Trabalho Prático Aeds III

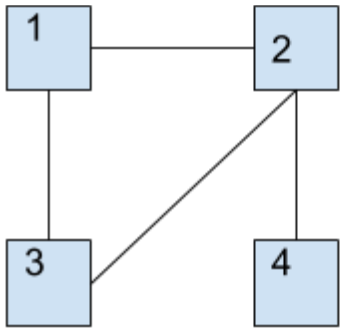
Pedro Nascimento Costa

TP2

Introdução:

A proposta do trabalho envolve desenvolver através de um grafo uma solução para um problema de labirinto, o labirinto consiste em diversos vértices, cada vértice podendo representar um tipo diferente de espaço. A solução é encontrada ao se achar o caminho mais curto do ponto de partida, “Vinícius”, até a saída. Inicialmente temos então três inteiros, dois n e m representando as dimensões n x m da matriz e t, representando o número de chaves que se é possível carregar, a seguir temos apenas a representação do labirinto através de uma matriz contendo alguns símbolos, “#” representa uma parede, “.” um espaço, letras maiúscula “C, D, H , S” portas, letras minúsculas “c, d, h, s” chaves, uma combinação de inteiros, por exemplo “00” um buraco de minhoca, que é praticamente um teleporte, por fim, as letras “V” e “E” representam o ponto inicial e o ponto de saída respectivamente.

Para solucionar o problema, foi utilizada um grafo por lista de adjacência e uma busca em largura(BFS) adaptada para tratar os diferentes tipos de vértice, essa adaptação é onde se concentrou o maior desafio do trabalho, pois cada caso de vértice é tratado da sua pŕopria maneira.



Grafo I

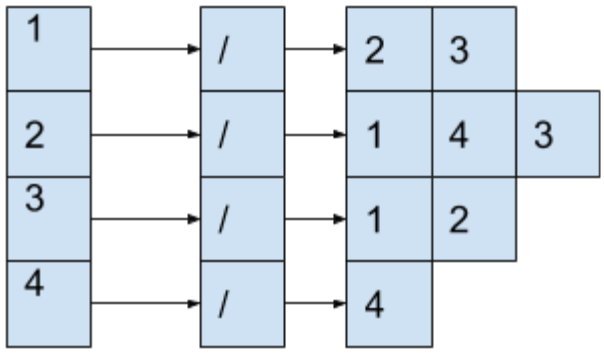


Figura I

O Grafo I e a Figura I representam de maneira simples a apresentação de um grafo não-direcionado através de uma lista de adjacência, que foi a representação de grafo usada na resolução do problema, nela para cada vértice do grafo temos a construção de uma lista contendo todos os vértices adjacentes a um mesmo vértice. No caso da resolução do problema, há um caso especial no qual a saída de um buraco de minhoca é sempre o primeiro vértice adjacente da lista.

Desenvolvimento:

Na implementação do trabalho foram utilizados dois TADs em junção com a main, um TAD é focado na implementação do Grafo através da lista de adjacência e que também faz a leitura de parte da entrada, o outro, na implementação do BFS adaptado. Além disso, foram criadas algumas estruturas de dados para ajudar na implementação, são utilizadas estruturas de fila e lista, relativamente comuns, temos um struct que representa o grafo, que contém as listas de adjacência, o número de arestas e o número de vértices, temos também um struct que representa cada vértice, nele está contida informação como qual o número(valor) daquele vértice, o caracter que ele representa no labirinto, um marcador se ele é buraco e caso seja, as coordenadas da sua saída representada na matriz de entrada, lembrando que o (0, 0) da matriz encontra-se no canto inferior esquerdo, por fim temos um struct que representa uma célula, que contém um apontador para a próxima célula, um vértice e um inteiro, a decisão por utilizar esses struct deve-se basicamente pela sua utilidade na organização das filas e listas.

Para o desenvolvimento, há três módulos, o primeiro monta, o da main, faz a primeira leitura de entrada, o m, n e t e então continua administrando os demais módulos, a seguir o módulo do Grafo realiza a leitura da matriz e faz uma representação dos vértices lidos em uma matriz de entrada, a partir dessa matriz, monta-se então o grafo de forma que não há arestas para fora do mesmo e para vértices paredes, dessa forma todo vértice está conectado aos vértices do lado e, em especial, o buraco tem uma possível aresta extra para sua saída.

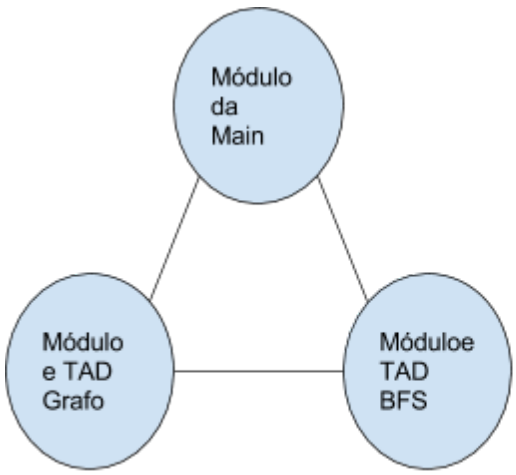


Figura II

A Figura II representa a dependência entre os módulos do programa, note que nenhum dos módulos é independente, por exemplo, o módulo do BFS depende da montagem do grafo e a saída depende da execução do BFS.

A entrada do programa consiste em três inteiros e uma matriz de caracteres, n e m, representam o número de linhas e o número de colunas respectivamentes da matriz que vem a

seguir na entrada, temos também um inteiro t, que representa o limite de chaves que é possível carregar durante aquela execução do programa, note que não é possível largar uma chave após pegá-la ou trocar de chave. A seguir temos então a matriz de caracteres, nela temos 12 possíveis caracteres e algumas combinações de números “00, 01, 31” por exemplo, uma combinação assim de números representa um buraco de minhoca, o primeiro número é a coordenada x e o segundo a y (x,y) da saída do buraco de minhoca. Os demais caracteres podem ser chaves “c, d, h, s”, portas “C, D, H, S”, o ponto inicial “V”, o ponto final “E”, paredes “#”, ou espaços “.”, no caso de uma chave, pode-se pegá-la ou não, numa porta, pode-se apenas passar por ela caso tenha a chave representada pela letra minúscula corresponde e ao chegar na saída, acha-se um possível caminho solução.

A saída é simplesmente um inteiro, que representa o tamanho do caminho mais curto da origem “V” até a saída “E”, caso não exista um caminho, a saída é -1. A saída é determinada por uma lista que contém todas as distâncias de caminhos que alcançaram o vértice “E”, no final, percorre-se a lista atrás do menor valor. Tanto a saída quanto a entrada estão no padrão ‘stdin’ e ‘stdout’.

Para se executar corretamente o programa, primeiro utilize o comando make, assim fazendo uso do makefile para compilar, em seguida, para executar, deve se digitar no terminal apenas: ./exec
em seguida entre com os dados de entrada, de acordo com o especificado acima.

Analisando a Complexidade:

Tempo:

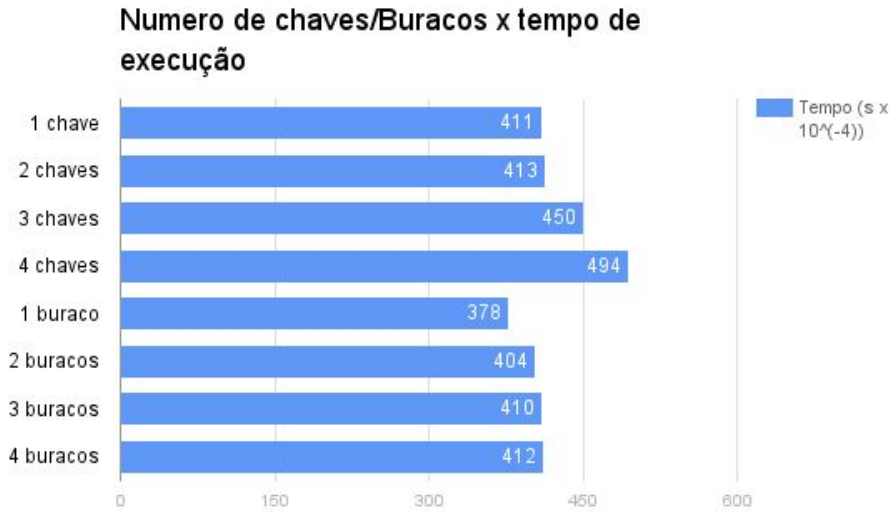
A criação da matriz da entrada é da ordem de O(n x m), a dimensão da matriz, a montagem do grafo é da ordem de O(v + a), sendo v o número de vértices e a o número de arestas, para cada vértice se insere todas as arestas possíveis, por fim a execução do BFS é da ordem de O((v + a)*2^(c + b)), sendo O(v + a) o tempo de execução normal de um BFS e 2^(c + b) o número máximo de execuções do BFS, sendo c o número de chaves e b o número de buracos de minhoca, pois para cada um deles existem duas distintas execuções do BFS.

Como conclusão, a ordem de execução do programa é de O((v + a)*2^(c + b)).

Gŕafico I

No Gráfico I temos a representação do tempo de execução em função da dimensão do labirinto
em segundos * 10^(-4). Nota-se que ao aumentar significativamente o tamanho do labirinto, o que mais ocorre é aumento na variância de possibilidades, que por consequência tende a gerar casos de maior tempo de execução.

Gráfico II



No Gráfico II temos a representação do tempo de execução em função do número de chaves sem variar o número de buracos e do número de buracos sem variar o número de chaves. Note que existe um pequeno aumento de tempo de execução a medida em que se aumenta o número de cada, o que é o esperado.

Espaço:

O espaço do programa deve-se principalmente ao tamanho n x m da matriz de entrada,
ou seja, do número de vértices, e do número de arestas, que determina o tamanho das listas de adjacência, pois os vértices são os itens sendo enfileirados ou listados e o espaço depende principalmente dessa quantidade enfileirada e listada.
O(v+a), sendo v o número de vértices e a o número de arestas.

Conclusão:

Considerando os limites do trabalho, que envolvem no máximo um tamanho 8x8 da matriz de entrada, ou seja no máximo 64 vértices, percebe-se que os tempos de execução e o espaço utilizado não tendem a ser muito grandes, entretanto, vale notar que a complexidade

do tempo é exponencial, logo caso o problema passe a ter cada vez um limite maior, a tendência é ‘explodir’. Isso se deve principalmente a variância nas possibilidades apresentadas por chaves e buracos de minhoca.

Referências:

https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm
http://www2.dcc.ufmg.br/livros/algoritmos/cap3/slides/c/completo1/cap3.pdf
http://www2.dcc.ufmg.br/livros/algoritmos/cap7/slides/c/completo1/cap7.pdf
http://www.inf.ufrgs.br/~tsrodrigues/utilidades/cormem.pdf