Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Trabalho 02 - Grafos

SCC05003 - Algoritmos e Estrutura de Dados II
Professora: Elaine Parros Machado de Sousa
Estagiário PAE: Evandro Ortigossa

Patrick Oliveira Feitosa 10276682

São Carlos, São Paulo 19/07/2020

1. Introdução

Esse trabalho tem como principal objetivo a implementação de um Grafo, o qual servirá para auxiliar na investigação do mundo de Kevin Bacon.

2. Descrição das Implementações

Para esse problema foi decidido que o grafo seria do tipo não-direcionado, no qual o vertices seriam os atores/atrizes e as arestas referência atoresa/atrizes que contracenaram em um mesmo filme. Abaixo você pode encontrar um exemplo visual.

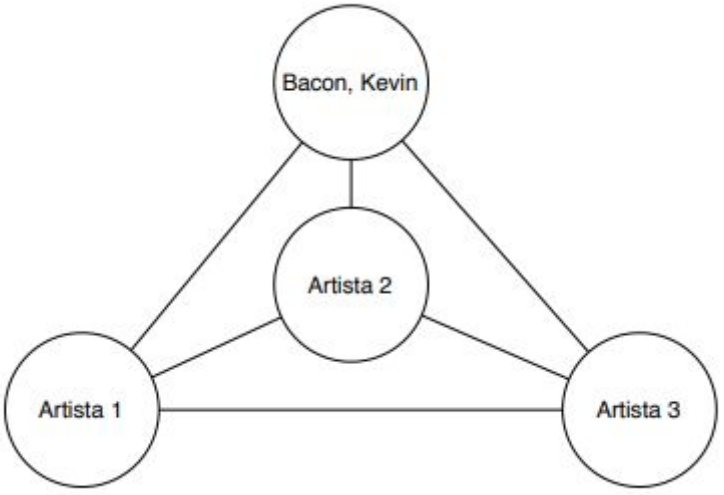

Figura 1:Ilustração do grafo

Nesse exemplo, vemos que os artista 1, 2, 3 e o KB estão todos conectado entre si, isso quer dizer que ele contracenaram juntos em algum filme. Além disso, cada aresta, conterá a informação de distância do nó seguinte, em relação ao KB. O valor de KB será dado pela aresta de menor distância. Nesse exemplo temos todos os Artistas com KB = 1.

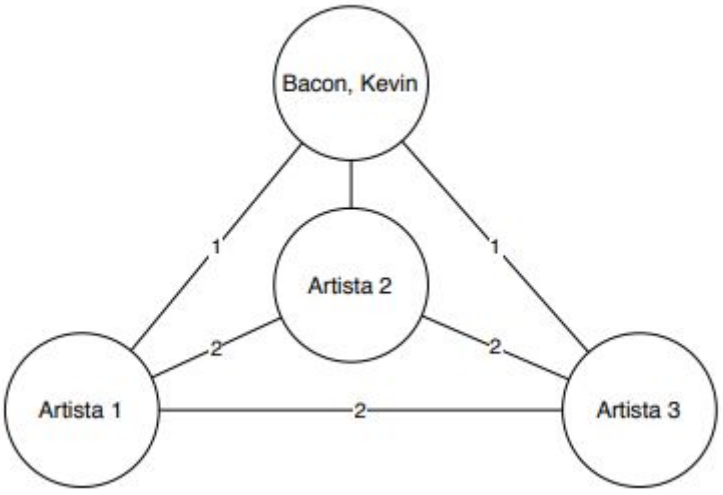

Figura 2:Ilustração do grafo

Para a construção do Grafo foi utilizada a estrutura de listas de adjacências, uma vez que a modulação do problema se mostrou mais simples de ser compreeendida. Abaixo você econtrará um diagrama referente a estrutura criada:

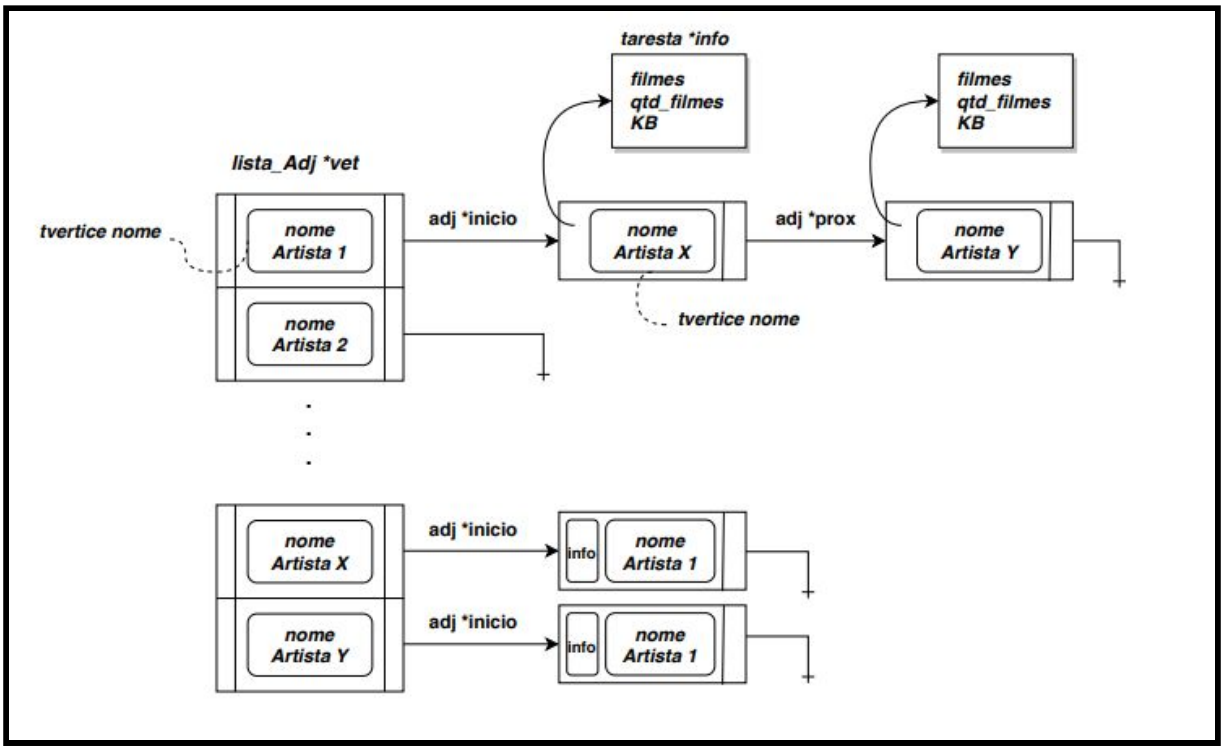

Figura 3: Estrutura do grafo criado com listas de adjacências

a. Grafo

O grafo é definido por uma *struct* de nome **tgrafo**. Nele contém a quantidade de vertices e a referência para um vetor do tipo **lista_Adj,** o qual armazenará em cada posicao um vertice e a referência para o inicio da lista de adjacência referente aquele vertice.

```
typedef struct {
    lista_Adj *vet;
    int num_vertices;
} tgrafo;
```

Figura 4: Estutura do grafo

```
typedef struct {
    tvertice nome;
    adj *inicio;

}lista_Adj;
```

Figura 5: Estutura de cada posicao de vet

b. Nós da lista

Os nós das lista, além da referência para o nó seguinte, terá as informações de vertices e da aresta.

```
typedef struct ADJ{
    taresta *info;
    tvertice nome;
    struct ADJ *prox;

} adj;
```

Figura 6: Estutura dos nos da lista

c. Aresta

Já na aresta conterá informações de filmes que aqueles dois artistas contracenaram juntos, além do valor de KB. Foi criado uma lista de filmes, uma vez que havia a possibilidade de dois atores terem contracenados mais de uma vez em filmes diferentes.

```
typedef struct {
    char **filmes;
    int qtd_filmes;
    int KB;

}taresta;
```

Figura 7: Estutura das arestas

d. Vértices

Já o vértice é dado pela seguinte string:

```
typedef char tvertice[TAM_NOME];
```

Figura 8: Estutura do grafo

3. Principais Funções

Nesse tópico irei explicar a respeito das principais funções, com qual lógica elas foram implementadas.

- ***void calculoKB****(tgrafo *grafo)*

Parâmetros:

grafo: ponteiro para o grafo

Função:

Essa função tem o intuito de percorrer as listas de adjacência, adcionando os valores KB, ou seja, as distância dos vertices em relação o nó de nome “Bacon, Kevin”.

Lógica:

Essa função apresenta 4 vetores que servem para controle no momento em que o algoritmos estiver percorrendo as listas de adjacência: ***artistasEnc***, guardará todos os artistas que já foram percorridos; ***artistasAPer***, guardará todos os artista que precisam ser percorridos; ***artistasAnter***, guarda o nome do artistas(vertices) que antecederam o vertices que precisam ser percorrido; e ***Kb***, guardará o valor de Kb referente aos vertices que precisam ser percorridos. Vale ressaltar que a primeira posição desses vetores apresentam a quantidade de elementos presentes nele, ou seja, se um vetor tiver só esse contador, apresenta o valor 1 na sua posicão, mas se outra elemento for adicionado esse valor passa a ser 2.

Inicialmente, setamos esses valores com as carateristicas para iniciar pelo vertice “Bacon, Kevin”, a partir desse momento a iteração se inicia. O primeiro vértice é retirado junto com as informções de Kb e de vértice anterior e, assim, é percorrido a sua lista adjacência preenchendo os valor de Kb, se o nó presente conter o nome do vértice anterior é atribuido o valor (Kb-1). Caso o vertices não estejam presentes ***artistasEnc,*** os vértices serão adicionados ao ***artistasAPer*** e os outros vetores serão setados com as caracteristica do vértice a ser percorrido. Essas insersões são sempre feitas no final do vetor, representando a função de uma fila. Após isso, o a iteração se repete e o vertice presente na primera posicao é retirado e será analisado, assim como o anterior.

- ***void inserirArtistas****(tgrafo *grafo, FILE *fp)*

Parâmetros:

grafo: ponteiro para o grafo

fp: ponteiro para o arquivo com as informações

Função:

Essa função irá percorrer as linhas dos arquivo e adicionará os vertices e arestas. A inserção dos vertices no ***vet*** é feita em ordem.

Lógica:

Inicialmente, cria-se as variáveis necessária. Após isso, o algoritmo realiza a leitura da primeira linha, o “\n”, caso exista no fim, é retirado. Com a leitura da linha, é realizado um split, dado pela função ***quebra_entrada***, a qual retorna um vetor de strings. A primeira posição se refere ao valor do tamanho do vetor, a segunda, ao filme, e o restante refere-se aos

artistas que contracenaram. Com esses resultados,insere-se os vertices em ordem no ***vet***, caso o vertices já exista não será feita nenhuma inserção. Posteriormente, é feita a inserçao das informações de aresta nas lista de adjacência. Isso é repetido até a leitura completa do arquivo.

- ***void pesquisarArtista****(tgrafo *grafo)*

Parâmetros:

grafo: ponteiro para o grafo

Função:

Essa função tem o intuito de percorrer o grafo printando as informações de KB, atores/atrizes e o filme que os dois contracenaram.

Lógica:

Inicialmente, o usuário digita o nome do artistas, seguindo o formato estabelecido. O "\n" é retirado. A partir desse momento, realiza-se a localização do vertice no vetor ***vet***, ao encontrar, a sua lista de adjacência é percorrida a fim de encontrar o vertice de menor KB, após encontra, essse valor é printado, junto com as informações de filme e nome do artista. Com isso, é realizado esse mesmo processo com o nome desse novo artista encontrado até chegar no "Bacon, Kevin" .

4. Interface

Foi criado uma interface na qual o usuário pode realizar as seguintes operações:

1. **Iniciar o grafo:** Inicia o grafo, criando os vertices, arestas e calculado o KB de todo o grafo. Para realizar as outras operações é necessária a execução dessa anteriormente.
2. **Número de Kevin Bacon(KB):** Realiza a pesquisa do valor de KB do artista que o usuário fornecer o nome, printando na tela o caminho até chegar no Kevin Bacon.
3. **Média e DP do Mundo:** Calcula os valores de média e desvio padrão e printa na tela.
4. **Finalizar operações:** Finaliza o programa, liberando os espaços de memória.

Se desejar realizar a visualização do grafo, basta usar o valor 5 na escolha da operação.

5. Método de Compilação e Execução

Para compilar e executar o algoritmos basta execultar os seguintes

```
make all  /*  Compila e Executa  */
```

Para verificar os teste de vazamentos de memórias, foi utilizado o programa ***valgrind***. Para realizar o teste, basta executar o comando a seguir e um arquivo de texto de nome “valgrind-out” será criado com a analise do código.

```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes --verbose
--log-file=valgrind-out.txt ./exe
```

Ao executar esse comando, o algoritmos será executado e você poderá realizar o seu uso. Após a finalização, o resultado será gerado.

Obs.: É necessária a geração do executável antes de realizar o teste. Além disso, recomenda-se que seja alterado o arquivo para o de teste. (“teste.txt”)

Se não possuir o programa, basta realizar a sua instalação:

```
sudo apt install valgrind  /* Ubuntu, Debian, etc.*/
sudo yum install valgrind  /* RHEL, CentOS, Fedora, etc.*/
```

6. Conclusão

Foi possível desenvolver o grafo para a representação do mundo de Kevin Bacon. As funções desenvolvidadas para calculo da média e devio padrão apresentaram um resultado surpreendentemente baixo, o que fortalece a teoria de seis graus de separação. Apesar desses resultados satisfatório, a função de ***calcularKB***, não apresenta tanta eficiência, uma vez que ao iniciar o grafo, o código fica executando alguns segundos em background, entretanto, após esses segundo, todas as funcionalidades estarão disponibilizadas.