Linguagem de Programação I - DIM0120

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 231, ramal 231, selan@dimap.ufrn.br
UFRN

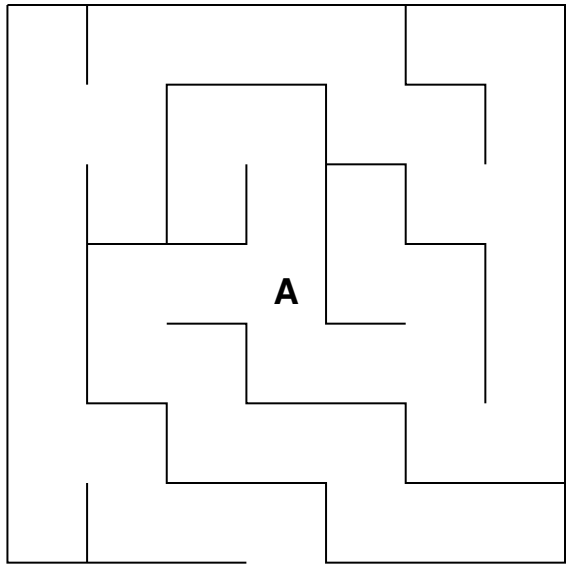
2018.1

Listas Lineares — Conteúdo

Introdução

Labirinto e o algorimto da mão-direita

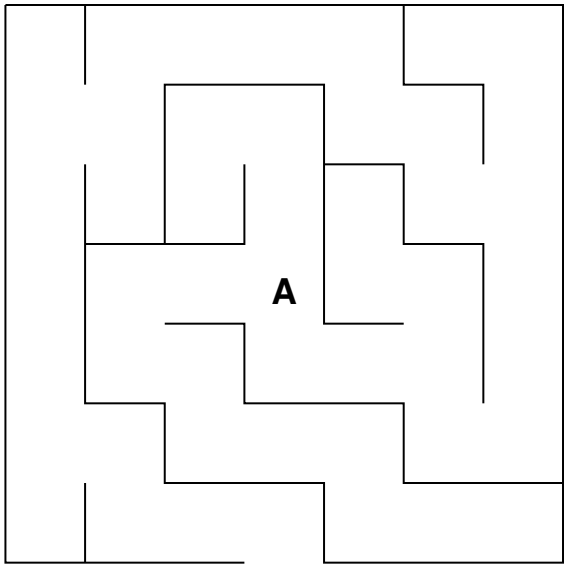
- Vamos abordar o problema de encontrar a solução para um labirinto.



Introdução

Labirinto e o algorimto da mão-direita

- Vamos abordar o problema de encontrar a solução para um labirinto.

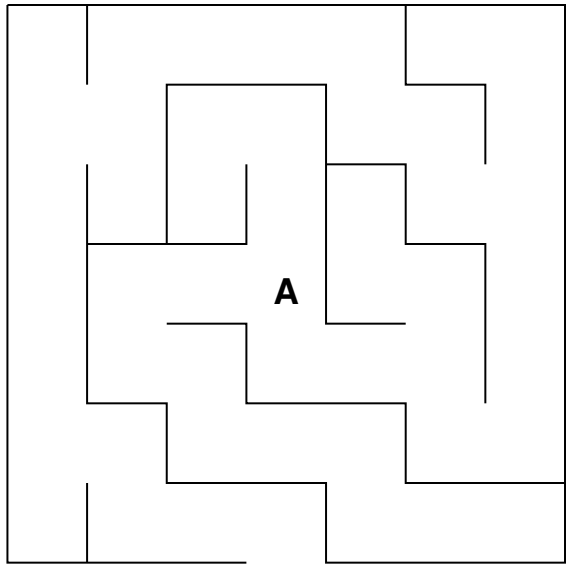


- Existe um algoritmo iterativo simples:

Introdução

Labirinto e o algorimto da mão-direita

- Vamos abordar o problema de encontrar a solução para um labirinto.

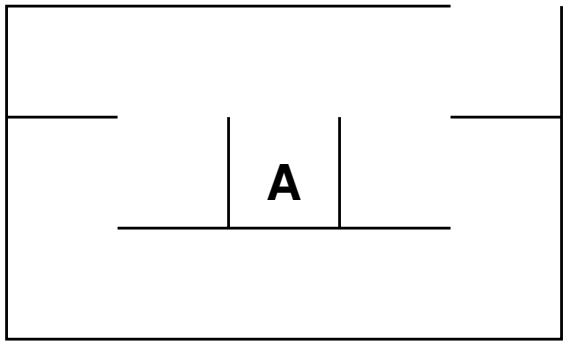


- Existe um algoritmo iterativo simples:

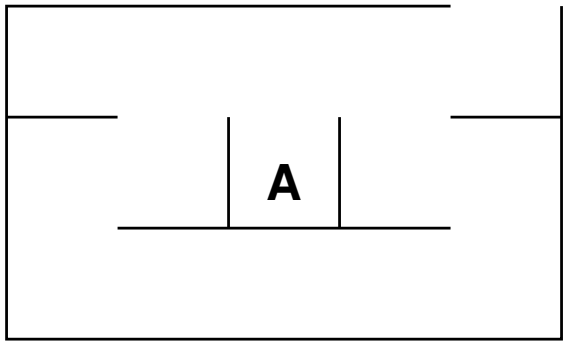
Algoritmo: Segue-parede (*Wall follower*)

1. Coloque sua mão direita contra parede.
2. **enquanto** *Não conseguir escapar do labirinto* **faça**
3. Ande pra frente mantendo a mão encostada na parede.

- Porém, para certas configurações de labirinto o algoritmo anterior **falha**!

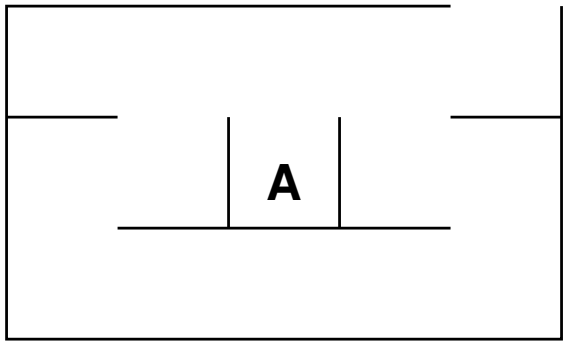


- Porém, para certas configurações de labirinto o algoritmo anterior **falha**!



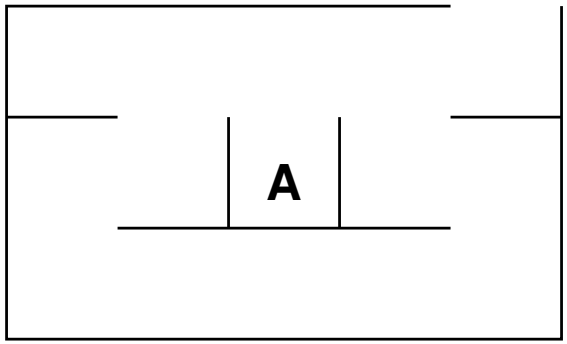
- Note que o algoritmo faz o ator ficar em laço infinito.

- Porém, para certas configurações de labirinto o algoritmo anterior **falha**!



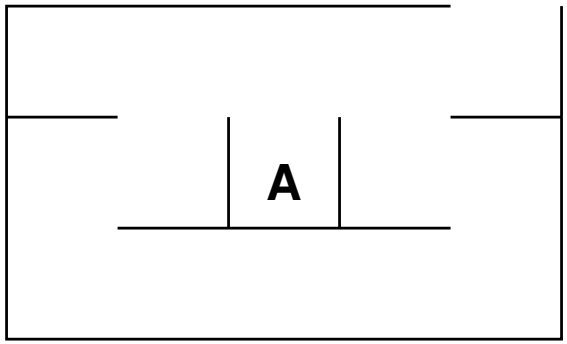
- Note que o algoritmo faz o ator ficar em laço infinito.
- Vamos tentar esboçar uma solução recursiva.

- Porém, para certas configurações de labirinto o algoritmo anterior **falha**!



- Note que o algoritmo faz o ator ficar em laço infinito.
- Vamos tentar esboçar uma solução recursiva.
  - ⋆ Precisamos achar uma simplificação do problema, criando subproblemas menores.

- Porém, para certas configurações de labirinto o algoritmo anterior **falha**!

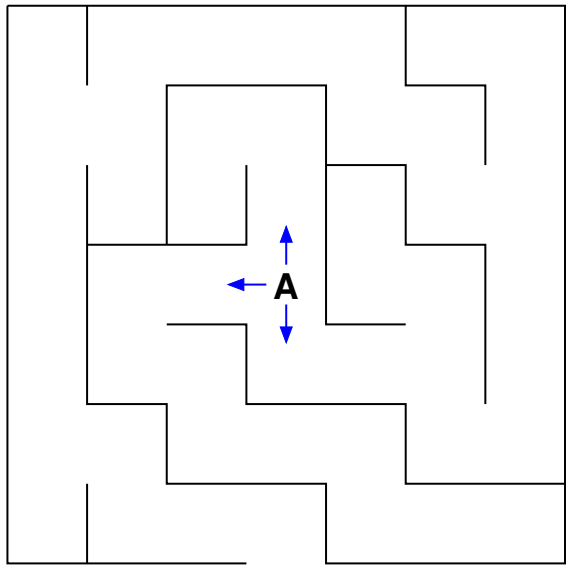


- Note que o algoritmo faz o ator ficar em laço infinito.
- Vamos tentar esboçar uma solução recursiva.
  - Precisamos achar uma simplificação do problema, criando subproblemas menores.
  - Também precisamos encontrar casos base e/ou casos simples de resolver.

Solução do Labirinto com Backtracking

Abordagem recursiva

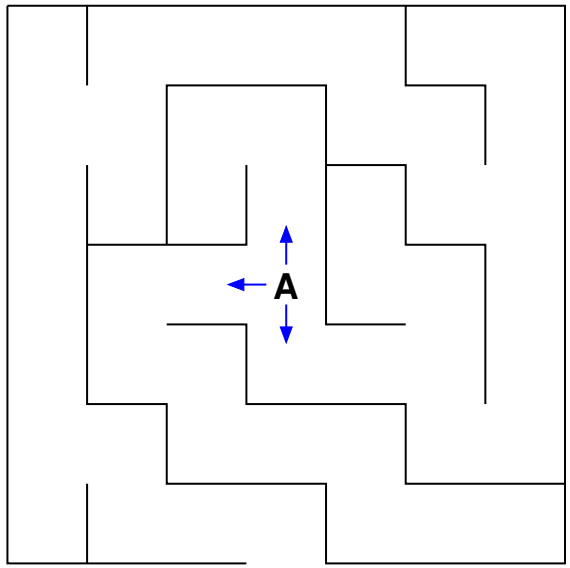
- Vamos considerar o labirinto original, para o qual existem 3 possíveis movimentos.



Solução do Labirinto com Backtracking

Abordagem recursiva

- Vamos considerar o labirinto original, para o qual existem 3 possíveis movimentos.

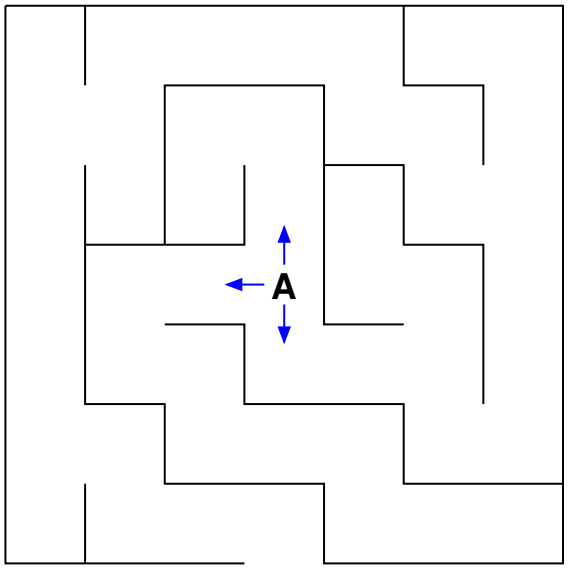


- A saída, se existir, deve estar ao longo de um destes 3 caminhos.

Solução do Labirinto com Backtracking

Abordagem recursiva

- Vamos considerar o labirinto original, para o qual existem 3 possíveis movimentos.

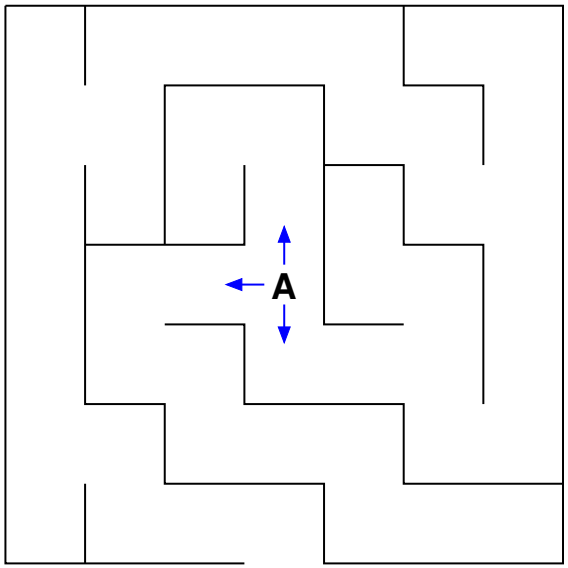


- A saída, se existir, deve estar ao longo de um destes 3 caminhos.
- Note também que estamos 1 passo mais próximo da solução.

Solução do Labirinto com Backtracking

Abordagem recursiva

- Vamos considerar o labirinto original, para o qual existem 3 possíveis movimentos.

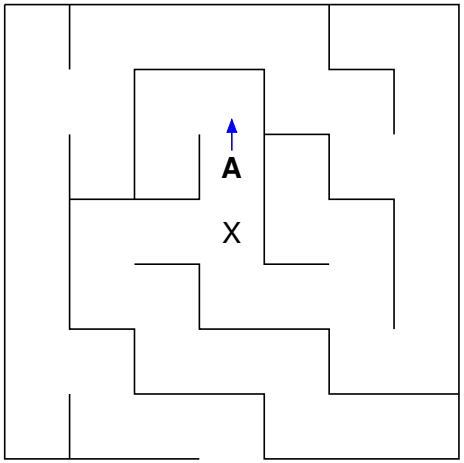


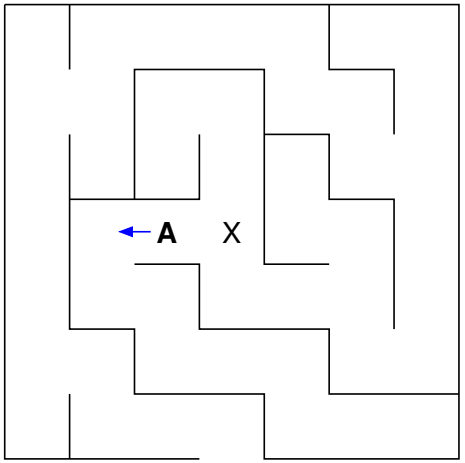
- A saída, se existir, deve estar ao longo de um destes 3 caminhos.
- Note também que estamos 1 passo mais próximo da solução.
- Isso quer dizer que o labirinto ficou mais simples ao longo de cada direção.

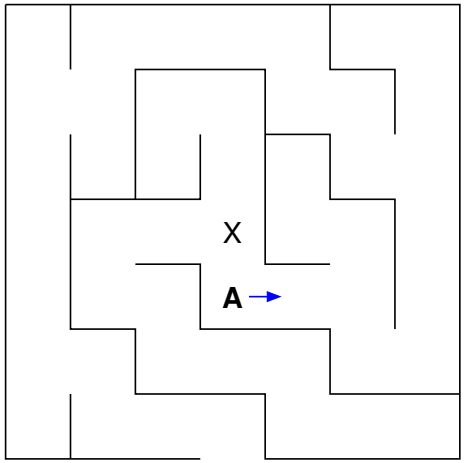
Solução do Labirinto com Backtracking

Abordagem recursiva (cont.)

- Existem, portanto, 3 (sub)labirintos mais simples:



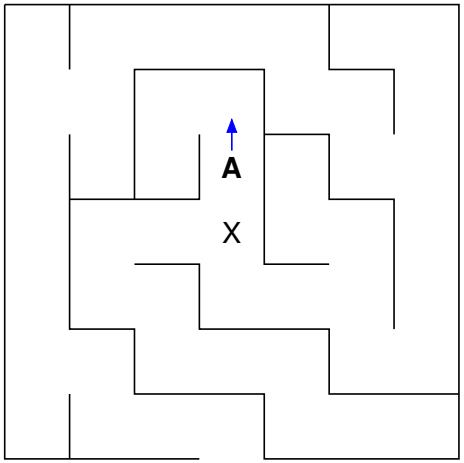


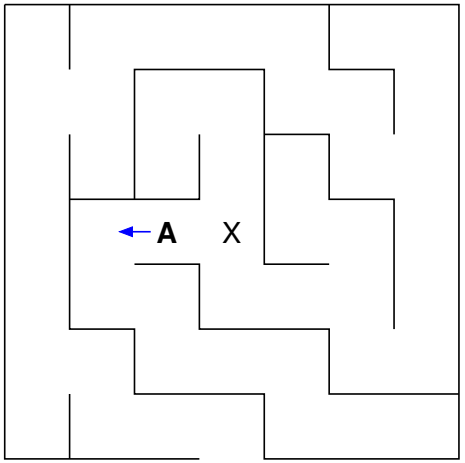


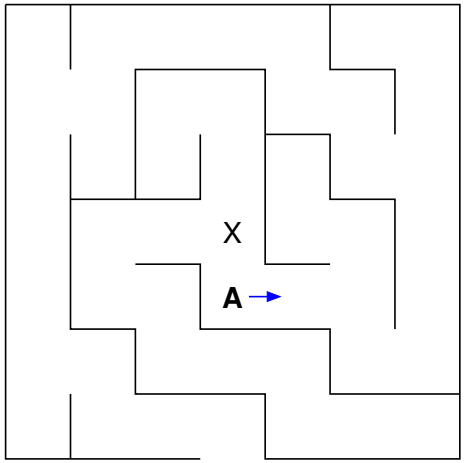
Solução do Labirinto com Backtracking

Abordagem recursiva (cont.)

- Existem, portanto, 3 (sub)labirintos mais simples:





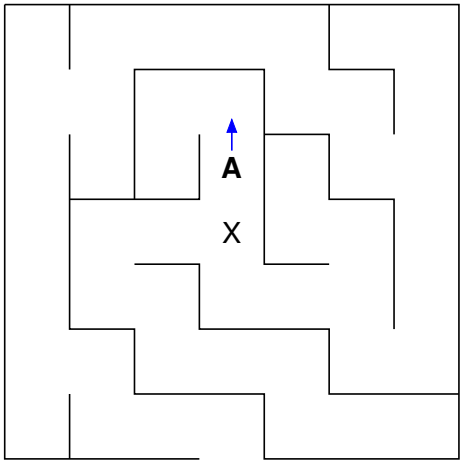


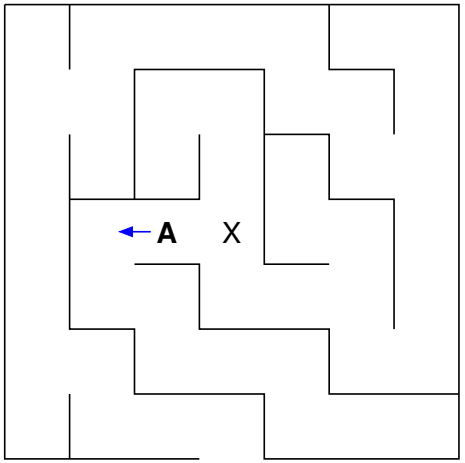
- Fazendo uma análise global, é fácil perceber que os dois primeiros levam a um beco sem saída.

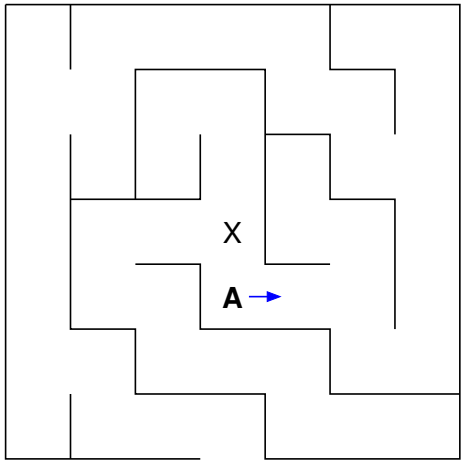
Solução do Labirinto com Backtracking

Abordagem recursiva (cont.)

- Existem, portanto, 3 (sub)labirintos mais simples:





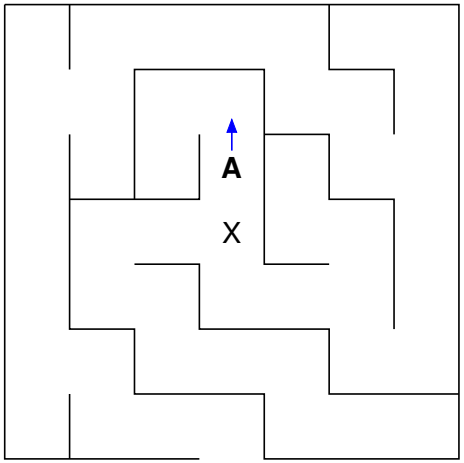


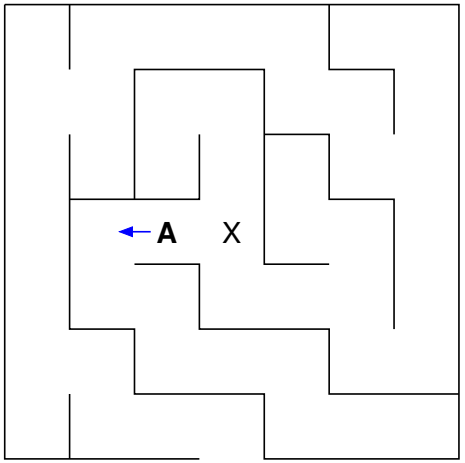
- Fazendo uma análise global, é fácil perceber que os dois primeiros levam a um beco sem saída.
- Porém, o algoritmo não tem capacidade de fazer uma "análise global" como a que fizemos.

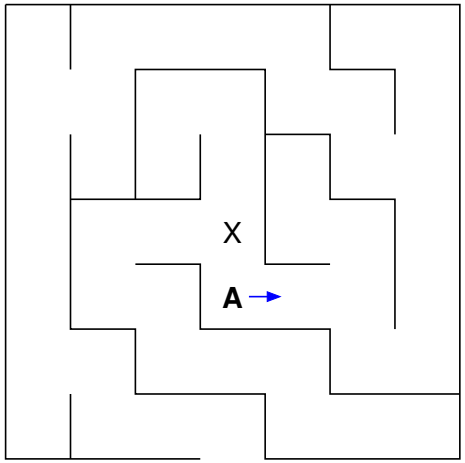
Solução do Labirinto com Backtracking

Abordagem recursiva (cont.)

- Existem, portanto, 3 (sub)labirintos mais simples:







- Fazendo uma análise global, é fácil perceber que os dois primeiros levam a um beco sem saída.
- Porém, o algoritmo não tem capacidade de fazer uma "análise global" como a que fizemos.
- Precisamos, então, identificar os casos simples para parar a recursão.

Solução do Labirinto com Backtracking

Casos simples da recursão

- Quais seriam os casos simples?

Solução do Labirinto com Backtracking

Casos simples da recursão

- Quais seriam os casos simples?
  - Ator achou a saída.

Solução do Labirinto com Backtracking

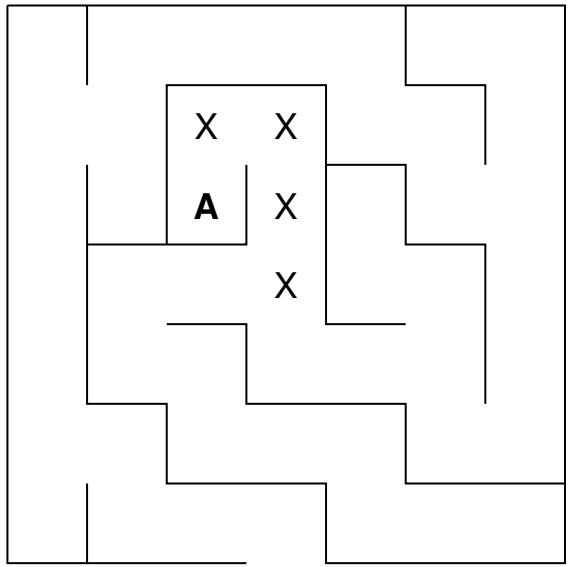
Casos simples da recursão

- Quais seriam os casos simples?
  - Ator achou a saída.
  - Ator alcançou um beco sem saída.

Solução do Labirinto com Backtracking

Casos simples da recursão

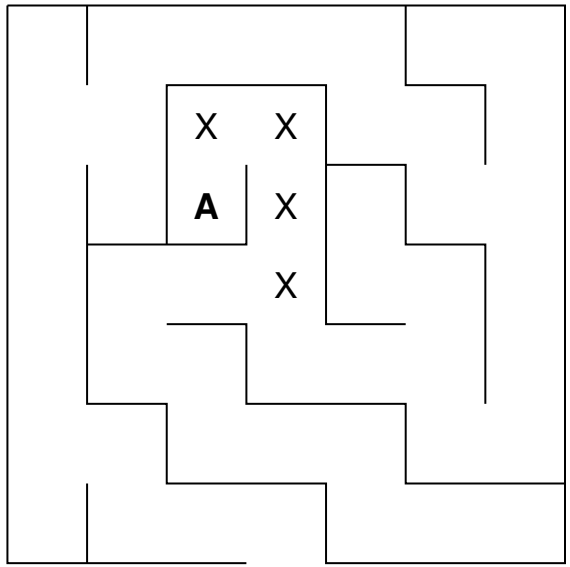
- Quais seriam os casos simples?
  - ⋆ Ator achou a saída.
  - ⋆ Ator alcançou um beco sem saída.
- Considere o primeiro sub-labirinto:



Solução do Labirinto com Backtracking

Casos simples da recursão

- Quais seriam os casos simples?
  - ⋆ Ator achou a saída.
  - ⋆ Ator alcançou um beco sem saída.
- Considere o primeiro sub-labirinto:



- Neste ponto não há mais uma célula livre para onde mover o ator, visto que as células livres já foram visitadas.

Solução do Labirinto com Backtracking

Casos simples da recursão (cont.)

- A solução recursiva fica mais fácil de codificar se, ao invés de verificar por células marcadas ao considerar uma direção de movimento, chamamos a recursão para a célula livre, independente de ser marcada ou não.

Solução do Labirinto com Backtracking

Casos simples da recursão (cont.)

- A solução recursiva fica mais fácil de codificar se, ao invés de verificar por células marcadas ao considerar uma direção de movimento, chamamos a recursão para a célula livre, independente de ser marcada ou não.
- No início da recursão verificamos os 2 casos básico (em um só lugar):

Solução do Labirinto com Backtracking

Casos simples da recursão (cont.)

- A solução recursiva fica mais fácil de codificar se, ao invés de verificar por células marcadas ao considerar uma direção de movimento, chamamos a recursão para a célula livre, independente de ser marcada ou não.
- No início da recursão verificamos os 2 casos básico (em um só lugar):
  1. Se a célula atual é a saída.

Solução do Labirinto com Backtracking

Casos simples da recursão (cont.)

- A solução recursiva fica mais fácil de codificar se, ao invés de verificar por células marcadas ao considerar uma direção de movimento, chamamos a recursão para a célula livre, independente de ser marcada ou não.
- No início da recursão verificamos os 2 casos básico (em um só lugar):
  1. Se a célula atual é a saída.
  2. Se a célula atual é marcada como visitada, não há solução ao longo do caminho que levou até esta célula.

Codificando a solução

- Precisamos definir uma representação para o labirinto, encapsulado na classe `Maze`.

Codificando a solução

- Precisamos definir uma representação para o labirinto, encapsulado na classe `Maze`.
- Em seguida, precisamos codificar uma função que resolver o labirinto, como em:

```
bool solve_maze( const Maze& mz, const Position& start )
```

Codificando a solução

- Precisamos definir uma representação para o labirinto, encapsulado na classe `Maze`.
- Em seguida, precisamos codificar uma função que resolver o labirinto, como em:

```
bool solve_maze( const Maze& mz, const Position& start )
```

- A função recebe (1) o labirinto na forma de uma classe que encapsula a estrutura de dados que escolhermos, (2) uma posição de onde começar a navegação, e retorna `true` se existir uma solução, ou `false` caso contrário.

Codificando a solução

- Precisamos definir uma representação para o labirinto, encapsulado na classe `Maze`.
- Em seguida, precisamos codificar uma função que resolver o labirinto, como em:

```
bool solve_maze( const Maze& mz, const Position& start )
```

- A função recebe (1) o labirinto na forma de uma classe que encapsula a estrutura de dados que escolhermos, (2) uma posição de onde começar a navegação, e retorna `true` se existir uma solução, ou `false` caso contrário.
- No caso de haver solução, o caminho deverá estar marcado dentro do objeto representando o labirinto.

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`
  - `bool is_outside(const Position& pos)`

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`
  - `bool is_outside(const Position& pos)`
  - `bool is_blocked(const Position& pos, const Direction& dir)`

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`
  - `bool is_outside(const Position& pos)`
  - `bool is_blocked(const Position& pos, const Direction& dir)`
  - `void mark_cell(const Position& pos )`

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`
  - `bool is_outside(const Position& pos)`
  - `bool is_blocked(const Position& pos, const Direction& dir)`
  - `void mark_cell(const Position& pos )`
  - `void unmark_cell(const Position& pos )`

Interface pública de `Maze`

- Quais seriam os métodos públicos da classe `Maze` para que a função (cliente) `solve_maze` consiga codificar o algoritmo discutido anteriormente?
  - `Position get_start_position()`
  - `bool is_outside(const Position& pos)`
  - `bool is_blocked(const Position& pos, const Direction& dir)`
  - `void mark_cell(const Position& pos )`
  - `void unmark_cell(const Position& pos )`
  - `bool is_marked(const Position& pos)`

Programa principal

- O programa principal poderia apresentar a seguinte estrutura.

Programa principal

- O programa principal poderia apresentar a seguinte estrutura.

Principal

1. Instancia um labirinto *maze* com base em um arquivo de definição.
2. Apresentar labirinto *maze* na tela.
3. **se** *solve_maze(maze, maze.get_start_position())* **então**
4. | Mostre *maze* com solução marcada na saída.
5. **senão**
6. | Labirinto sem solução.

Prorgamando o solucionador

- Função que tenta solucionar o labirinto, mas possui um erro.

Prorgamando o solucionador

- Função que tenta solucionar o labirinto, mas possui um erro.

Maze solver

```
bool solve_maze( const Maze& mz, const Position& start ) {
    if ( mz.is_outside( start ) ) return true;
    if ( mz.is_marked( start ) ) return false;
    mz.mark_cell( start );
    for_each ( Direction dir in {NORTH, EAST, SOUTH, WEST} ) {
        if ( not mz.is_blocked( start, dir ) )
            if ( solve_maze( mz, walk_to_cell( star, dir ) )
                return true;
    }
    return false;
}
```