Documentação TP0 - O Hit do Verão

Bruno M. Monteiro

25/03/2018

1 Introdução

Este trabalho teve como objetivo o cálculo da propagação de uma música, o *hit do verão*. As leis de propagação da música foram dadas da seguinte forma:

- Se uma pessoa escuta a música e tem menos de 35 anos, ela **gosta** da música e a **compartilha** para todos os seus familiares.
- Se uma pessoa escuta a música e tem pelo menos 35 anos, ela **não gosta** da música e não a compartilha.
- Inicialmente, só uma pessoa escuta a música.
- Outra pessoa só escuta a música se esta for para ela compartilhada.

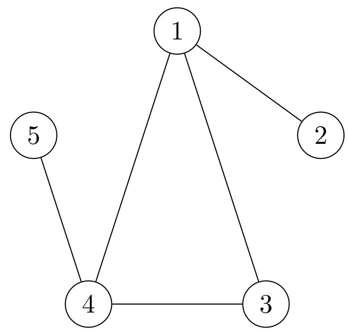
Então, dado o número de pessoas envolvidas, as idades de cada uma delas, e as relações familiares, a tarefa é descobrir quantas pessoas, ao final da propagação, terão gostado da música.

2 Solução do Problema

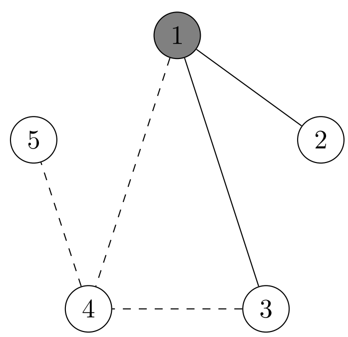
Para solucionar o problema, o modelei por um grafo não direcionado, em que um vértice representa uma pessoa e uma aresta representa uma relação familiar. Por exemplo, para a seguinte entrada (no formato especificado):

```
5
5
1 12
2 32
3 27
4 36
5 12
1 2
3 1
1 4
3 4
4 5
1
```

o grafo ficaria da seguinte forma:



Uma simples observação é que, como só nos interessa a propagação da música, apenas arestas entre pessoas “jovens” interessam, ou seja, se pelo menos um dos vértices é uma pessoa com idade maior ou igual a 35 anos, a aresta é desconsiderada (representada no desenho seguinte por uma aresta tracejada).



Então, basta realizar uma busca partindo do vértice inicial (representado em cinza), e verificar quantos vértices foram visitados ao final da busca.

Uma busca em profundidade (DFS) foi então feita. O algoritmo está elucidado em forma de pseudocódigo a seguir:

Algorithm 1 Busca em Profundidade

Input: Grafo G, vértice k

```
function DFS(i)
    Marca i como visitado
    for all aresta e = (i, j) ∈ E(G) do
        if j não foi visitado then
            DFS(j)
DFS(k)
```

3 Análise de Complexidade Assintótica

3.1 Complexidade Assintótica de Tempo

O programa faz duas coisas principais: lê a entrada e executa a **Busca em Profundidade**. Vamos então analisar as complexidades separadamente.

Denotaremos por n o número de pessoas e m o número de relações familiares.

3.1.1 Leitura dos dados: $\mathcal{O}(n + m)$

Nessa etapa, cada uma das n idades é lida, e depois cada uma das m relações (arestas), em tempo $\mathcal{O}(1)$. Assim, a complexidade é $\mathcal{O}(n + m)$.

3.1.2 Busca em Profundidade: $\mathcal{O}(n + m)$

Para saber a complexidade da **Busca em Profundidade**, basta perceber que cada vértice só é visitado no máximo uma vez, e que cada aresta é processada no máximo duas vezes (uma para cada vértice em que ela incide). Dessa forma, é evidente que sua complexidade é $\mathcal{O}(n + m)$.

Assim, a complexidade assintótica de tempo do programa é $\mathcal{O}(n + m) + \mathcal{O}(n + m) = \mathcal{O}(n + m)$.

3.2 Complexidade Assintótica de Espaço

Além de variáveis simples, apenas 3 vetores são armazenados em memória. Dois deles, o vetor de visitados e o vetor contendo as idades das pessoas, possuem n posições. Portanto, seu custo de espaço é $\mathcal{O}(n)$.

O outro vetor, e a principal estrutura de dados do programa, é a representação do grafo. Tal representação é feita por um vetor de listas encadeadas: na posição i do vetor, estão todas as arestas que incidem no vértice i.

Portanto, cada aresta é representada duas vezes (uma para cada vértice em que ela incide). Com n posições, e $\mathcal{O}(m)$ arestas representadas, o grafo ocupa espaço $\mathcal{O}(n + m)$ na memória.

Assim, a complexidade assintótica de espaço do programa é $\mathcal{O}(n + m)$.

4 Análise Experimental

Para realizar a análise experimental, foi criado um programa em *Python* para gerar casos de teste automaticamente, para um número definido de vértices e uma certa probabilidade de criação de aresta entre dois vértices, utilizando uma biblioteca que cria grafos com tais especificações. Além disso, um programa em *C++* foi utilizado para gerar os grafos e marcar o tempo de execução do programa. Essa linguagem foi escolhida porque ela possui formas mais precisas de medir o tempo que a biblioteca **<time.h>**.

Os grafos foram gerados com probabilidade de 10% de criação de aresta entre dois vértices, de forma que o número esperado de arestas é:

$m \approx 0.1 \times \frac{n^2-n}{2} \Rightarrow m \in \mathcal{O}(n^2)$

Então, o algoritmo, que é $\mathcal{O}(n+m)$, vai gastar tempo $\mathcal{O}(m)$ para executar, ou seja, linear no número de arestas.

Fazer isso é melhor que, por exemplo, variar n mantendo m constante. O motivo disso é que, se mantém-se m constante, o grafo fica cada vez mais esparso, e a busca fica cada vez mais curta, de forma que o comportamento assintótico não será observado.

O algoritmo foi então testado para cerca de 500 grafos. Fazendo uso da linguagem R, os resultados obtidos foram colocados em forma de gráfico:

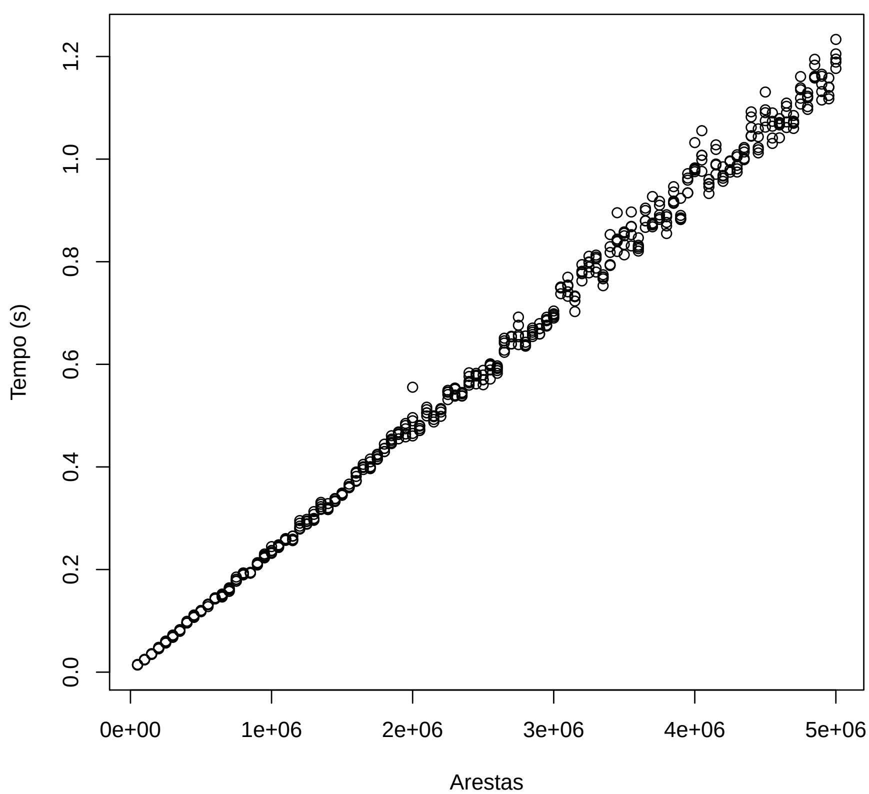


Figura 1: Tempo de execução do programa

5 Conclusão

Neste trabalho, o problema *O Hit do Verão* foi resolvido em tempo linear, modelando o problema por um **grafo** e fazendo uso de uma DFS. A utilização de uma lista de adjacência foi escolhida (em vez de uma matriz de adjacência) com o intuito de reduzir o uso de memória, que também ficou linear.

Por fim, a complexidade foi ilustrada e verificada por meio de análise experimental, na qual verificou-se que o algoritmo tem um comportamento, de fato, linear no número de arestas.