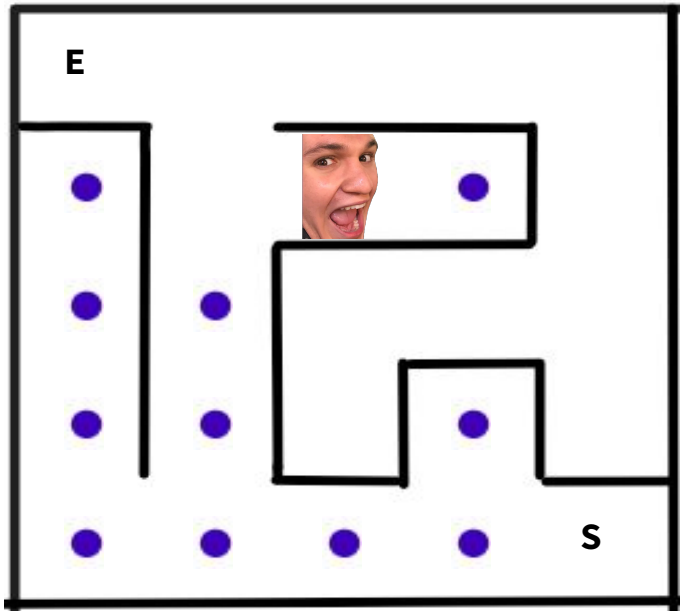
Problema do Labirinto

_ _ _



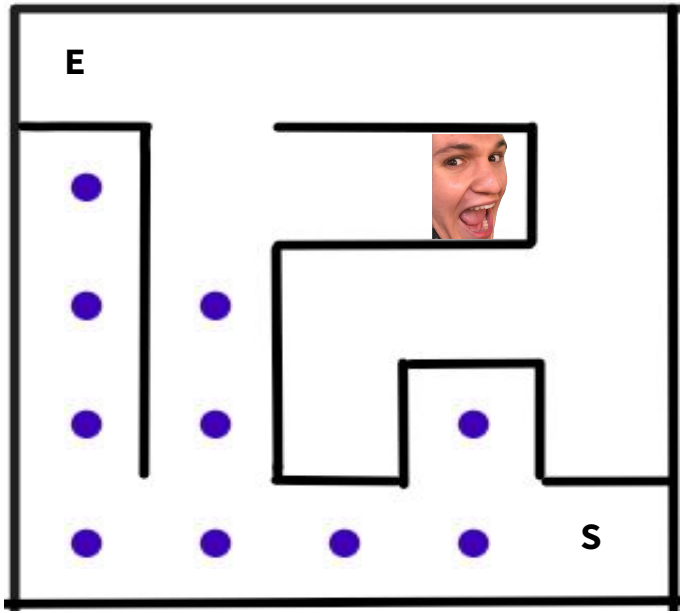
Problema do Labirinto

_ _ _



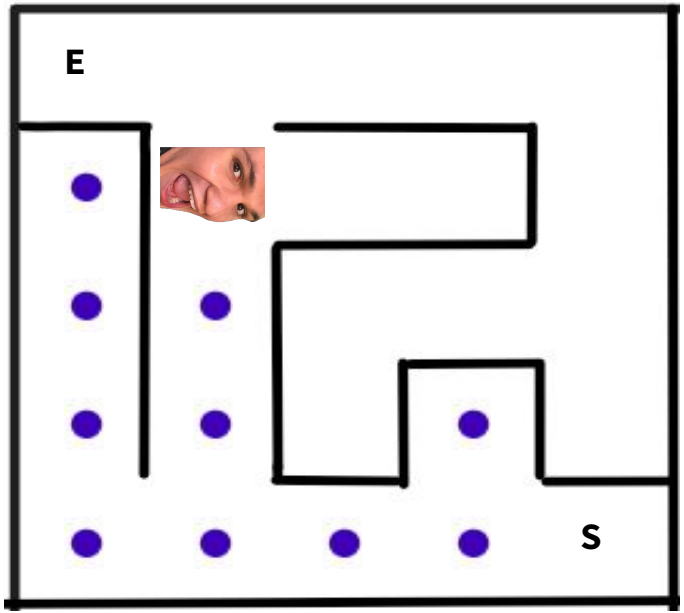
Problema do Labirinto

_ _ _

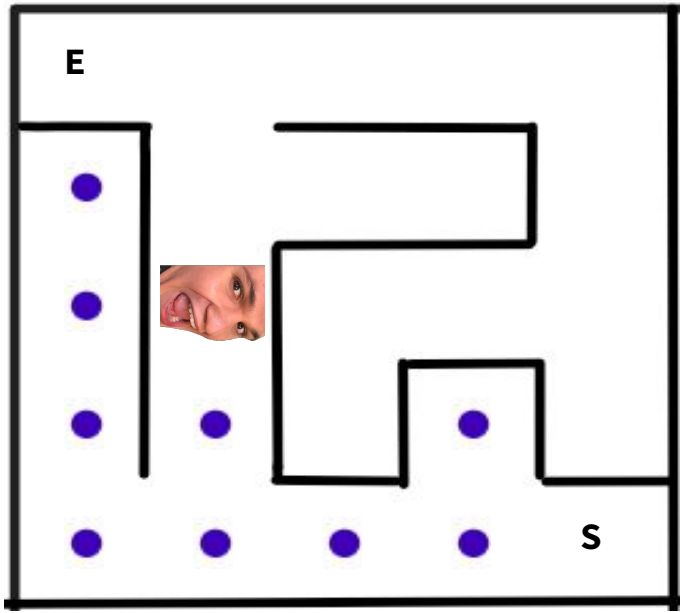


Problema do Labirinto

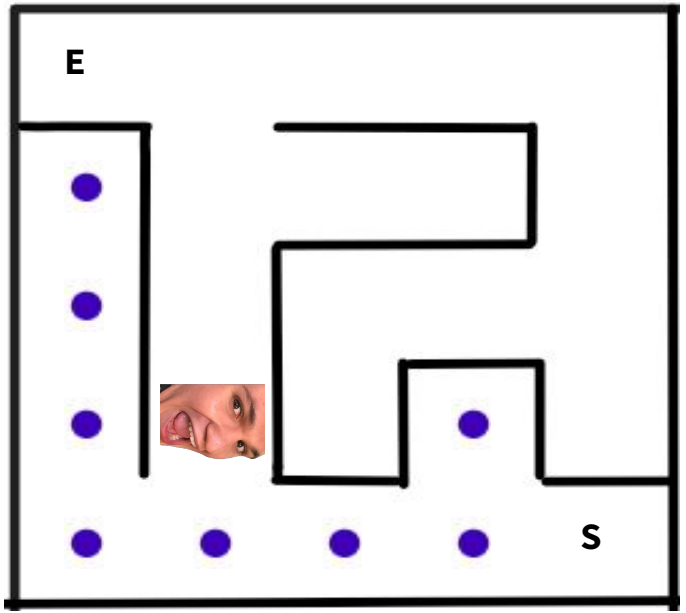
_ _ _



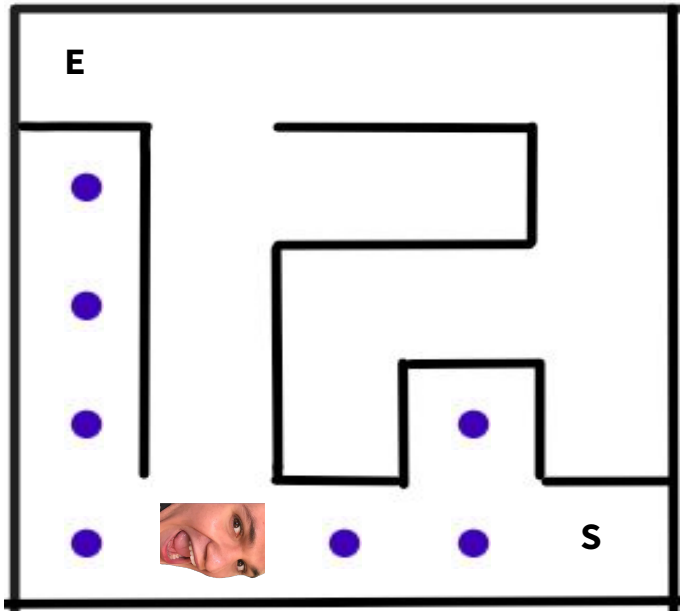
Problema do Labirinto



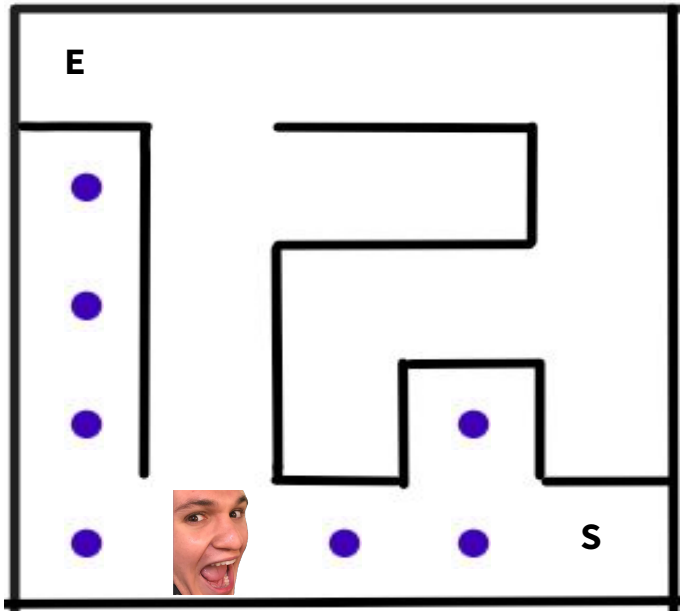
Problema do Labirinto



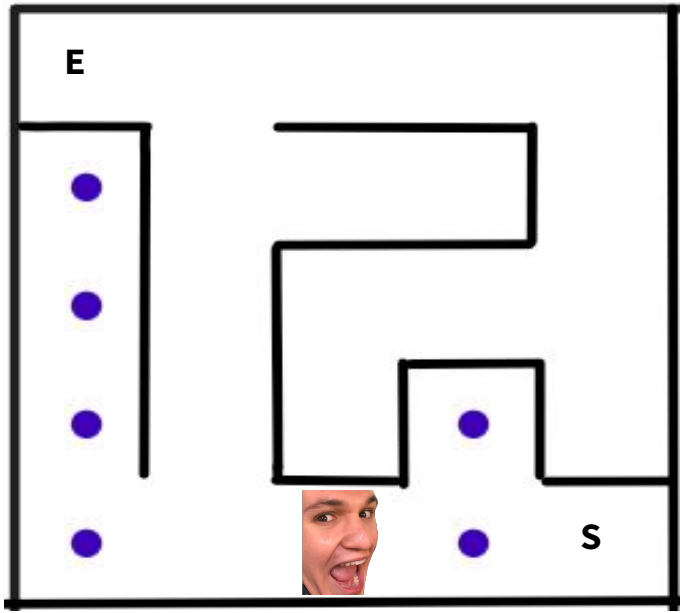
Problema do Labirinto



Problema do Labirinto

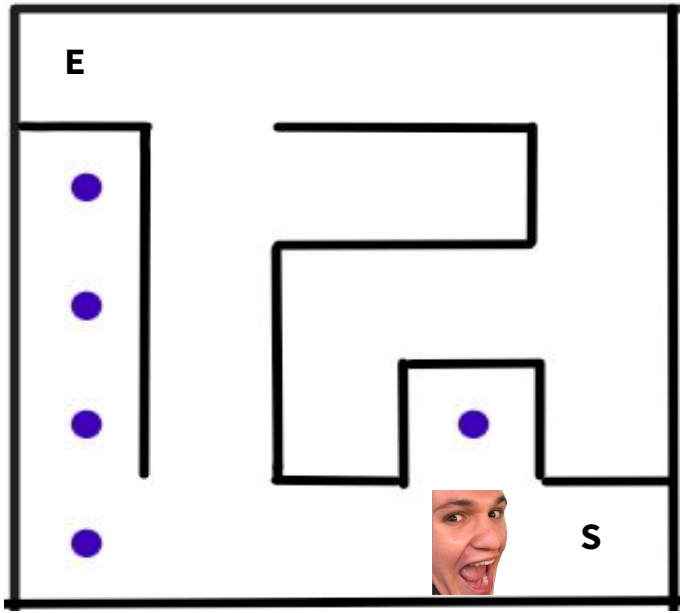


Problema do Labirinto

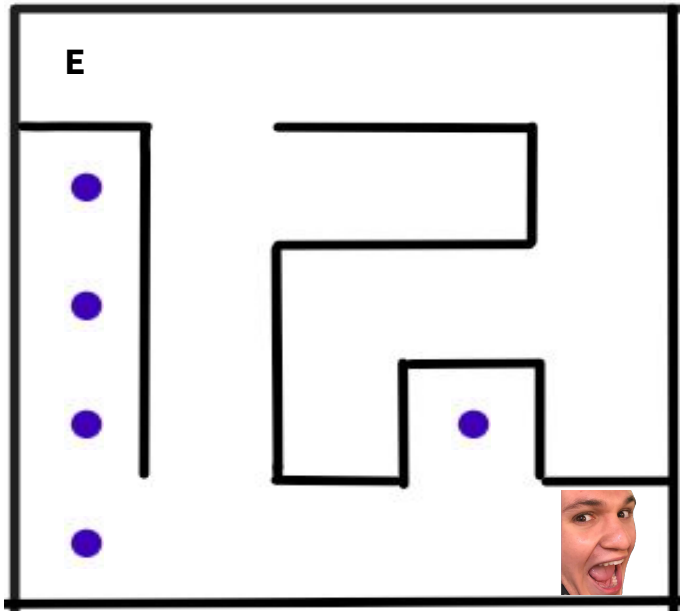


Problema do Labirinto

_ _ _



Problema do Labirinto



```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<pair<ll,ll>> mov = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
vector<vector<bool>> vis;
ll linha_final = 5, coluna_final = 5;

bool flag = false;

void andar(ll lin, ll  col){
    if(flag)return;

    if(lin == linha_final && col == coluna_final){
        flag = true;
        return;
    }
    if(vis[lin][col])return;
    vis[lin][col] = true;

    for(int i = 0; i < mov.size(); i++){
        ll nova_linha, nova_coluna;
        nova_linha = lin + mov[i].first;
        nova_coluna = col + mov[i].second;
        andar(nova_linha, nova_coluna);
    }
    return;
}
```

Sugestão de Problemas

- Grid Paths - CSES - Difícil
- Labyrinth - CSES

Problema do Sudoku

- Problema: dado um Sudoku parcialmente resolvido, determinar se existe solução para aquele problema;

- Vídeo de explicação do problema do Sudoku