Teoria dos Grafos - COS242

Prof. Daniel Ratton e Fábio Botler

Trabalho - Parte 1

Aluno: Patrick Carneiro Trindade
DRE: 119052764

1 Introdução

1.1 O programa criado

Para este trabalho foi criado um conjunto de funções e estruturas (struct) em Go - uma linguagem tipada e compilável criada pela Google - a fim de explorar grafos (gerar árvores, encontrar diâmetros, distâncias etc).

O código pode ser encontrado no github, no link abaixo:
https://github.com/ptk-trindade/graph-project

1.1.1 Estrutura do código

O código possui 2 arquivos principais, com as funções que realizam as buscas e alguns arquivos secundários que servem apenas para chamar as funções e fazer a manipulação de arquivos.

Arquivo	Descrição
graphList	Funções que leem o grafo em forma de lista
graphMatrix	Funções que leem o grafo em forma de matriz
main	A função main deste arquivo permite que o usuário rode e interaja com o programa
tests	Uma função criada durante a para o tópico 3 deste relatório (Estudos de caso)

1.2 A máquina utilizada

Para rodar este programa foi utilizada uma máquina Windows, com um processador Intel i7 6ª geração e 24GB de memória RAM.

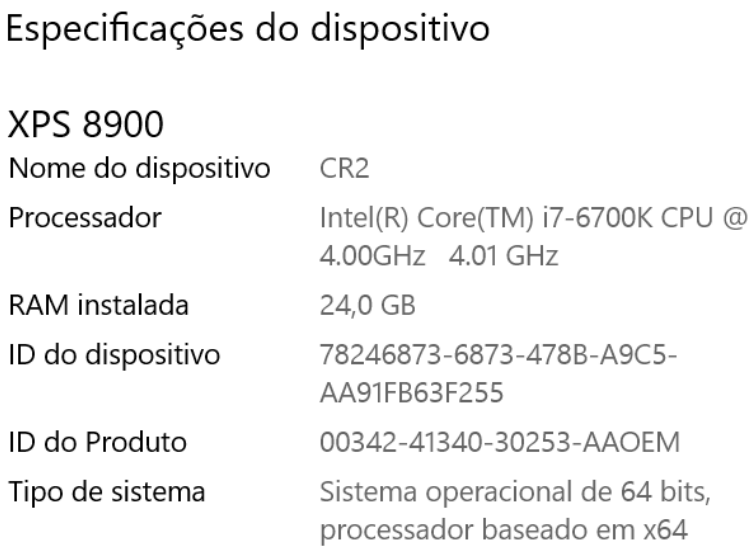


imagem 1: especificações da máquina utilizada

1.3 Os grafos analisados

Para este relatório, foram analisados 5 grafos, que possuem as características descritas abaixo:

nome	vértices	arestas	grau			mediana	compo-nentes
			máximo	mínimo	médio		
grafo_2	968	14204	59	2	29,32	31	5
grafo_3	9989	343126	149	1	68,69	74	10
grafo_4	100054	3433247	148	1	68,63	72	20
grafo_5	1000022	17666652	87	1	35,33	37	20
grafo_6	10000036	26666787	23	1	5,33	5	25

Essas e algumas buscas pelo grafo podem ser encontradas na pasta “output_log” do repositório. Mais informações sobre os arquivos de saída na seção a seguir.

2 Input e Output (I/O)

Para input do grafo, o usuário deve criar um arquivo txt. O nome do arquivo será perguntado assim que o programa começa sua execução, o arquivo porém deverá ter o seguinte formato:

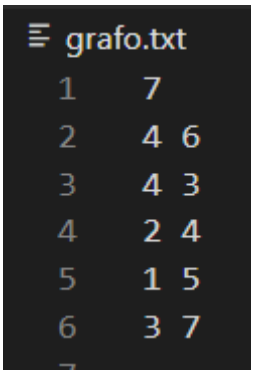


imagem 2: arquivo de input “grafo.txt”

Neste, a primeira linha representa o número de vértices do grafo, e cada linha seguinte representa uma aresta (a ordem dos vértices em cada linha, assim como a ordem das linhas de arestas não geram diferença).

O programa pode gerar então os seguintes arquivos de output:

I/O	nome	descrição
input	grafo.txt*	Número de vértices e arestas do grafo
output	output.txt	Informações básicas sobre o grafo
output	path.txt	Os caminhos entre os vértices escolhidos pelo usuário
output	tree.txt	A BFS ou DFS gerada a partir do vértice fornecido
output	diameter	O diâmetro do grafo

*Não precisa ser necessariamente esse nome, mas é recomendado não utilizar o nome de um dos arquivos de output.

3 Estudos de caso

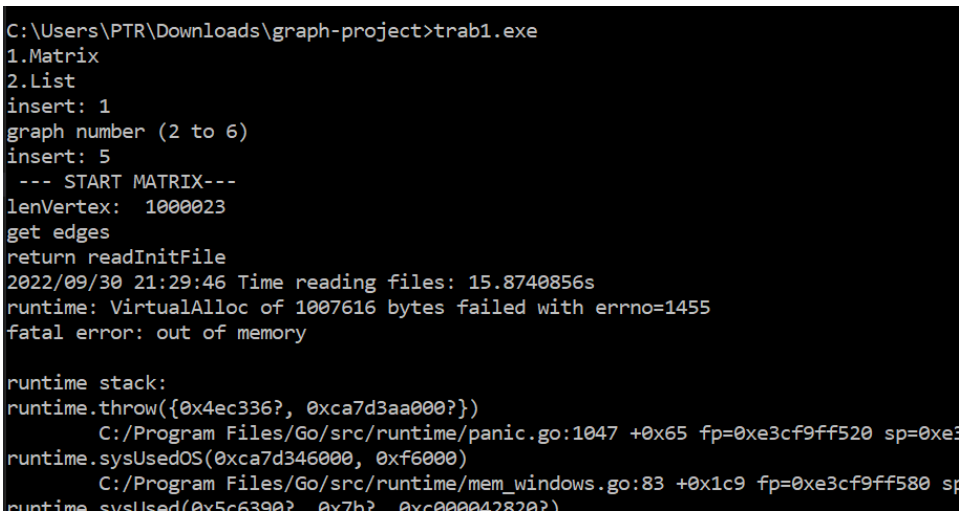
Foram feitas diversas comparações com o programa rodando com lista e matriz de adjacência, conforme pode ser visto abaixo. (Obs: Nos cálculos de complexidade **n** é o número de vértices e **m** o número de arestas).

3.1 Quantidade de memória utilizada

De forma geral, a lista de adjacência é mais eficiente quando temos grafos mais esparsos, enquanto a matriz de adjacência é indicada para grafos mais completos. Isso porque a complexidade de memória da lista é de O(n+m), enquanto na matriz ela é de ordem $O(n^2)$

	lista de adjacência	matriz de adjacência
grafo_2	5.5 MB	5.6 MB
grafo_3	32.0 MB	107.3 MB
grafo_4	352.2 MB	8555.8 MB
grafo_5	1422.2 MB	-*
grafo_6	2488.1 MB	-*

*Não foi possível gerar a matriz para os grafos 5 e 6, no meio da execução o programa exibia um erro de falta de memória e parava a sua execução.



erro de falta de memória

3.2 e 3.3 Tempo de execução BFS e DFS

Para cada grafo, foram escolhidos 1000 vértices aleatórios, então rodou-se primeiro a BFS e depois a DFS para cada um destes vértices, calculado o tempo total de execução das buscas.

	lista de adjacência		matriz de adjacência	
	BFS	**DFS**	**BFS**	**DFS**
grafo_2	38.98ms	118.02ms	448.41ms	473.62ms
grafo_3	657.62ms	3.43s	32.25s	26.79s
grafo_4	10.59s	30.78s	2921.74s	2326.479s
grafo_5	112.33s	219.19s	-*	-*

grafo_6	938.71s	1205.98s	-*	-*

*Não foi possível rodar o programa para estes grafos

3.4 Pai dos vértices

Encontramos também os pais dos vértices 10, 20 e 30 tanto para a busca em largura (BFS) como para a busca em profundidade (DFS). Para cada um, executamos a busca começando nos vértices 1, 2 e 3. (O hífen '-' significa que o vértice não pertence a mesma componente em que a busca foi realizada).

BFS

pais de:	**pais de 10**			**pais de 20**			**pais de 30**		
começando no vértice:	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
grafo_2	775	47	513	-	-	-	-	-	-
grafo_3	-	-	6984	-	-	-	-	-	7250
grafo_4	-	-	-	-	-	-	-	-	73580
grafo_5	433213	496560	1480	630893	836334	694539	-	-	-
grafo_6	-	-	-	9914409	1571310	152149	-	-	-

DFS

pais de:	**pais de 10**			**pais de 20**			**pais de 30**		
começando em:	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
grafo_2	895	895	895	-	-	-	-	-	-
grafo_3	-	-	8072	-	-	-	-	-	8948
grafo_4	-	-	-	-	-	-	-	-	63701
grafo_5	252178	252178	252178	352162	836334	352162	-	-	-
grafo_6	-	-	-	9990331	9990331	9990331	-	-	-

3.5 Distância entre vértices

Obs: O hífen ‘-’ indica que os vértices não estão na mesma componente.

distância entre:	10 e 20	10 e 30	20 e 30
grafo_2	-	-	2
grafo_3	-	1	-
grafo_4	-	-	-
grafo_5	4	-	-
grafo_6	-	-	-

3.6 Componentes conexas

componentes	qtd de componentes	maior	menor
grafo_2	5	500	31
grafo_3	10	5000	10
grafo_4	20	50000	10
grafo_5	20	500000	10
grafo_6	25	5000000	10

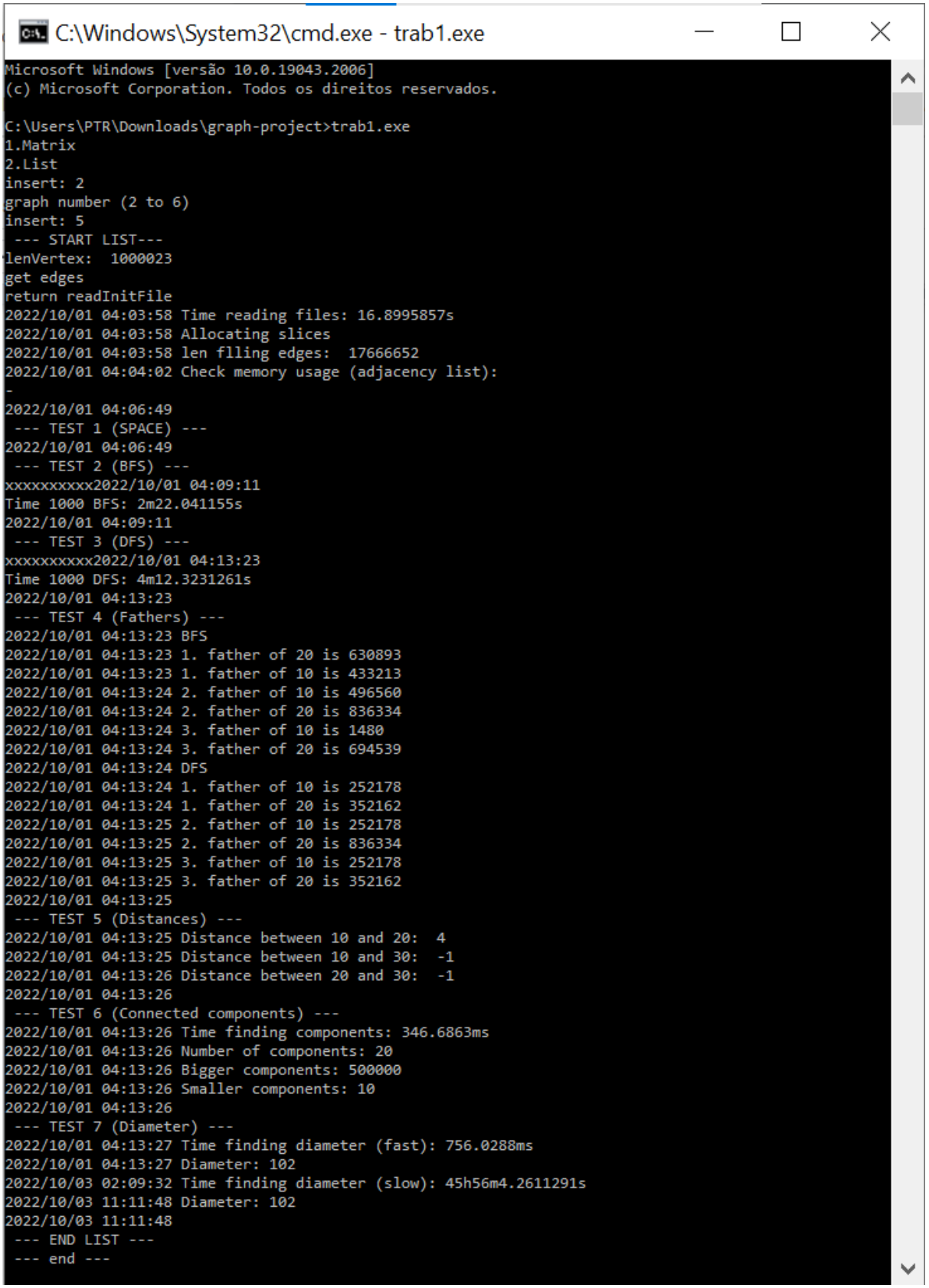
3.7 Diâmetro do grafo

Para calcular o diâmetro exato do grafo, o programa roda uma BFS para cada um dos nós, o que possui uma complexidade de tempo de $O(n^2+m^2)$.

Foi criada também uma função para calcular o diâmetro do grafo “rapidamente” - em complexidade de tempo de $O(n+m)$. Para cada componente do grafo, o algoritmo roda a BFS em um vértice qualquer e depois roda uma BFS do vértice mais distante deste vértice qualquer. Essa solução é ótima para grafos sem ciclos (árvores), mas para outros grafos a distância encontrada pode não ser a maior do grafo.

	real		aproximado	
	diâmetro	**tempo de execução**	**diâmetro**	**tempo de execução**
grafo_2	5	41.15ms	4	< 1ms
grafo_3	9	7.02s	9	3.99ms
grafo_4	31	1068.81s	31	57.059ms
grafo_5	102	165364.26s	102	587.39ms
grafo_6	-	-	999	5.42s

Para o último grafo, por conta do tempo que levaria, não era viável rodar o algoritmo para encontrar o diâmetro real do grafo.



rodando testes para um dos grafos