Universidade de São Paulo

EP3 – Labirintos Perfeitos

São Paulo
2015

Objetivo

Dado um inteiro N (e opcionalmente uma seed), gerar o caminho da solução para cada par de coordenadas, vindas da entrada padrão, formadas por 2 inteiros maiores que 0 e menores que N. A solução deve ter tempo proporcional ao tamanho do caminho entre esses pontos.

Solução

A forma de resolver proposta para resolver EP foi criar uma árvore, representando os possíveis caminhos do labirinto. Assim, devido a propriedade do labirinto perfeito, percorremos a árvore do nó de cada ponto até a raiz, ou até que um dos caminhos intercepte o outro. Quando se encontrarem, temos o caminho do ponto 1 até a interseção, e o caminho da interseção até o ponto 2. Isso faz com que o tempo gasto seja proporcional ao caminho percorrido. Seja C1 e C2 o primeiro e segundo caminho, respectivamente, e gerados simultaneamente. Assim, percorreremos a árvore no máximo C vezes, sendo C o maior valor entre C1 e C2. O pior caso é aquele em que um dos caminhos passa pelo ponto de origem do outro, sem interceptar em nenhum outro lugar, e mesmo assim, tem tempo O(C), e no caso, C é o tamanho do caminho entre os pontos dados.

Classes:

O código original de Maze.java foi alterado, para gerar o labirinto e montar a solução.

Uma nova foi criada, **MatrixVert**, que é a representação das possíveis posições do labirinto. Ela contém os atributos **x**, **y**, que equivalem aos índices da matriz do labirinto, e **ID**, que representa esses índices como um inteiro, para poder ser identificado. Essa classe ainda possui os métodos **int X()** e **int Y()**, que retornam as coordenadas x e y, respectivamente, e **int XYtoID (int x, int y, int N)**, que retorna um determinado inteiro, para um x e y.

Variáveis:

Além das já contidas em *Maze.java* orignal, foram acrescentadas 5 novas:

MatrixVert [] edgeTo:

Similar ao edgeTo de uma dfs, cada vez que uma parede do labirinto é “derrubada”, colocamos em edgeTo que esse novo caminho pode ser alcançado passando pela posição anterior. No fundo, criamos uma árvore com os possíveis caminhos.

MatrixVert [] Vertices:

Vetor que contém todas as posições possíveis do labirinto.

MatrixVert V:

V é a representação de uma posição do labirinto. É identificada através de um inteiro.

boolean [] path e boolean [] path2:

Semelhante ao marked[] de dfs, guardam que posições foram visitadas ao caminhar pelo labirinto (no caso, a árvore).

Métodos:

Alguns métodos foram modificados, para que a exigência do EP fosse satisfeita, e outros foram acrescentados:

void generate (int x, int y):

Esse método já estava contido em Maze.java, mas cada vez que ele decide que parede apagar, colocamos em edgeTo[ ] esse caminho. Assim, temos o controle de todos os possíveis caminhos desse labirinto.

MatrixVert pathTo (int start1, int start2, int end, int N):

Esse método recebe os IDs dos pontos, e monta um caminho até a origem, (1, 1), até que os caminhos se encontrem.

Como o labirinto é perfeito, ou seja, só existe um único caminho entre 2 pontos, fazemos um caminho entre esses dois pontos até a origem (devido a forma em que o labirinto foi criado), até que eles passem pelo mesmo lugar, e vamos guardando esse caminho em path[ ] e path2[ ]. Em outras palavras, percorremos a árvore, partindo do nó de cada ponto, simultaneamente, até a raiz, ou até que eles um deles tenha passado por um lugar já visitado. Então, retornamos esse ponto de interseção.

void drawsolve (int start1, int start2, int end):

Agora que já sabemos onde os caminhos se encontram, fazemos novamente esse caminho, mas agora até esse ponto de interseção encontrado, desenhando esse caminho.