

ACH2002

Aula 13

Técnicas de Desenvolvimento
de Algoritmos -
**Backtracking (tentativa e
erro)**

Aulas passadas

- Técnicas de programação:
 - **Divisão e conquista:**
 - **Programação dinâmica:**
 - **Algoritmos gulosos:**

Aulas passadas

- Técnicas de programação:
 - **Divisão e conquista**: solução do problema original pode ser obtida combinando soluções de subproblemas preferencialmente disjuntos (indução fraca)
 - **Programação dinâmica**: solução do problema original pode ser obtida combinando soluções de subproblemas que se sobrepõem – para evitar reexecuções precisa armazenar soluções em uma estrutura de dados (normalmente uma matriz) (indução forte)
 - **Algoritmos gulosos**: quando sempre pegar a próxima melhor escolha compõe a solução ótima global

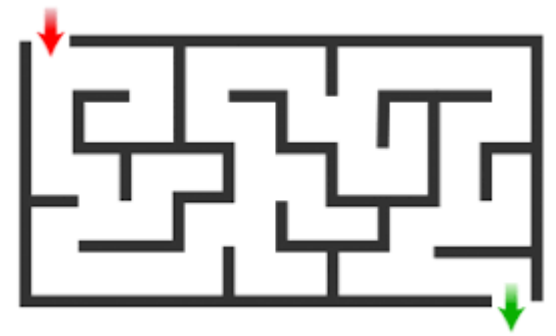
Aula de hoje

- **Backtracking (tentativa e erro)**

Tentativa e erro - conceitos

- Como o próprio nome diz:
 - tentar enquanto estiver errado
- Técnica de projetos de algoritmos:
 - útil para problemas cuja **solução consiste em tentar (potencialmente) todas alternativas propostas;**

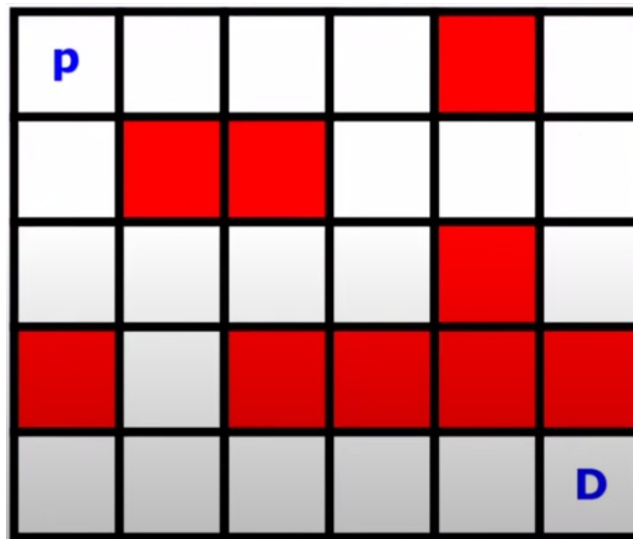
Tentativa e erro - exemplo do labirinto



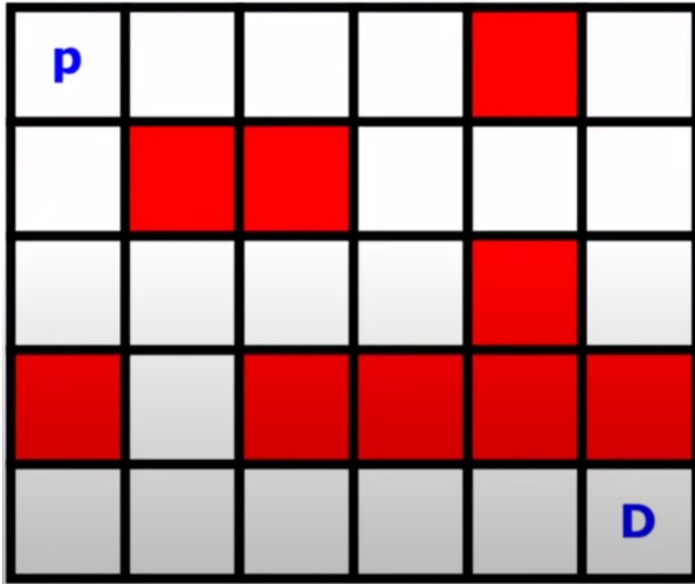
- Dado um **labirinto**, encontre um caminho da entrada à saída:
 - em cada interseção, você tem que decidir se:
 - segue direto;
 - vira à esquerda;
 - vira à direita;
 - você não tem informação suficiente para escolher corretamente;
 - cada escolha leva a outro conjunto de escolhas
 - uma ou mais sequências de escolhas pode ser a solução.

Tentativa e erro - exemplo do labirinto

- Você deseja buscar um elemento qualquer dentro de uma região (representada por uma matriz):
 - existem posições inacessíveis;
 - você desconhece a posição do elemento buscado;
 - em cada iteração, você deve decidir para onde ir na matriz;
 - você não tem informação suficiente para escolher a direção;
 - cada escolha leva a outro conjunto de escolhas;
 - uma ou mais sequência de passos pode ser a solução.



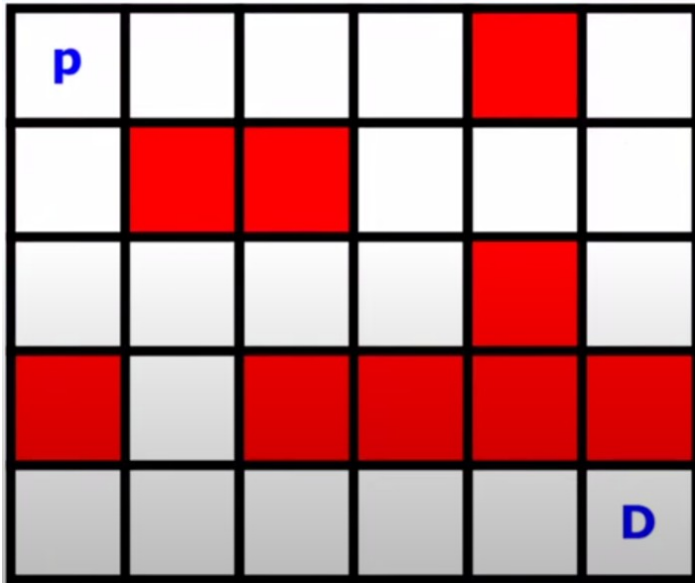
Tentativa e erro - exemplo do labirinto



- O que seria um algoritmo exaustivo (força bruta) para este problema?

Cada célula: → ↓ ← ↑

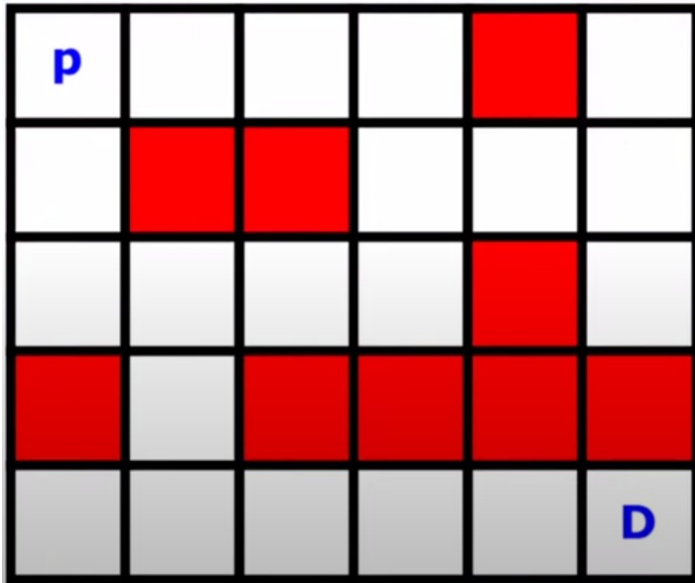
Tentativa e erro - exemplo do labirinto



- O que seria um algoritmo exaustivo (força bruta) para este problema?
 - Testar todos os possíveis caminhos, ou seja, todas as possíveis combinações de setinhas

Cada célula: → ↓ ← ↑

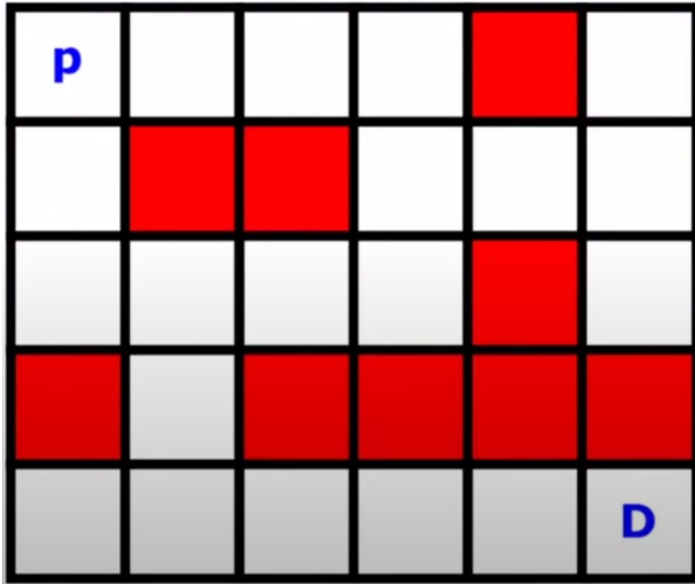
Tentativa e erro - exemplo do labirinto



Cada célula: → ↓ ← ↑

- O que seria um algoritmo exaustivo (força bruta) para este problema?
 - Testar todos os possíveis caminhos, ou seja, todas as possíveis combinações de setinhas
- Qual seria a complexidade, para uma matriz $n \times m$?
 - Qual o nr de possíveis caminhos?
 - Qual a complexidade de testar cada um?
 - Total:

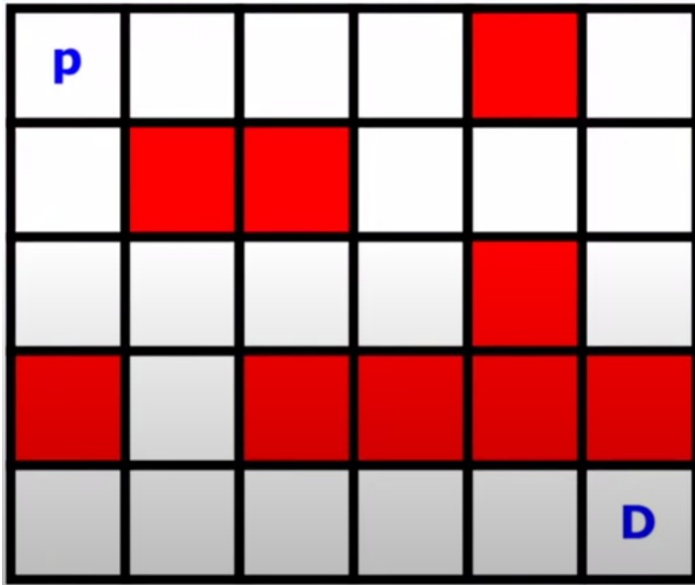
Tentativa e erro - exemplo do labirinto



- O que seria um algoritmo exaustivo (força bruta) para este problema?
 - Testar todos os possíveis caminhos, ou seja, todas as possíveis combinações de setinhas
- Qual seria a complexidade, para uma matriz $n_x m$?
 - Qual o nr de possíveis caminhos?
 $O(4^{n_x m}) = O(2^{n_x m})$
 - Qual a complexidade de testar cada um?
 - Total:

Cada célula: → ↓ ← ↑

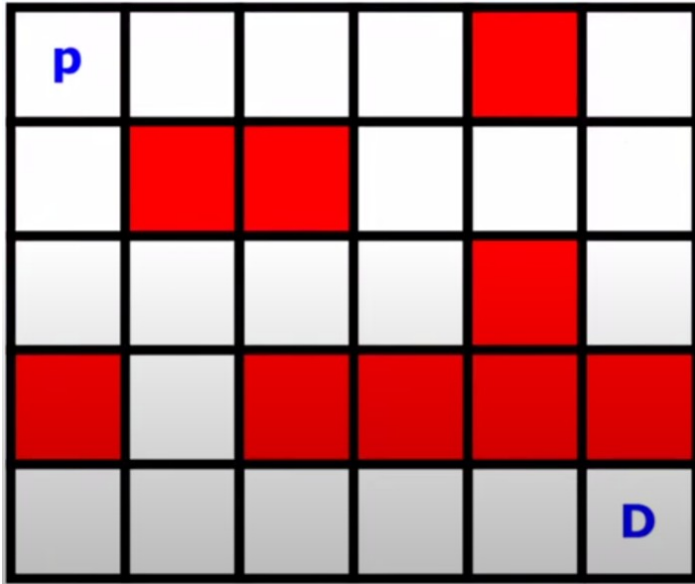
Tentativa e erro - exemplo do labirinto



Cada célula: → ↓ ← ↑

- O que seria um algoritmo exaustivo (força bruta) para este problema?
 - Testar todos os possíveis caminhos, ou seja, todas as possíveis combinações de setinhas
- Qual seria a complexidade, para uma matriz $n_x m$?
 - Qual o nr de possíveis caminhos?
 $O(4^{n_x m}) = O(2^{n_x m})$
 - Qual a complexidade de testar cada um?
 $O(n_x m)$
 - Total: $O(2^{n_x m})$

Tentativa e erro - exemplo do labirinto



- O que seria um algoritmo exaustivo (força bruta) para este problema?
 - Testar todos os possíveis caminhos, ou seja, todas as possíveis combinações de setinhas
- Qual seria a complexidade, para uma matriz $n_x m$?
 - Qual o nr de possíveis caminhos?
 $O(4^{n_x m}) = O(2^{n_x m})$
 - Qual a complexidade de testar cada um?
 $O(n_x m)$
 - Total: $O(2^{n_x m})$

Cada célula: → ↓ ← ↑

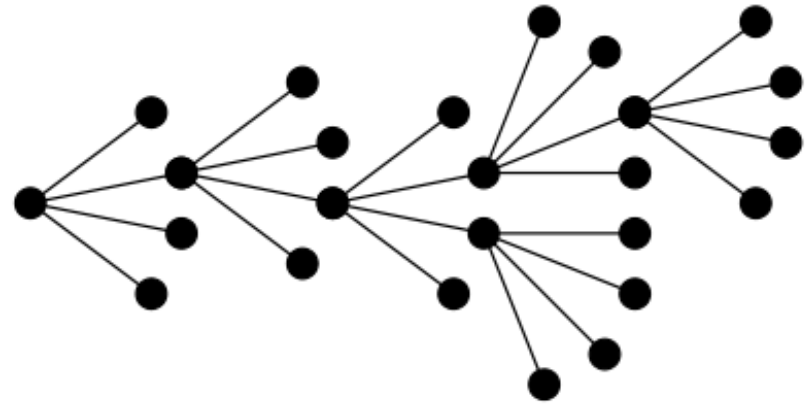
Dá para fazer algo um pouco melhor?

Podemos, durante a construção de um caminho, perceber que ele “vai dar ruim” e desistir de segui-lo, voltando um passo para trás para tentar em outra direção.

Isso é backtracking.

Tentativa e erro - conceitos

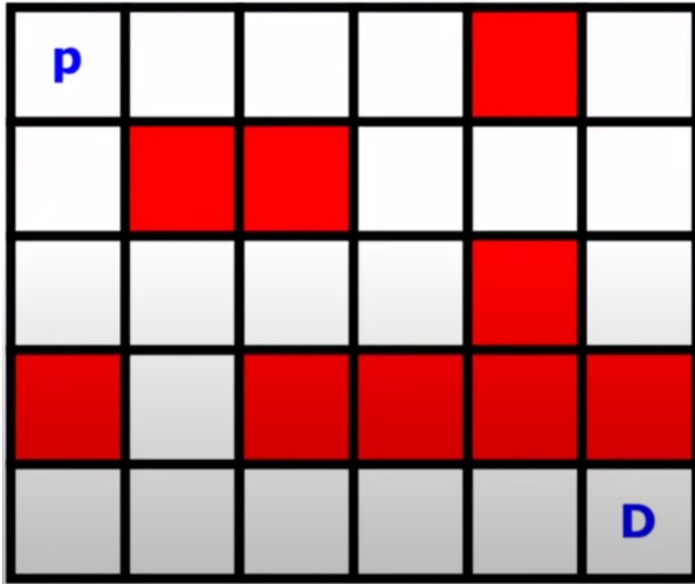
- Como o próprio nome diz: tentar enquanto estiver errado
- Técnica de projetos de algoritmos:
 - útil para problemas cuja **solução consiste em tentar (potencialmente) todas alternativas propostas**;
 - então o problema se resume a: como organizar/explorar todas as alternativas?
 - idéia: decompor o processo em um número finito de subtarefas que devem ser exploradas;
 - processo geral pode ser visto com processo de tentativas que constrói e percorre uma árvore de subtarefas.



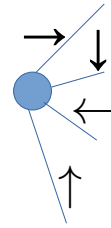
Tentativa e erro - conceitos

- Algoritmos com esta técnica não têm uma regra fixa de computação.
- Funcionamento geral:
 - passos para obtenção da solução final são tentados e registrados;
 - caso um passo não leve à solução final, são retirados e apagados do registro.
- Pesquisa na árvore de solução: muitas vezes tem crescimento exponencial
 - Nestes casos pode-se usar heurísticas e/ou algoritmos aproximados que podem não garantir uma solução ótima, mas rápida (subótima)

Tentativa e erro - exemplo do labirinto



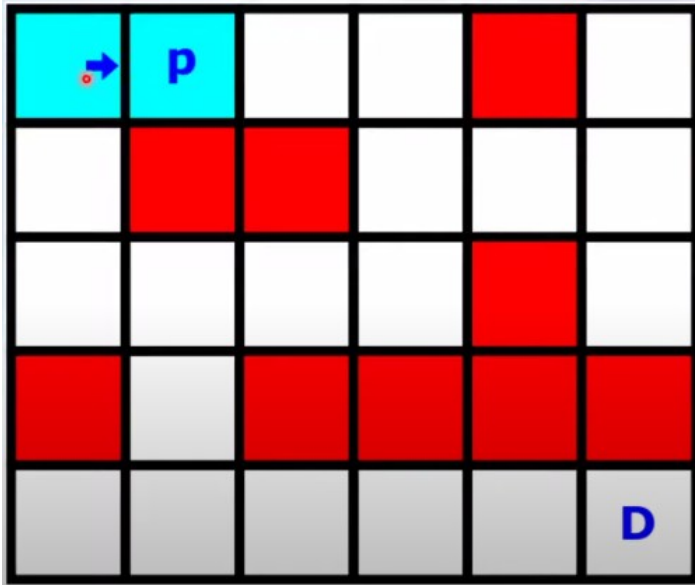
Estratégia: → ↓ ← ↑



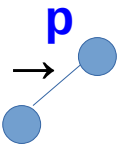
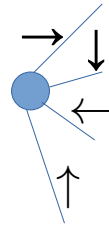
<https://www.youtube.com/watch?v=iVsN6ZnVx4Q&t=650s>

Do instante 14:29 até 17:33

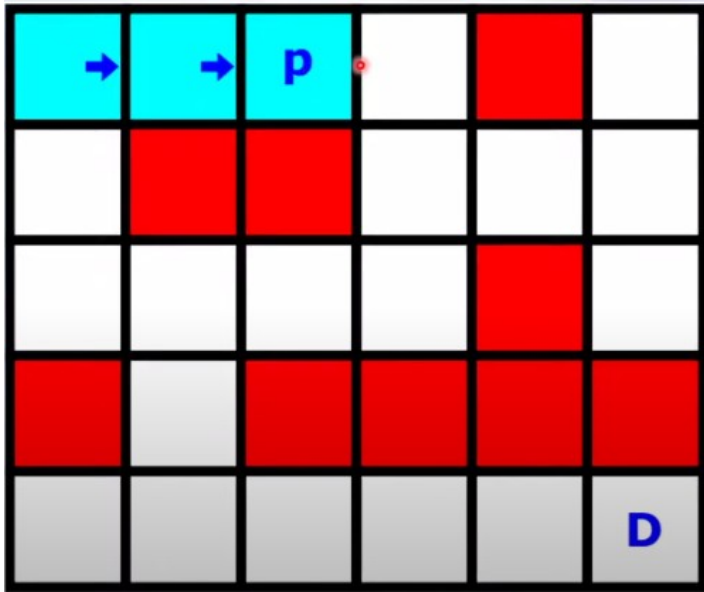
Tentativa e erro - exemplo do labirinto



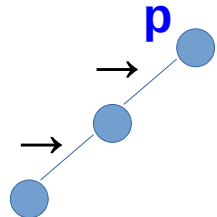
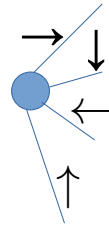
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



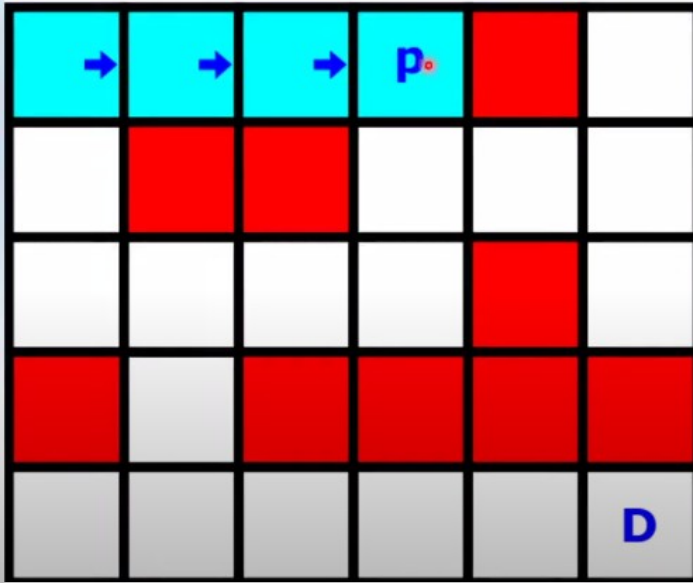
Tentativa e erro - exemplo do labirinto



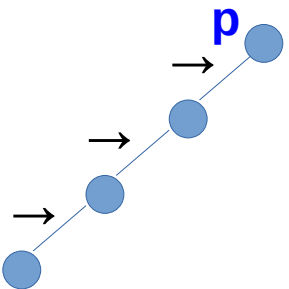
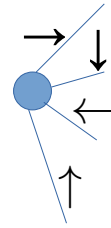
Estratégia: → ↓ ← ↑



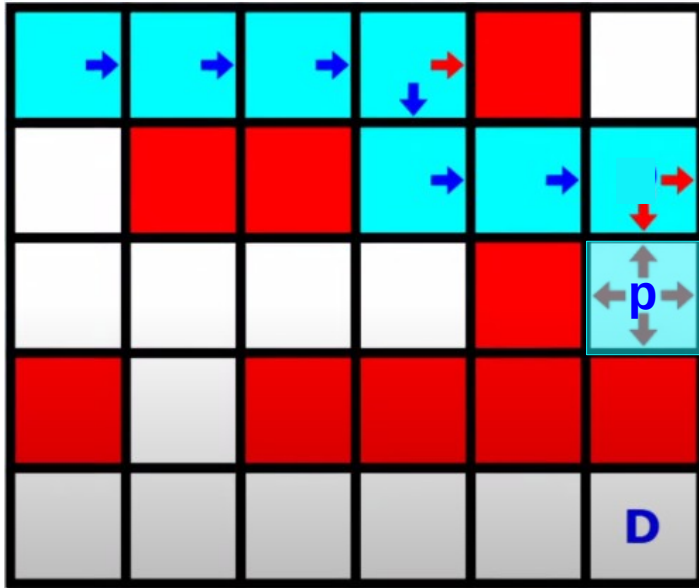
Tentativa e erro - exemplo do labirinto



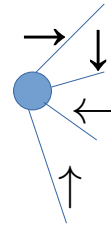
Estratégia: → ↓ ← ↑



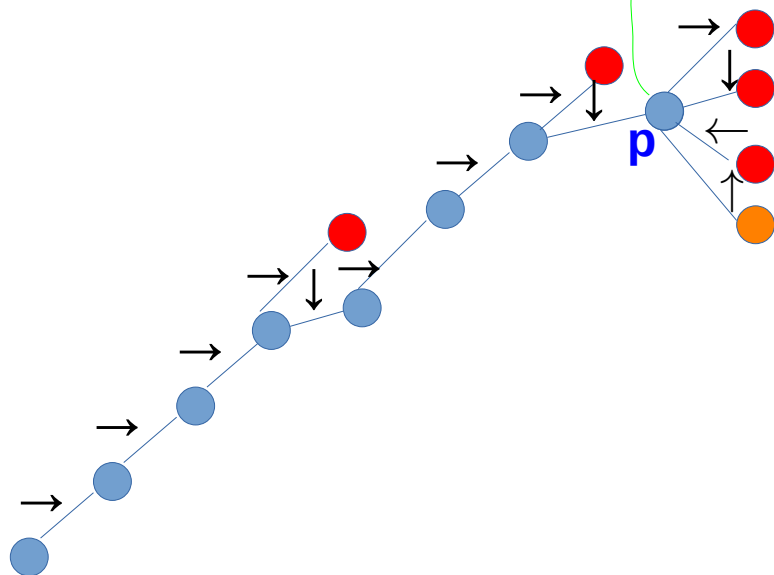
Tentativa e erro - exemplo do labirinto



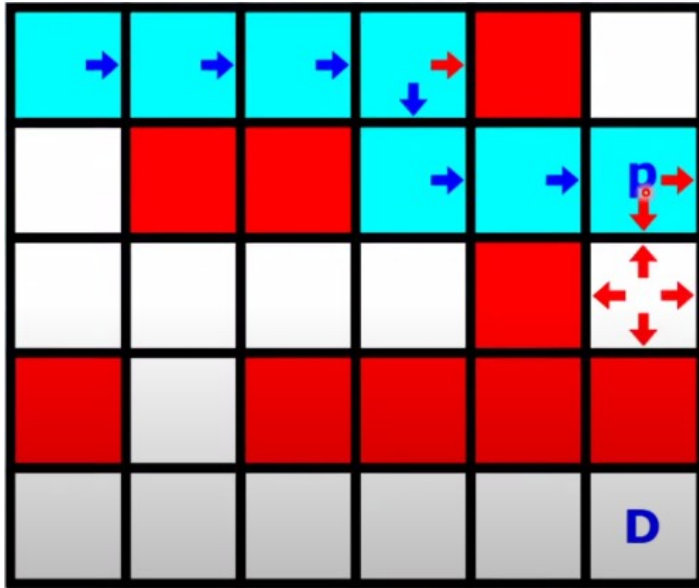
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



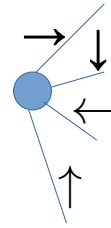
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



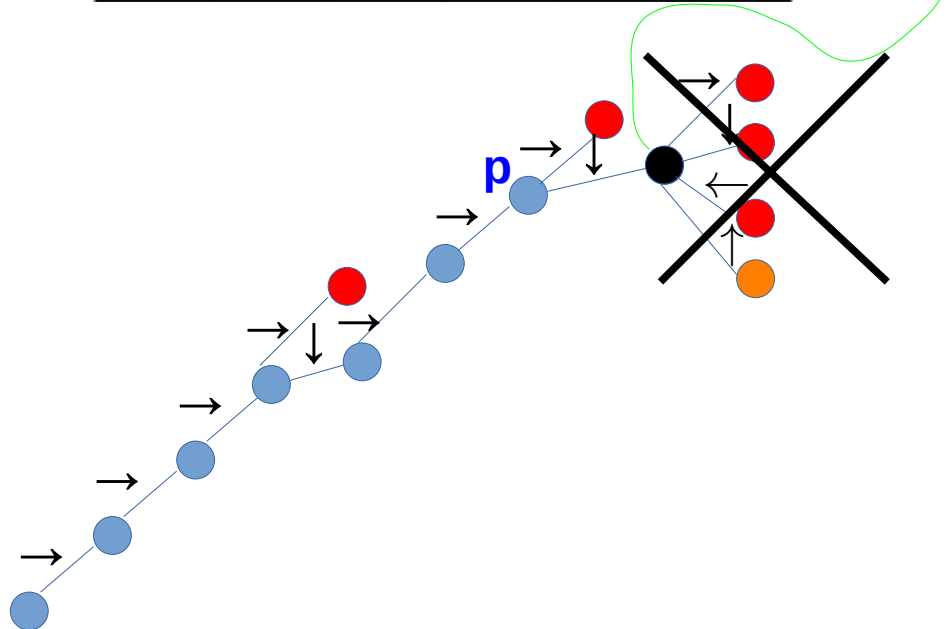
Tentativa e erro - exemplo do labirinto



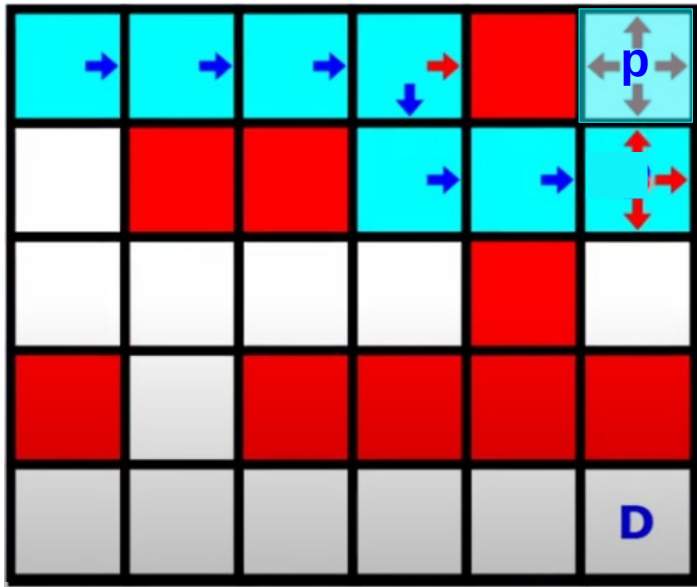
Estratégia: → ↓ ← ↑



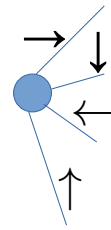
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



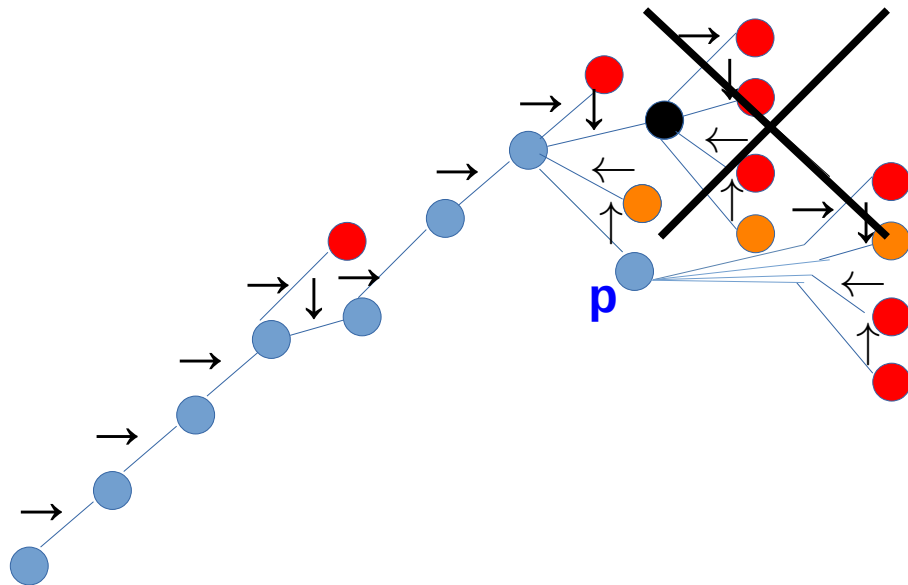
Tentativa e erro - exemplo do labirinto



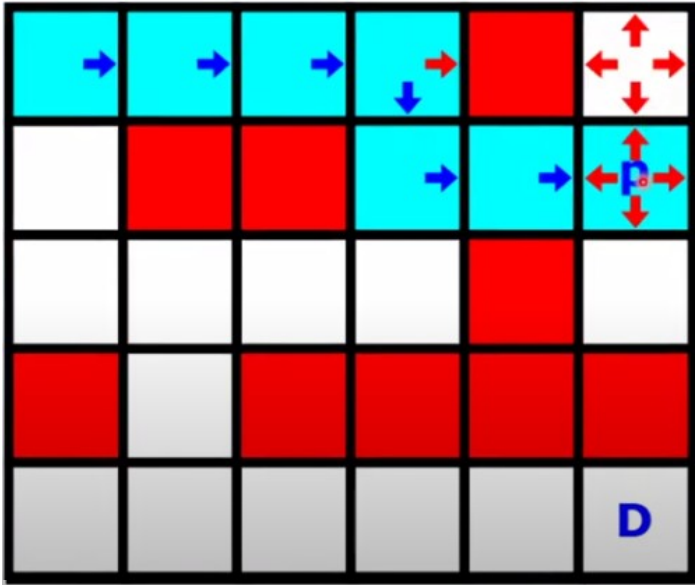
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



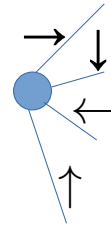
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



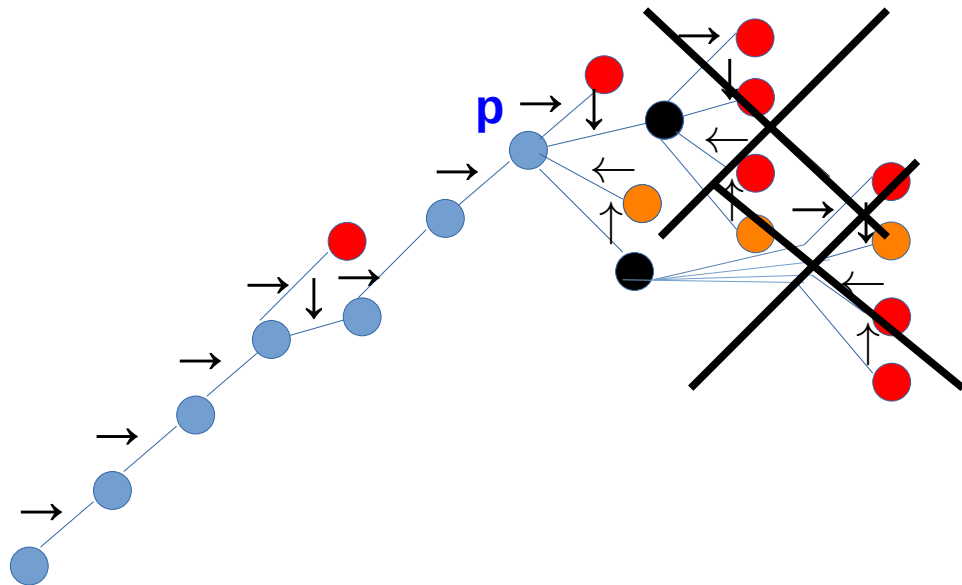
Tentativa e erro - exemplo do labirinto



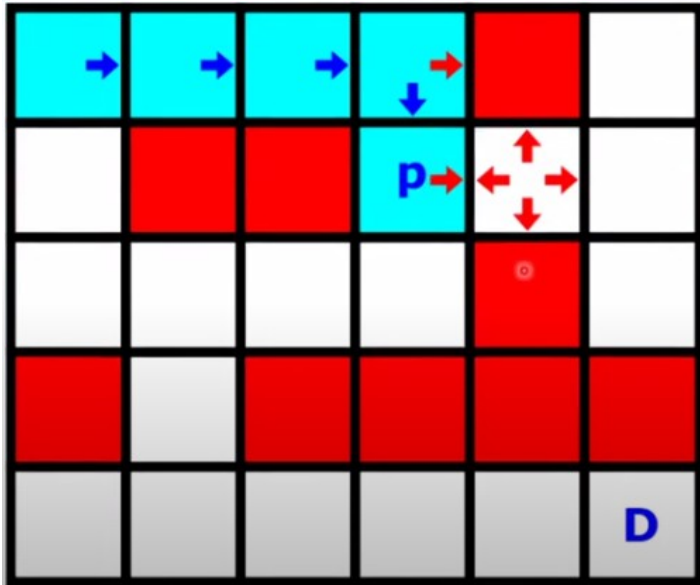
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



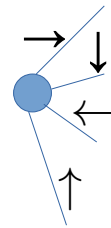
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



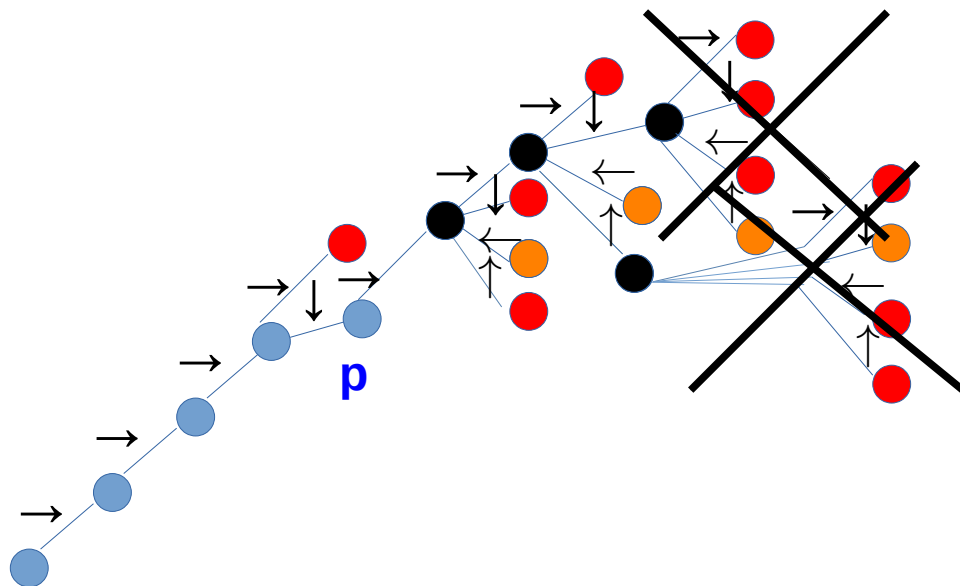
Tentativa e erro - exemplo do labirinto



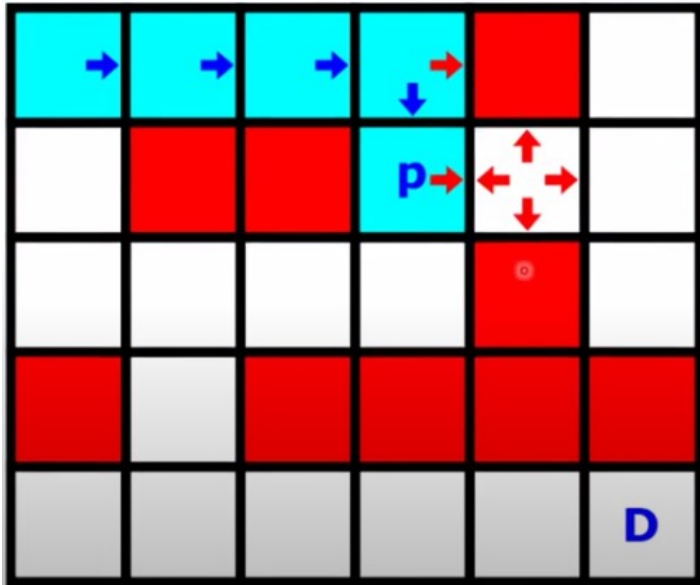
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



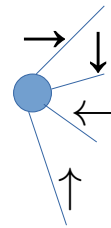
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



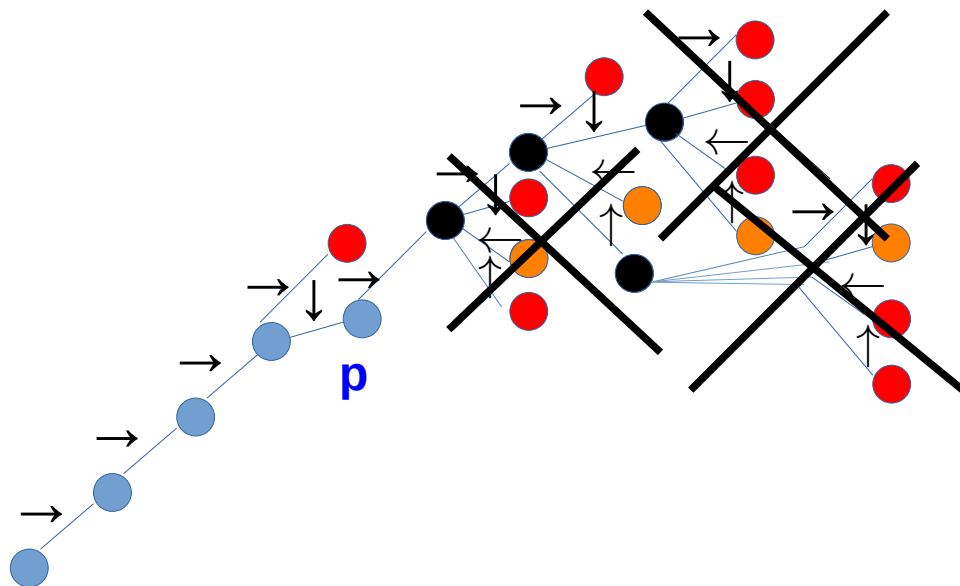
Tentativa e erro - exemplo do labirinto



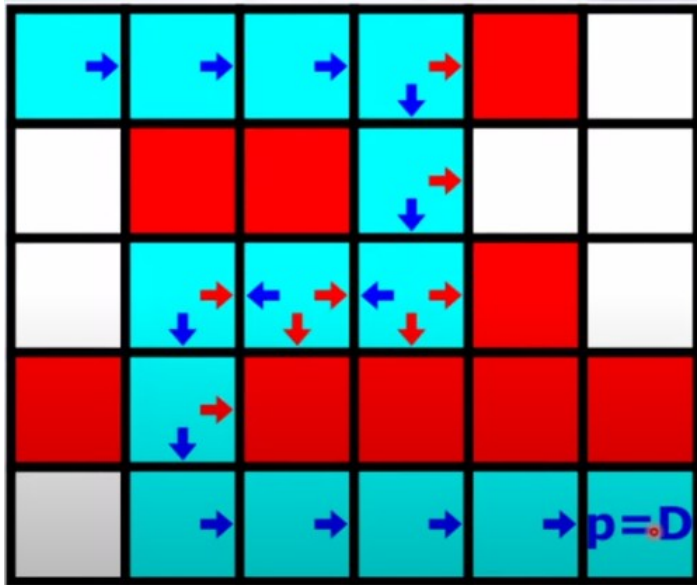
Estratégia: → ↓ ← ↑



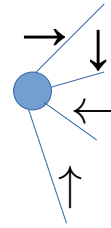
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



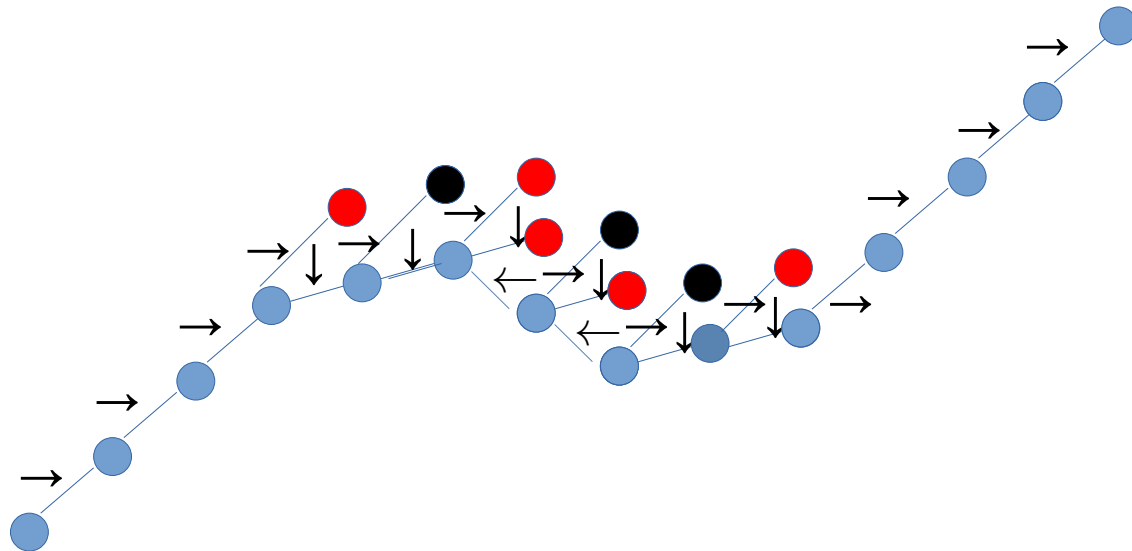
Tentativa e erro - exemplo do labirinto



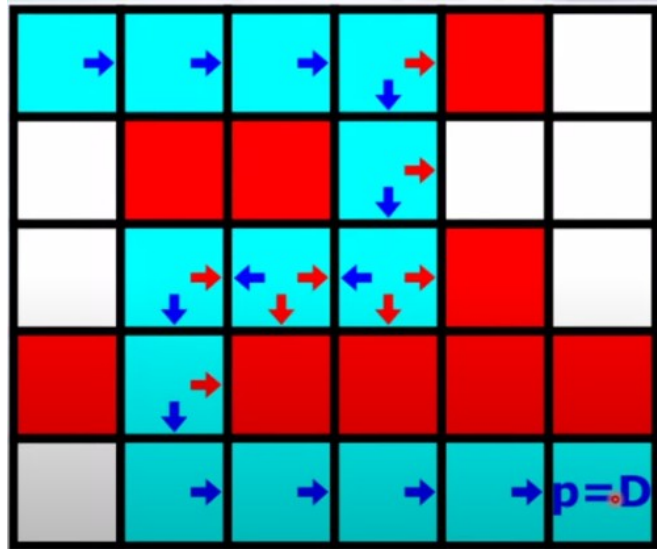
Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$



- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)

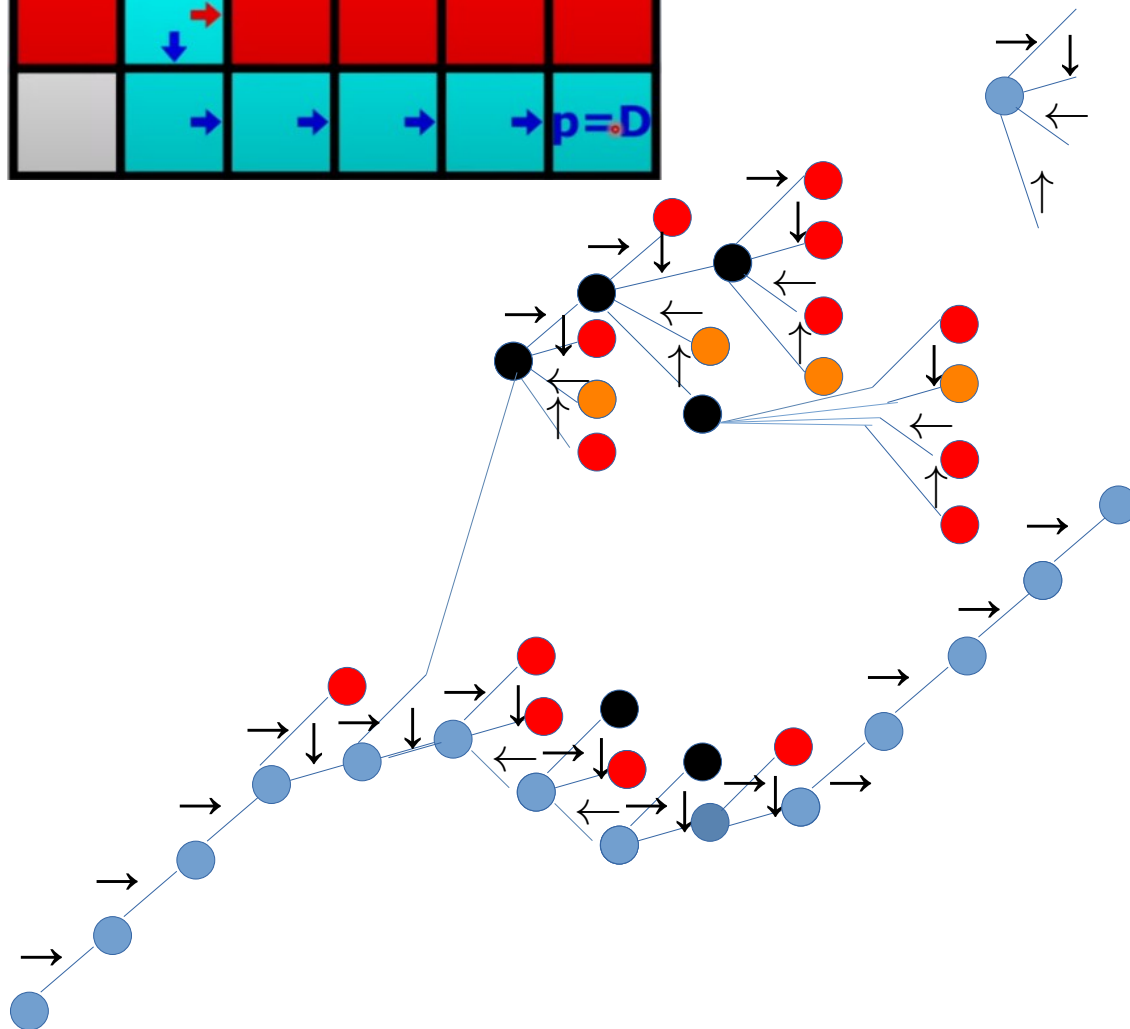


Tentativa e erro - exemplo do labirinto



Estratégia: \rightarrow \downarrow \leftarrow \uparrow

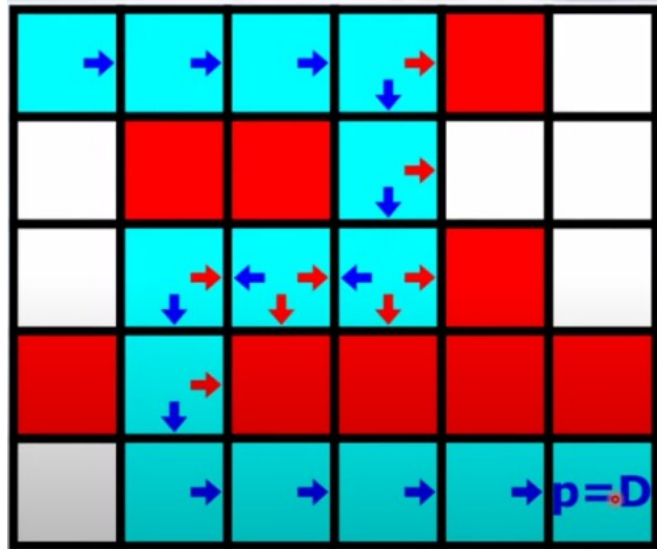
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



Árvore completa que foi percorrida

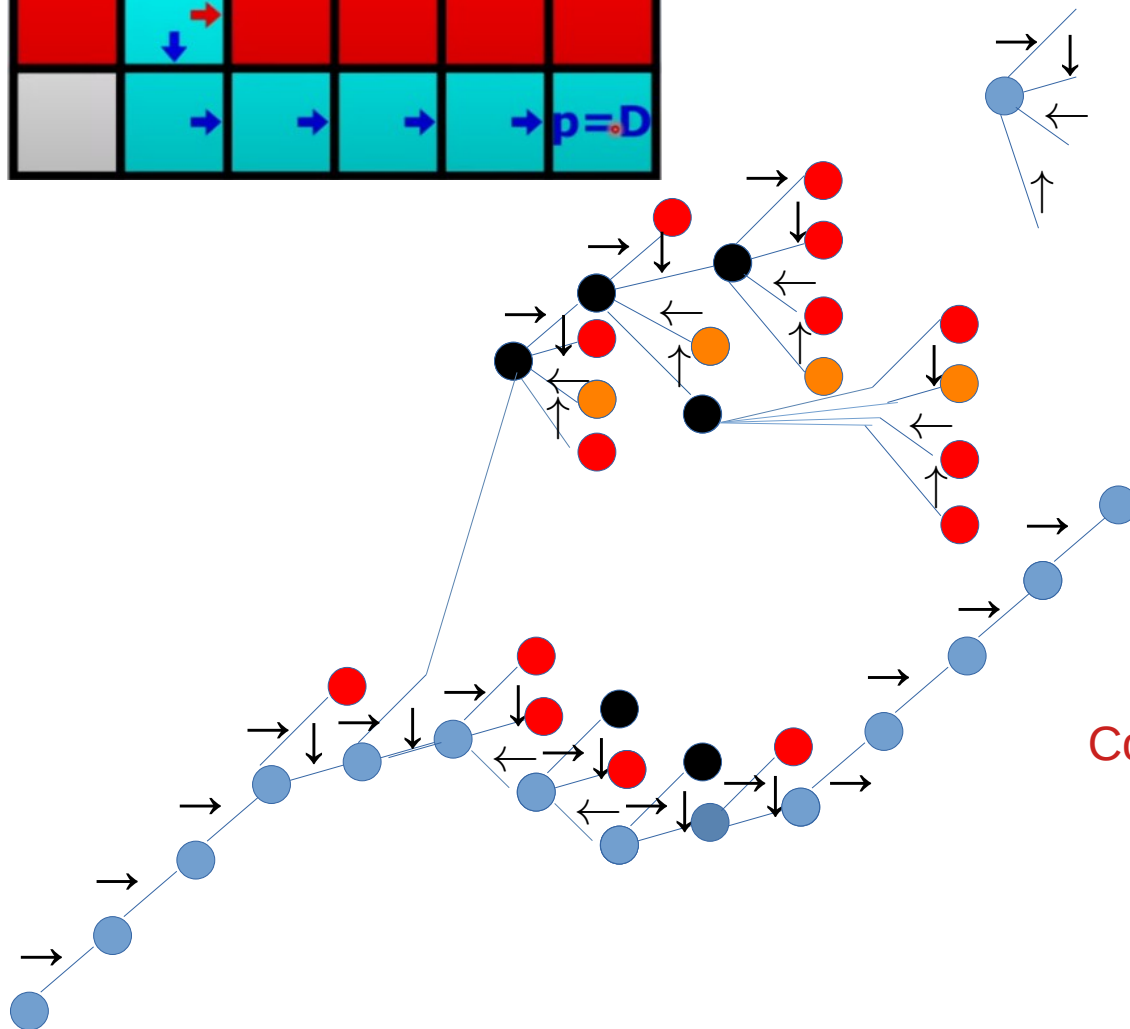


Tentativa e erro - exemplo do labirinto



Estratégia: \rightarrow \downarrow \leftarrow \uparrow

- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)

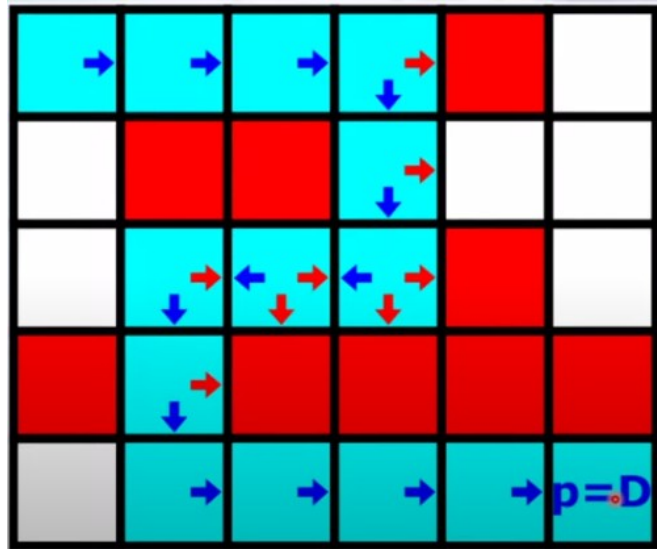


Árvore completa que foi percorrida

Como faço para “imprimir” o caminho?

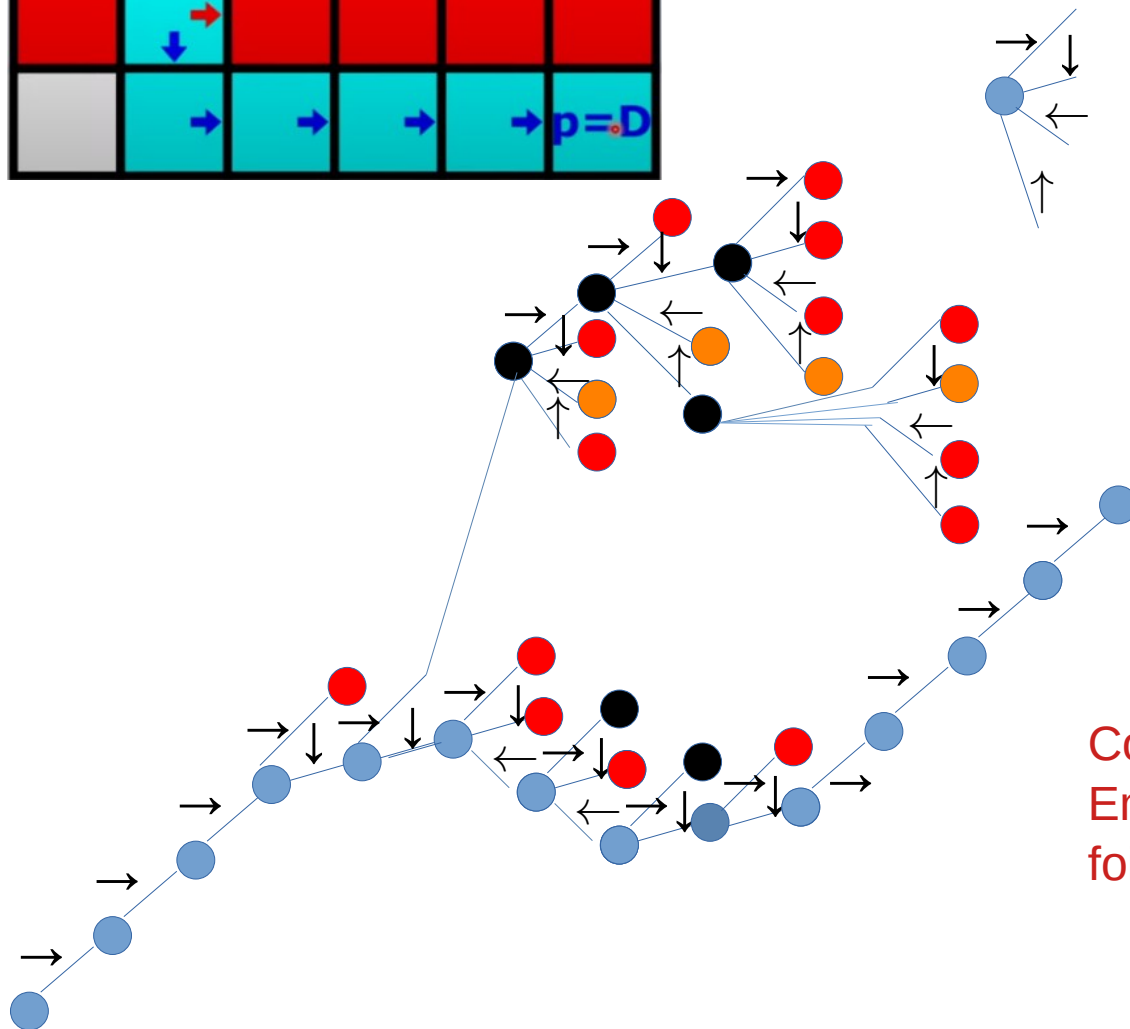


Tentativa e erro - exemplo do labirinto



Estratégia: \rightarrow \downarrow \leftarrow \uparrow

- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



Árvore completa que foi percorrida

Como faço para “imprimir” o caminho?
Em cada nó (célula da matriz) guardo quem foi o nó anterior, e imprimo de trás para frente

● : inviável

● : já esgotei as possibilidades (voltar)

É o caminho mais curto?

A 6x6 grid world environment. The start cell is at (5,1) and the goal cell is at (1,6). Obstacles are red cells at (1,5), (2,5), (3,5), (4,1), (4,3), (4,4), (4,5), (4,6), (5,1), and (5,5). Blue arrows show a path from (5,1) to (1,6) with the label 'p=D' at the goal.

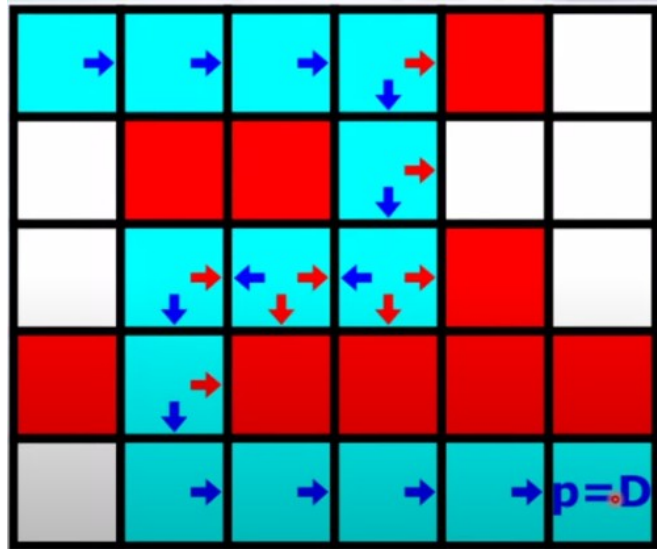
● : inviável

● : já esgotei as possibilidades (voltar)

É o caminho mais curto? NÃO

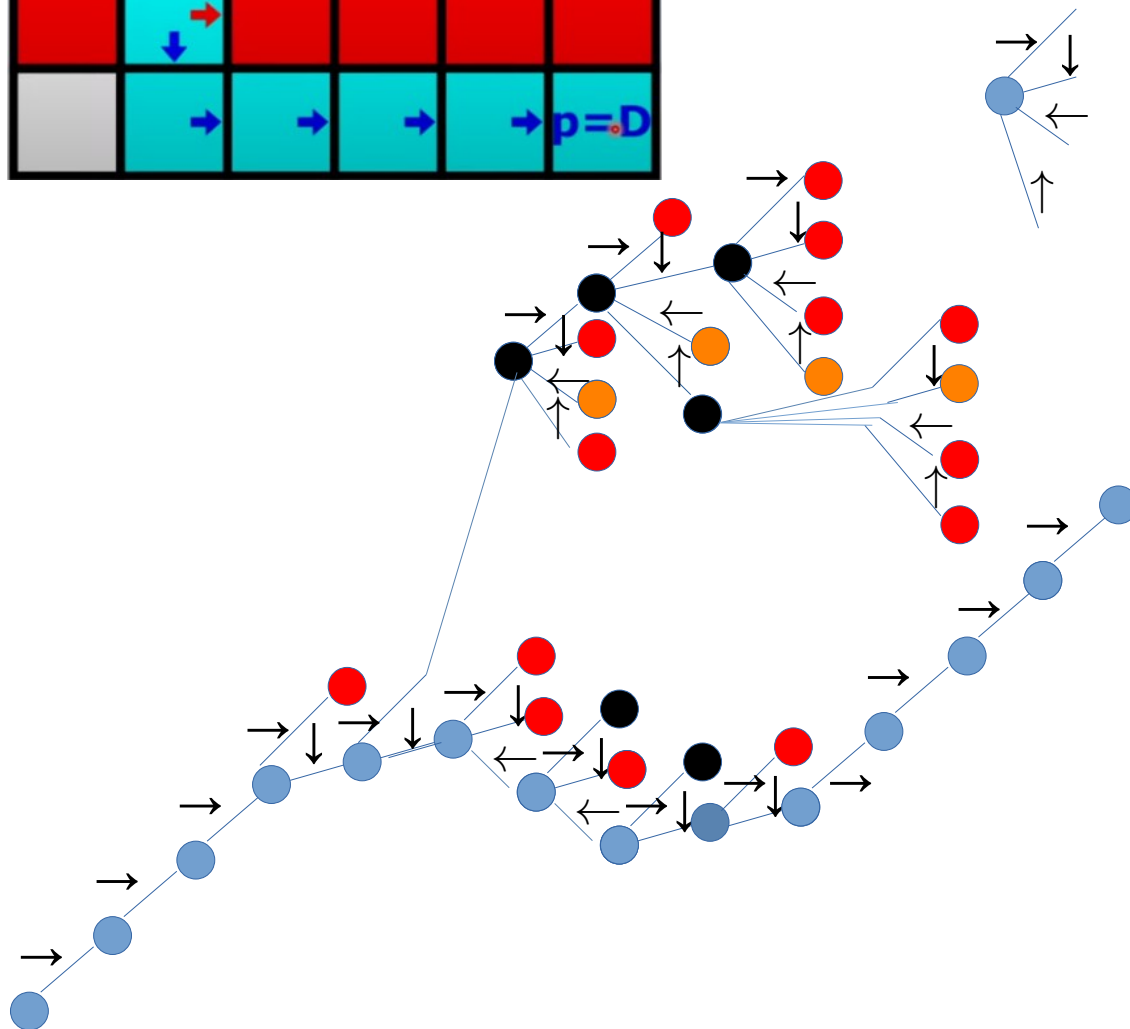
A complex directed graph with nodes of four colors: blue, red, black, and orange. The graph is composed of several interconnected components. On the left, a sequence of blue nodes is connected by horizontal arrows pointing right. This sequence branches into two main paths. The upper path features a cluster of black nodes, some of which are connected to red and orange nodes. The lower path also features a cluster of black nodes, similarly connected to red and orange nodes. Both paths eventually merge back into a single sequence of blue nodes on the right, which continues with horizontal arrows. Numerous small black arrows are scattered throughout the graph, indicating specific directions of flow or relationships between nodes. The overall structure suggests a complex network or a state transition system.

Tentativa e erro - exemplo do labirinto



Estratégia: \rightarrow \downarrow \leftarrow \uparrow

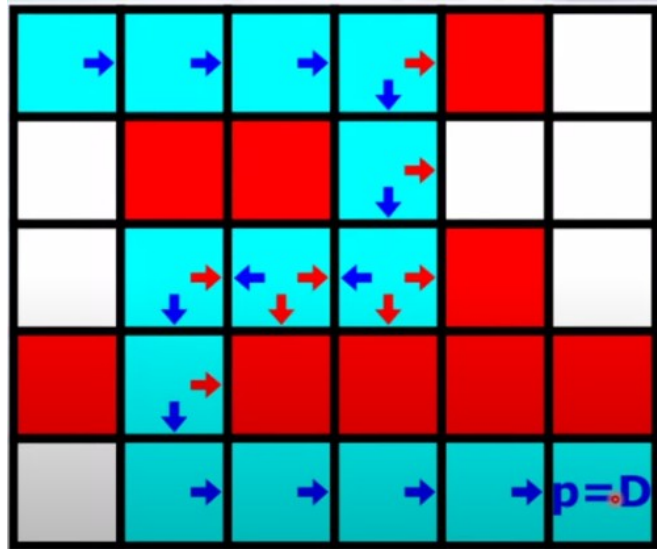
- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



Complexidade?

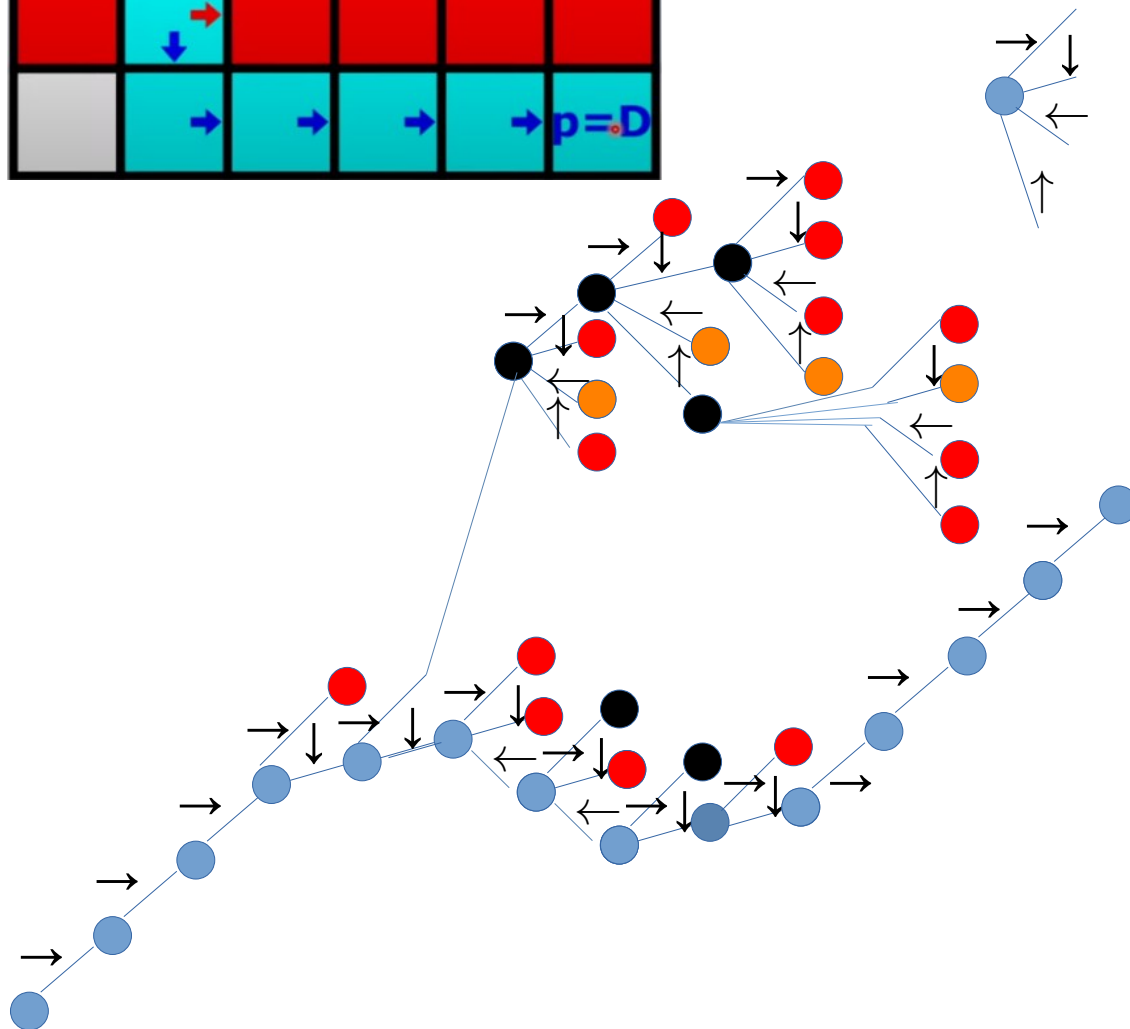


Tentativa e erro - exemplo do labirinto



Estratégia: $\rightarrow \downarrow \leftarrow \uparrow$

- : viável
- : inviável
- : já foi visitado
- : já esgotei as possibilidades (voltar)



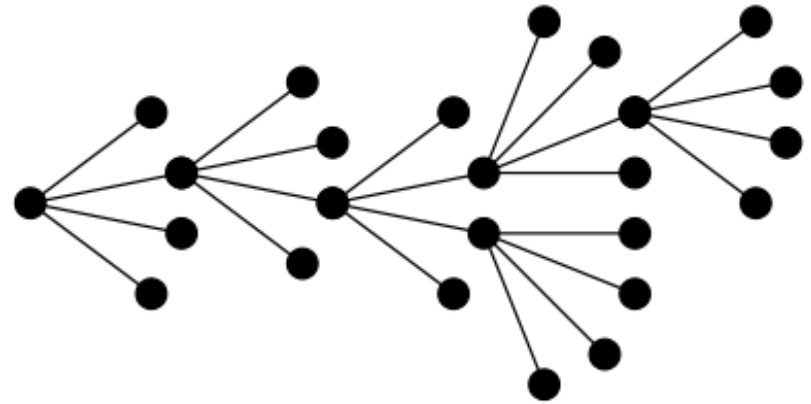
Complexidade?

Continua $O(2^{n \times m})$, mas na prática mais rápido que o força bruta



REPETINDO:Tentativa e erro - conceitos

- Como o próprio nome diz: tentar enquanto estiver errado
- Técnica de projetos de algoritmos:
 - útil para problemas cuja **solução consiste em tentar (potencialmente) todas alternativas propostas**;
 - então o problema se resume a: como organizar/explorar todas as alternativas?
 - idéia: decompor o processo em um número finito de subtarefas que devem ser exploradas;
 - processo geral pode ser visto com processo de tentativas que constrói e percorre uma árvore de subtarefas.



REPETINDO:Tentativa e erro - conceitos

- Algoritmos com esta técnica não têm uma regra fixa de computação.
- Funcionamento geral:
 - passos para obtenção da solução final são tentados e registrados;
 - caso um passo não leve à solução final, são retirados e apagados do registro.
- Pesquisa na árvore de solução: muitas vezes tem crescimento exponencial
 - Nestes casos pode-se usar heurísticas e/ou algoritmos aproximados que podem não garantir uma solução ótima, mas rápida (subótima)

Tentativa e erro - conceitos

- Voltando ao funcionamento geral:
 - passos para obtenção da solução final são tentados e registrados;
 - caso um passo não leve à solução final, são retirados e apagados do registro.
- **Como registrar e remover os passos com o que sabemos até agora?**

Tentativa e erro - conceitos

- Voltando ao funcionamento geral:
 - passos para obtenção da solução final são tentados e registrados;
 - caso um passo não leve à solução final, são retirados e apagados do registro.
- **Como registrar e remover os passos com o que sabemos até agora?**

Possível solução: **recursividade!!!**

Tentativa e erro - recursividade

- Registrando e removendo **com recursividade**:
 - Exemplo: fatorial do número 5

```
fatorial (n)
  se n < 3
    retorna n
  senão
    retorna n *
      fatorial (n-1)
  fim se
```

```
n = 5
retorna n * fatorial (4)
```

registra dados chamada 0
faz chamada recursiva

```
n = 4
retorna n * fatorial (3)
```

registra dados chamada 1
faz chamada recursiva

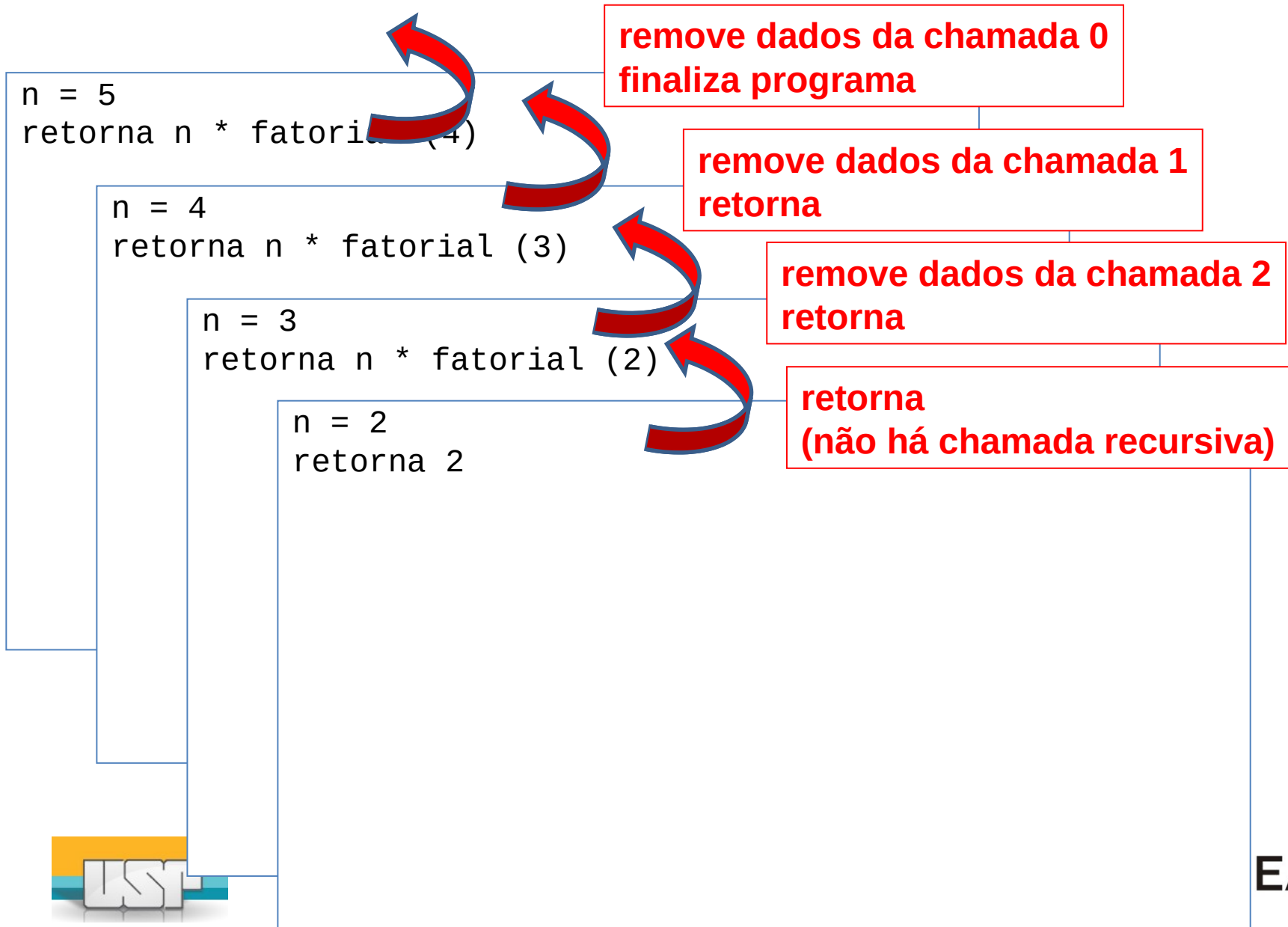
```
n = 3
retorna n * fatorial (2)
```

registra dados chamada 2
faz chamada recursiva

```
n = 2
retorna 2
```

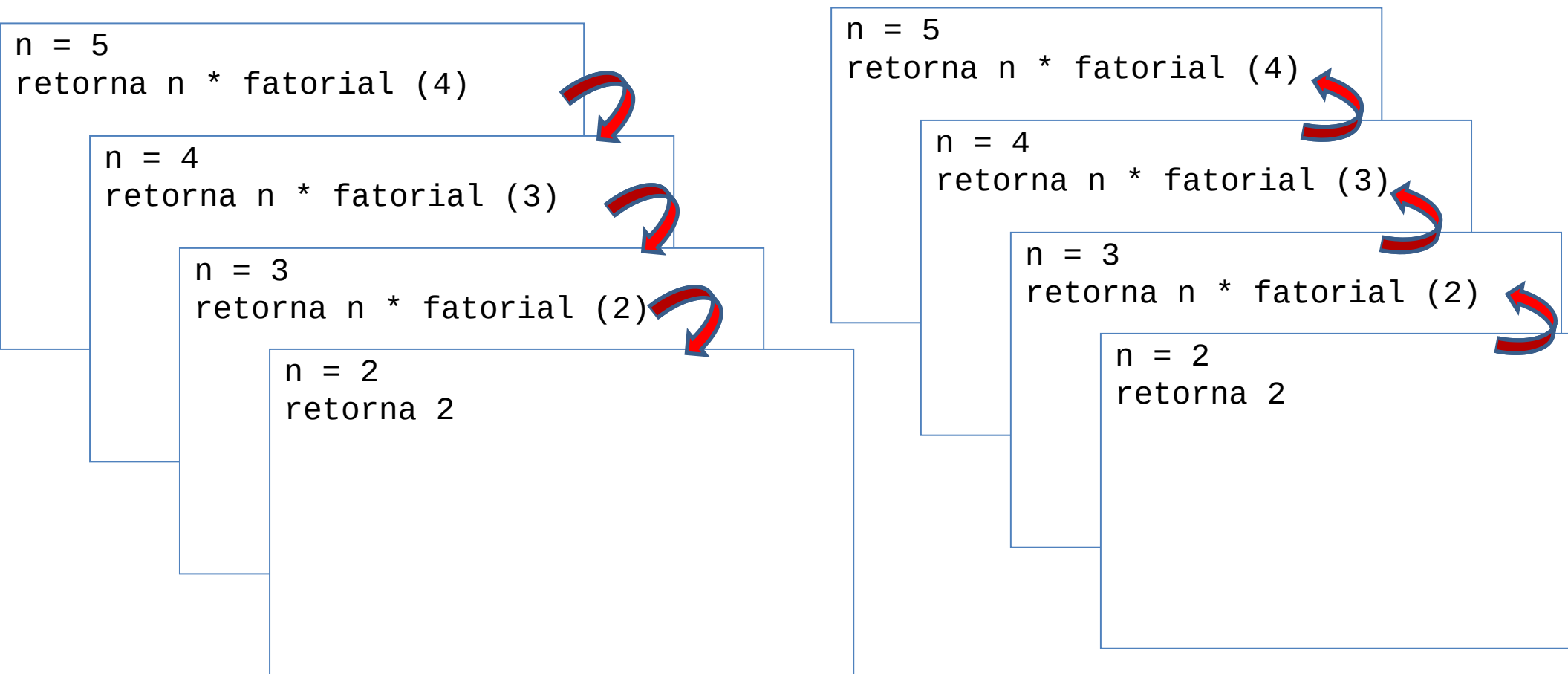
Tentativa e erro - recursividade

- Registrando e removendo **com recursividade**:



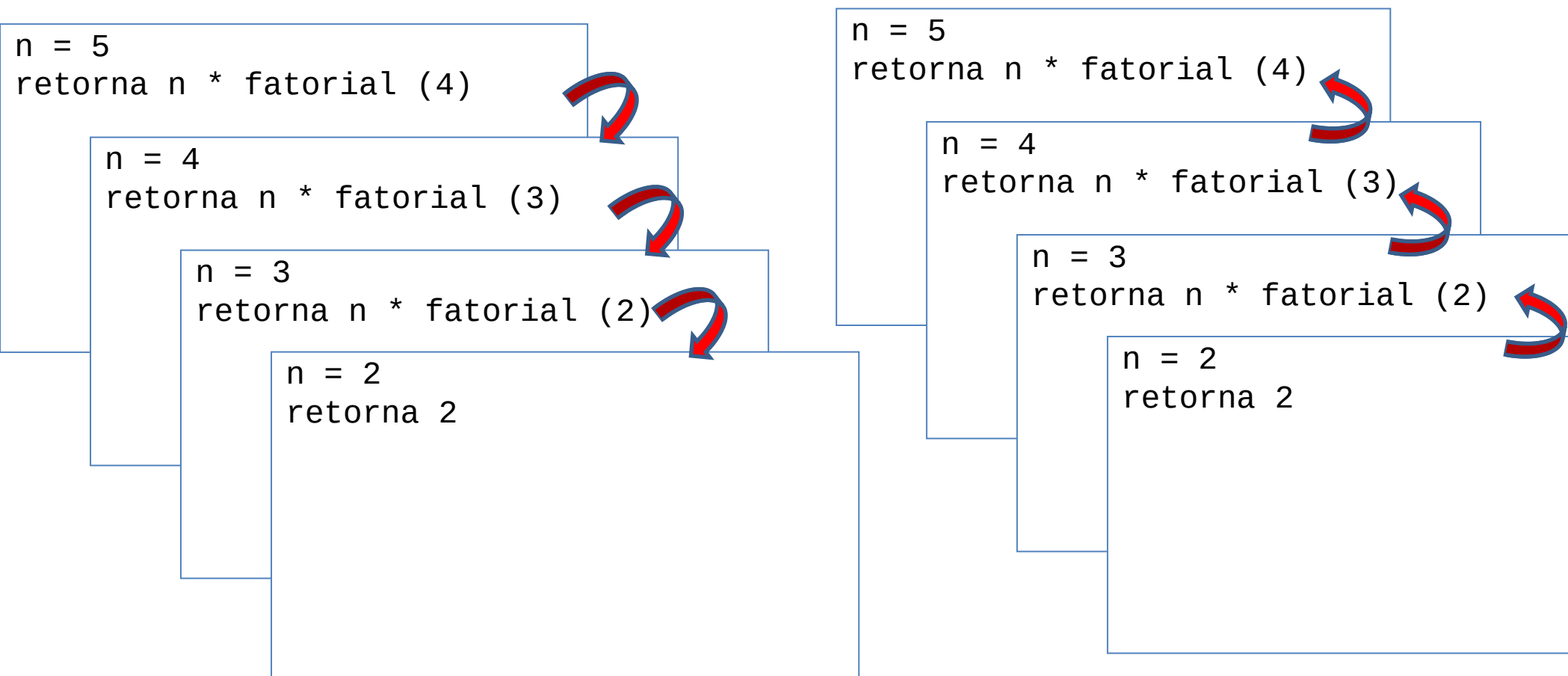
Tentativa e erro - recursividade

- O que você precisa fazer para que a recursividade registre e remova os dados?



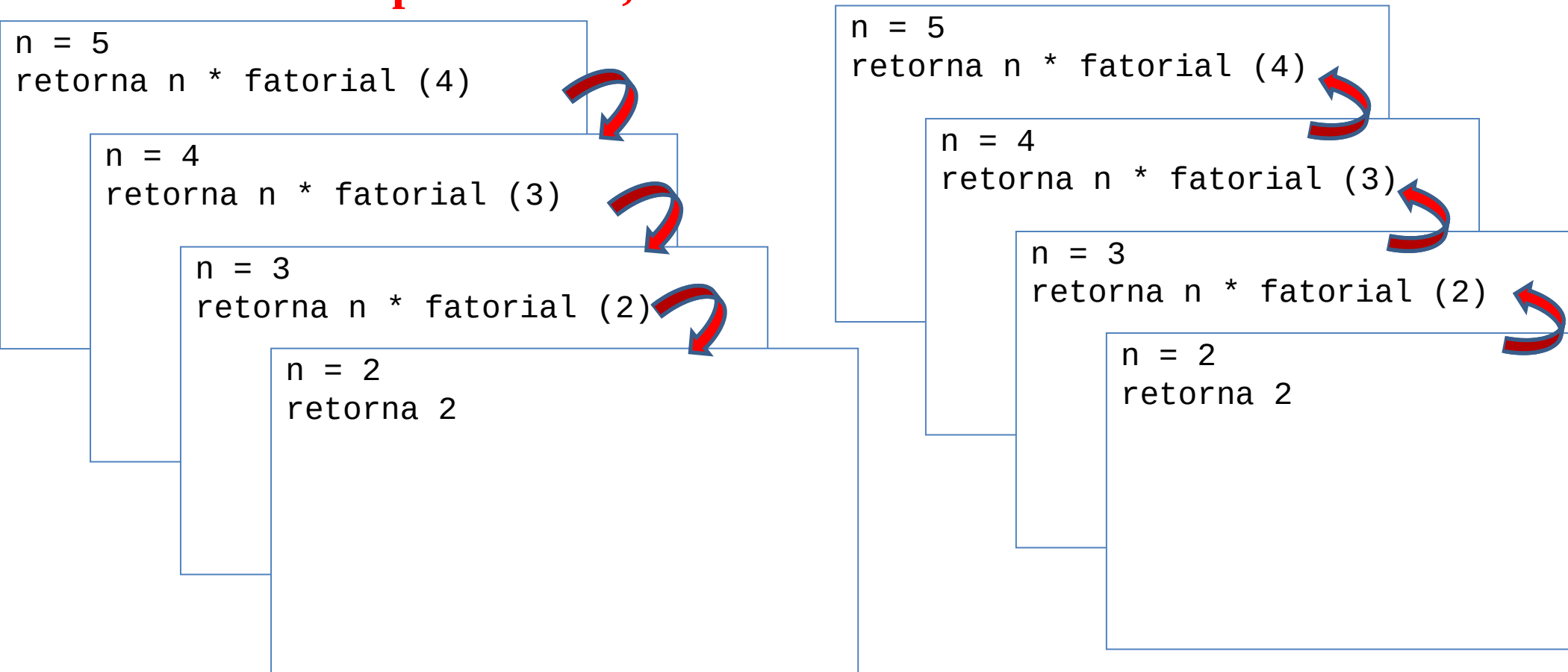
Tentativa e erro - recursividade

- O que você precisa fazer para que a recursividade registre e remova os dados?
- **NADA!!! SIMPLEMENTE USAR RECURSIVIDADE!**



Tentativa e erro - recursividade

- O que você precisa fazer para que a recursividade registre e remova os dados?
- **NADA!!! SIMPLESMENTE USAR RECURSIVIDADE!**
- Às vezes precisa sim, vamos ver no ex do Passeio do Cavalo



Backtracking - Solução genérica

tenta (no_atual)

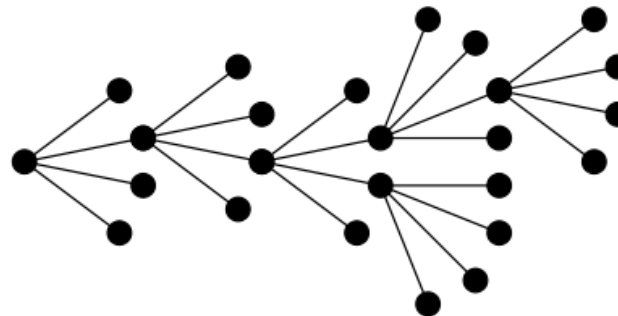
```
se promissor(no_atual)
```

se já_é_uma_solução(no_atual)

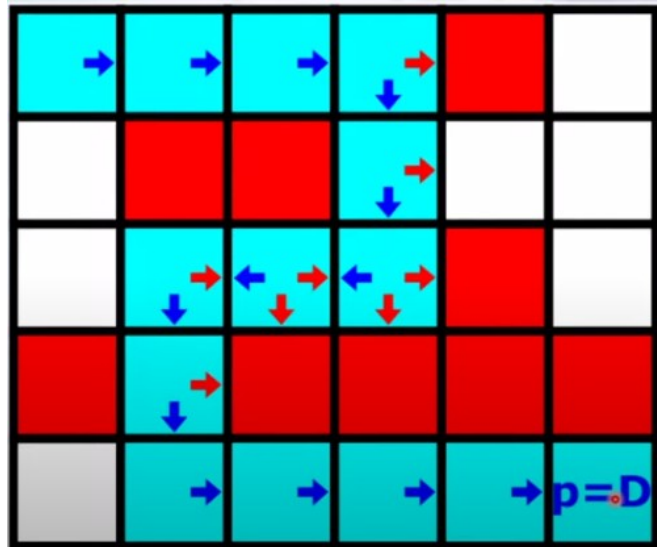
```
imprime/armazena_solução (no_atual)
```

senão para cada filho de no_atual

tenta(filho)



Tentativa e erro - exemplo do labirinto



Código do labirinto:

<https://www.youtube.com/watch?v=iVsN6ZnVx4Q&t=650s>

Do instante 19:49 até 27:42

Tentativa e erro - outro exemplo

Coloração de grafos

- Você deseja colorir um mapa com no máximo 4 cores: vermelho, amarelo, verde e azul
 - Países adjacentes devem ter cores diferentes
 - Em cada iteração, você deve decidir que cor pinta um país, dadas as cores já atribuídas aos vizinhos
 - Você não tem informação suficiente para escolher as cores previamente
 - Cada escolha leva a outro conjunto de escolhas
 - Uma ou mais sequências de passos pode ser a solução

Tentativa e erro - outro exemplo

Quadrado Latino

- Dada uma matriz $n \times n$, completá-la com b diferentes símbolos de modo que não haja repetição de símbolo na mesma linha nem na mesma coluna
- A matriz pode vir com algumas células já preenchidas

B			D
	D	B	
	C	D	
D			C

Tentativa e erro - outro exemplo

Sudoku

- Dada uma matriz 9×9 , completá-la de forma que cada linha, coluna e seção 3×3 contenha todos os dígitos entre 1 e 9.

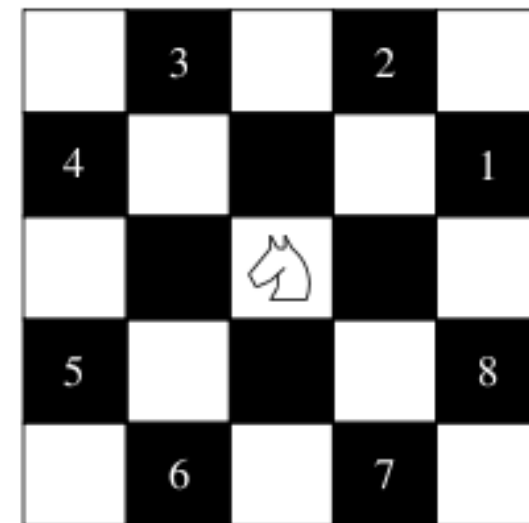
1		7			6	4	5	
	2	5	3	4				8
	6				1		7	
	5	3					2	9
6	1				9	8		
			6		2			7
		1		9	3	2		
		8						
	4			7	8	5	9	1

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

Tentativa e erro - exemplo

Passeio do cavalo no tabuleiro de xadrez (livro Ziviani)

- tabuleiro com $n \times n$ posições
- cavalo se movimenta segundo as regras do xadrez (próximo passo: soma 1 casa em x e 2 em y – ou 2 casas em x e 1 em y)
- A partir de uma posição inicial, o problema consiste em encontrar, se existir um passeio do cavalo com n^2-1 movimentos, visitando todos os pontos do tabuleiro uma única vez



Tentativa e erro - exemplo

Passeio do cavalo no tabuleiro de xadrez (livro Ziviani)

- O tabuleiro pode ser representado por uma matriz t $n \times n$
- A situação de cada posição pode ser representada por um inteiro para recordar o histórico das ocupações:
 - $t[x,y] = 0$, campo $\langle x, y \rangle$ não visitado,
 - $t[x,y] = i$, campo $\langle x, y \rangle$ visitado no i -ésimo movimento, $1 \leq i \leq n^2$

Tentativa e erro - exemplo

Passeio do cavalo no tabuleiro de xadrez (livro Ziviani)

- Uma solução para um tabuleiro de tamanho 8 x 8

1	60	39	34	31	18	9	64
38	35	32	61	10	63	30	17
59	2	37	40	33	28	19	8
36	49	42	27	62	11	16	29
43	58	3	50	41	24	7	20
48	51	46	55	26	21	12	15
57	44	53	4	23	14	25	6
52	47	56	45	54	5	22	13

Passeio do Cavalo - algoritmo bem alto-nível

Tenta ()

inicializa seleção de movimentos

faça

seleciona próximo candidato ao movimento

se aceitável

registra movimento

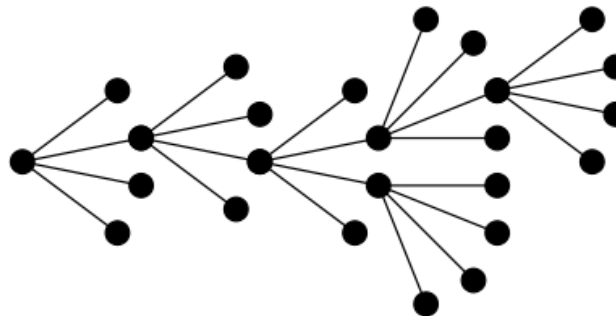
se tabuleiro **não** está cheio

tenta novo movimento

se não *bem sucedido*

apaga registro anterior // backtracking

enquanto não (movimento foi *bem sucedido* ou acabaram-se os candidatos a movimento)



Passeio do Cavalo

```
#define N 8  /* Tamanho do lado do tabuleiro */
```

```
#define FALSE 0
```

```
#define TRUE 1
```

```
int i, j, t[N][N], a[N], b[N];
```

```
short q;
```

/ a e b indicam quanto devo somar na linha e coluna, respectivamente, da posição atual para ir para outra posição válida */*

```
int main(int argc, char *argv[]) { /* programa principal */
```

```
    a[0] = 2; a[1] = 1; a[2] = -1; a[3] = -2;
```

```
    b[0] = 1; b[1] = 2; b[2] = 2; b[3] = 1;
```

```
    a[4] = -2; a[5] = -1; a[6] = 1; a[7] = 2;
```

```
    b[4] = -1; b[5] = -2; b[6] = -2; b[7] = -1;
```

```
    for (i = 0; i < N; i++) for (j = 0; j < N; j++) t[i][j] = 0;
```

```
    t[0][0] = 1; /* escolhemos uma casa do tabuleiro */
```

```
    Tenta(2, 0, 0, &q);
```

```
    if (!q) { printf("Sem solucao\n"); return 0; }
```


```
    for (i = 0; i < N; i++)
```

```
        { for (j = 0; j < N; j++) printf(" %4d", t[i][j]); putchar('\n'); }
```

```
    return 0;
```

```
}
```

deslocamentos possíveis

0				
0	4		3	
	5			2
				
	6			1
		7	0	

“booleano” se o movimento foi bem sucedido ou não

posições x,y atuais

nr do movimento

Passeio do Cavalo

```
void Tenta(int i, int x, int y, short *q)
{
    int u, v;
    int k = -1;
    short q1;

    /* inicializa selecao de movimentos */
    do { ++k; q1 = FALSE;
        u = x + a[k]; v = y + b[k];
        if (u >= 0 && u < N && v >= 0 && v < N) // movimento dentro do tabuleiro
            if (t[u][v] == 0) // movimento para um lugar ainda não visitado
            {
                t[u][v] = i; // registra movimento
                if (i < N * N) { /* tabuleiro nao esta cheio */
                    Tenta(i + 1, u, v, &q1); /* tenta novo movimento */
                    if (!q1)
                        t[u][v] = 0; /* nao sucedido apaga registro anterior */
                }
                else q1 = TRUE;
            }
    } while (!(q1 || k == 7));

    *q = q1;
}
```

k varia de 0 a 7
(movimentos do cavalo)

Tenta ()

inicializa seleção de movimentos

faça

seleciona próximo candidato ao movimento

se aceitável

registra movimento

se tabuleiro não está cheio

tenta novo movimento

se não bem sucedido

apaga registro anterior // backtracking

enquanto não (movimento foi bem sucedido ou
acabaram-se os candidatos a movimento)



Passeio do Cavalo

```
void Tenta(int i, int x, int y, short *q)
{ int u, v; int k = -1; short q1;
  /* inicializa selecao de movimentos */
  do { ++k; q1 = FALSE;
    u = x + a[k]; v = y + b[k];
    if (u >= 0 && u < N && v >= 0 && v < N) // movimento dentro do tabuleiro
    if (t[u][v] == 0) // movimento para um lugar ainda não visitado
    { t[u][v] = i; // registra movimento
      if (i < N * N) { /* tabuleiro nao esta cheio */
        Tenta(i + 1, u, v, &q1); /* tenta novo movimento */
        if (!q1)
          t[u][v] = 0; /* nao sucedido apaga registro anterior */
      }
      else q1 = TRUE;
    }
  } while (!(q1 || k == 7));
  *q = q1;
}
```

Tenta ()
inicializa seleção de movimentos
faça
seleciona próximo candidato ao movimento
se aceitável
registra movimento
se tabuleiro **não** está cheio
tenta novo movimento
se **não** bem sucedido
apaga registro anterior // backtracking
enquanto **não** (movimento foi bem sucedido ou
acabaram-se os candidatos a movimento)

Qual a complexidade ?



Passeio do Cavalo

```
void Tenta(int i, int x, int y, short *q)
{ int u, v; int k = -1; short q1;
  /* inicializa selecao de movimentos */
  do { ++k; q1 = FALSE;
    u = x + a[k]; v = y + b[k];
    if (u >= 0 && u < N && v >= 0 && v < N) // movimento dentro do tabuleiro
    if (t[u][v] == 0) // movimento para um lugar ainda não visitado
    { t[u][v] = i; // registra movimento
      if (i < N * N) { /* tabuleiro nao esta cheio */
        Tenta(i + 1, u, v, &q1); /* tenta novo movimento */
        if (!q1)
          t[u][v] = 0; /* nao sucedido apaga registro anterior */
      }
      else q1 = TRUE;
    }
  } while (!(q1 || k == 7));
  *q = q1;
}
```

Tenta ()
inicializa seleção de movimentos
faça
seleciona próximo candidato ao movimento
se aceitável
registra movimento
se tabuleiro **não** está cheio
tenta novo movimento
se **não** bem sucedido
apaga registro anterior // backtracking
enquanto **não** (movimento foi bem sucedido ou
acabaram-se os candidatos a movimento)

Qual a complexidade ?

Nr de possibilidades de preenchimento:

$1 \times 8 \times 8 \times \dots \times 8$ (n^2 vezes)

=



Passeio do Cavalo

```
void Tenta(int i, int x, int y, short *q)
{ int u, v; int k = -1; short q1;
  /* inicializa selecao de movimentos */
  do { ++k; q1 = FALSE;
    u = x + a[k]; v = y + b[k];
    if (u >= 0 && u < N && v >= 0 && v < N) // movimento dentro do tabuleiro
    if (t[u][v] == 0) // movimento para um lugar ainda não visitado
    { t[u][v] = i; // registra movimento
      if (i < N * N) { /* tabuleiro nao esta cheio */
        Tenta(i + 1, u, v, &q1); /* tenta novo movimento */
        if (!q1)
          t[u][v] = 0; /* nao sucedido apaga registro anterior */
      }
      else q1 = TRUE;
    }
  } while (!(q1 || k == 7));
  *q = q1;
}
```

Tenta ()
inicializa seleção de movimentos
faça
seleciona próximo candidato ao movimento
se aceitável
registra movimento
se tabuleiro **não** está cheio
tenta novo movimento
se não bem sucedido
apaga registro anterior // backtracking
enquanto não (movimento foi bem sucedido ou
acabaram-se os candidatos a movimento)

Qual a complexidade ? $O(2^{n^2})$:

Nr de possibilidades de preenchimento:

$1 \times 8 \times 8 \times \dots \times 8$ (n^2 vezes)

$= O(8^{(n-1)^2}) = O(2^{n^2})$



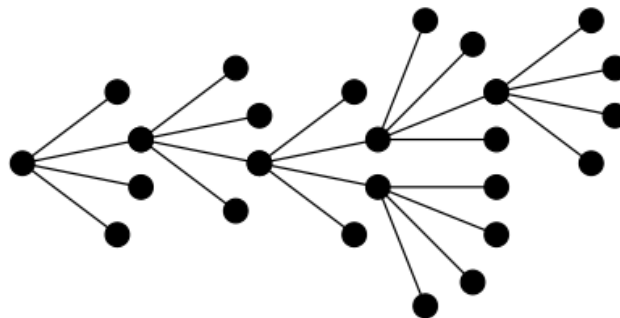
Lembrando....

Tentativa e erro - conceitos

- Algoritmos com esta técnica não têm uma regra fixa de computação.
- Funcionamento geral:
 - passos para obtenção da solução final são tentados e registrados;
 - caso um passo não leve à solução final, são retirados e apagados do registro.
- Pesquisa na árvore de solução: **muitas vezes tem crescimento exponencial**
 - Nestes casos pode-se usar **heurísticas** e/ou algoritmos aproximados que podem não garantir uma solução ótima, mas rápida (subótima)

Vantagens do backtracking

- A organização do problema (na verdade do **espaço de busca das soluções**) já te permite a não testar combinações que não têm chance de ocorrer. Ex: se no primeiro movimento já sai do tabuleiro, não testa os movimentos seguintes a partir desse primeiro
- Para alguns problemas a organização do **espaço de busca** permite o uso de heurísticas que, além de tornar a busca mais eficiente, ainda consegue garantir a solução ótima! (cuidado: pode ter heurísticas que não garante otimalidade...)
- Ex: Branch and bound

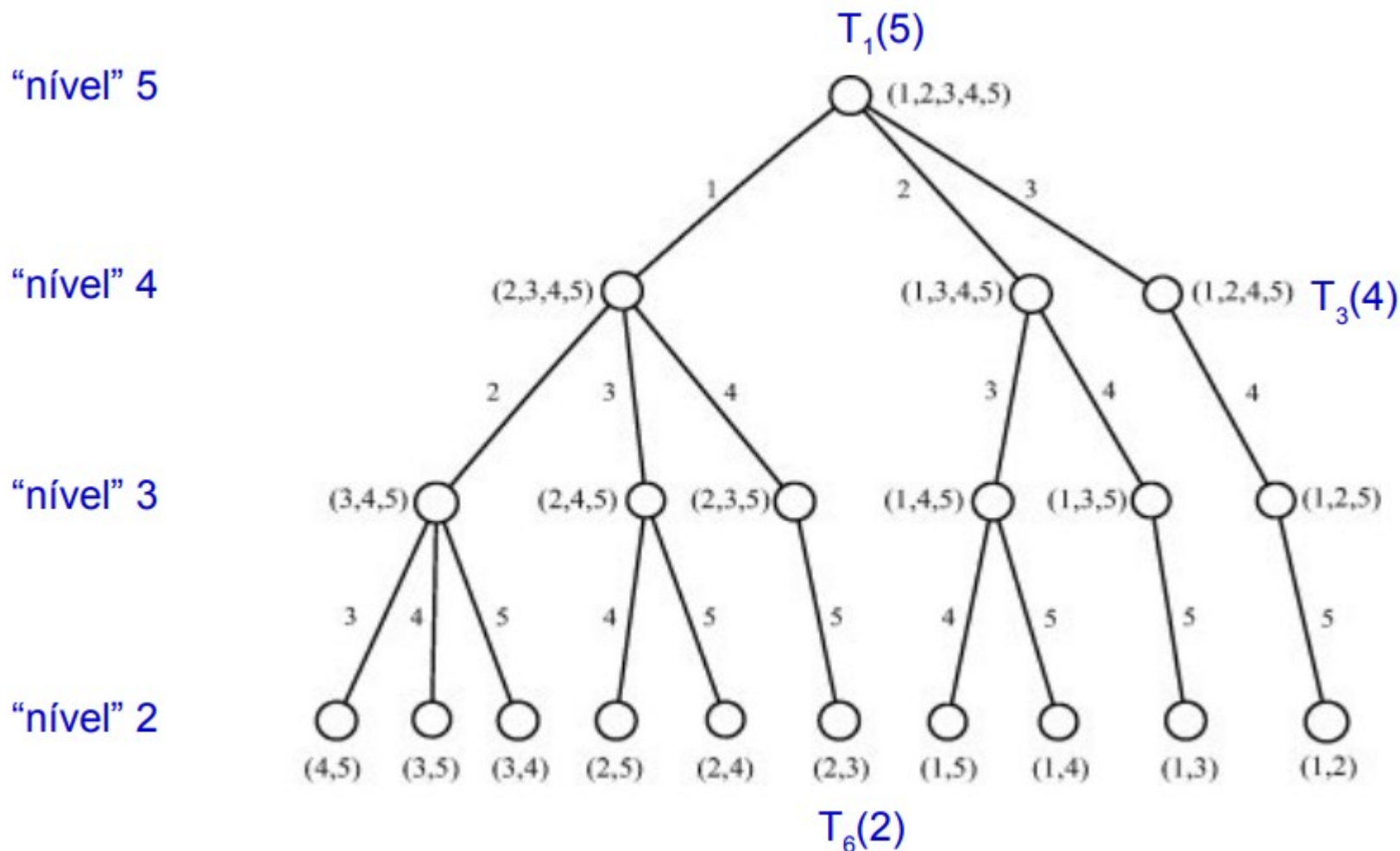


Exemplo de branch and bound

Cada nó é um subconjunto de elementos, e recebe uma nota

Quero o nó do nível 2 de maior nota

Heurística: um subconjunto nunca tem nota maior que seu superconjunto



Exemplo de branch and bound

Cada nó é um subconjunto de elementos, e recebe uma nota

Quero o nó do nível 2 de maior nota

Heurística: um subconjunto nunca tem nota maior que seu superconjunto

Exemplo: $N=5$, $D=2$:

S (melhor subconjunto até agora) = ●

$\text{melhor_medida} = 200$

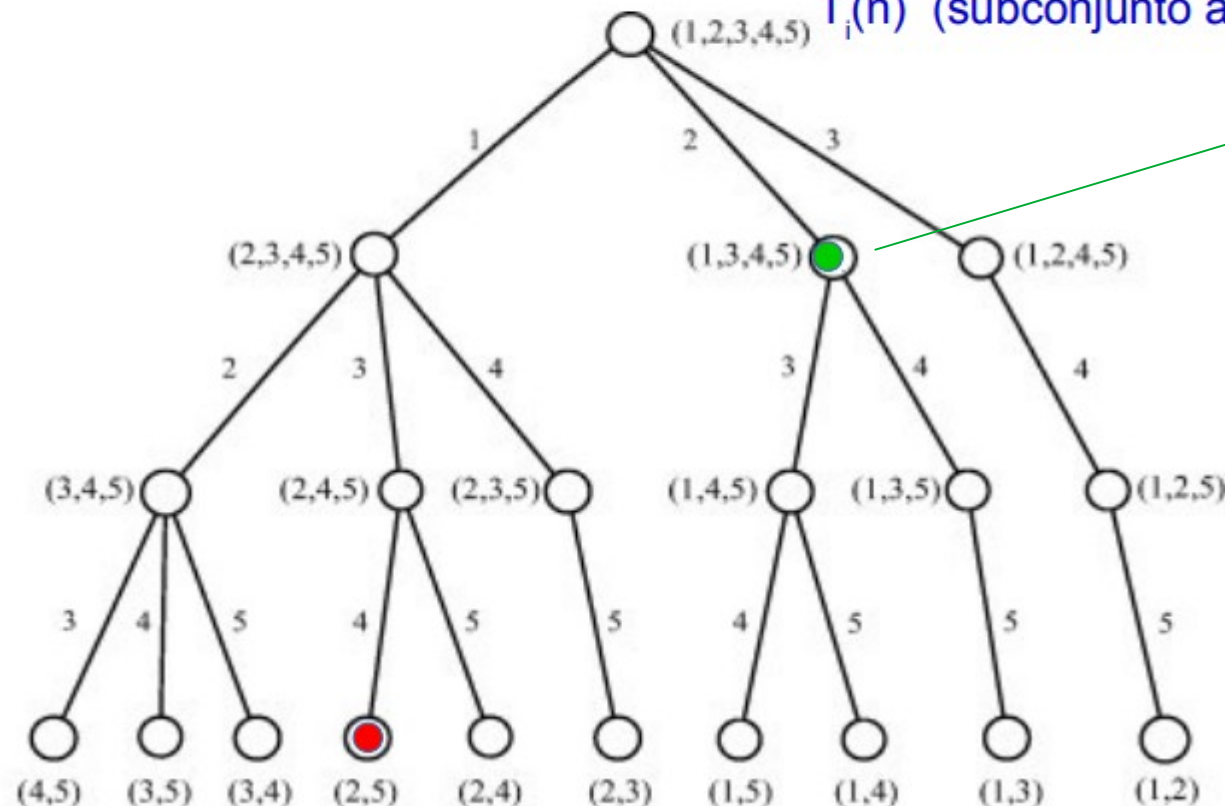
$T_i(n)$ (subconjunto atual) = ●

'nível" 5

'nível" 4

'nível" 3

'nível" 2



Nota = 180

Exemplo de branch and bound

Cada nó é um subconjunto de elementos, e recebe uma nota

Quero o nó do nível 2 de maior nota

Heurística: um subconjunto nunca tem nota maior que seu superconjunto

Exemplo: $N=5$, $D=2$:

S (melhor subconjunto até agora) = ●

$\text{melhor_medida} = 200$

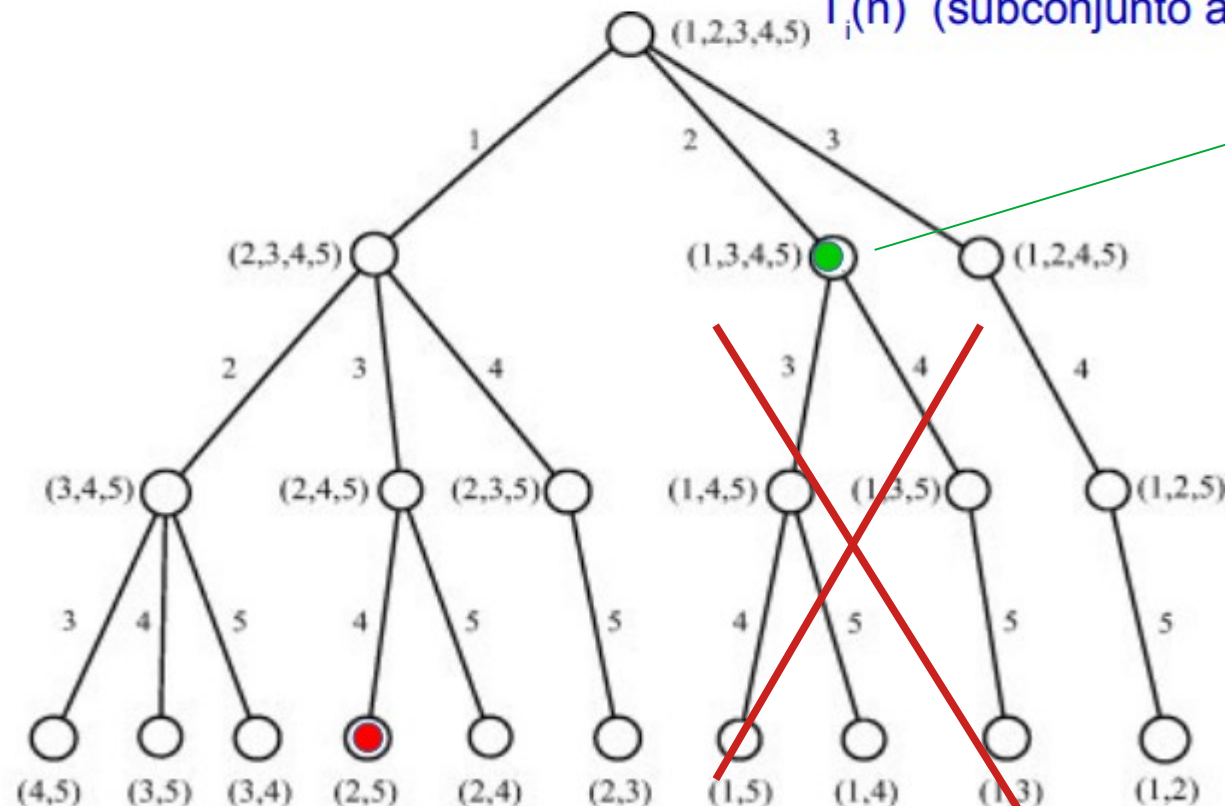
$T_i(n)$ (subconjunto atual) = ●

'nível' 5

'nível' 4

'nível' 3

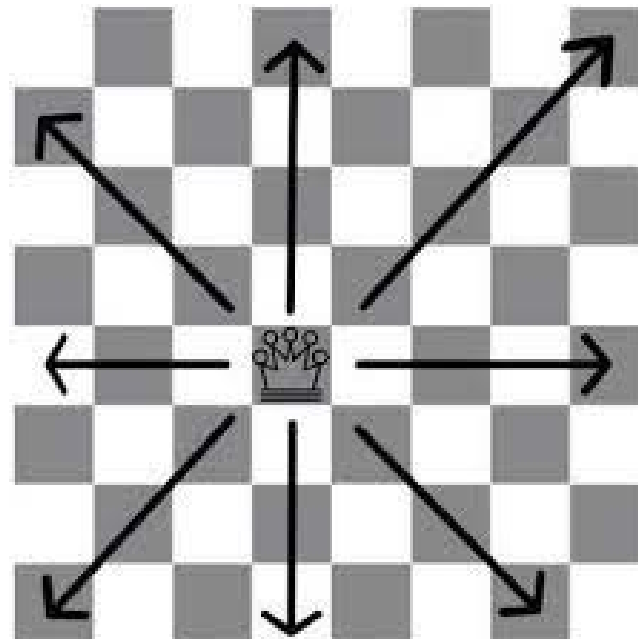
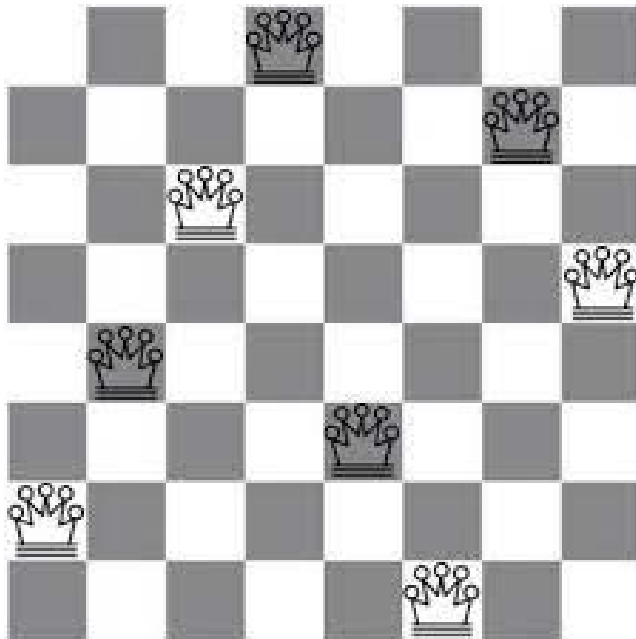
'nível' 2



Nota = 180

Exercícios

- Altere o programa do Passeio do Cavalo para permitir que o usuário escolha a posição inicial do cavalo (e experimente executar com diferentes valores!)
- **Problema das oito rainhas:** ainda considerando um tabuleiro de xadrez, escreva um algoritmo que seja capaz de distribuir oito rainhas pelo tabuleiro (8x8) de forma que nenhuma rainha seja atacada pela outra (ou seja, encontrar oito posições que não compartilhem a mesma linha, coluna ou diagonal)



Referências

- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 3a. Edição, 2011 (texto base), cap 2.3
- Notas de aula – Profa. Fátima L. S. Nunes – EACH-USP