Teoria dos Grafos – COS 242

Bruno Dantas de Paiva, Eduardo Guedes de Seixas

Escola Politécnica – Universidade Federal do Rio de Janeiro (UFRJ)

Rio de Janeiro – RJ – Brasil

Introdução

A ideia principal do trabalho é desenvolver uma biblioteca para manipular grafos, que seja capaz de agregar também algoritmos essenciais para o trabalho.

Implementação

A implementação ocorreu na linguagem `C++`, onde optamos por um conjunto de funções. Contudo, foi necessária a adaptação (variáveis sublinhadas) das `structs` previamente implementadas para auxiliar a utilização dos dados que representavam os grafos nos algoritmos implementados. Elas são:

```
grafoVector
int numVertices;
vector<int>* adjVector;
bool bipartido;
vector<int> bipartite_1;
vector<int> bipartite_2;
```

```
grafoMatriz
int numVertices;
bool **adjMatriz;
bool bipartido;
vector<int> bipartite_1;
vector<int> bipartite_2;
```

```
grafoVectorComPeso
int numVertices;
double**adjMatriz;
bool directed;
```

```
grafoMatrizComPeso
int numVertices;
vector<pair<int,double>>*adjVector;
bool bipartido;
```

Imagem 1: Estruturas adaptadas para facilitar a integração do grafo com o código.

a- Principais observações sobre o projeto:

1- Para a detecção do emparelhamento máximo, foi implementado o algoritmo de Hopcroft-Karp, cuja complexidade é $O(\sqrt{n} * m)$. Contudo, foram notados erros de implementação, gerando um tempo de execução muito alto para os grafos propostos.

2- Durante a execução da biblioteca, foi utilizada a flag `-O3` para otimização do código, pois, por testes de caso, pode-se observar que esta gerou um desempenho melhor no tempo de execução, diminuindo-o.

b- Estudo de Caso

Analisando todos os arquivos de grafos de teste, obtivemos os seguintes resultados:

Grafo	Bipartível(s/n)	Emp. Max	Tempo(s)
1	S	500	0.000569
2	S	500	0.001106
3	S	5000	0.02265
4	S	5000	0.032485
5	S	50000	4.74875
6	S	50000	2.2463
7	N	0	0.000009
8	N	0	0.000009
9	N	0	0.000009
10	S	500000	512.736

Tabela 1: Estudos de casos para os grafos requisitados. Parte 1.

Grafo	Ciclo Negativo	Distancia(1,10)	Distancia(2,20)	Distancia(3,30)	Tempo (s)
ER_50	S	-Inf	-Inf	-Inf	0.00485
ER_100	S	-Inf	-Inf	-Inf	0.017602
ER_500	S	-Inf	-Inf	-Inf	3.98899
ER_1000	S	-Inf	-Inf	-Inf	51.7165
ER_1500	S	-Inf	-Inf	-Inf	238.465

Tabela 2: Estudos de casos para os grafos requisitados. Parte 2.

C – Observações:

1- A implementação para a detecção do grafo bipartido foi executada de maneira incorreta, pois o algoritmo só executava uma bfs para um único vértice. Contudo, em um dos casos teste (grafo_teste_8.txt), possuia mais de 1000 componentes conexas, o que não impediria ele ser bipartível ou não.

2- Além disso, foi observado que a distância de quase todos os vértices dos grafos para a execução do algoritmo de Bellman-Ford passavam por ciclos negativos, o que fazia com que a distância entre eles fosse muito baixa (Para fins didáticos foi considerado o valor de -inf). Contudo, haviam vértices onde o grau de saída deles era 0, ou seja, o caminho deles até os outros vértices do grafo possuia uma distância infinita.

3- Durante a implementação do algoritmo de Bellman-Ford, foram implementadas as otimizações propostas em aula, de modo que o algoritmo parasse no exato momento que nenhum outro vértice fosse atualizado, por meio de uma flag, e, para a detecção de ciclos, bastou executar o algoritmo para mais um vértice e checar se a flag previamente citada foi atualizada, caso sim, haveria ciclo negativo.

4- Também, foi requisitado que a função de Bellmanford retornasse uma matriz[i,j] de distâncias, onde, representasse a distância de i até o vértice j. Para isto, foram implementadas duas funções, de modo que a real implementação do algoritmo de Bellman-Ford fosse utilizada como núcleo de uma função secundária, onde nesta foi realizada a inserção dos vetores distâncias em uma matriz, para, caso o usuário quisesse, salvar a matriz em um arquivo .txt por meio de uma variável booleana salve.

5- Para a definição dos tempos do algoritmo de Bellman-Ford, foi realizada a impressão da distância de todos os vértices para todos os outros vértices, ou seja, a impressão de toda a matriz distância , deste modo, foi possível obter um tempo mais fidedigno do proposto pelo trabalho.

D- Repositório:

Todo o trabalho e implementações podem ser encontrados no repositório abaixo:
https://github.com/DantasB/Graph-Library