Dijkstra

Aluno: Adams Vietro Codignotto da Silva - 6791943

Professor: Alneu

1 Introdução

O algoritmo de Djikstra é utilizado para descobrir o menor caminho entre dois vértices, dado um grafo com arestas cujos pesos são positivos. Este programa executa o algoritmo dado um arquivo de entrada contendo a quantidade de arestas e seus respectivos pesos.

2 Objetivos

Utilizar o mapa rodoviário dado e interpretá-lo como um grafo. Após isso, utilizando o algoritmo de Djikstra, encontrar a menor distancia de uma cidade a outra, no melhor desempenho possível.

3 Implementação

3.1 Estrutura

Foram utilizadas duas estruturas: uma lista de vértices (um para cada cidade), e uma matriz. Ambos são declarados estaticamente, porém, modificando o cabeçalho VERTICES no arquivo header, pode-se alterar o tamanho de ambos sem causar danos ao algoritmo. A lista dos n vértices é composta pela struct grafo, sendo este declarado globalmente e estático:

```
typedef struct
{
    int distancia; //distancia do node ate a origem
    int existe; //se ainda esta no grafo ou nao
    int anterior; //anterior no caminho
} grafo;
```

O grafo é montado a partir de um arquivo de entrada (também declarado no cabeçalho ***INPUT***), o qual deve conter na primeira linha a quantidade de cidades, e depois o nome das cidades (sem espaços) em cada linha.

A matriz é declarada globalmente como ***int MPeso[VERTICES][VERTICES]***, e esta é utilizada para representar o peso entre o vértice *i* e *j*: ou seja, o campo ***MPeso[i][j]*** contem o peso da aresta entre os vértices *i* e *j*. Caso este seja 0, não há uma aresta entre os mesmos. Esta é montada a partir de um arquivo de entrada (declarado no cabeçalho ***INPUT2***), que contem na primeira linha a quantidade de arestas (caminhos) existentes, e após isso a cidade de origem, destino e seu peso em cada linha.

3.2 Algoritmo

O Djikstra utilizado percorre n vértices vizinhos ao vértice de origem, procurando o menor caminho e o removendo (simplesmente setando a flag *existe* como 0, indicando que este é inexistente). Após isso, verifica se o peso deste é menor que o peso previamente encontrado, se a aresta entre o vértice atual e de origem existe, e se é necessário fazer o relaxamento do mesmo. Isso é repetido ate que o vértice de destino seja encontrado.

```
void dijkstra()
{
    int interacao;
    for(interacao=0; interacao<arestas; interacao++) //enquanto existirem
        arestas a serem percorridas
    {
        int i;
        int v = acha_menor(); //v eh a menor aresta vizinha a source
        G[v].existe = 0; //Remove a menor aresta de G
        for (i = 0; i < arestas; i++){
            if (G[i].existe)//vertice deve estar no grafo
                if(MPeso[v][i] != 0)//a aresta entre o menor e o atual deve
                    existir
                    if((G[v].distancia + MPeso[v][i] < G[i].distancia || G[i].
                        distancia == 0))
                    {
                        //caso a distancia do menor + distancia entre o atual
                            e o menor
                        //seja menor que a distancia do no sendo visitado ou
                            caso ela nao tenha sido inicializada
                        /*----RELAXAMENTO DE [V][I]----*/
                        G[i].distancia = G[v].distancia + MPeso[v][i]; //
                            vertice sendo visitado recebe peso do menor +
                            distancia entre menor e sendo visitado
                        G[i].anterior = v; //caminho mais proximo eh salvo
                    }
        }
        if(v==dest) {
                return;
        }
    }
}
```

O resultado é uma lista invertida do caminho de origem ate o destino (que depois é invertida novamente e o caminho é finalmente impresso). A lista inicia-se no vértice de destino e é percorrida de trás para frente, onde ***G[i].anterior*** contém o índice do próximo vértice a ser visitado.

4 Resultados

O algoritmo se mostrou bastante estável, com complexidade $O(n \log(n))$, onde n é a quantidade de vértices. Os caminhos, sempre que existentes, são garantidos de serem encontrados.

A implementação do algoritmo utilizando uma estrutura de grafos em matriz se mostrou relativamente simples, se comparado com uma estrutura de grafos em listas. A estrutura de matriz acaba utilizando mais espaço de memória, porém a velocidade dos acessos são muito mais rápidas.

5 Conclusão

Para grafos pequenos, a estrutura em matriz é mais recomendada, porém, o uso de memória tem ordem quadrática, ou seja, para grafos muito grandes, o recomendado é o uso de listas dinâmicas. O algoritmo de Dijkstra é indicado em ambos os casos, porém ele é mais rápido utilizando matrizes ao invés de listas, apesar de a primeira ocupar mais memória.