Trabalho Prático 0

1. Introdução

O objetivo deste trabalho foi solucionar o problema de se conhecer a aprovação do trabalho feito pelo DJ Victor Diniz dado um grupo de pessoas, as relações familiares entre elas e a primeira pessoa a escutar a música. Através destas informações, podemos simular o compartilhamento do hit de carnaval entre as pessoas e realizar a contagem ao final da simulação a quantidade de pessoas que gostaram da música. O critério estabelecido para determinar se uma pessoa gosta ou não da música é a idade, sendo que pessoas com 35 anos ou mais não gostam da música e não repassam para os familiares, e as demais gostam da música e compartilham com os familiares.

2. Modelagem do problema

A modelagem utilizada na solução se baseia na criação de um grafo não orientado, onde os vértices representam as pessoas e as arestas são as relações familiares entre elas. Através deste grafo podemos simular o compartilhamento entre as pessoas, caminhando sobre ele e determinando se as pessoas gostaram ou não da música através do critério fornecido.

Dentre as entradas que o programa recebe, uma delas especifica a identificação da primeira pessoa a escutar a música do DJ. Para simular o compartilhamento da música a partir desta primeira pessoa, foi feito um caminhamento no grafo a partir do vértice correspondente à pessoa identificada. A estratégia para o caminhamento utilizada foi a do *Deep First Search*, em que se dá preferência para o caminhamento sobre vértices cada vez mais distantes do vértice de origem. O critério para o avanço no grafo se dá pelo critério de aprovação da música, e caso uma pessoa não goste da música, os vértices adjacentes ao vértice correspondente não são visitados através dele.

A abordagem utilizada para armazenar os dados do grafo foi a de listas de adjacência. A escolha desta abordagem foi influenciada pela informação dada na especificação que diz que pode-se considerar que os grafos seriam esparsos, ou seja, a quantidade de arestas é muito menor que o quadrado da quantidade de vértices.

Um grafo nesta solução é formado por um vetor de tamanho fixo V, onde são guardados os V vértices e suas informações. Cada vértice possui uma lista encadeada que guarda as informações das arestas que estão conectando este vértice a outros. A vantagem em se utilizar listas de adjacências é não desperdício de memória com o armazenamento das arestas.

3. Análise Teórica do Custo Assintótico

3.1. Análise Teórica do Custo Assintótico de Tempo

Para analisar o custo assintótico de tempo, a solução desenvolvida pode ser encarada pela composição de 5 partes: criação do grafo e inserção de vértices, ordenação do vetor de vértices, relacionamento dos vértices, simulação do compartilhamento e liberação da memória alocada. Estes 5 passos estão detalhados nas próximas subseções, bem como conclusão sobre o custo assintótico de tempo.

3.1.1. Criação do grafo e inserção de vértices

A criação das estruturas do grafo é constante, logo $O(1)$. Neste passo, a complexidade dominante é a de inserção de vértices no grafo que é da ordem de $O(V)$, sendo V o número de vértices do grafo.

A inserção de um vértice consiste no armazenamento de uma nova estrutura de dados criada para uma pessoa. A inserção propriamente dita no vetor de vértices é $O(1)$, mas ainda é necessário um laço que executa a inserção no vetor V vezes, sendo V a quantidade de pessoas que o usuário estabelece através da entrada.

3.1.2. Ordenação do vetor de vértices com base no id de cada pessoa

A solução implementada foi pensada para suportar identificadores em forma de texto, para que outras informações como nome, CPF, RG ou qualquer identificador único possa ser utilizado. A decisão foi tomada com o intuito de aplicar conhecimentos aprendidos na matéria de AEDS II, como ordenação e busca eficiente em um vetor.

Esta decisão aumenta a complexidade de recuperação do vértice de uma pessoa, porque requer uma busca no vetor de vértices para se encontrar determinado vértice. Este problema pode ser facilmente solucionado com a utilização de outras estruturas de dados que possuem indexação de dados por chaves de texto, como por exemplo hashmaps que têm a complexidade para a recuperação de dados da ordem de $O(1)$. Por falta de tempo para implementação de uma hashmap estável, optei por implementar melhorias na busca de pessoas no vetor de vértices para reduzir a complexidade deste procedimento para a ordem de $O(log(n))$, sendo n a largura do vetor.

O passo de ordenação foi pensado para reduzir o tempo de busca no vetor de vértices, que acontece várias vezes no passo de relacionamento dos vértices. A ordenação possibilita a busca binária no vetor, que é melhor explicada na próxima subseção referente a este passo. O algoritmo de ordenação utilizado é o *Quicksort* com complexidade da ordem de $O(V \times log(V))$, sendo V a quantidade de vértices armazenados no vetor.

3.1.3. Relacionamento dos vértices

A entrada fornece o número E de arestas do grafo. Um laço é executado E vezes, lendo as informações de uma aresta que contém dois identificadores *id1* e *id2*. Para relacionar as duas pessoas que são identificadas por *id1* e *id2*, uma busca no vetor de vértices precisa ser feita.

O método de busca escolhido foi a busca binária, que consiste basicamente na divisão do vetor pela metade sucessivamente. A cada divisão, o algoritmo escolhe uma das metades comparando se o id da pessoa que o vértice que está dividindo armazena é maior ou menor do que o id que se procura. Assim, a complexidade da busca no vetor de vértices têm complexidade da ordem de $O(log(V))$, sendo V o número de vértices que o vetor guarda.

Após a recuperação dos dois vértices v1 e v2, a relação é feita inserindo um item na lista encadeada de v1 que guarda um ponteiro para v2, e outro em v2 para v1 para que se crie arestas não direcionadas. A inserção dos itens nas listas têm complexidade $O(1)$.

Com o laço executando E vezes duas buscas de complexidade $O(log(V))$, a complexidade final do passo de relacionamento dos vértices é de $O(E \times log(V))$, sendo E o número de arestas do grafo, e V o número de vértices.

3.1.4. Simulação do compartilhamento e contagem de pessoas que gostaram da música

O algoritmo desenvolvido para a simulação do compartilhamento consiste no caminhamento pelo grafo construído até agora. A estratégia para o caminhamento no grafo utilizada foi a mesma do algoritmo de busca *Deep First Search*, que consiste na exploração recursiva das arestas do vértice atual.

A entrada do programa nos fornece o identificador da primeira pessoa a escutar a música, que é buscada no vetor de vértices. A busca por esta pessoa é $O(log(V))$, sendo V o número de vértices.

O algoritmo começa com o vértice especificado pela entrada e com um contador de pessoas que gostaram começando com 0. À cada vértice que é visitado, o algoritmo seta a *flag listened* para indicar que a pessoa já escutou a música e assim representar também a visita no vértice. Caso a idade da pessoa seja menor que 35 anos, incrementamos o contador e exploramos os vértices que estão conectados (familiares da pessoa). Assim, recursivamente caminhamos pelo sub-grafo de pessoas que gostaram da música e quem têm familiares que também gostaram da música.

O cenário que gera o pior caso do algoritmo é aquele em que todas as pessoas gostam da música, ou seja, têm menos de 35 anos. Assim, cada vértice seria visitado uma vez, e suas arestas também seriam exploradas uma vez. Com V e E sendo o número de vértices e o número de arestas no grafo, respectivamente, podemos definir a complexidade deste

algoritmo como $O(V + E)$, por que a complexidade para se recuperar vértices e arestas é $O(1)$.

3.1.5. Liberação da memória alocada no programa

Este passo tem complexidade assintótica de tempo proporcional à complexidade assintótica de espaço, que é melhor detalhada na próxima seção. Isso de deve ao fato de que para liberar a memória alocada, cada estrutura deve ser individualmente liberada. A liberação de memória tem custo assintótico de tempo constante, logo a complexidade para se liberar a memória alocada é de $O(V + E)$.

3.1.6. Complexidade assintótica de tempo final

Através da análise, podemos concluir que os dois passos mais custosos assintoticamente são o de ordenação do vetor de vértices e de relacionamento dos vértices, com complexidades da ordem de $O(V \times log(V))$ e $O(E \times log(V))$. Assim, desenvolvendo a equação resultante da soma das complexidades, chegamos na complexidade assintótica de tempo final de $O(E \times V \times log(V))$.

3.2. Análise Teórica do Custo Assintótico de Espaço

3.2.1. Pessoa

A complexidade de espaço para se guardar os dados de uma pessoa é constante. Uma pessoa guarda:

- Seu identificador único, sendo uma string dinamicamente alocada de no máximo 265 caracteres;
- Sua idade;
- Uma *flag* que indica se a pessoa já escutou a música;

3.2.2. Listas e Células

Uma célula da lista guarda:

- Um ponteiro genérico para o item que ela guarda. No caso deste problema, todas as listas encadeadas guardam as arestas em forma de referência para vértices.
- Um ponteiro para a próxima célula da lista;
- Um ponteiro para a célula anterior à ela.

Como células só guardam ponteiros, que têm complexidade de espaço constante, a complexidade de espaço de uma célula também é constante.

Uma lista encadeada guarda:

- O tamanho atual da lista;
- Um ponteiro para a primeira célula da lista;
- Um ponteiro para a última célula da lista;
- Um ponteiro para a última célula que foi acessada da lista;
- A posição que a última célula que foi acessada se encontra.

Como uma lista é responsável pelo controle das células que guardam as informações, e cada célula têm complexidade de espaço constante, a lista tem complexidade de espaço $O(n)$, sendo n a quantidade de células que a lista guarda.

3.2.3. Grafo e vértices

Um vértice guarda:

- Um ponteiro genérico, que aponta para os dados guardados pelo vértice. No caso deste problema, todos os vértices apontam para as respectivas pessoas que representam. As estruturas que representam uma pessoa são alocadas dinamicamente;
- Uma lista encadeada que guarda referências para outros vértices, representando as arestas. A lista é alocada dinamicamente.

Um vértice precisa de 4 alocações de memória para ser criado. Como a lista encadeada guarda as arestas do vértice, e a complexidade de espaço de uma lista é $O(n)$, sendo n a quantidade de células, a complexidade de espaço de um vértice é $O(A)$, sendo A a quantidade de vértices que estão conectados à este vértice.

Um grafo guarda:

- O número de vértices do grafo;
- Um vetor que guarda todos os vértices do grafo, que é alocado dinamicamente.

Todas as arestas estão representadas como as células das listas que cada vértice guarda. A soma da quantidade de células de todas as listas encadeadas guardadas pelos V vértices resulta em $2 \times E$ arestas, pois uma aresta é adicionada nos dois vértices criando duas células diferentes. A quantidade de alocações para se criar a estrutura do grafo resulta em $4 \times V + 2 \times E$. Assim, a complexidade de espaço para se guardar um grafo é da ordem de $O(V + E)$.

3.2.4. Complexidade assintótica de espaço final

Como neste programa todas as estruturas alocadas se resumem na construção de um grafo, a complexidade de espaço final é dada pela complexidade de espaço deste, ou seja, $O(V + E)$. Todas as outras alocações são referentes à variáveis auxiliares, e estas alocações têm quantidade constante e não têm representatividade no crescimento assintótico.

4. Análise do Experimento

O experimento feito utiliza os 10 casos de teste fornecidos pelo monitor. Os programas *time* e *valgrind* foram utilizados para a coleta de tempo de execução e quantidade de memória alocada, respectivamente. O experimento foi feito usando uma máquina com processador Intel Core™ i7-3630QM CPU 2.40GHz × 8 e memória RAM de 7,5 GiB.

Tempo de execução

O processador citado faz 34.29 IPC (Instructions Per Clock cycle), e utilizando a frequência de 2.40GHz conseguimos fazer uma aproximação de quantas instruções por segundo o processador executa:

$$2,4 \times 10^9 \times 34,29 = 8,2296 \times 10^{10}$$

Assim, chegamos à medida de $8,2296 \times 10^{10}$ IPS (Instructions per second).

Para confirmar a análise assintótica de tempo teórica foi feito o cálculo de $E \times V \times log(V)$, sendo E e V a quantidade de arestas e vértices estabelecidas pelos casos de teste. O cálculo foi feito a fim de obter uma valor teórico de quantas instruções o programa faria para cada caso de teste.

Utilizando a quantidade de instruções que o programa faria e a quantidade de instruções estimada que o processador da máquina utilizada executa, conseguimos calcular um tempo estimado em segundos de quanto o programa levaria para executar com cada caso de teste.

No experimento, foi observado o tempo de execução do programa para cada caso de teste. Os tempos de execução foram coletados 5 vezes para cada caso de teste, e o valor final utilizado corresponde à média destes 5 tempos observados.

Abaixo se encontram os valores encontrados e o gráfico gerado a partir dos dados da tabela.

Caso de teste	Nº de Vértices (V)	Nº de Arestas (E)	Qtd. de instruções estimada	Tempo estimado (s)	Tempo observado (s)
1	5	5	17,47425011	0,000000000212334136 6	0

2	7	9	53,24117652	0,0000000006469473185	0
3	100	900	180000	0,000002187226597	0
4	500	7275	9817503,391	0,0001192950252	0,01
5	1000	24375	73125000	0,0008885608049	0,03
6	2000	360000	2376741597	0,02888040241	0,31
7	5000	960000	17755056021	0,215746282	0,94
8	7500	1460000	42431920834	0,5156012544	1,44
9	10000	1960000	78400000000	0,9526586954	2,23
10	20000	7840000	674401503320	8,194827249	8,94

Tabela 1: Quantidade de tempo estimada e observada do programa para cada caso de teste.

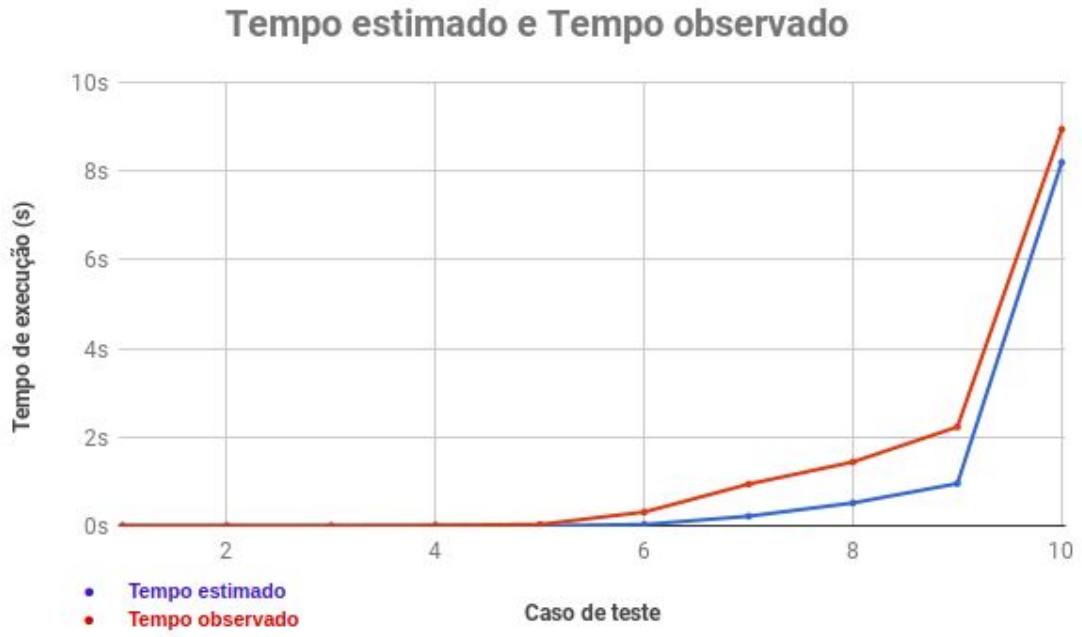

Gráfico 1: Comparação gráfica entre o tempo de execução esperado e observado do programa para cada caso de teste.

Através da análise do gráfico podemos concluir que a complexidade assintótica de tempo se confirma, pois podemos considerar que a diferença observada no tempo de execução esperado e observado se deve às outras instruções que não são contabilizadas na quantidade de instruções esperada de um programa com complexidade assintótica $O(E \times V \times log(V))$ e à concorrência do processo com outros programas do computador. A criação do grafo, a simulação de compartilhamento e a liberação de memória ainda possuem uma influência considerável no tempo de execução para tais casos pois não têm custo assintótico constante.

Alocações feitas e complexidade de espaço

Utilizando o programa *valgrind* foram coletadas a quantidade de alocações de memória feitas pelo programa em sua execução para cada caso de teste. Os dados da coleta se encontram abaixo, bem como a quantidade de alocações esperada de acordo com a análise de complexidade assintótica de espaço que corresponde à $4 \times V + 2 \times E$.

Caso de teste	Nº de Vértices (V)	Nº de Arestas (E)	Qtd. de alocações esperada	Qtd. de alocações observada
1	5	5	30	37
2	7	9	46	53
3	100	900	2200	2207
4	500	7275	16550	16557
5	1000	24375	52750	52757
6	2000	360000	728000	728007
7	5000	960000	1940000	1940007
8	7500	1460000	2950000	2950007
9	10000	1960000	3960000	3960007
10	20000	7840000	15760000	15760007

Tabela 2: Quantidade de alocação de memória esperada e observada para cada caso de teste.

Podemos observar uma pequena diferença de 7 alocações entre a quantidade esperada e a observada, que se deve à alocações de variáveis auxiliares e à alocação de memória do próprio grafo construído. Apesar da diferença, os valores observados confirmam a complexidade da ordem de $O(V + E)$ concluída a partir da análise de custo assintótico de espaço, uma vez que constantes são desconsideradas na análise assintótica.

5. Conclusão

Pode-se afirmar pela assertividade das saídas que o programa gera que a solução resolve o problema proposto. A adição de suporte para outros tipos de identificação de pessoas aumenta o custo assintótico de tempo, mas trouxe uma funcionalidade interessante para a solução possibilitando à quem utiliza o programa o uso de nomes, CPFs e qualquer identificador único que possa ser representado em forma de texto. Como mencionado, o problema do aumento de complexidade pode ser solucionado com a continuidade do trabalho na implementação de outras estruturas de dados que possibilitam recuperação de dados através de chaves únicas mais eficiente.