ADA - Projeto 2

60477 Miguel Valido
60694 Pedro Fernandes

Maio 2023

1 Resolução do Problema

Para resolver este problema, decidimos usar o algoritmo de pesquisa em largura, com duas filas, uma que representa o passo atual, e outra que representa o passo seguinte. Enquanto se percorre o passo atual, vai-se adicionando os sucessores dum dado nó à outra fila. Quando uma fila fica vazia, trocamos as duas. Desta maneira, é possível saber em que passo estamos (número de trocas efetuadas). Se as duas filas eventualmente ficarem vazias em simultâneo, é porque não existe forma de alcançar o buraco (stuck).

Cada elemento na fila é codificado num inteiro que indica a sua posição no campo.

Para impedir a possibilidade de ciclos infinitos, registamos numa matriz de booleanos, semelhante à matriz de entrada, as posições já visitadas pela pesquisa (essa posição já esteve na fila). Uma posição é considerada como alcançada, se a partir desta, a esfera tiver ido contra um obstáculo.

Para evitar a repetição de cálculos, fazemos uso dum grafo em listas de adjacências, que vai registando as posições previamente alcançadas a partir duma outra posição. Desta maneira, reduzimos drasticamente o cálculo de caminhos entre vários casos de teste, pois uma vez percorrido, não volta a ser calculado.

2 Legenda

- R - número de linhas da matriz de entrada
- C - número de colunas da matriz de entrada

3 Complexidade Temporal

Para cada nó, existe a necessidade de percorrer nas quatro direções até colidir com um obstáculo, cair do mapa ou alcançar o buraco. Por este motivo, no pior caso, percorre-se a linha inteira e a coluna inteira onde este nó se encontra. Isto apenas acontece uma vez para cada nó, pois nas pesquisas consequentes os

destinos estão guardados como sucessores num grafo em listas de adjacências. Por isso, no pior caso, existe a necessidade de percorrer todas as casas do campo, e para cada casa do campo, a necessidade de descobrir os seus sucessores. Ao criar um grafo com listas de adjacências, é possível reduzir o custo entre casos de teste diferentes.

As funções seguintes descrevem as operações implementadas, e as suas complexidades temporais:

Função	Descrição	Complexidade
tracePath	Percorrer numa direção	$O(R) \vee O(C)$
tracePaths	Percorrer todas as direções	$O(R + C)$
skipTrace	Saltar para os destinos calculados anteriormente	$O(4) = O(1)$
processNode	Processamento dum nó	$O(R + C)$
solve	Resolução do problema	$O(R \times C \times (R + C))$

4 Complexidade Espacial

Neste algoritmo, fizemos uso de duas filas implementadas em listas ligadas, uma matriz de booleanos e uma matriz de listas de adjacências. Cada lista de adjacências tem no máximo tamanho 4. As filas estão sempre a variar de tamanho, nunca atingindo $R \times C$ elementos.

Demos uso às seguintes estruturas de dados, cujas complexidades espaciais estão descritas:

Estruturas de dados	Descrição	Complexidade
ready e waiting	Filas dos nós a percorrer	$O(R \times C)$
reached	Matriz de nós percorridos	$\Theta(R \times C)$
skipGraph	Matriz de listas de adjacências	$\Theta(R \times C)$
-	Total	$\Theta(R \times C)$

5 Conclusões

Uma das alternativas estudadas foi usar vetores de inteiros ou objetos do tipo *Node* especificados com os campos necessários em vez de codificar os nós em inteiros. Uma desvantagem de codificar os nós é o ligeiro custo de descodificação constante. Um ponto forte é a reutilização de resultados previamente calculados, através do uso de um grafo, que reduz o custo total entre vários casos de teste.