

# Programação com retrocesso (backtracking)

Mário César San Felice

Aula 25 de Algoritmos e Estruturas de Dados 1  
DC - UFSCar  
*felice@ufscar.br*

24 de junho de 2019

Dado um problema computacional

Podemos resolvê-lo usando enumeração por força bruta

Ou seja, testar todas as soluções candidatas

Em geral isso é pouco eficiente, pois:

- Verificar se uma solução é viável pode ser custoso
- O número de soluções candidatas costuma ser muito grande

# Técnica: Backtracking

Programação com retrocesso (backtracking) é um método mais eficiente para fazer busca exaustiva

Aplicável quando as soluções candidatas podem ser construídas incrementalmente

Nele, múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas

Ideia central é retroceder quando detectar que a solução candidata é inviável

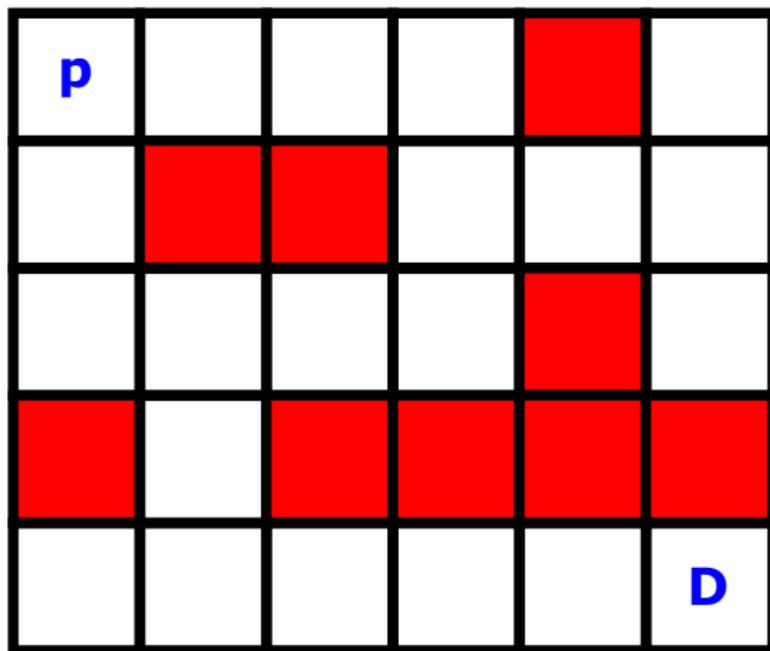
# Algoritmo: Backtracking

Algoritmos com backtracking em geral utilizam *recursão*

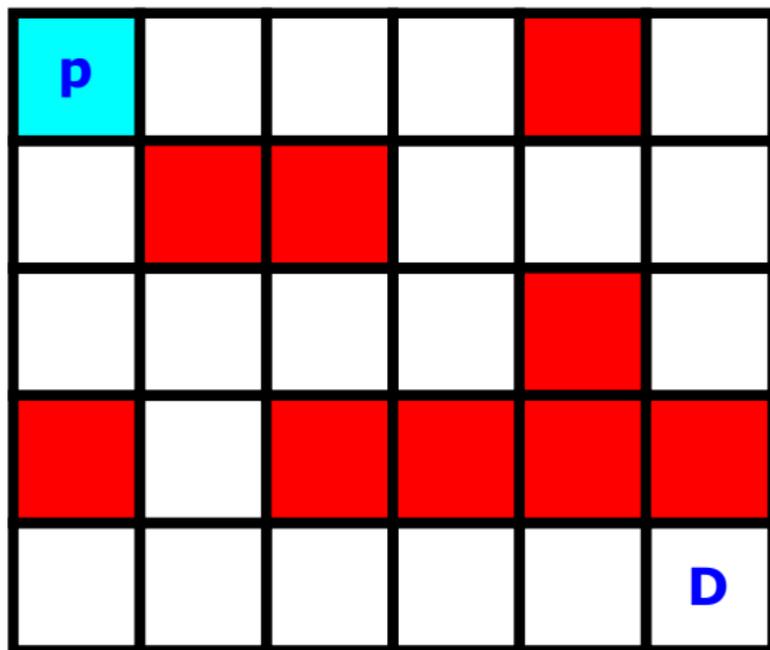
Pois esta facilita o mecanismo de retrocesso

```
backtrackingRec(solParcial p) {  
    se (p é válida) {  
        se (p é solução)  
            imprima p;  
        enquanto (puder estender p para algum s)  
            backtrackingRec(s);  
    }  
}
```

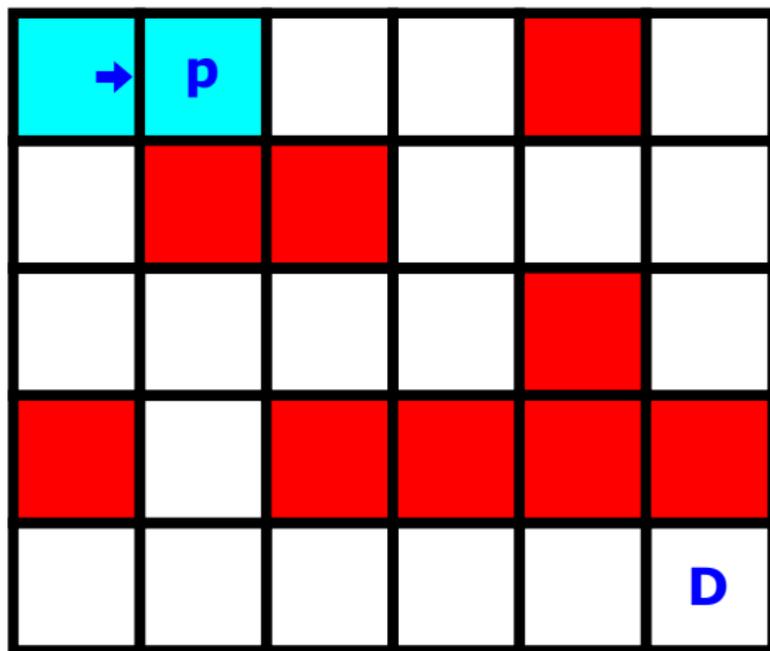
# Exemplo: Labirinto



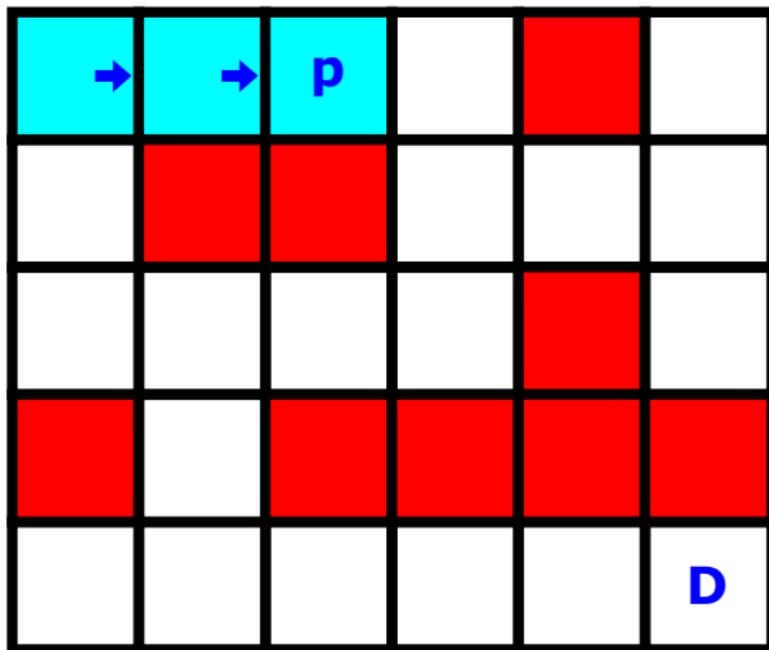
# Exemplo: Labirinto



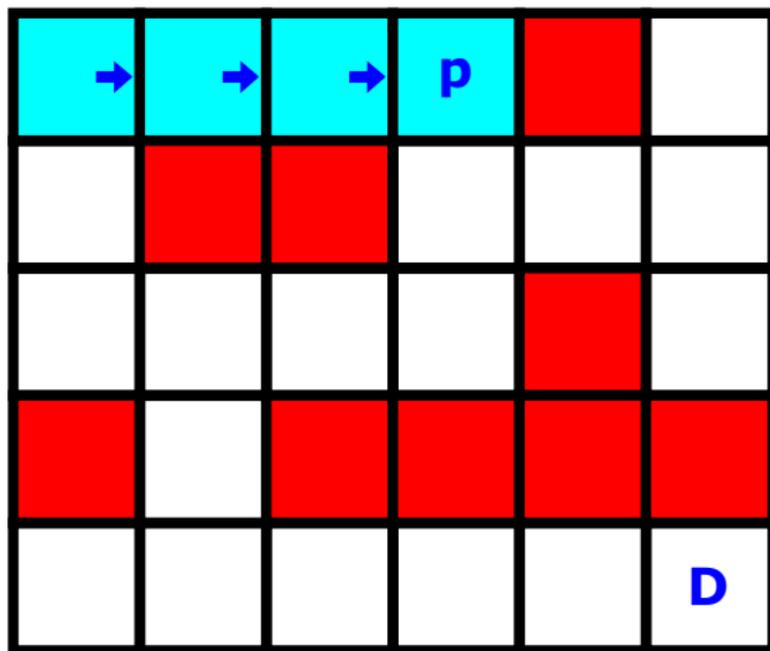
# Exemplo: Labirinto



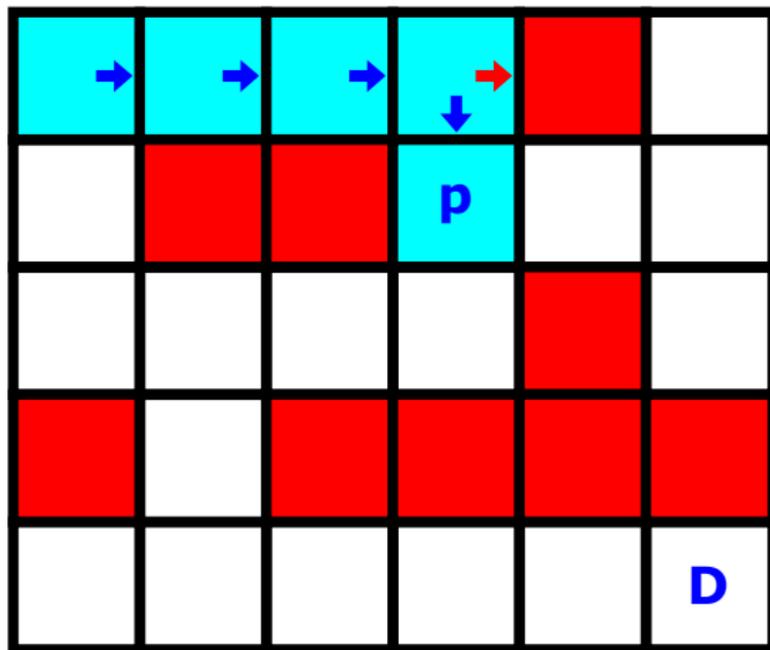
# Exemplo: Labirinto



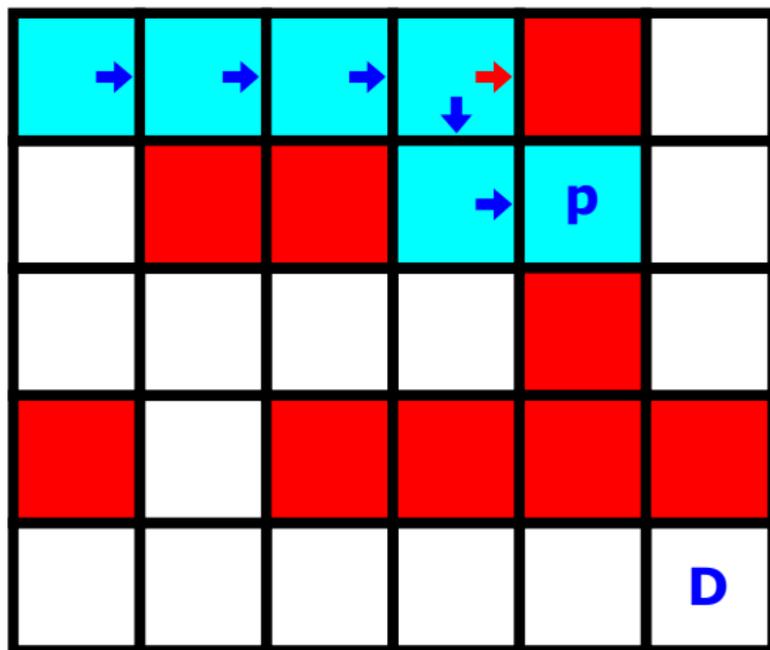
# Exemplo: Labirinto



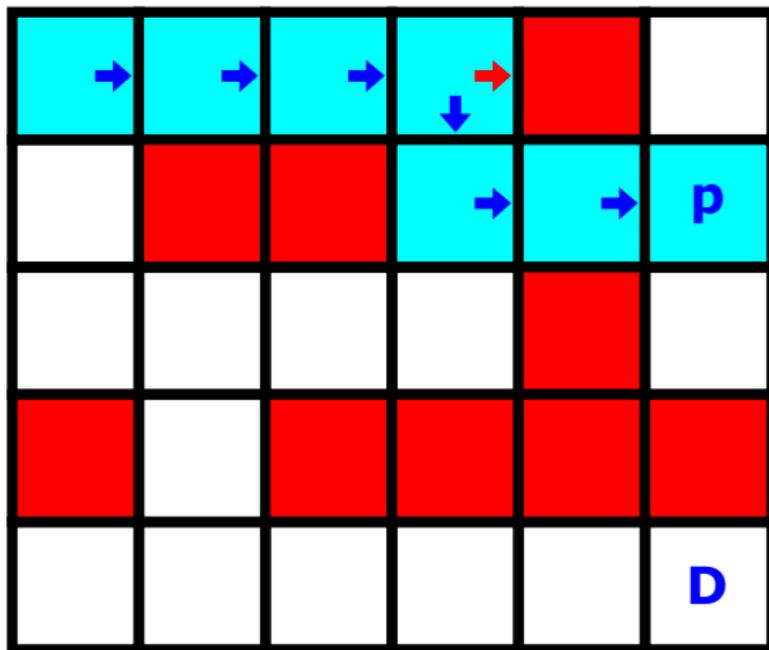
# Exemplo: Labirinto



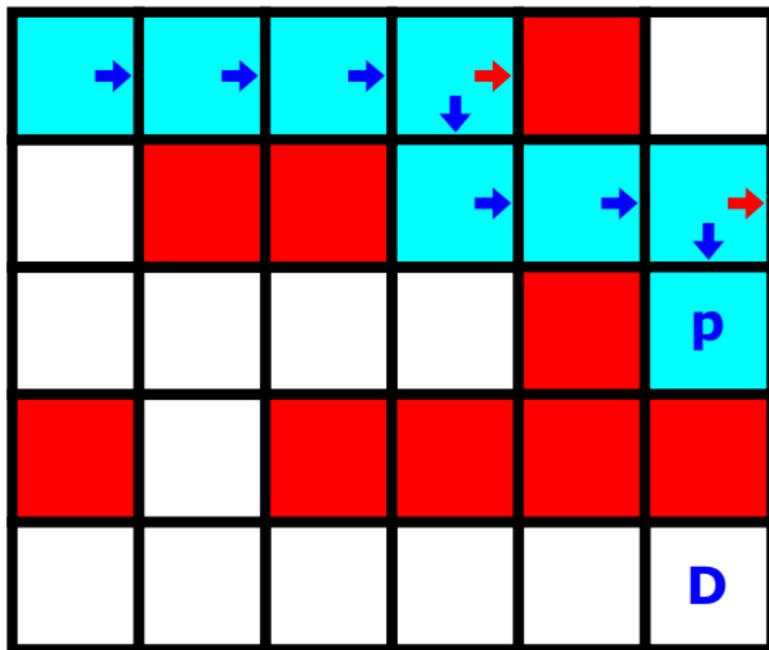
# Exemplo: Labirinto



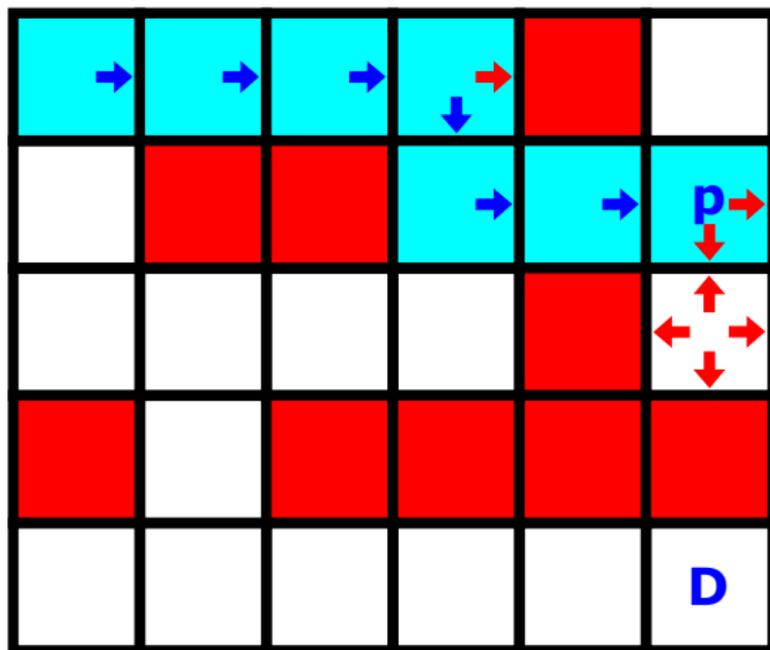
# Exemplo: Labirinto



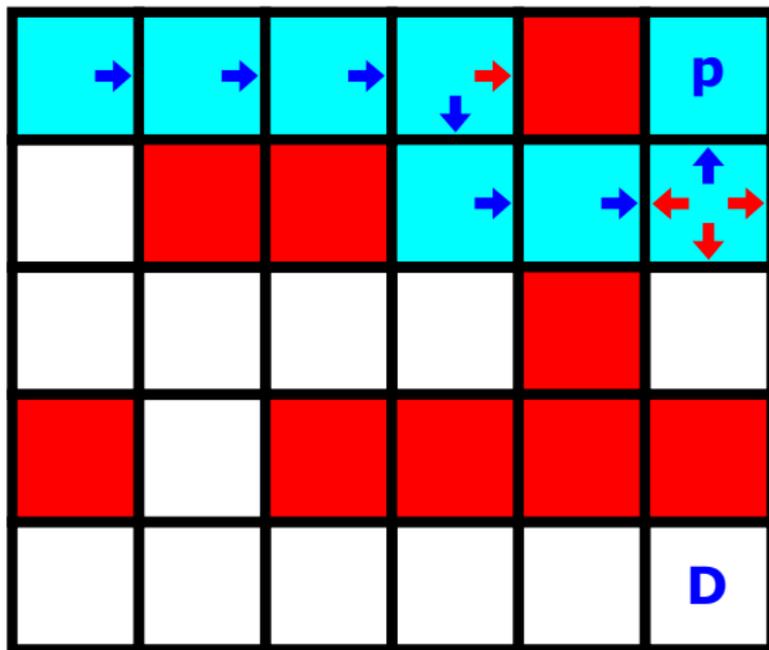
# Exemplo: Labirinto



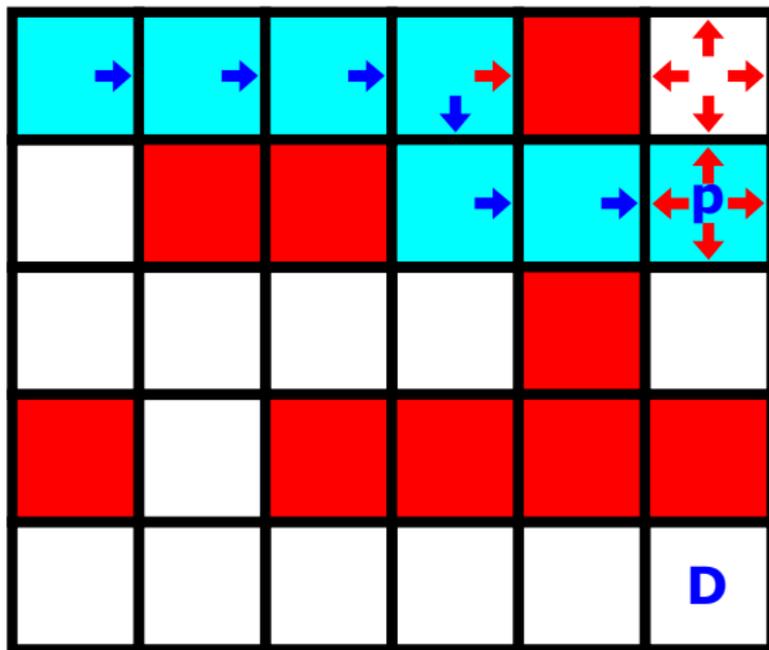
# Exemplo: Labirinto



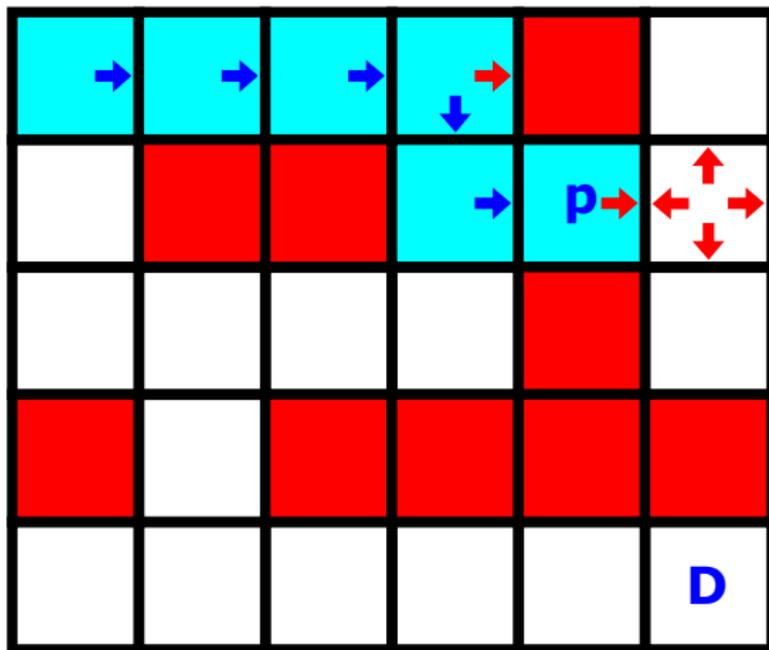
# Exemplo: Labirinto



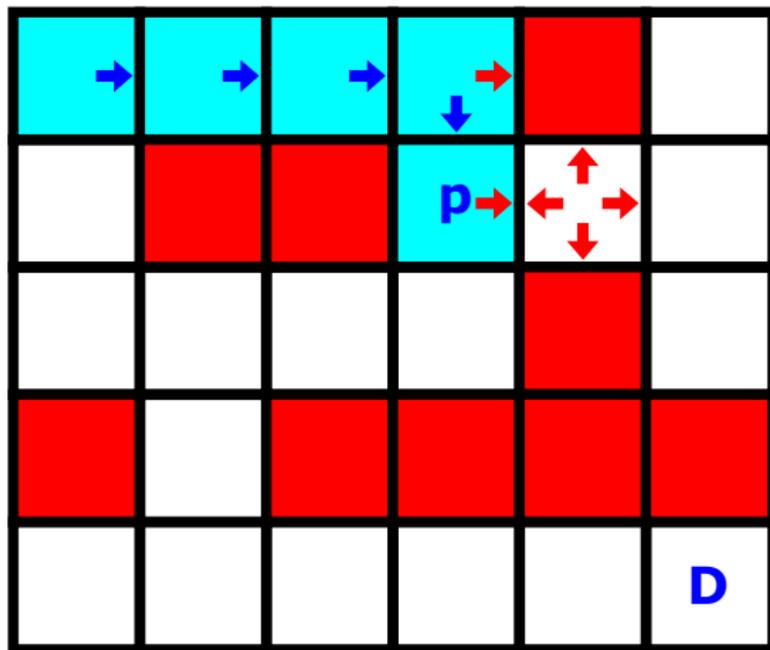
# Exemplo: Labirinto



# Exemplo: Labirinto

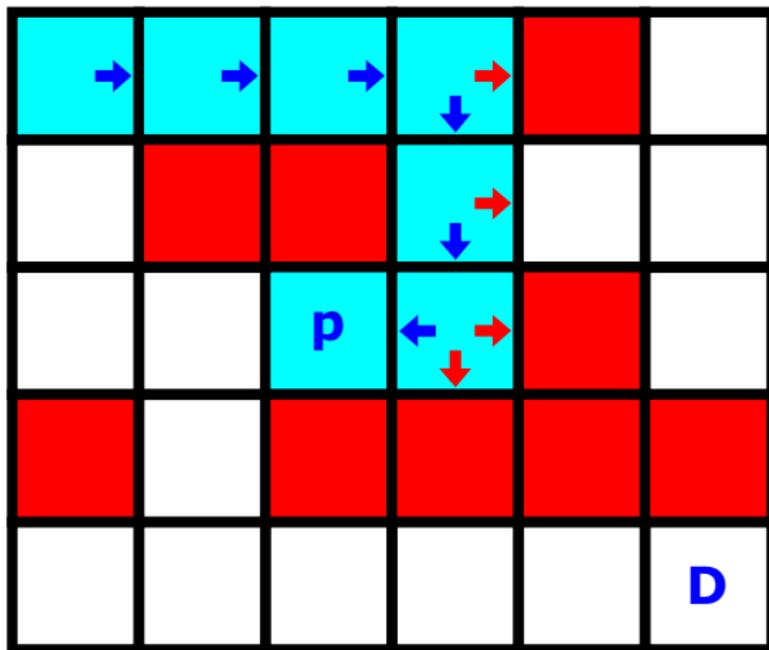


# Exemplo: Labirinto

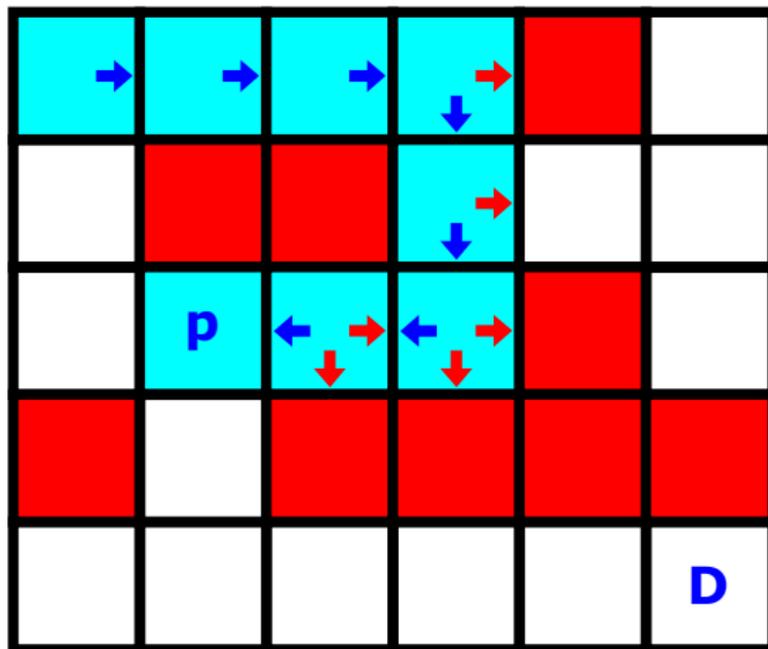




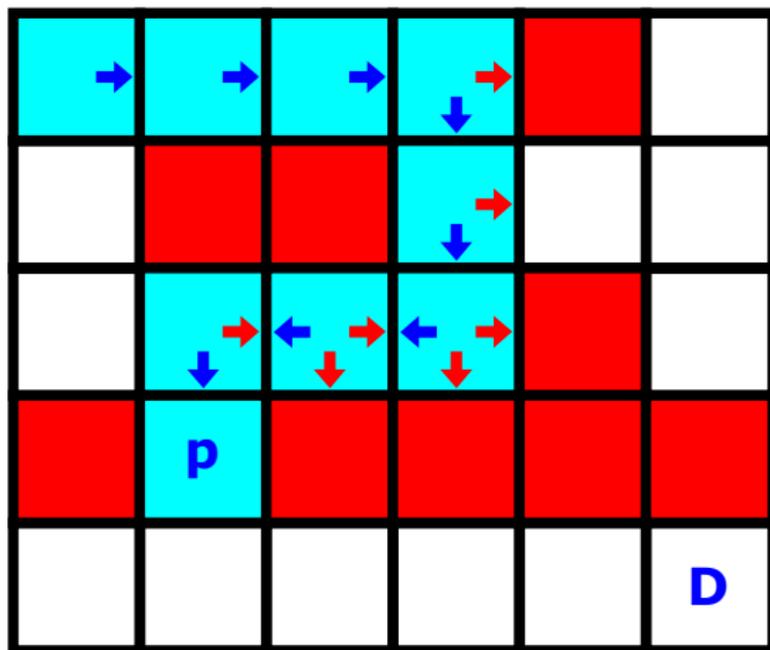
# Exemplo: Labirinto



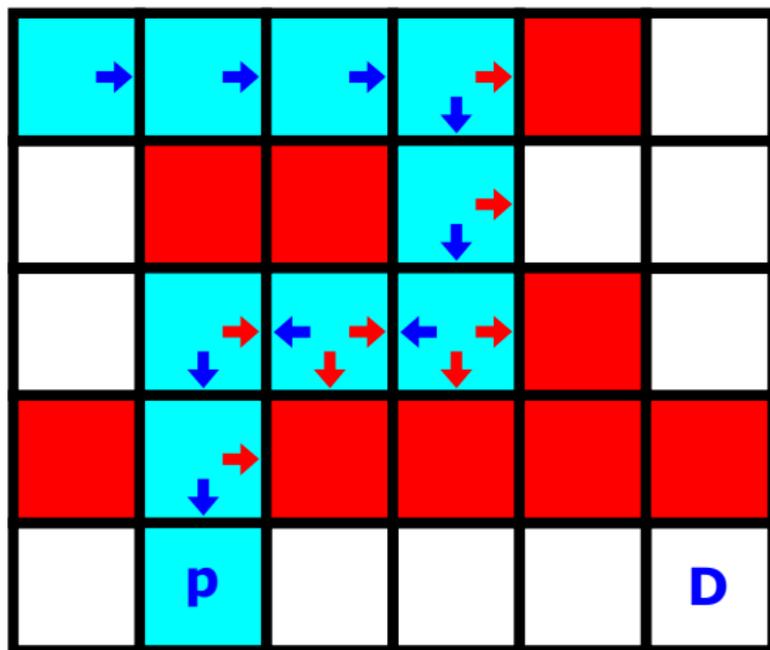
# Exemplo: Labirinto



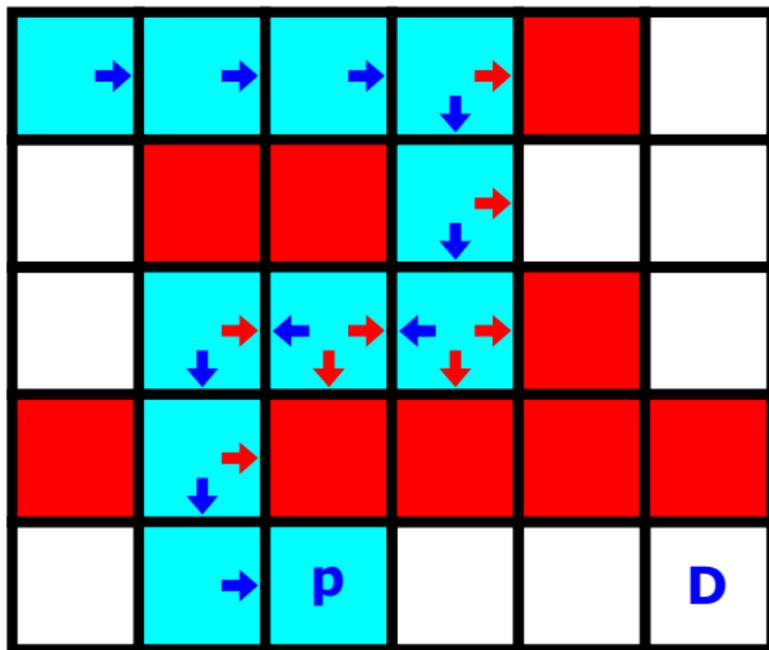
# Exemplo: Labirinto



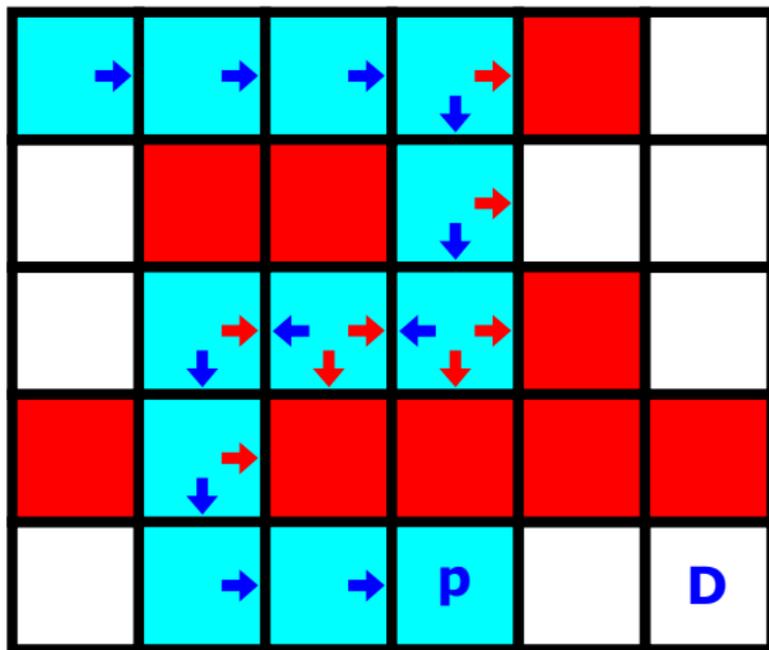
# Exemplo: Labirinto



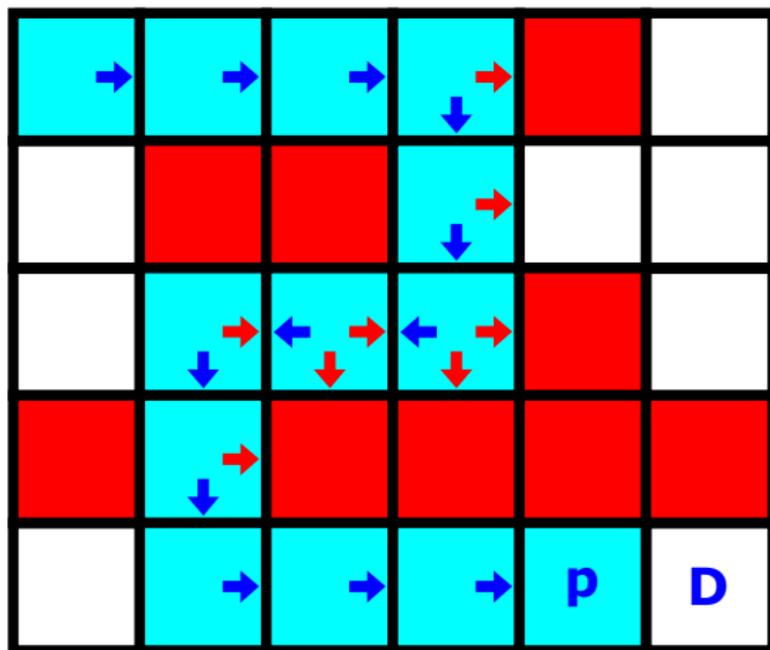
# Exemplo: Labirinto



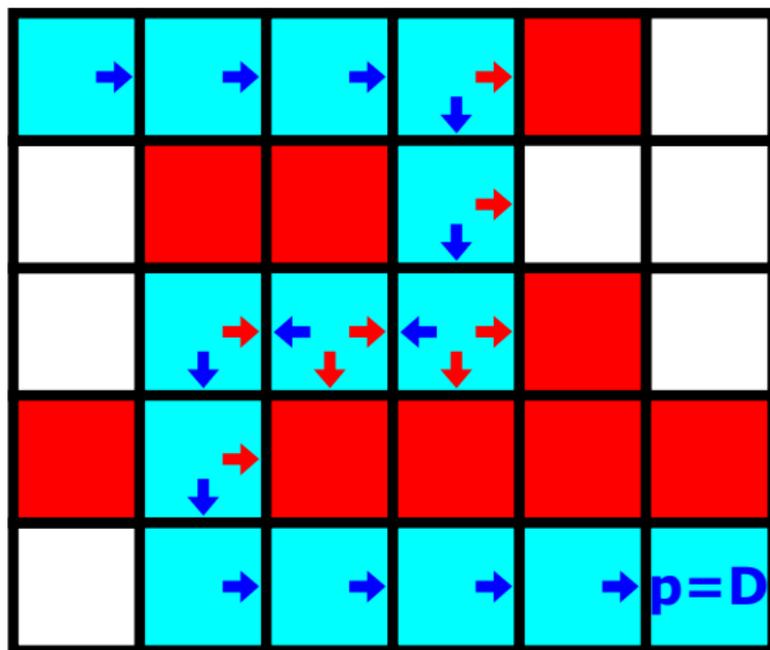
# Exemplo: Labirinto



# Exemplo: Labirinto



# Exemplo: Labirinto



# Algoritmos: Labirinto

```
#define NUM_MOV 4

int desLin[NUM_MOV] = {0, 1, 0, -1};
int desCol[NUM_MOV] = {1, 0, -1, 0};

typedef struct labirinto
{
    int numLins;
    int numCols;
    int **pos;
} * Labirinto;

void imprimeLabirinto(Labirinto lab);
```

# Algoritmos: Labirinto

```
int posValida(Labirinto lab, int lin, int col)
{
    if (lin >= 0 && lin < lab->numLins &&
        col >= 0 && col < lab->numCols)
        if (lab->pos[lin][col] == 0)
            return 1;
    return 0;
}
```

```
int resolveLabirinto(Labirinto lab)
{
    return resolveLabirintoR(lab, 0, 0,
        lab->numLins - 1, lab->numCols - 1);
}
```

# Algoritmos: Labirinto

```
int resolveLabirintoR(Labirinto lab, int lin, int col,
                    int linDest, int colDest) {
    int achou = 0, k = 0;
    if (posValida(lab, lin, col)) {
        lab->pos[lin][col] = 1;
        if (lin == linDest && col == colDest) {
            imprimeLabirinto(lab);
            achou = 1; }
        while (achou == 0 && k < NUM_MOV) {
            achou = resolveLabirintoR(lab,
                lin + desLin[k], col + desCol[k],
                linDest, colDest);
            k++; }
        lab->pos[lin][col] = 0; }
    return achou; }
```

Note que, a primeira solução encontrada nem sempre é a melhor.

*Exercício:* Modificar o algoritmo anterior para encontrar todas as soluções possíveis.

*Desafio:* Modificar o algoritmo anterior para resolver o problema dos movimentos dos cavalos. (Dica: é necessário modificar o padrão dos movimentos e o critério de sucesso.)

# Observações: Backtracking

Numa enumeração por força bruta pura, um algoritmo poderia

- considerar todas as concatenações possíveis de posições do labirinto,
- e descartar cada uma que não corresponda a um caminho válido da origem ao destino.

Isso é muito ineficiente, pois só o número de concatenações distintas já corresponde a um fatorial do tamanho dos caminhos.

# Observações: Backtracking

Backtracking é mais eficiente que enumeração por força bruta, pois:

- sempre parte de soluções parciais viáveis e
- quando detecta que uma solução parcial é inviável,
- essa parcial é descartada, eliminando assim diversas soluções candidatas inviáveis que derivam da parcial.

Ainda assim, como o número de soluções candidatas costuma ser muito grande, para muitos problemas backtracking ainda é ineficiente, particularmente para instâncias grandes.

No caso do labirinto, o número de soluções cresce exponencialmente no número de posições do labirinto. Aproximadamente  $O(3^{(nm)})$ .

# Observações: Backtracking

É importante destacar que, o caminho que as soluções parciais percorrem

- nem sempre corresponde a um caminho no problema sendo resolvido,
- mas é um caminho lógico na árvore de decisão sendo percorrida.

Nosso próximo exemplo deixa isso mais claro.

# Exemplo: Quadrado Latino

<b>b</b>			<b>d</b>
	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>

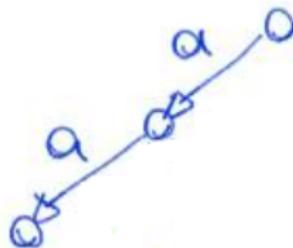
# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>		<b>d</b>
	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>



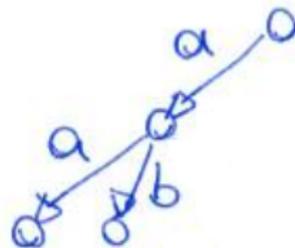
# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>a</b>	<b>d</b>
	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>



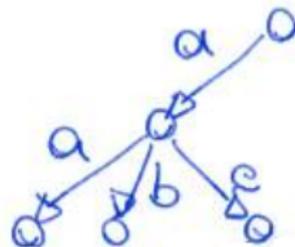
# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>b</b>	<b>d</b>
	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>



# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>c</b>	<b>d</b>
	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>







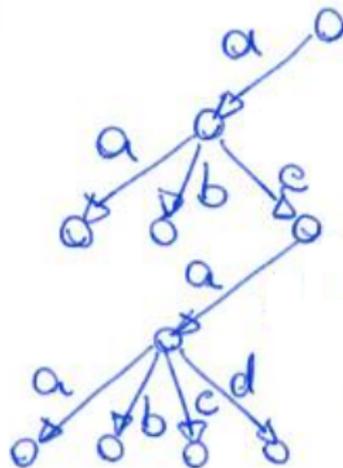






# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>c</b>	<b>d</b>
<b>a</b>	<b>d</b>	<b>b</b>	
	<b>c</b>	<b>d</b>	
<b>d</b>			<b>c</b>







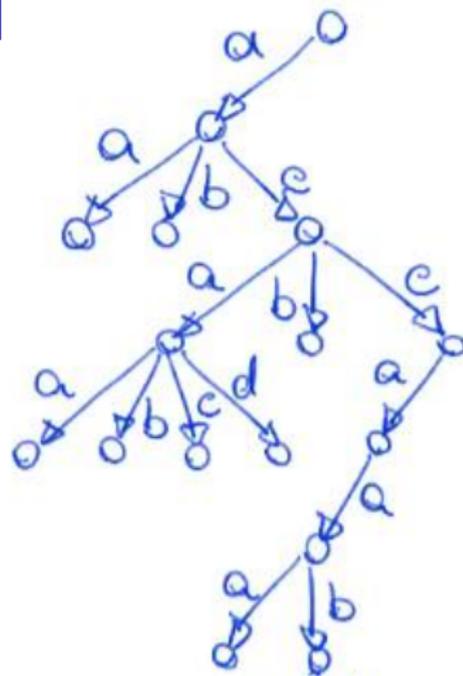






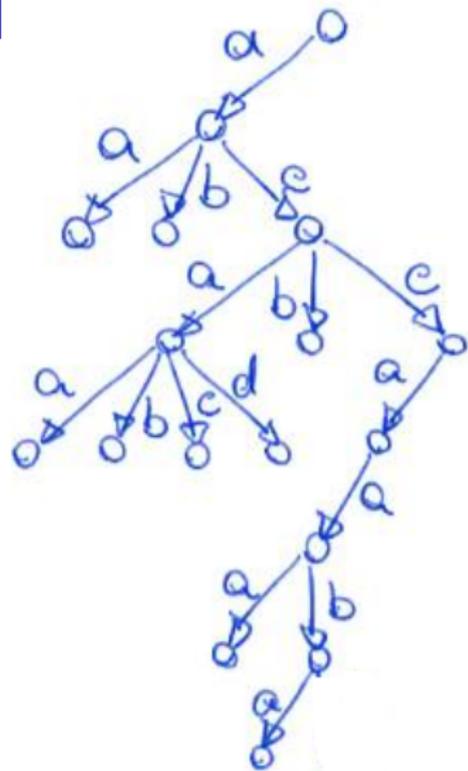
# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>c</b>	<b>d</b>
<b>c</b>	<b>d</b>	<b>b</b>	<b>a</b>
<b>a</b>	<b>c</b>	<b>d</b>	<b>b</b>
<b>d</b>			<b>c</b>



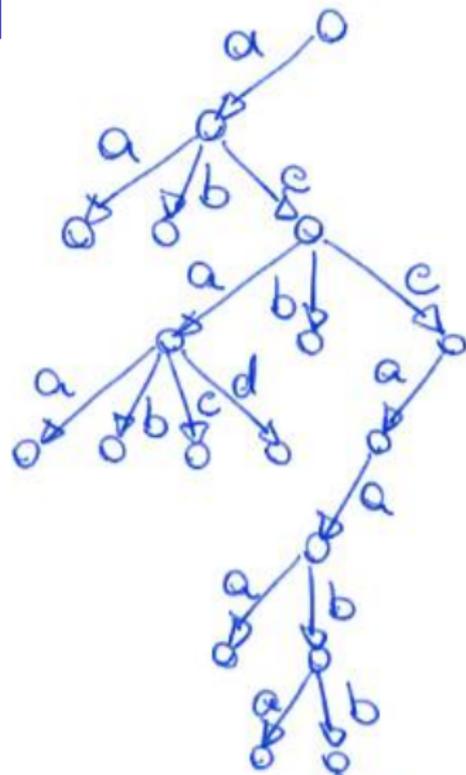
# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>c</b>	<b>d</b>
<b>c</b>	<b>d</b>	<b>b</b>	<b>a</b>
<b>a</b>	<b>c</b>	<b>d</b>	<b>b</b>
<b>d</b>	<b>a</b>		<b>c</b>



# Exemplo: Quadrado Latino

<b>b</b>	<b>a</b>	<b>c</b>	<b>d</b>
<b>c</b>	<b>d</b>	<b>b</b>	<b>a</b>
<b>a</b>	<b>c</b>	<b>d</b>	<b>b</b>
<b>d</b>	<b>b</b>		<b>c</b>





# Algoritmos: Quadrado Latino

```
typedef struct matriz
{
    int numLins;
    int numCols;
    int **pos;
} * Matriz;
```

```
// funções auxiliares
```

```
void imprimeQLatino(Matriz m);
int qLatinoResolvido(Matriz m);
int proxPosLin(Matriz m, int lin, int col);
int proxPosCol(Matriz m, int lin, int col);
```

# Algoritmos: Quadrado Latino

```
int atribValida(Matriz m, int lin, int col, int l)
{
    for (int i = 0; i < m->numLins; i++)
        if (m->pos[i][col] == l ||
            m->pos[lin][i] == l)
            return 0;
    return 1;
}

// função inicial que chama a principal
int resolveQLatino(Matriz m);
```

# Algoritmos: Quadrado Latino

```
int qLatR(Matriz m, int lin, int col, int l) {
    int achou = 0, k = 1;
    if (atribValida(m, lin, col, l)) {
        m->pos[lin][col] = 1;
        if (qLatinoResolvido(m)) {
            imprimeQLatino(m);
            achou = 1; }
        while (achou == 0 && k <= m->numLins) {
            achou = qLatR(m,
                proxPosLin(m, lin, col),
                proxPosCol(m, lin, col), k);
            k++; }
        m->pos[lin][col] = 0; }
    return achou; }
```

# Algoritmos: Quadrado Latino

*Eficiência:* o número de chamadas da função recursiva `qLatR` pode chegar a  $O(n^{(n^2)})$ , sendo

- o  $n$  da base o número de valores testados por posição vazia
- e o  $(n^2)$  no expoente o número de posições vazias possíveis.

Na prática, espera-se que as podas na árvore de decisão reduzam significativamente esse valor.

*Exercício:* Implementar as funções auxiliares e a função inicial do algoritmo anterior.

*Desafio:* Modificar o algoritmo anterior para resolver o Sudoku. Qual função deve ser modificada?

# Curiosidades: Backtracking

Backtracking é a base do branch-and-bound, uma técnica importante para resolver problemas de otimização discreta.

Backtracking pode ser utilizada para resolver problemas de satisfação de restrições e puzzles, como:

- Problema das 8 rainhas
- Palavras-cruzadas
- Criptoaritmética

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array}$$