Documentação do Trabalho Pratico 0 de AEDS III

Aluno – Jesimon Barreto Santos (2016070093)

1 Introdução

O ano é 2020 e o DJ Victor Diniz acaba de lançar o hit do carnaval. A música é envolvente, e uma vez que alguém com menos de 35 anos escuta ela, a pessoa gosta da música e compartilha a música com todos seus familiares (e mais ninguém). Infelizmente pessoas com 35 anos ou mais odeiam a música e não a compartilham. Sua função como o amigo de Vitor é calcular quantas pessoas gostaram da bela melodia composta pelo artista belo-horizontino. Para isso você recebe uma lista com as idades de cada pessoa, as relações familiares dessas pessoas, e a primeira pessoa a ouvir a música.

2 Abordagem do Problema

Com base no problema apresentado, foi induzido uma abordagem em grafos para resolvê-lo, de maneira que pressupõe e facilita visualização facilmente.

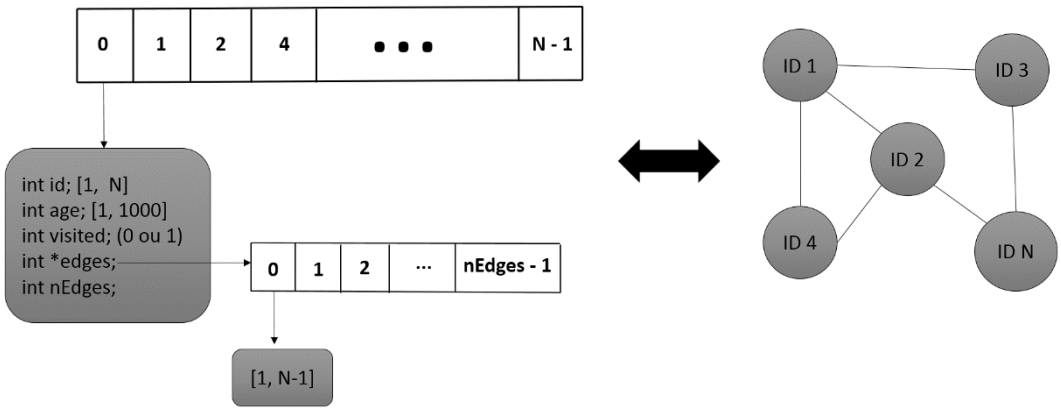

Figura 1. Arquitetura de Representação de dados

Na figura 1 é apresentada a forma que os dados, já apresentados como grafos, são tratados nesse trabalho prático, a abordagem explicita uma estrutura de lista de adjacência. Dois métodos são mais conhecidos para representação de grafos, matriz de adjacência e lista de adjacência, o formato de lista foi escolhido pelo motivo que o grafo é não direcionado, se matriz fosse escolhida se tornaria uma matriz quadrada cuja a metade da matriz seria redundância, o que não é necessário.

As arestas são representadas por uma lista de inteiros, presente em cada vértice (posição na lista), que, em cada posição, guarda a posição de um outro vértice válido na lista. É importante

deixar claro que, não é usado o ID como referência, mas sim a posição dele na lista, essa atitude tem o objetivo diminuir processamento na busca. Seguindo a mesma ideia, esse programa considera que os dados apresentados inicialmente já estão com seus IDs ordenados.

2.1 Análise Geral das Principais Funções

Para solução do problema foram necessários algoritmo binário de busca, um algoritmo de verificação de dualidade de associação entre dois vértices e a função e busca. O algoritmo de busca implementado foi binária, o método de busca serve para passar de id para posição que o vértice ocupa na lista. Seguindo, o algoritmo de **verifyDuplicate** verifica se a associação já foi feita anteriormente caso sim não associa novamente. Por último, método de busca no grafo é baseado em fila, alocação e método de controle da fila aloca e desaloca a todo momento dados na fila.

3 Complexidade

Nessa seção são apresentados as complexidades de tempo e espaço alocado para execução desse programa.

3.1 Complexidade de tempo

Visualização geral, o que atrasa é a verificação de relações duplicadas, o que consequentemente, traria problemas para abordagem de busca proposta. Foram separadas 3 complexidades que afetam significativamente no tempo de execução do programa proposto.

No pior caso, a função verifyDuplicate proporciona ao programa uma complexidade, no pior caso, de O(MN), cujo **M** é o número de relações feitas e **N** número de vértices, já que executa duas buscas binarias O(N/2), logo depois analisa qual o vértice com menos arestas e faz uma busca linear O(N-1), porém o vértice pode se conectar a todos os outros menos o próprio, o que determina a ordem de N*M é o fato de que esse processo é executado para cada relação feita, ou seja, O([(2*(N/2)+(N-1)]*M), o que no pior caso, apresenta um algoritmo de ordem O(NM) relacionado ao número de associações entre vértices e número de vértices.

Para **busca** (função **search**), segue uma ideia baseada em fila, cujo o primeiro valor a entrar é a posição do vértice com o primeiro id, último valor dado no arquivo. Para isso, inicialmente, é feita uma busca binaria no vetor O(N/2). Depois disso, um **while** envolve as próximas atividades, esse é executado com base em uma fila que serão inseridos os vértices a quem forem sendo compartilhados. Na função **visit**, chamada dentro do while, apresenta O(N) caso a pessoa compartilhe, existe um for que verifica se cada aresta já foi inserida na fila a serem visitados e se já foram visitados, se não para dos dois casos então serão inseridos na fila, isso apresenta O(N-1), a fila pode conter N-1 elementos no pior caso, representado pela função **exist**. Porém, essa função anteriormente citada, é chamada para cada aresta do vértice visitado e que compartilhou, pode ser N-1 porque um vértice não tem ligação com ele mesmo, ou seja O((N-1)*(N-1)). Ainda dentro do **while**, apresenta um for que elimina a primeira posição, a que acabou de ser visitada, mas antes copia todos os dados para a posição anterior e libera a última posição

que apresenta O(N). Dessa forma, O(N*([N-1]*[N-1])*N), ou seja, ordem de $O(N^4)$. O que irá prevalecer entre O(M*N) e $O(N^4)$.

3.2 Complexidade de Espaço

Nesse caso, são alocados N vértices, cada vértice possui, 4 inteiros e um ponteiro para inteiro. Cada inteiro ocupa 4 bytes, e cada ponteiro 8 bytes, assim cada vértice ocupa 24 bytes. Além disso, possui uma fila, com um inteiro e um ponteiro para inteiro, assim mais 12 bytes. Analise a acrescentar é que cada um dos ponteiros de inteiro, tanto do vértice quanto da fila pode apresentar até N inteiros.

4 Experimentos

Os testes foram feitos em um computador com processador i5, 2.6 GHz e memória de 6 GB. Inicialmente, foram executados os experimentos dos 10 casos de teste. Apresentados na tabela e no gráfico 2d, na figura 2, o primeiro relaciona o número de vértice de cada teste com o tempo em segundos, o segundo relaciona o número de relações(arestas).

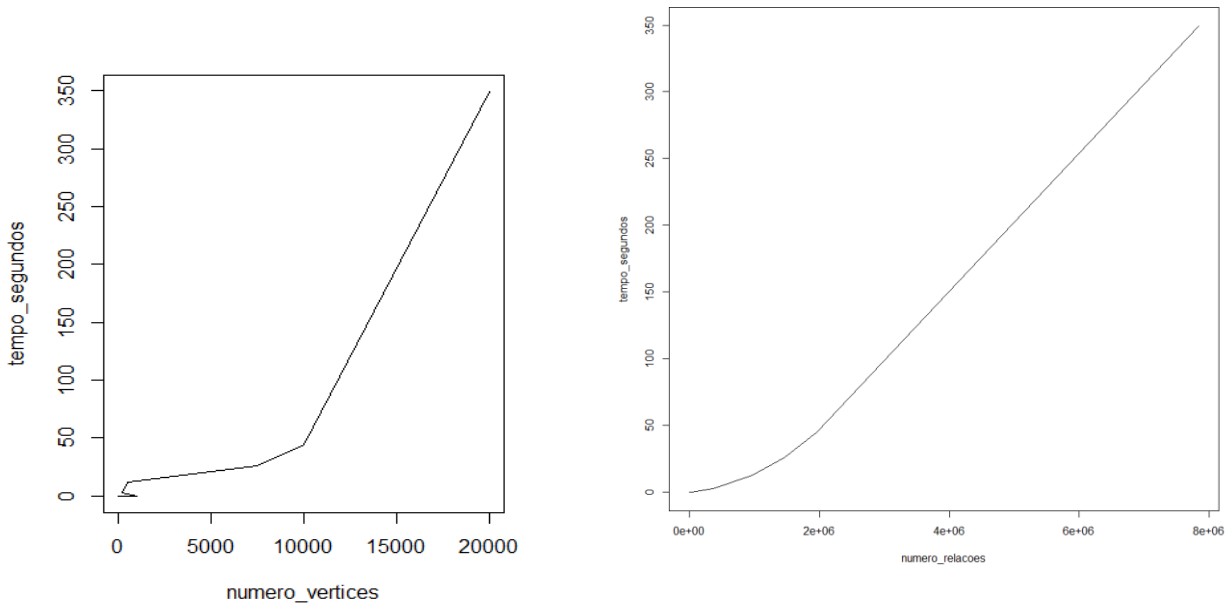

Figura 2. Referente se refere à os casos de teste feitos, uma mostra a relação número de vértices-tempo e outra, número de arestas-tempo.

Na tabela 1 e figura 3 está sendo apresentada a analise do comportamento, na prática, do programa. O experimento consistiu em usar o caso de teste 10, apresentado em conjunto com a documentação do TP, variando apenas o número de arestas e o vértice de início se manteve fixado. Apesar de apresentar o resultado, essa tabela vem com objetivo de julgar apenas o tempo de execução, pois o resultado varia de acordo com as relação apresentadas, e isso não foi tratado quando foi gerado as variações do arquivo “teste_10.txt”. Em termos práticos, o

arquivo teve seu número M de arestas reduzidos e as linhas de relações que excediam M foram apagadas.

Tabela 1. Analise de tempo variando número de Arestas no caso de teste 10.

Tempo (s)	Número Vértices	Número Arestas	Resultado
0.002	20000	784	1
3.146	20000	980000	1
7.508	20000	1960000	1
17.298	20000	3920000	1
349.228	20000	7840000	6877

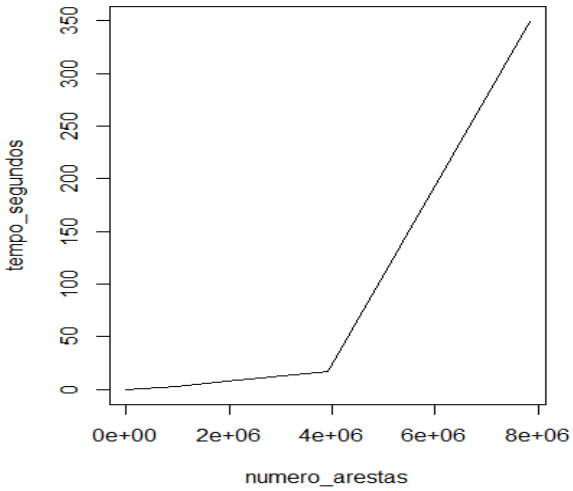

Figura 3. Apresenta Gráfico do experimento feito com variações do caso de teste 10.

A apresentação dos dados segue a ordem de complexidade apresentada na seção 3. Os arquivos gerados extras seguem em anexo com o código.

5 Conclusão

Esse programa foi criado para resolver um problema de alcance e aceitação de uma música com base na idade das pessoas que ouvem e se compartilham ou não. Foi usado grafo para modelar e apresentar uma solução, que se demonstra eficaz para resolução do problema, com base nos testes feitos, apesar de apresentar uma alta complexidade de tempo.