Análise e síntese de algoritmos

Hugo Guerreiro 83475
João Sousa 83487

1 Introdução

Foi-nos apresentado um problema que consistia em desenvolver um software para ajudar o Sr.João a organizar fotos para o seu álbum de fotografias. Contudo, ele só consegue comparar duas fotos de cada vez e dizer qual delas foi tirada primeiro.

Deste modo, o software tem por objetivo organizar as fotos temporalmente caso possível, senão indica se a informação dada é insuficiente ou incoerente para gerar uma ordem temporal.

2 Solução

2.1 Descrição da solução

Dada a natureza do problema, é conveniente solucioná-lo utilizado grafos como representação, onde cada fotografia é um vértice e cada ligação realizada pelo Sr.João uma aresta.

Dada a representação, a organização temporal pretendida é obtida através de uma ordenação topológica do grafo. Caso possua ciclos então a informação é incoerente pois não é possível gerar uma ordenação topológica em grafos não acíclicos. Caso o grafo possua mais que uma ordenação topológica é porque a informação é insuficiente.

2.1.1 Unicidade

"Se uma ordenação topológica tem a propriedade de todos os pares de vértices consecutivos na ordem de classificação serem conectados por arestas, essas arestas formam um caminho Hamiltoniano dirigido no DAG. Se existe um caminho Hamiltoniano, a ordenação topológica é única(...)"[2].

2.2 Passos da solução

1ºpasso: Construir o grafo a partir do input recebido.
2º Passo: Aplicar uma pesquisa em profundidade ao grafo introduzido
3º Passo: Caso o grafo contenha ciclos então o software devolve "Incoerente", caso

contrário verificamos se a ordenação obtida é única.
4ºpasso: Se a ordenação for única, o output será a ordenação obtida, caso contrário devolvemos "Insuficiente".

2.2.1 Pseudocódigo

```
1: L ← Empty list that will contain the sorted nodes
2: getInputAndBuildGraph()
3: while there are unmarked nodes do
4:    select an unmarked node n
5:    visit(n)
6: end while
7: verifyUnicity(L)
8:
9: function VISIT(node n)
10:     if n has a temporary mark then stop
11:     end if
12:     if n has not been visited yet then
13:         mark n temporarily
14:         for each node m with an edge from n to m do
15:             visit(m)
16:         end for
17:         mark n permanently
18:         unmark n temporarily
19:         add n to head of L
20:     end if
21: end function
22:
23: function VERIFYUNICITY(List L)
24:     for each two consecutive nodes in L, n and m do
25:         if n contains a conection to m then continue
26:         else  return false(it has more than one possible topological order)
27:         end if
28:     end for
29:      return true(the topological order is unique)
30: end function
```

3 Análise teórica

3.1 Complexidade

Tal como descrito anteriormente, a nossa solução está dividida em três passos. O primeiro passo tem como complexidade $\theta(V+E)$ pois para construir o grafo é necessário

percorrer todos vértices e todas as arestas obrigatoriamente.

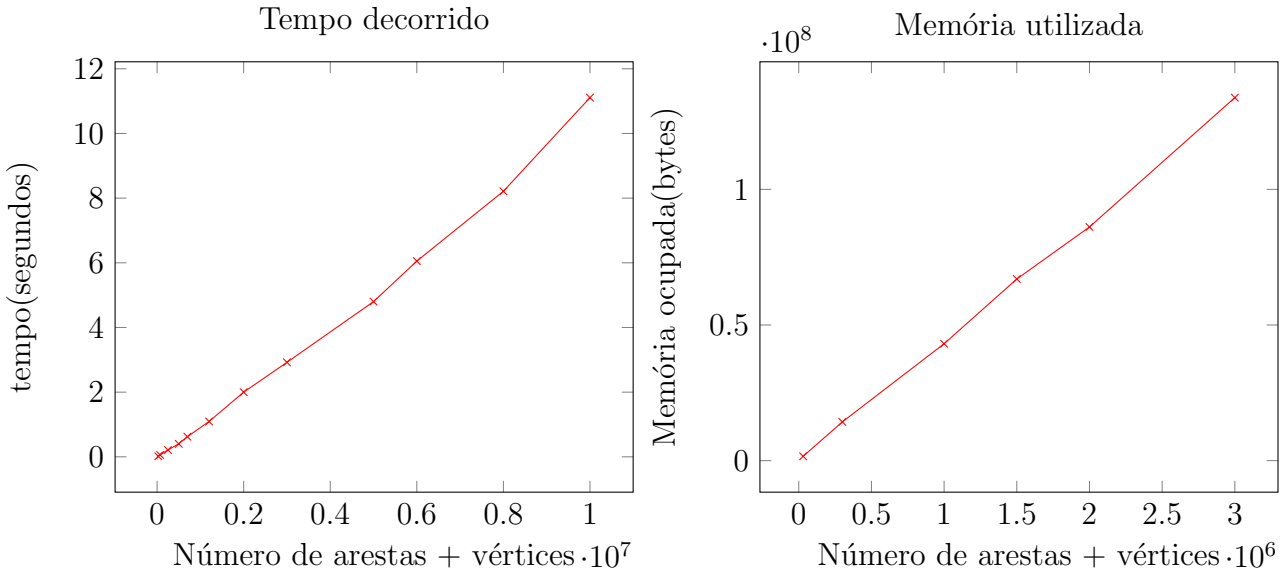
O segundo passo consiste na aplicação de uma pesquisa em profundidade tendo esta como complexidade também $\mathcal{O}(V + E)$[3].

O último e terceiro passo é $\mathcal{O}(V + E)$ pois no pior caso terá de verificar todas as ligações de todos os vértices. Deste modo, a complexidade total do programa irá ser de $\mathcal{O}(V + E)$. Em termos de espaço utilizado, o algoritmo DFS é linear ($\mathcal{O}(V)$)[1] pois, no pior caso, necessita de guardar todos os vértices na lista ordenada.

3.2 Demonstração de resultados

Para correr os testes para esta demonstração de resultados foi utilizado um computador com um processador Intel Core i7-4720HQ @ 2.60Hz 2.59 GHz com 8,00 GB de RAM, numa máquina virtual Ubuntu 16.04.2 LTS com 4 cores dedicados.

Para obter os valores dos testes corridos, utilizámos a função **time** para obter o tempo de execução e o profiler **valgrind** para analisar a quantidade de memória alocada. Para gerar os testes, utilizámos o gerador de instâncias disponibilizado na página da disciplina.



Tal como podemos observar pelos gráficos gerados, tanto o gráfico de tempo demorado por número de vértices como a memória utilizada correspondem ao esperado, sendo aproximadamente lineares.

References

[1] https://en.wikipedia.org/wiki/Depth-first_search

[2] https://en.wikipedia.org/wiki/Topological_sorting

[3] Cormen, Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. *Introduction to Algorithms (3rd ed.)*, MIT Press and McGraw-Hill.ISBN 0-262-03384-4.