Teoria dos Grafos
Prof.: Daniel Ratton Figueiredo
Alunos: Eduardo Guedes de Seixas e Bruno Dantas Paiva.

Trabalho de Disciplina – Parte 1

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 criação de duas `structs` para auxiliar a utilização dos dados que representavam os grafos. Elas são:

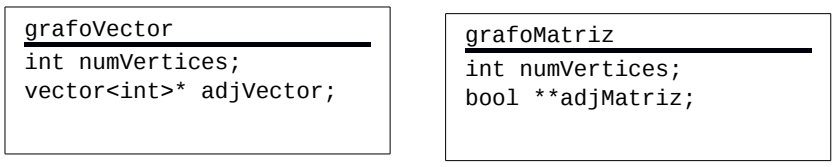

Imagem 1: Estruturas criadas para facilitar a integração do grafo com o código.

a- Principais observações sobre o projeto:

1- Foi utilizado o container `vector` em vez da utilização de `arrays` no `grafoVector` por conta da alocação dinâmica de memória e da facilidade de realocação. Contudo, não foi utilizado no `grafoMatriz`, o que pode ser considerado uma falha de projeto, pois poderia ter reduzido o gasto de memória na Matriz de adjacência.

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:

Atributos			Grafos					
			as_graph.txt		dblp.txt		live_journal.txt	
			Lista	Matriz	Lista	Matriz	Lista	Matriz
Utilização de Memória (Mb)			2.2	1049.7	125.2	-	579.1	-
Tempo 1000 BFS (s)			0.79	1513.68	197.13	-	924.85	-
Tempo 1000 DFS (s)			1.02	1863.13	287.78	-	2036	-
Árvore								
Busca	Vértice	Pai						
BFS	1	10	1		1226753		2	
		20	1		843078		2	
		30	1		80251		2	
	2	10	2		1226753		2	
		20	5		843078		2	
		30	21		367956		2	
	3	10	3		-		2	
		20	1		-		2	
		30	1		-		2	
DFS	1	10	1		1226753		3942876	
		20	1		843078		231067	
		30	1		1395366		76383	
	2	10	2		1226753		2	
		20	1		843078		2	
		30	1		1395366		2	
	3	10	3		-		2	
		20	67		-		2	
		30	11274		-		2	
Vértices			Distância					
10 e 20			1		7		2	
10 e 30			2		5		2	
20 e 30			2		7		2	

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

Atributos	Grafos					
	as_graph.txt		dblp.txt		live_journal.txt	
	Lista	Matriz	Lista	Matriz	Lista	Matriz
Quantidade de Componentes conexas	1		116442		1	
Tamanho da maior componente conexa	32385		1183247		3997962	
Tamanho da menor componente conexa	32385		1		3997962	
Diâmetro	11		-		-	
Tempo do cálculo do Diâmetro (s)	20.59		-		-	

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

C – Observações:

1- Nota-se que a DFS possui um tempo duas vezes maior que a BFS. Contudo, teoricamente, o tempo gasto deveria ser igual, pois a complexidade de ambas é `O(n+m).` Tal fato pode ser observado pois a DFS não teve o mesmo tratamento que a BFS, onde tratamento está relacionado a otimização do código. Pode-se observar que a busca em profundidade não teve um cuidado tão grande e poderia ter sido bem melhor otimizada, porém, em virtude do tempo, tal otimização não foi executada.

2- Além disso, o Diâmetro não foi possível de ser calculado por conta da complexidade alta `O(n³)` que, para grafos com um alto número de vértices `(dblp e live_journal)` demorariam mais de 160 horas para ter os resultados explicitados. Com isso, poderia ter sido implementado o código do `pseudo-diâmetro`, porém, pela mesma razão da não otimização da DFS (tempo) esta não foi implementada.

3- Observa-se que a implementação das Componentes Conexas não foi a maneira ótima `O(n+m)`, pois o grupo encontrou dificuldades na utilização do container `list` da `standard template library`, com isso, a função foi implementada de maneira ingenua, o que acarretou numa demora na execução do código.

D- Repositório:

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