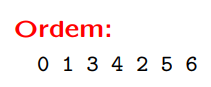
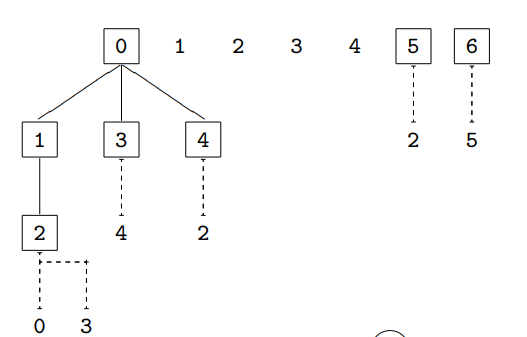
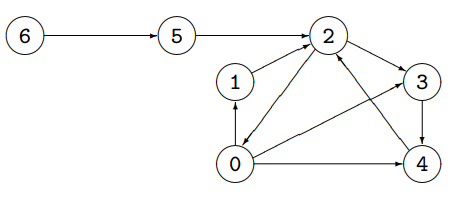
Was it a Dream  
Relatório



# Resolução do Problema

Para a resolução do problema apresentado neste trabalho, utilizamos o algoritmo Breadth-First Search Traversal, em português, procura em largura.



Este algoritmo percorre a estrutura de dados, neste caso uma matriz do tipo char, de forma sistemática e em ordem de largura. Começa no ponto de partida, escolhido por input, e vai explorando todos os caminhos possíveis, esquerda direita cima e baixo, antes de passar para a posição seguinte.

O algoritmo mantém uma fila de posições a serem exploradas. A posição inicial e colocado na fila e marcado como visitado. Em seguida enquanto a fila não estiver vazia o algoritmo remove a posição na frente da fila, explora os caminhos possíveis, marca-os como visitados e os caminhos que não terminam fora do mapa ou num no já visitado são colocados na parte de trás da fila. Este processo é repetido até a fila ficar vazia ou até chegar ao Hole.

# Complexidade Temporal

Na nossa solução, para alem de recorrermos à matriz de jogo, que é percorrida para ser colocada uma limitação nas suas bordas, com um X, utilizamos também uma matriz *found*, que guarda todas as posições que já foram visitadas do tamanho da matriz de jogo a ser percorrida, e uma fila de espera.

Assim, a complexidade temporal da nossa solução é:

- Percorrer a matriz de jogo -

- Criação e inicialização da matriz Found -

- Execução do algoritmo de resolução - O

- Total -

O algoritmo de resolução do problema

# Complexidade Espacial

Na nossa implementação, fazemos uso de duas matrizes cujo seu tamanho depende do tamanho de jogo. Para alem das matrizes utilizamos também uma fila de espera. Concluímos que a complexidade espacial iria ser:

- Matriz de jogo -

- Matriz Found -

- Fila *waiting* -

- Total -

# Conclusões

Ao analisarmos o problema que nos foi apresentado neste trabalho, o primeiro algoritmo que pensamos utilizar foi o de pesquisa em profundidade. Apercebemos-mos, ao analisarmos com mais detalhe o enunciado, os problemas que esses algoritmos poderiam trazer relativamente à complexidade temporal como espacial.   
 A solução foi otimizada por fim com o algoritmo de pesquisa em largura, já mencionado.

*Inicialmente otimizamos uma solução onde era preenchida uma tabela de tamanho C x L onde C era o tamanho do percurso a percorrer e L era os tipos de itens que podemos carregar, incluindo o vazio. Este preenchimento era feito através de programação dinâmica. Posteriormente, de modo a melhorar a complexidade espacial, passamos a utilizar um array, também preenchido através de programação dinâmica, de tamanho L onde L são os tipos de itens que podemos carregar, incluindo o vazio. Deste modo conseguimos reduzir o espaço utilizado na nossa solução em C.*

*Depois desta alteração não encontramos possíveis melhoramentos*

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente