Trabalho Prático 2: Andando na Física

Algoritmos e Estruturas de Dados III - Lucas Fonseca Mundim - 2015042134

1 Introdução

O objetivo do TP2 é aprender a trabalhar com grafos e operações relevantes a ele. Em especial, nesse trabalho prático a tarefa a ser realizada é calcular a menor distância entre dois pontos em um mapa peculiar, com portas, chaves e buracos de minhoca (*wormholes*).

2 Solução do Problema

Para a solução do problema, utilizou-se uma variante entre o algoritmo de Busca em Largura (*BFS*) e o algoritmo de *Dijkstra*. Os algoritmos citados são utilizados em grafos para encontrar o menor caminho entre dois pontos, precisamente o que é necessário para esse trabalho.

Para a utilização do algoritmo supracitado, foram criados *TADs* representando o *grid* recebido pelo usuário, as células desse *grid* e o grafo criado com os dados recebidos. Depois de receber os dados do usuário, foi necessário interpretá-los. Isso foi feito da seguinte maneira: as dimensões do grid são recebidas e, para cada linha seguinte, armazenava-se o dado da célula em um *struct*; ao final do armazenamento, percorre-se o grid, preenchendo um grafo (com as dimensões iguais ao número de células no grid) de forma que cada aresta entre dois vértices é dada por um único dígito 1 na casa relevante no grafo. Note que na maioria dos casos, as arestas são bidirecionais, salvo em casos de *wormholes* e portas, até então, fechadas.

Depois disso, armazenou-se todas as posições das chaves em um vetor estático. Essas posições serão usadas para seguir a lógica de força bruta implementada no trabalho.

Para a caminhada, foi utilizado o algoritmo citado inicialmente que, dado um grafo e um ponto inicial, calcula a distância entre o ponto escolhido e todos os outros pontos, retornando a distância entre o inicial e um ponto de interesse. Isso foi pensado inicialmente com intenção de implementação de algoritmos de programação dinâmica, mas descartado posteriormente. Note que, se não houver caminho possível entre os pontos, é retornado um número padronizado grande e fora de alcance das situações do trabalho, indicando a impossibilidade de "sair da Física".

O algoritmo foi usado então para uma "caminhada" seguindo a lógica de força bruta. Inicialmente calcula-se a distância entre o ponto inicial e o final e armazena-se o valor recebido. Feito isso, o algoritmo caminha até uma chave e dessa chave até a saída, calculando novamente o caminho e retornando esse valor. Se o novo valor for menor que o anterior, ele é substituído. Isso é feito de forma iterativa para todas as chaves e, caso possam ser carregadas duas ou mais chaves, para todas as combinações possíveis de chaves. Ao final, o menor valor é retornado para o programa principal e condicionalmente impresso.

Note que, para as portas fechadas, a cada chave coletada pelo aluno, todas as portas relativas à ela são abertas (tendo em vista que não consome-se tempo para abri-las).

Ao final, toda a memória alocada é liberada e o programa é encerrado.

3 Análise de Complexidade

3.1 Complexidade de Tempo

3.1.1 Abrir e Fechar Portas

Ambas as funções possuem apenas dois *for's* aninhados, portanto a complexidade de tempo de ambas é da ordem de $O(n * m)$, onde n e m são as dimensões do grid passado pelo usuário.

3.1.2 Criação do Grafo e Criação do Mapa

A criação do grafo a partir do grid dado pelo usuário possui complexidade $O(n*m)$ pelo mesmo motivo da função anterior. Caso encontre um *wormhole* entretanto essa complexidade pode alterar-se, devido ao *loop while* inserido na função, sendo $O(n*m*w)$ nesses casos, onde w é o número de *wormholes* sequenciais.

3.1.3 Cálculo de Menor Caminho (Dijkstra)

O algoritmo utilizado para calcular o menor caminho é baseado em Dijkstra e, como tal, possui dois *for's* aninhados, levando a complexidade da função para $O(v^2)$, onde v é o número de vértices do grafo, dado por $n * m$.

3.1.4 Caminhada

O custo temporal da "caminhada" pelo grafo em busca do menor caminho utiliza a função de Dijkstra em seu interior múltiplas vezes, obtendo uma complexidade então de $O(k * v^2)$, onde k é o número de chaves que o aluno pode pegar.

3.1.5 Complexidade Total

Dadas as ordens de complexidade de tempo acima, a complexidade total é a maior dentre elas. Como as maiores são dependentes de n e m, a maior dentre elas é $O(v^2)$, que pode ser vista como $O((n * m)^2)$.

4 Avaliação Experimental

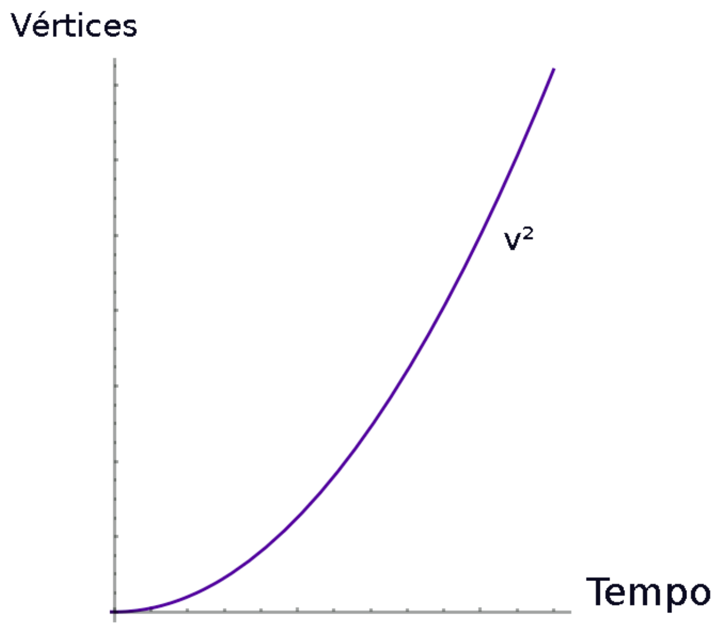
4.1 Complexidade de Espaço

São criadas dinâmicamente duas matrizes para armazenar os dados, sendo elas:

- Grid: $O(m * n)$, onde m e n são as dimensões dadas pelo usuário;
- Matriz de Adjacência: $O(v^2)$, onde v = m*n, representando o número de vértices do grafo.

4.2 Análise Experimental

Dadas as limitações de dimensão do trabalho prático, os casos teste gerados para teste de tempo foram pequenos demais para serem conclusivos, obtendo tempo abaixo de 1 segundo mesmo para mapas 10x10 (que resultavam em matrizes 100x100), tornando os dados insuficientes para a produção de um gráfico. Segue abaixo o gráfico então da complexidade total de tempo do programa.



5 Conclusão

O trabalho permitiu concluir que, em casos onde o grafo possui uma densidade baixa de arestas, mostra-se pouco eficiente a utilização de uma matriz para trabalhar com o grafo. Porém, como o trabalho limitou o número de vértices, tornou-se viável a opção, possibilitando a resolução do problema com implementações de força bruta.

Trabalhar com mapas contendo portas e buracos de minhoca pode se mostrar um tanto quanto trabalhoso, considerando as condições do buraco de minhoca. Além disso, dado que o grafo não era ponderado e era direcionado, era possível utilizar um simples algoritmo de BFS para resolver o problema, mesmo com a opção de utilização de um Dijkstra.