Relatório do Trabalho Prático nº 2

Grafos

Hugo Cruz
Nº a23010

Professor: *Luis Ferreira*

Licenciatura Engenharia de Sistemas Informáticos
Estrutura de Dados Avançados

25 de maio de 2024

Conteúdo

1. Introdução

Nesta projeto, iremos explorar a aplicação de conceitos avançados de teoria de grafos, onde utilizamos C para resolver este problema. O problema envolve a criação de um grafo, onde teremos de desenvolver funcionalidades algumas funcionalidades básica relacionas com grafos como, por exemplo: Adicionar vértice remover vértice, entre outros... Outro ponto é a utilização de estruturas de dados dinâmicos, que se trata de uma estrutura que se adapta a qualquer tipo de dados, e não trabalha apenas com 1 ficheiro em específico.

Relativamente a criação do meu grafo, que se trata de um grafo orientado pesado, eu utilizei listas de adjacências, uma das estruturas lecionadas ao longo desta UC. Esta estrutura de dados permite o acesso ao próximo valor da lista com bastante facilidade. Para realizar todas estas funcionalidades de forma autónoma, implementamos funcionalidades relacionadas com ficheiros. Onde encontramos funcionalidades como carregar grafo através de um ficheiro de texto, guardar o grafo em ficheiro não manipuláveis, e carregar grafos através dos mesmos, de forma a torna o programa mais versátil.

Por fim foi nos sugeridos a implementação de algoritmos para encontrar o caminho com menor peso, verificar se existe caminho entre dois vértices, calcular a soma total de uma origem a um destino, entre outros.

2. Problema

Fase 2 – Grafos

Nesta segunda fase, pretende-se aplicar conceitos avançados de teoria dos grafos e programação em C para resolver um problema computacional com grau de complexidade maior, relacionando estruturas de dados, algoritmos de procura e técnicas de otimização. O objetivo é desenvolver uma solução capaz de calcular o somatório máximo possível de inteiros a partir de uma matriz de inteiros de dimensões arbitrárias, considerando regras específicas de conexão entre os inteiros.

Assim, procure implementar as funcionalidades seguintes:

1. Definir uma estrutura de dados GR para representar um grafo. Esta estrutura deve ser capaz de representar grafos dirigidos e deve suportar um número variável de vértices. A implementação deve incluir funções básicas para criação do grafo, adição e remoção de vértices e arestas;

2. Após definir a estrutura de dados GR, pretende-se modelar o problema utilizando grafos. Cada elemento da matriz de inteiros será representado por um vértice no grafo. As arestas entre vértices devem representar a possibilidade de somar dois elementos adjacentes na matriz, sob uma regra de conexão específica que poderá ser configurada pelo utilizador (por exemplo, apenas elementos na mesma linha ou coluna, não permitindo diagonais, ou qualquer outra regra);

3. Carregamento para a estrutura de dados da alínea anterior GR dos dados de uma matriz de inteiros constante num ficheiro de texto. A operação deverá considerar matrizes de inteiros com qualquer dimensão, sendo os valores separados por vírgulas. A título de exemplo, o ficheiro de texto deverá respeitar o formato seguinte:

4. Implementar operações de manipulação de grafos, incluindo procura em profundidade ou em largura, para identificar todos os caminhos possíveis que atendem às regras de conexão definidas. Desenvolver também uma função para calcular a soma dos valores dos vértices num dado caminho;

5. Utilizar as estruturas e algoritmos desenvolvidos para encontrar o caminho que proporciona a maior soma possível dos inteiros na estrutura GR, seguindo a regra de conexão estabelecida. O programa deve fornecer tanto a soma máxima quanto o caminho (ou caminhos, se existirem múltiplos caminhos com a mesma soma máxima) que resulta nessa soma;

3. Trabalho Desenvolvido

No desenvolvimento projeto, como já foi referido anteriormente, utilizei listas de adjacência. As listas de adjacência, trata-se de uma estrutura de dados dinâmica, sendo uma opção valida para desenvolver este projeto, optei pela mesma.

Estas foi as estruturas de dados utilizadas para os Vértices:

```
typedef struct Vertice
{
    int id;
    struct Vertice *nextV;
    Adjacente *nextA;

} Vertice;
```

Figura 1

Esta é composta por apenas um campo que é o id que identifica cada vértice. Visto que se trata de uma lista existe um apontador para si mesmo, e um apontador para a lista de adjacências.

Como podemos ver na figura 2 cada vertices irá ter a sua lista de adjacecias que irá permitir a manipulcao de dados. Em contexto de desenvolvimeto refere-se:

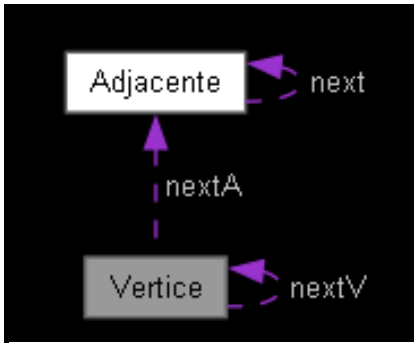

Figura 3

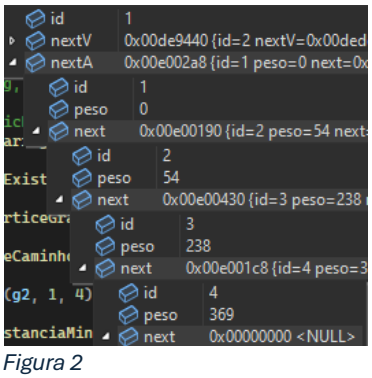

Figura 2

Em termos de adjacências:

```c
typedef struct Adjacente
{
    int id;
    int peso;
    struct Adjacente *next;

} Adjacente;
```

Figura 4

Figura 5

Esta estrutura representa uma adjacência, cada adjacência é composta por dois campos, sendo esse o id de destino, e o peso que corresponde a distância entre o vértice origem e o destino. A mesma contém um apontador para si mesma de forma a criar uma lista.

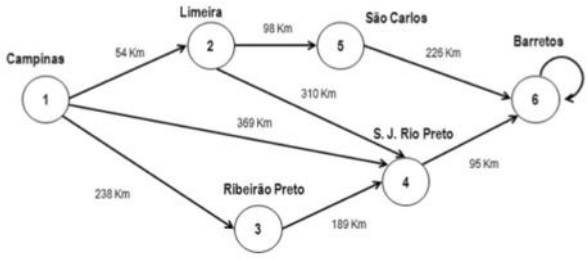

Figura 6

Visto que já contemos tudo o que é necessário para criar um grafo, apenas adaptamos a nossa estrutura Grafo onde a mesma aponta para o início de uma lista de adjacência de vértices

```c
typedef struct Grafo
{
    Vertice *inicioGrafo;

} Grafo;
```

Figura 7

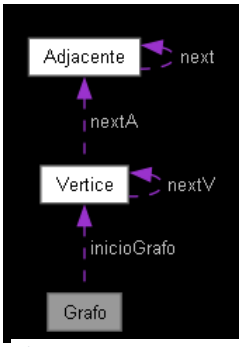

Figura 8

Ao analisar a estrutura Grafo, podemos ver que ela é eficiente para representar relações complexas entre elementos, permitindo a manipulação eficiente de dados. Após definir a nossa estrutura inicializamos a criação de funcionalidade correspondentes aos grafos.

4. Carregamento de dados

O carregamento de dados é uma etapa crucial para o bom funcionamento deste projeto. Neste caso, os dados são carregados a partir de um ficheiro de texto. A abordagem utilizada para carregar os dados é a seguinte:

Cada linha ou coluna do ficheiro de texto representa um vértice num grafo. Para garantir a criação do número correto de vértices, é sempre criado o maior número entre as linhas e as colunas. Isto garante que todos os vértices necessários são criados, independentemente da estrutura do ficheiro de texto.
Os dados entre os pontos e vírgulas representam o peso de cada adjacência. Por exemplo, se tivermos a linha "7;53;183;439;863", isto significa que temos adjacências do vértice 1 para os vértices 2, 3, 4, 5 e 6 com pesos de 7, 53, 183, 439 e 863, respetivamente.

Assim, a partir de um ficheiro de texto simples, é possível construir um grafo complexo com vértices e arestas. Esta abordagem facilita aa criação de grafos ajustado pois é mais simples remover linhas colunas de um ficheiro de texto.

5. Problemas que não consegui resolver

Durante o desenvolvimento deste projeto, deparei-me com alguns desafios que não consegui superar completamente. Embora tenha conseguido implementar o algoritmo de Dijkstra, que é fundamental para a resolução de muitos problemas em teoria dos grafos, onde me abriu portas para realizar funcionalidades como verificar se existe caminho entre dois vértice calcular o peso mínimo entre dois vértices e criar o grafo de caminho mínimo, não consegui implementar Breadth-First Search – BFS e o Depth-First Search – DFS.

6. Conclusão

Neste projeto, exploramos conceitos avançados de teoria de grafos e aplicamos em programação em C para resolver um problema complexo. A estrutura de dados que representa o meu grafo foi fundamenta para modelar a matriz de inteiros e definir as regras de adjacências entre os vértices.

O algoritmo de Dijkstra desempenho um papel crucial no meu trabalho. Através da mesma consegui elaborar maio o meu trabalho e implementar funcionalidade interessantes relacionados com caminhos.

Além de todas estas novidades para mim, ainda consegui complementar melhor os meus conhecimentos relativamente a manipulação de ficheiros.

Embora tenha enfrentado algumas dificuldades na compreensão e no desenvolvimento deste trabalho, sinto que aprendi bastante sobre o tema, e tenho como objetivo de melhorar e implementar os caminhos que ficaram em falta