Estruturas de Dados Avançadas

David Amorim Carvalho

Nº 27973 – Regime Diurno

Ano letivo 2023/2024

Licenciatura em Engenharia de Sistemas Informáticos

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Identificação do Aluno

David Amorim Carvalho

Aluno número 27973, regime diurno

Licenciatura em Engenharia de Sistemas Informáticos

RESUMO

Neste trabalho realizado no âmbito da cadeira de Estrutura de dados avançados estará presente a aplicação dos conceitos sobre listas ligadas e grafos lecionados pelo professor Luís Ferreira.

ÍNDICE

1. Introdução 2

1.1. Objetivos 2

2. Desenvolvimento 3

2.1. Alínea 1: 3

2.2. Alínea 2: 4

2.3. Alínea 3 5

2.4. Alínea 3 VII

3. Conclusão X

ÍNDICE DE FIGURAS

Figura 1 - Struct Grafo 3
Figura 2- Struct Vertices 3
Figura 3 - Struct Adjacentes 3
Figura 4 - Desenho do grafo 4
Figura 5- Estrutura do ficheiro 4
Figura 6- Procura em profundidade 6
Figura 7 - Procura em largura.....6
Figura 8 - Estrutura vértice distâncias..... VII

1. Introdução

Este trabalho foi realizado no âmbito da cadeira de Estrutura de dados avançados na qual tem como objetivo a aplicação prática dos conceitos lecionados na disciplina de Estrutura de dados avançados pelo professor Luís Ferreira.

1.1. Objetivos

- Definir uma estrutura de dados para representar um grafo;
- Carregamento para a estrutura de dados a partir um ficheiro.txt e ficheiro binário;
- Guardar a estrutura de dados para um ficheiro binário;
- Implementar algoritmo de procura em profundidade;
- Implementar algoritmo de procura em largura;
- Determinar e calcular o caminho máximo.

2. Desenvolvimento

2.1. Alínea 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.

```
typedef struct Grafo {
    struct Vertice* vertInicio;
    int vertices;
}Grafo;
```

Figura 1 - Struct Grafo

```
typedef struct Vertice {
    int id;
    bool visitado;
    struct Vertice* proxVertice;
    struct Adjacente* proxAdjacente;
}Vertice;
```

Figura 2- Struct Vertices

```
typedef struct Adjacente {
    int id;
    int peso;
    struct Adjacente* proxAdjacente;
}Adjacente;
```

Figura 3 - Struct Adjacentes

Para definir a estrutura do grafo foram criadas essas três estruturas de dados que juntas constituem o **grafo**.

O grafo possui uma **lista de vértices** que representa todos os vértices presentes no grafo, e a lista de vértices possui uma **lista de adjacentes** que representa as conexões existentes a partir de cada vértice.

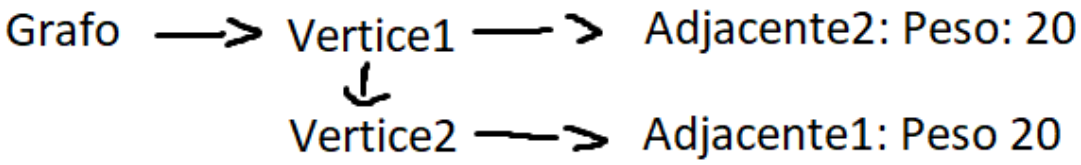


Figura 4 - Desenho do grafo

2.2. Alínea 2:

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.

```
0;6;1;0;0
6;0;2;2;6
1;2;0;1;0
0;2;1;0;5
0;5;0;5;0
```

Figura 5- Estrutura do ficheiro

Para fazer o carregamento do ficheiro txt e do ficheiro binário, e também para guardarem binário comecei por criar uma lista de inteiros (estrutura de dados).

```
typedef struct ListaInteiros {
    int num;
    struct ListaInteiros* proxInteiro;
}ListaInteiros;
```

Arranque do programa:

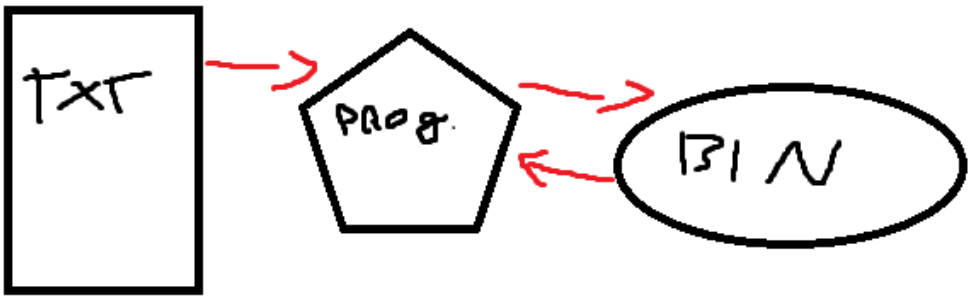
Para o programa arrancar foi criado uma função que verifica se existe um ficheiro binário, se o ficheiro binário não existir é carregado o grafo a partir de um ficheiro txt e guardado num ficheiro binário, para que a próxima vez que o programa foi aberto ser carregado a partir de um ficheiro binário.

Carregamento txt:

Para fazer o carregamento do ficheiro txt foi lido o ficheiro utilizado o **getc** que lê o ficheiro caracter a caracter para dentro de da lista de inteiros, tendo todos os dados do ficheiro na lista de inteiros é carregado a lista de inteiros para o grafo.

Carregamento e guardar binário:

Para guardar o grafo em binário, primeiro é carregado a informação de um grafo numa lista de inteiros e após isso é guardado a informação da lista de inteiros, quando o programa precisar de carregar do binário, carrega para a lista de inteiros e só depois é carregado para o grafo.



2.3. Alínea 3

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.

Procura em profundidade:

Para criar o algoritmo de procura em profundidade (DFT), começo por a partir de um vértice (vértice início) visitar cada adjacente não visitado, ao visitar o adjacente marco o mesmo como visitado e coloco-o na **stack** (stack foi criada com a estrutura lista de inteiros), se chegar num vértice sem adjacentes retiro o vértice da stack e repito estes passos ate a stack ficar vazia.

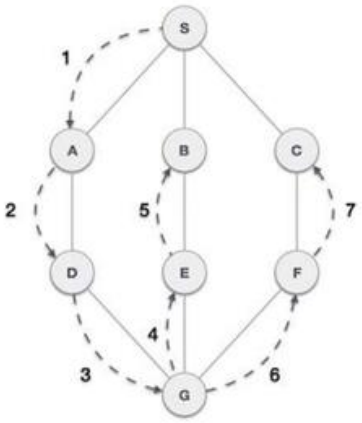


Figura 6- Procura em profundidade

Procura em largura:

Para criar o algoritmo de procura em largura (BFT) começo por a partir de um vértice (vértice início) visitar cada adjacente não visitado, ao visitar o adjacente marco o mesmo como visitado e coloco-o na **queue** (queue foi criada com a estrutura lista de inteiros), se chegar num vértice sem adjacentes retiro o primeiro vértice da queue e repito estes passos ate a queue ficar vazia.

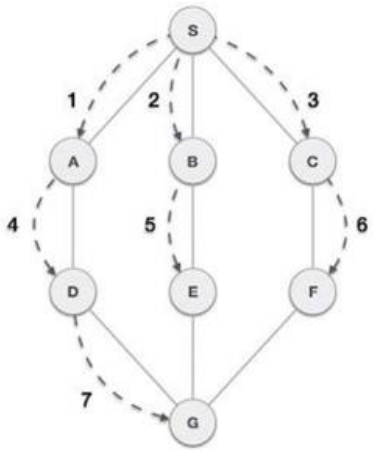


Figura 7 - Procura em largura

2.4. Alínea 3

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;

Para determinar a soma e caminho máximo optei por utilizar o algoritmo **Dijkstra** que é normalmente utilizado para determinar o caminho mais curto, logo implementei o algoritmo e modifiquei o para determinar o caminho mais longo.

Para implementar o algoritmo comecei por criar uma estrutura nova chamada **VerticeDistancias** que contem o id do vértice, um inteiro correspondente á distancia do vértice início até ao vértice em si, e um apontador para o vértice lista anterior.

```
typedef struct VerticeDistancia {
    int id;
    int distancia;
    struct VerticeDistancia* proxVerDist;
    struct VerticeDistancia* antVerDist;
}VerticeDistancia;
```

Figura 8 - Estrutura vértice distâncias

Com a estrutura implementado começa por inicializar a lista de distâncias com os ids dos vértices e com as distâncias todas a -1, exceto a do vértice início que fica com a distância a 0. E também é criada uma lista de inteiros para colocar todos os vértices por visitar que no início contem todos os vértices menos o vértice do início.

Com todos esses passos feitos começo a implementar o algoritmo: No algoritmo é começado por visitar o vértice adjacente com menor distancia do vértice inicial, no vértice é examinado as suas adjacências e em cada adjacência é calculado a distancia do inicio a essas adjacências, e se a distancia calculada for maior á distancia presente na lista de distancias, é guardada a distancia na lista de distancias e por fim é tirado o vértice da lista de vértices por visitar, repetindo todos os passos até não haver vértices por visitar.

Bibliografia

- Material fornecido nas aulas.

3. Conclusão

Este relatório detalhou a aplicação de 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 trabalho permitiu uma aplicação prática dos conhecimentos adquiridos na unidade curricular.

Os objetivos do trabalho foram parcialmente alcançados ficando a alínea 4 (calcular caminho máximo) não totalmente funcional, devido ao utilizar o algoritmo Dijkstra, sendo o algoritmo feito para calcular caminhos curtos e não caminhos longos, mesmo após modifica-lo nem sempre determina o caminho mais longo. Uma forma de melhorar o trabalho nesse aspeto seria implementar uma função que analise todos os caminhos possíveis e determine entre todas as distâncias calculadas desses caminhos o maior caminho.