UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS - ICEX
ALGORITMOS E ESTRUTURAS DE DADOS II

Ingrid Rosselis Sant'Ana da Cunha

TP1

Belo Horizonte
2016

● Introdução:

O problema proposto ao TP1 é encontrar um caminho em uma matriz extraída de um arquivo texto, dado os pontos de entrada e saída, sabendo que o algoritmo só pode passar num caminho de 0s e só pode mover-se em quatro direções, sem diagonais. Ao final, o trabalho deverá gerar um arquivo que contém a solução, ou seja, um caminho entre a entrada e a saída, ou escrever o número 0, indicando que não há solução para o mapa dado.

● Implementação:

Para a implementação do trabalho, além da estrutura Labirinto, contida na especificação do trabalho, foram utilizadas as seguintes estruturas para formar a pilha auxiliar:

- ✓ **TipoItem**, contendo os inteiros x e y e um vetor de direções dir;
- ✓ **Apontador**, um ponteiro para a estrutura Celula;
- ✓ **Celula**, estrutura que contém um TipoItem e um Apontador para a próxima célula e;
- ✓ **TipoPilha**, contendo os Apontadores para o topo e para o fundo da pilha e o seu tamanho.

Deve ser ressaltado, primeiramente, que o mapa pertencente à estrutura Labirinto e a matriz solução foram tratados como um único vetor para facilitar a implementação e desalocação dos espaços. As funções são divididas em duas partes: uma referente às funções gerais do trabalho e outra com as operações que a pilha pode realizar. Na primeira parte temos:

Testes: Faz testes referentes ao inteiro relacionado ao tipo de caminhamento escolhido e à extensão do arquivo de saída, ambos passados como parâmetros da main. Recebe o vetor de strings argv, passa argv[3], o inteiro do caminhamento, para o tipo int usando atoi() e testa se é 0 ou 1. Em seguida, testa se argv[2], o nome do arquivo de saída, possui como fim de string ".txt". Por fim, se todos os testes não apresentaram nenhum resultado falso, devolve o inteiro do algoritmo de caminhamento. Sua chamada é feita na main.

TestaExtensao: A função serve para testar se a extensão passada no nome do arquivo de saída corresponde a ".txt". Para o nome do arquivo de entrada, se a extensão estiver errada o arquivo não é encontrado e, logo, não é aberto, mas o arquivo de saída pode acabar sendo gerado com outra extensão ou sem uma. Recebe argv[2], separa a parte da extensão e retorna 1 se está correta e 0 se não. É chamada na função testes.

LeLabirinto: Essa função é utilizada para tratar o arquivo de entrada, extraindo suas informações e passando para a estrutura Labirinto. Ela foi um pouco alterada em relação à especificação do trabalho, seu tipo agora é void e ela recebe, além de argv[1], o ponteiro da estrutura Labirinto que armazenará o mapa, para não criar e devolver um ponteiro, tornando-a mais simples. Basicamente, a função abre com fopen() o arquivo de entrada e testa se obteve sucesso (se f != NULL), retira com um fscanf() as variáveis da primeira linha do arquivo (tamanho N, x e y como ponto de entrada e sx e sy como ponto de saída do mapa), aloca espaço para o mapa do Labirinto, lê a matriz mapa do arquivo e grava no vetor lab->mapa e, por fim, fecha o arquivo. Não possui retorno.

CaminhaLabirintoRecursivo: A função percorre o mapa do labirinto procurando um caminho da entrada até a espada. Sua implementação foi totalmente baseada no algoritmo apresentado na especificação do trabalho, sendo assim, ela recebe o ponteiro para estrutura Labirinto, os inteiros x e y referentes à posição de entrada e a matriz solução. Primeiro os x e y atuais são comparados aos da espada (condição de parada da recursão), depois o ponto atual é marcado como 1 na matriz solução, os limites do mapa são testados e as 4 chamadas recursivas (para y+1, x+1, y-1 e x-1) são feitas. Se o algoritmo não tiver opção de caminhar para nenhum de seus pontos vizinhos, o ponto é marcado com 0 na matriz solução e a função devolve falso para onde foi chamada anteriormente. O programa retornará true para sua chamada inicial na main se chegar à espada ou false se não conseguir encontrar um caminho possível.

CaminhaLabirintoIterativo: O caminhamento iterativo foi baseado no recursivo o máximo possível, tanto que as duas funções possuem os mesmos parâmetros e retornos e a pilha auxiliar implementada faz praticamente o mesmo papel da pilha do sistema no CaminhaLabirintoRecursivo. A principal diferença entre as duas pilhas é o fato de que a pilha do sistema retorna para a função no ponto onde a chamada foi feita e a pilha implementada não possui essa capacidade, por isso esta guarda o ponto por onde o caminhamento passa e um vetor com 4 inteiros - dir[0] para direita, dir[1] para baixo, dir[2] para esquerda e dir[3] para cima. Isso possibilita testar se há um possível caminho e, caso seja necessário, voltar para um ponto anterior que possua mais de uma passagem sem perder qual a última direção explorada.

Exemplificando o processo que a função executa: Primeiro, o x e y atuais são comparados com os da espada e depois é testado se há passagem nesse ponto. Caso exista, a função marca o ponto na matriz solução como 1, empilha o ponto atual, seta a flag (variável int) como true, simbolizando que existe um possível caminho ali, testa os limites do mapa para as próximas posições e se há passagem pelos vizinhos na ordem y+1, x+1, y-1 e x-1. Independente se há passagem ou não, o espaço no vetor dir correspondente àquele vizinho é marcado com 1. Se o caminhamento chegar a um beco, todos os vizinhos serão 1 e não haverá passagem, para esse caso a flag é setada como false e o programa cai no segundo caso, onde o vetor de direções é repassado para descobrir se todos as direções possíveis foram testadas. Caso não tenham sido, o programa volta com o mesmo ponto para seguir para outra parte do mapa. Se tudo foi testado e não há outra saída naquele ponto, a matriz solução é marcada com 0 ali e é gerado uma parede no mapa do labirinto, impedindo que aquela posição seja revisitada. Por fim, a posição anterior é desempilhada e o processo é repetido. Os pontos de parada para função são a chegada na espada, onde a posição é marcada como 1 e é retornado verdadeiro depois de liberar o espaço da pilha, ou o algoritmo ter voltado até a entrada e a pilha ter ficado vazia, simbolizando que não foi encontrado um caminho possível e retornado falso.

EscreveArquivo: Essa função possui como parâmetros o nome do arquivo de saída, o tamanho da matriz/vetor solução, o valor retornado pela função de caminhamento utilizada e a matriz solução. Seu funcionamento é bem simples, ela cria o arquivo de saída e testa se obteve sucesso, avalia o inteiro passado (0 ou 1) e grava no arquivo de saída a matriz solução ou o número 0. Por fim, a função fecha o arquivo e devolve 1, significando que o arquivo foi gravado com sucesso.

Libera_Memoria e **Libera_Pilha:** Como depois de toda a implementação ter sido feita foi notado que os free() usados para liberar o espaço alocado estavam dispersos (e, no caso da pilha, repetidos antes de cada return da função CaminhaLabirintoIterativo) pela main e pelas funções, foi criada a função Libera_Memoria, que recebe os ponteiros para estrutura Labirinto e para a matriz solução e dá free no ponteiro do mapa da estrutura Labirinto e nos dois principais. Como é gerado um erro ao tentar desalocar uma pilha que não existe quando o caminho recursivo é chamado, foi criada a função Libera_Pilha, que recebe o ponteiro para a pilha e dá free em todos os ponteiros de próximo, do topo, do fundo e o principal da pilha. Nenhuma das duas funções possui retorno. A Libera_Memoria é chamada ao fim da main e a Libera_Pilha antes dos return da função de caminhamento iterativo.

As funções referentes à pilha foram as estudadas em sala de aula. Brevemente:

FPVazia: Função recebe o ponteiro para a pilha e tem a função de inicializá-la e torná-la vazia, restaurando todas as condições de uma pilha nova (tamanho 0, fundo = topo e sem próximo, ou seja, apenas a cédula-cabeça). Sem retorno.

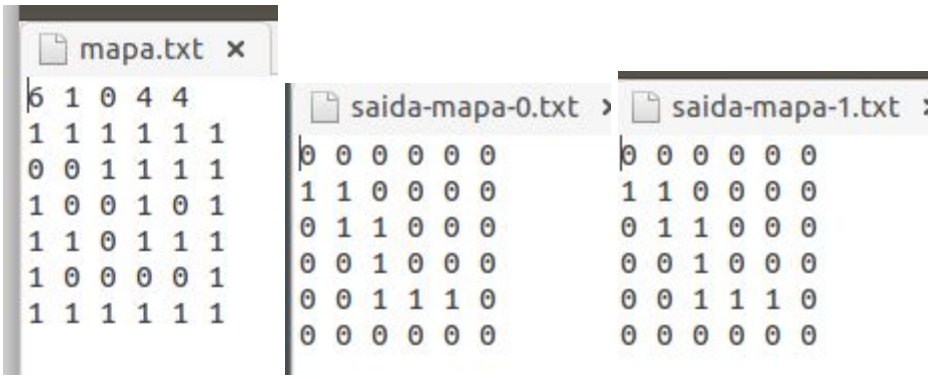
Vazia: Função que recebe um ponteiro para a pilha e devolve se ela está vazia ou não. Foi utilizada como condição do while no caminhamento iterativo.

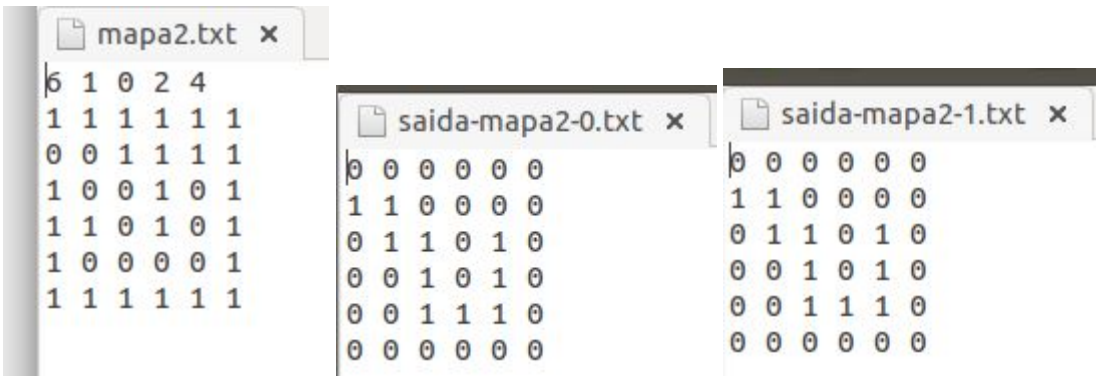
Empilha: Função recebe uma estrutura TipoItem e um ponteiro para pilha e adiciona o item ao topo. Sem retorno.

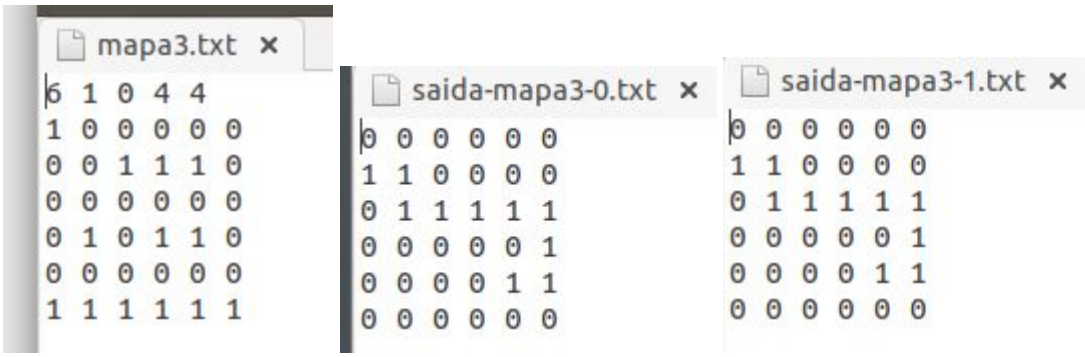
Desempilha: Função recebe um ponteiro para a pilha e retira o elemento do topo desta, devolvendo-o.

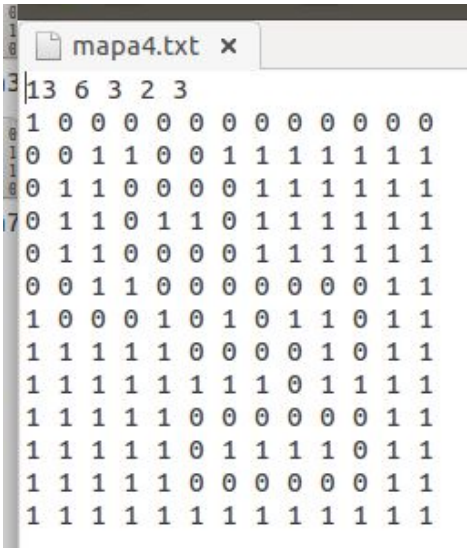
- **Resultados:**

Para testar o programa, além dos dois casos de teste que foram disponibilizados, foram usados mais 10 mapas diferentes. As imagens a seguir são os mapas originais, a saída recursiva e a saída iterativa, respectivamente:



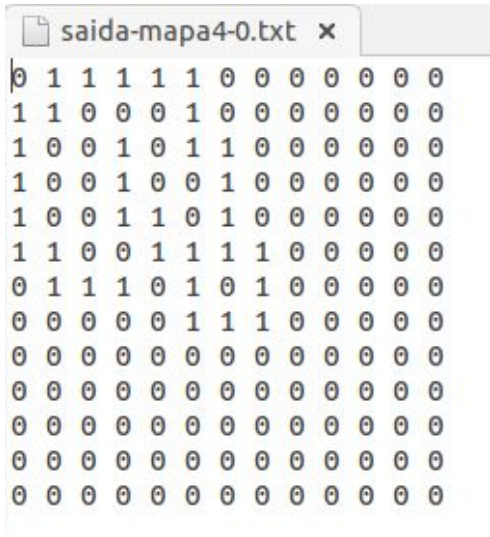






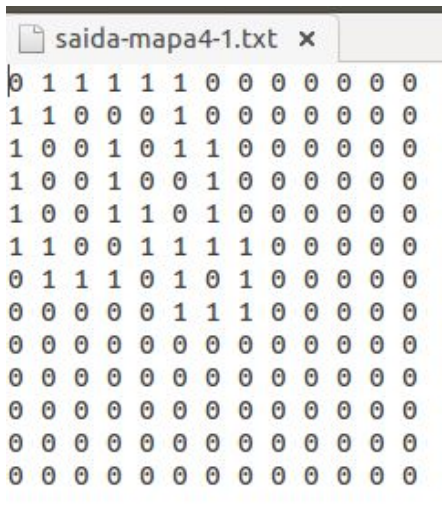
mapa4.txt

```
13 6 3 2 3
1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 1 1 1 1 1 1 1
0 1 1 0 0 0 0 1 1 1 1 1 1 1
0 1 1 0 1 1 0 1 1 1 1 1 1 1
0 1 1 0 0 0 0 1 1 1 1 1 1 1
0 0 1 1 0 0 0 0 0 0 0 1 1 1
1 0 0 0 1 0 1 0 1 1 0 1 1 1
1 1 1 1 1 1 0 0 0 0 1 0 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 1 1
1 1 1 1 1 1 0 1 1 1 1 0 1 1
1 1 1 1 1 1 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
```



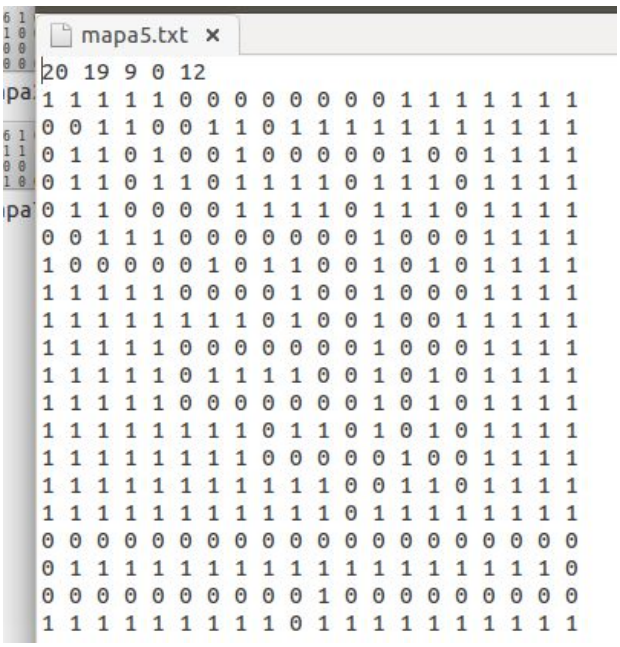
saida-mapa4-0.txt

```
0 1 1 1 1 1 0 0 0 0 0 0 0
1 1 0 0 0 1 0 0 0 0 0 0 0
1 0 0 1 0 1 1 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0 0
1 0 0 1 1 0 1 0 0 0 0 0 0
1 1 0 0 1 1 1 1 0 0 0 0 0
0 1 1 1 0 1 0 1 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```



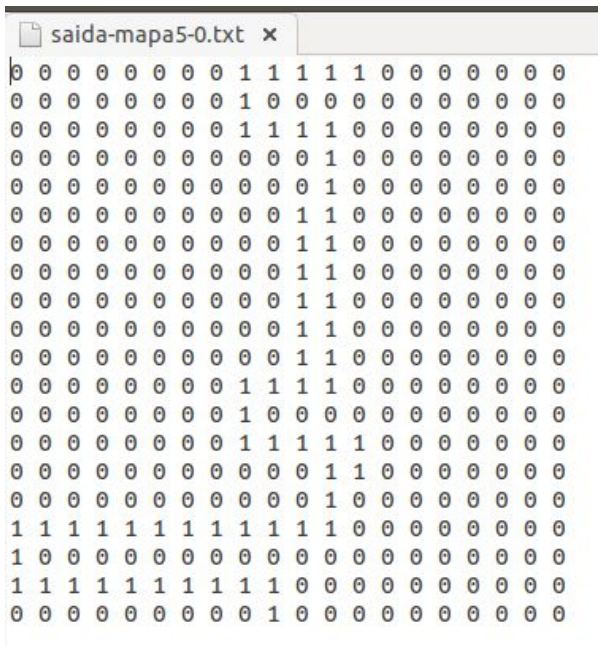
saida-mapa4-1.txt

```
0 1 1 1 1 1 0 0 0 0 0 0 0
1 1 0 0 0 1 0 0 0 0 0 0 0
1 0 0 1 0 1 1 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0 0
1 0 0 1 1 0 1 0 0 0 0 0 0
1 1 0 0 1 1 1 1 0 0 0 0 0
0 1 1 1 0 1 0 1 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```



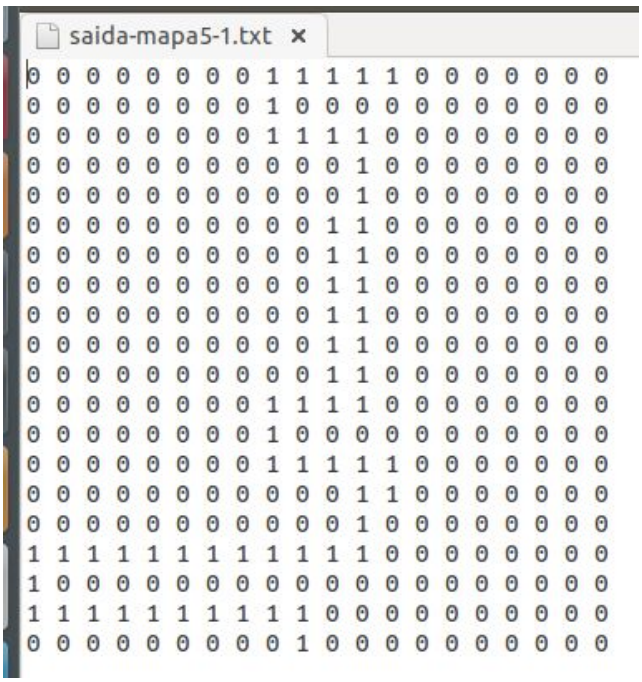
mapa5.txt

```
20 19 9 0 12
1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 1 1 1 1
0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1
0 1 1 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1
0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1
1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
```



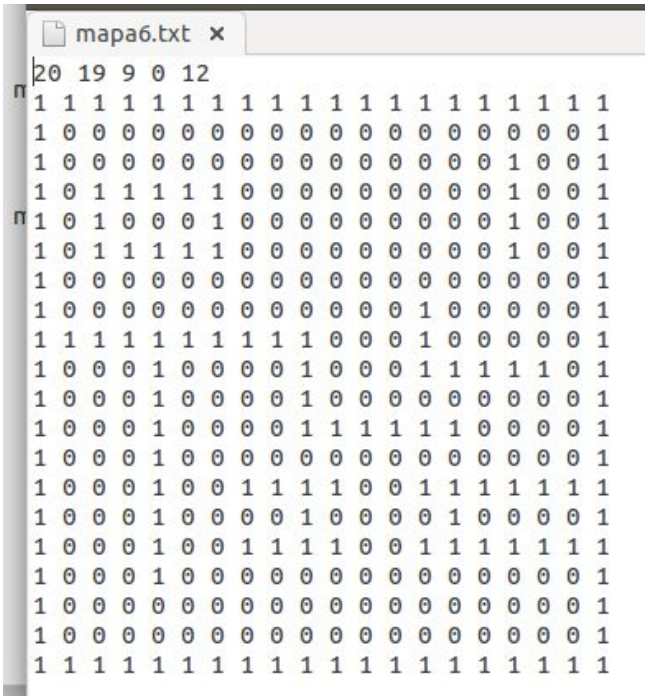
saida-mapa5-0.txt

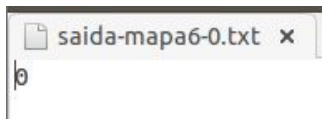
```
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
```

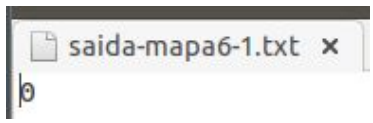


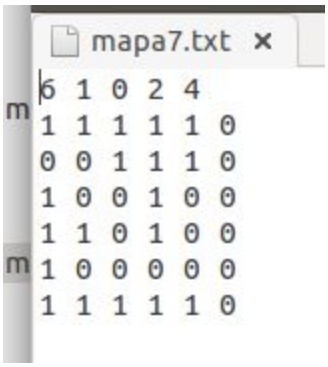
saida-mapa5-1.txt

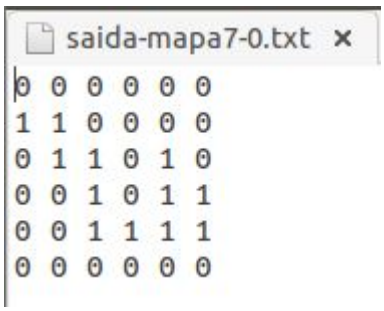
```
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
```


mapa6.txt
20 19 9 0 12
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1
1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1
1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 1 0 1
1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

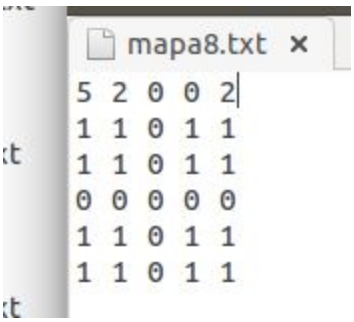

saida-mapa6-0.txt
0

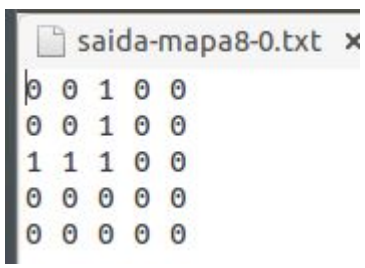

saida-mapa6-1.txt
0

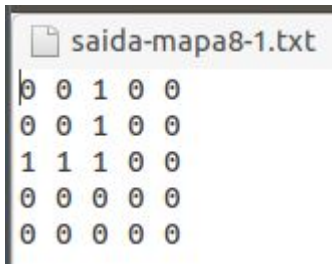

mapa7.txt
6 1 0 2 4
1 1 1 1 1 0
0 0 1 1 1 0
1 0 0 1 0 0
1 1 0 1 0 0
1 0 0 0 0 0
1 1 1 1 1 0


saida-mapa7-0.txt
0 0 0 0 0 0
1 1 0 0 0 0
0 1 1 0 1 0
0 0 1 0 1 1
0 0 1 1 1 1
0 0 0 0 0 0


saida-mapa7-1.txt
0 0 0 0 0 0
1 1 0 0 0 0
0 1 1 0 1 0
0 0 1 0 1 1
0 0 1 1 1 1
0 0 0 0 0 0


mapa8.txt
5 2 0 0 2
1 1 0 1 1
1 1 0 1 1
0 0 0 0 0
1 1 0 1 1
1 1 0 1 1


saida-mapa8-0.txt
0 0 1 0 0
0 0 1 0 0
1 1 1 0 0
0 0 0 0 0
0 0 0 0 0


saida-mapa8-1.txt
0 0 1 0 0
0 0 1 0 0
1 1 1 0 0
0 0 0 0 0
0 0 0 0 0

lab20x20.txt

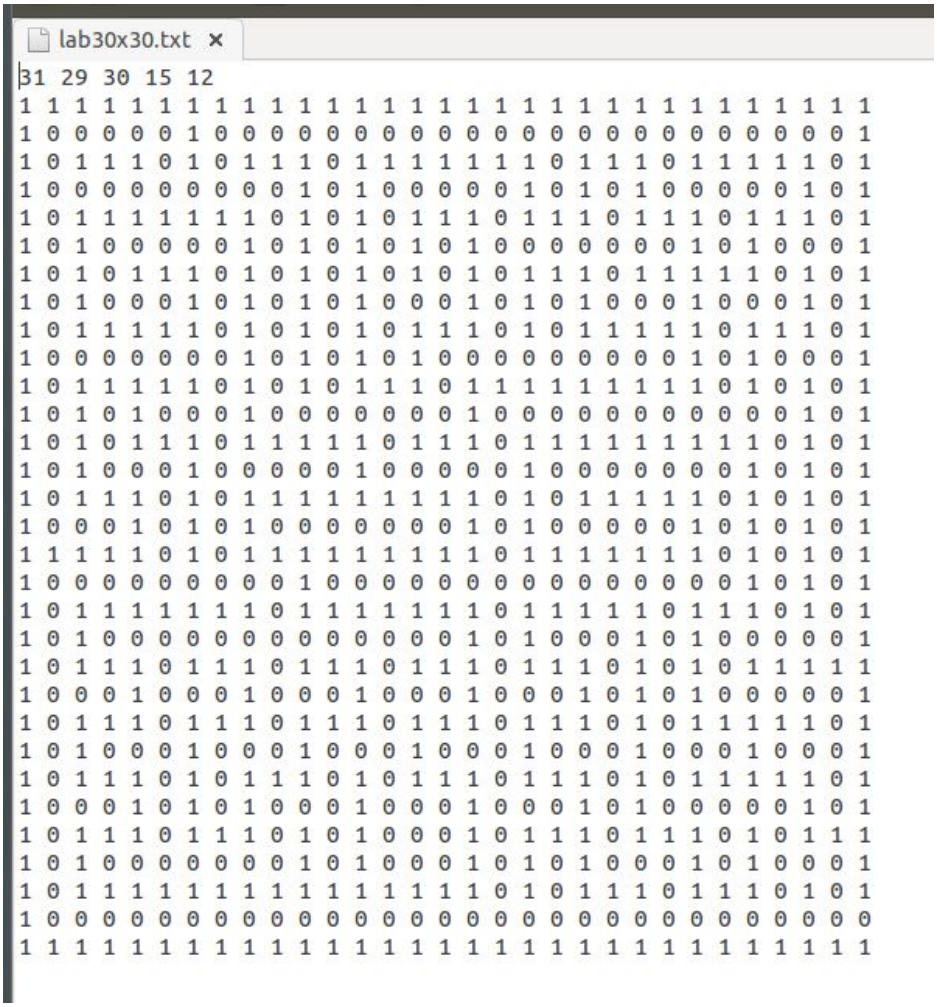
```
19 5 0 7 5
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1
1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1
1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1
0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1
1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1
1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1
1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1
1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1
1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1
1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 1
1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1
1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1
1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

saida-lab20x20-0.txt

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0
1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0
0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0
0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

saida-lab20x20-1.txt

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0
1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```



Complexidade: Para encontrar um ponto em um mapa deve-se andar um certo número de posições a partir do ponto de entrada. Sendo o mapa uma matriz quadrada, tem-se, no máximo, n^2 posições possíveis para deslocar. Logo, o melhor caso possível para o caminhamento iterativo seria ter a posição de entrada igual à posição da espada, sendo necessário apenas uma comparação e tendo complexidade igual a **O(1)**; o caso médio dependeria da probabilidade que cada ponto dentro da matriz possui e; o pior caso seria passar por todos os nodos antes de encontrar a espada, sendo necessárias n^2 comparações, tendo complexidade igual a **O(n^2)**. Considerando o espaço utilizado, a pilha implementada guarda/retira um ponto visitado (pode fazê-lo mais de uma vez para o mesmo ponto, se necessário), então o espaço requerido não ultrapassa n^2, sendo n o tamanho da dimensão da matriz.

Para o cálculo do caminhamento iterativo, a linha de pensamento foi semelhante. Porém, a equação de recorrência gerou várias dúvidas à respeito de qual seria sua forma correta, sendo assim, a fórmula aqui apresentada é uma tentativa baseada no raciocínio que será apresentado a seguir:

Tomando $x = n^2$.

»$T(x) = T(x - 1) + 1$, para $x > 1$

»$T(1) = 1$, para $x = 1$

$T(x - 1) = T(x - 2) + 1$

$T(x - 2) = T(x - 3) + 1$

.

.

.

$T(x - i + 1) = T(x - i) + i.1$ $x - i = 1 \rightarrow i = x - 1$

$T(x) = T(x - (x - 1)) + \square(i = 1 \rightarrow x)\ i$

$T(x) = T(1) + (x(x + 1))/2$

$T(x) = 1 + x^2/2 + x/2$

$T(n^2) = n^4/2 + n^2/2 + 1 \rightarrow$ **$O(n^4)$**

Para chegar a essa equação foi considerado, por exemplo, a equação de recorrência do fatorial. Nesse caso, cada passada diminui em 1 no tamanho do n considerado, até chegar ao ponto de parada 1. No caminhamento na matriz, cada vez que a função é chamada também diminui em 1 o número de "possibilidades" restantes. Só que o n, neste caso, é n^2 porque a matriz é nxn. Outro ponto é que em equações de recorrência geralmente é considerado o número de chamadas da função, contudo, no caso do caminhamento, penso que seria errado considerar 4 vezes a chamada porque a cada vez só uma chamada é considerada. Se existe apenas um caminho, a função é chamada uma vez pra cada ponto. Se for necessário voltar àquele ponto, a chamada feita anteriormente não será feita de novo.

Baseado nas complexidades aqui apresentadas o algoritmo tem complexidade polinomial. Existem algoritmos melhor elaborados para teoria dos grafos que possuem complexidade menor, pois utilizam outras abordagens, como usar grafos e a ideia de heurística - são feitas várias contas matemática para determinar qual é a maior probabilidade do ponto de saída desejado estar próximo a determinada direção e assim o algoritmo toma um rumo mais específico, em vez de testar caminhos. Exemplos desse são o algoritmo de Dijkstra, a busca por abrangência e o A*.

- **Conclusão:**

As principais dificuldades para este trabalho foram encontrar a equação de recorrência e o caminhamento iterativo - não sua implementação, mas entender seu uso da pilha auxiliar e debugar os primeiros problemas vindos da implementação. A equação de recorrência gerou dúvidas pertinentes e, no fim, não trás a certeza de que está correta. A implementação do algoritmo recursivo é mais simples, quase como se existe alguma mágica ali. Já a implementação iterativa requer cuidado com muitos detalhes, como alteração da ordem de alguma instrução. No meu caso, demorou até descobrir que duas instruções estavam em posições trocadas e que daí vinha o erro. Trabalhar com a pilha foi desafiante e sempre havia algo para adicionar até que o resultado parecesse com o recursivo. O último problema que foi resolvido foi o fato de que o ponto que possuísse 4 direções para seguir, nunca passava pela última direção prevista(no caso, para cima), este foi resolvido alterando o código para um do while que adiciona o ponto atual na pilha logo depois do teste se há passagem.

Por fim, utilizar o algoritmo passado na especificação para implementar o caminhamento recursivo e o iterativo mostrou os pontos fortes e fracos de cada um deles. As soluções apresentadas pelo caminhamento recursivo sempre foram bem limpas e direto ao ponto de saída, sem muitas voltas ou becos. Já o caminhamento iterativo, nas primeiras soluções do algoritmo, dava muitas voltas e tinha muitos 1s em lugares onde não havia saída, quando não tinha problemas de segmentação ou deixavam pontos que ligavam o caminho como 0s na solução. Pode-se concluir que a implementação recursiva é, de certa forma, um pouco mais fácil, mas utiliza mais espaço pelas chamadas da função e é mais lenta que a enquanto a iterativa.

- **Referências:**

https://en.wikipedia.org/wiki/Depth-first_search

http://www.redblobgames.com/pathfinding/a-star/introduction.html