Cinéfilos

Solução e análise do Algoritmo

Eduardo Lima Dornelles*
Faculdade de Informática - PUCRS

22 de junho de 2016

Resumo

Este relatório busca pela solução para o segundo problema proposto na disciplina de Algoritmos e Programação III, do primeiro semestre de 2016: Descobrir o número de Bacon de um ator qualquer.

Serão apresentados uma solução criada para gerar a estrutura do grafo de relações de atores e filmes, além de demais algoritmos para acharmos o número de bacon de qualquer ator e quais deles possuem o maior número.

1 Introdução

Dentro do escopo da disciplina de Algoritmos e Programação III o segundo problema de 2016/I pode ser visto da seguinte forma: Recebendo três arquivos de texto (.txt) um informando uma lista de atores, o outro de filmes e o terceiro relacionando os filmes aos seus atores participantes, precisamos criar uma estrutura relacionando os atores que fizeram determinados filmes. Tendo uma estrutura correta podemos verificar o numero de bacon de um ator qualquer que contem na estrutura.

O numero de bacon se define por: partindo de um ator qualquer temos que verificar se ele esteve em um filme com alguém que esteve em um filme , ... , que esteve em um filme com Kevin Bacon. Por exemplo: o ator Joãozinho contracenou em um mesmo filme que Kevin Bacon, logo seu número de Bacon é 1. Entretanto o ator Pedrinho nunca atuou com Kevin Bacon, porém já atuou em um outro filme em que o Joãozinho estava, sendo assim o número de Bacon de Pedrinho é 2, pois ele atuou em um filme com o ator tal que por sua vez atuou em outro filme com Kevin Bacon.

Na figura 1 mostrada na próxima página, podemos verificar dois casos retirados do site https://oracleofbacon.org/ mostrando que Morena Baccarin possui número de Bacon 2 e Caio castro tem numero 4.

*eduardo.dornelles.001@acad.pucrs.br

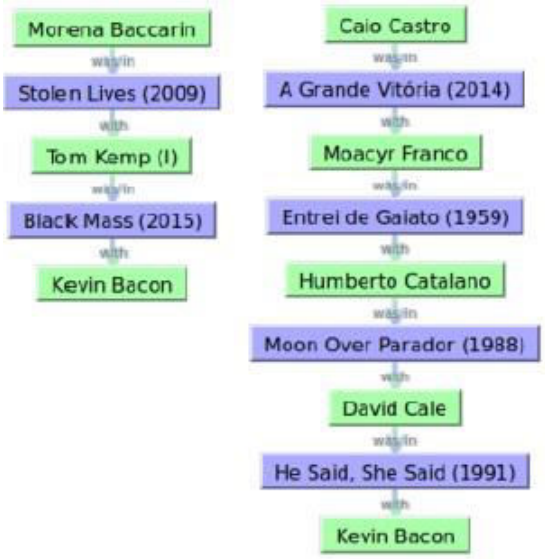

Figura 1

Para tentar resolver esse problema analisaremos uma possível solução para montar a estrutura bem como para gerar o número de Bacon.

2 Interpretando o Problema

Analisando a tarefa proposta, percebemos que ela poderia ser construída da seguinte maneira: construir um grafo não dirigido e valorado, onde os vértices são os atores e as arestas são os filmes que relacionam os atores participantes de determinado filme.

Analisando os três arquivos de texto percebemos que o primeiro (Atores.txt), contem em cada linha um número de identificação seguido do nome do ator. O segundo (Filmes.txt) também um número de identificação porém seguido do nome do filme. Já o terceiro (FilmeAtores.txt) possuí uma lista de relações de id ator para id filme em cada linha.

Para montar as ligações (arestas) do grafo partindo dessas três informações espalhadas nessas três listas seria muito trabalhoso pois teríamos que para cada relação id filme a id ator, percorrer a lista de filmes para achar o nome do filme e percorrer a lista de atores para achar o nome do ator.

3 Construindo uma Estrutura

Antes de montarmos o grafo precisamos criar uma estrutura que interprete a relação de atores de um determinado filme que estão contidas nos três arquivos de texto.

Para montar tal estrutura utilizaremos os HashMaps que nos permitem guardar uma lista de valores para uma determinada chave. Iremos criar quatro HashMaps, que são eles:

Mapa Atores - Ira guardar os valores do arquivo Atores.txt sendo cada chave um numero inteiro contendo o id do ator tendo como valor uma String com seu nome.

Mapa Filmes - Guardará os valores do arquivo Filmes.txt sendo cada chave um numero inteiro contendo o id do filme tendo como valor uma String com seu nome.

Mapa Relação - Vai guardar os valores do arquivo FilmesAtores.txt sendo cada chave um numero inteiro contendo o id do filme, que tem como valor um tipo lista de inteiros que ira guardar os ids de atores relacionados a tal filme.

Mapa Final - Esse mapa será preenchido analisando os dados dos anteriores, ele ira guardar uma chave do tipo String que será o nome do filme e ela terá como valor uma Lista do tipo String que ira conter os nomes dos atores participantes do filme.

A montagem dos três primeiro mapas (*Atores*, *Filmes* e *Relação*) é bastante trivial. Basicamente é pegar as informações contidas nos arquivos e usar para popular os respectivos mapas. Já a montagem do mapa *Final* é feita da seguinte maneira: iteramos pelo mapa *Relação* e para cada laço buscamos no mapa *Filmes* o nome do filme relacionado aquele id. Seguido de uma busca no mapa *Atores* pelo nome de todos os atores que estão na lista de valores id do mapa *Relação* da respectiva chave. Em pseudocódigo fica o seguinte:

Algoritmo 1 Função Preencher *Mapa Final*

```
1:  Método CriaMapaFinal()
2:    Para cada iteração no mapa Relação Faça
3:          filmeID ← Chave
4:          listaIdAtores ← Valores
5:          listaNomeAtores ← Nova Lista Vazia
6:             Para cada id de listaIdAtores Faça
7:                   listaNomeAtores.adiciona(mapaAtor(id).valor)
8:             fim para
9:          mapaFinal.adiciona(mapaFilme(filmeID).valor, listaNomeAtores )
10:    fim para
11: fim método
```

Note que todos os mapas são atributos e nenhum é declarado dentro do método.

3.1 Construindo o grafo

Com o *MapaFinal* criado e preenchido corretamente podemos agora começar a montar o grafo, como dito anteriormente ele será valorado e não dirigido.

Para isso iremos criar duas classes, uma chamada Vértice e outra Aresta, elas terão a seguintes estruturas:

```
Estrutura Vértice {
        String item;
        boolean visitado;
        Lista de Arestas listaAdj;
              }
```

```
Estrutura Aresta {
              Vértice v;
              String filme;
                    }
```

Como os vértices são os atores, para criar eles basta varrer o mapa atores e para cada ator criamos um novo vértice contendo o nome dele como valor no atributo *item.*

Repare também que há uma variável booleana chamada *visitado,* ela será usada mais pra frente quando estivermos andando pelo grafo. Também, há uma lista de arestas, ou seja é a lista com quais outros vértices esse vértice esta ligado. Dentro da estrutura da aresta além de conter o vértice em que esta conectado, existe um atributo chamado *filme* que ira guardar o valor dessa aresta, que no nosso caso interessa ser o nome do filme ao qual os dois atores ligados contracenaram juntos.

Com os vértices criados vamos agora para parte de montar as arestas. Para fazer isso iremos fazer o seguinte: vamos percorrer todo o M*apaFinal* e para cada iteração pegamos a chave (nome do filme) e o seu respectivo valor (lista de nomes dos atores).

Possuindo essas informações, podemos fazer a conexão de todos os atores que estão nessa lista valorando a aresta com o nome do filme em questão. Em pseudocódigo fica o seguinte:

Algoritmo 2 Função Criar Arestas

```
1:  Método CriaArestas()
2:    Para cada iteração no mapa MapaFinal Faça
3:      filme ← Chave
4:      listaAtores ← Valores
5:          Para x ← 0 até x < tamanho da listaAtores Faça
6:                  Para y ← 0 até y < tamanho da listaAtores Faça
7:                     se (y<=x ) então
8:                          avança próximo laço
9:                     senão
10:                         criaAresta(listaAtores[x], listaAtores[y], filme)
11:                    fim se
12:                 fim para
13:         fim para
14:  fim método
```

Observe que cada a iteração no *mapaFinal* conectamos todos atores que fizeram um determinado filme no grafo. os 'Paras' das linhas 5 e 6 irão fazer com que cada ator seja ligado com os outros na linha 10. Já o teste da linha 7 garante que o algoritmo não ira tentar ligar um ator com ele mesmo e nem fazer ligações que já foram feitas anteriormente.

3.2 Algoritmo de Bacon

Agora que estamos com o grafo G = (V,A) devidamente criado, podemos construir o algoritmo que nos permitam saber o numero de Bacon de um ator qualquer.

O número de Bacon nada mais é do que a distancia (numero de arestas) mínima entre o ator escolhido e Kevin Bacon. Por exemplo: um ator que possua o numero de

Bacon 3, significa que ele deve que percorrer 3 arestas até chegar no Bacon, passando por 2 vértices. Para resolver esse problema utilizamos a pesquisa BFS (Breadth-first search) que nos permite percorrer o grafo em largura, pois assim, partindo de qualquer ator podemos verificar todos os atores que estão a uma determinada distancia dele.

Partindo de Kevin Bacon (*atrOrigem*) pegamos todos os atores que estão conectados a ele e verificamos se algum deles é o ator que estamos procurando, caso encontrarmos retornamos o valor 1 (distancia). Senão andamos no grafo e pegamos todos os atores que estiverem a uma distancia 2 e assim sucessivamente até achar o ator em questão, em pseudocódigo fica assim:

Algoritmo 3 Função Geradora de Bacon

```
1:  Método numBacon(String atrOrigem, String atrDestino)
2:    distancia ← 0
3:       se (atrOrigem = atrDestino )
4:          retorna distancia
5:       fim se
6:    Vertice origem ← LocalizaVertice(atrOrigem)
7:    Vertice destino ← LocalizaVertice(atrDestino)
8:    limparVertVisitados()
9:    Lista<Vertice> ListaAtual ← nova Lista Vazia
10:   Lista<Vertice> ListaAuxiliar ← nova Lista Vazia
11:   filaAtual.adiciona(origem)
12:   origem.Visitado ← verdadeiro
13:      Enquanto (ListaAtual <> vazia) Faça
14:         se (ListaAtual não contem origem) então
15:            distancia ← distancia + 1
16:         fim se
17:         se (ListaAtual contem destino) então
18:            retorna distancia
19:         fim se
20:         Para cada Vertice v de ListaAtual Faça
21:            v.Visitado ← verdadeiro
22:                  Para cada Aresta adj de v.listaAdj Faça
23:                     se (adj.Vertice não foi visitado) então
24:                        ListaAuxiliar.adiciona(adj.Vertice)
25:                        adj.Vertice.Visitado ← verdadeiro
26:                     fim se
27:                  fim para
28:          fim para
29:         ListaAtual ← ListaAuxiliar
30:         ListaAuxiliar ← nova Lista Vazia
31:      fim enquanto
32:   retorna -1
33: fim método
```

Nas linhas 6 e 7 temos um simples método que nos localiza a referencia dos vértices de cada ator no grafo, precisamos disso para saber de qual vértice iremos partir do grafo e qual queremos chegar. A linha 8 possuí um método local também bem simples porém muito importante, ele faz com que todos os vértices do grafo fiquem com o status "Não visitado", pois precisamos garantir que estejam todos assim senão pode ocorrer problemas no caminhamento. O enquanto da linha 12 é quem será responsável por percorrer o grafo, ele ira iniciar com o vértice *origem* e a cada repetição ele valida se o vértice *destino* está contido na lista. Caso não esteja, ele passa para linha 20 e entrará no Para que é responsável por pegar todos os vértices que estão na L*istaAtual* e internamente (com outro Para, linha 22) pegar todos os vértices que estão a uma *distancia + 1* adicionando eles a *ListaAuxiliar.*

Para evitar que esse andamento pelo grafo não tenha problemas de ciclos e pegue as distancias corretamente, precisamos marcar os Vértices como visitados sempre que adicionarmos ele a lista, isso é garantido nas linha 21 e 25 e servira para logo em seguida, na linha 23 verificarmos se esse vértice já não foi visitado anteriormente. Caso já tenha sido, significa que encontramos um outro caminho de arestas até ele, porém de maior ou igual distancia, logo não nos interessa.

3.3 Variações do Algoritmo

O algoritmo 3 mostrado anteriormente nos resolve um dos problemas solicitados que é o de gerar o numero de bacon de um ator qualquer, porém aplicando pequenas adaptações podemos re-utilizar ele para resolver os outros problemas que abrangem esse artigo.

O segundo problema pede que informemos o nome do ator, ou atores que possuem o maior numero de bacon. Para fazer isso simplesmente percorremos o grafo até os vértices que possuem a maior distancia de bacon e salvamos eles em uma lista.

Para sabermos se chegamos na maior distancia, basta verificarmos se há Vértices ainda não visitados na *distancia + 1*, caso haja prosseguimos até chegar no laço em que não ira ter nenhum vértice na *distancia + 1*. Em pseudocódigo fica assim:

Algoritmo 4 Função Localizar Atores na Distancia máxima

```
1:  Método maiorNbacon()
2:     Vertice origem ← LocalizaVertice("Kevin Bacon")
3:     limparVertVisitados()
4:     listaAtores ← nova Lista Vazia
5:     Lista<Vertice> ListaAtual ← nova Lista Vazia
6:     Lista<Vertice> ListaAuxiliar ← nova Lista Vazia
7:     filaAtual.adiciona(origem)
8:     origem.Visitado ← verdadeiro
9:     res ← Verdadeiro
10:       Enquanto (res = Verdadeiro) Faça
11:           contador ← 0
12:               Para cada Vertice v de ListaAtual Faça
```

```
13:                 v.Visitado ← verdadeiro
14:                         Para cada Aresta adj de v.listaAdj Faça
15:                             se (adj.Vertice não foi visitado) então
16:                                 ListaAuxiliar.adiciona(adj.Vertice)
17:                                 contador ← contador + 1
18:                                 adj.Vertice.Visitado ← verdadeiro
19:                             fim se
20:                         fim para
21:             fim para
22:         se (contador = 0) então
23:                 Para cada Vértice v de listaAtual Faça
24:                         listaAtores.adiciona(v.nomeAtor)
25:                 fim para
26:             res ← falso
27:         fim se
28:         ListaAtual ← ListaAuxiliar
29:         ListaAuxiliar ← nova Lista Vazia
30:     fim enquanto
31:  retorna listaAtores
32: fim método
```

Basicamente controlamos com um contador se chegamos no final do Grafo, a linha 10 vai acumulando o contador para cada vértice em uma *distancia + 1* encontrada, caso não encontre ninguém fica em 0 e cai na situação da linha 15 onde pegamos a lista contendo os atores do fim do grafo e matamos a estrutura de repetição.

O terceiro problema é: Apresentar o número de atores que possuem um determinado número de Bacon. Essa questão se resolve com um algoritmo semelhante ao *Algoritmo 3* . A única diferença é que ao invés de somente retornarmos o numero referente a distancia de um ator, retornamos a quantidade de atores que estão na distancia solicitada por parâmetro (Tamanho da *listaAtual* na distancia em questão). Em pseudocódigo fica:

Algoritmo 5 Função Geradora de Atores a uma distancia X
Requer: distancia >= 0

```
1:  Método atoresDistancia(Inteiro distancia)
2:      atual ← 0
3:      se (distancia = 0) então
4:          retorna 1
5:      fim se
6:      Vertice origem ← LocalizaVertice("Kevin Bacon")
7:      limparVertVisitados()
8:      Lista<Vertice> ListaAtual ← nova Lista Vazia
9:      Lista<Vertice> ListaAuxiliar ← nova Lista Vazia
10:     filaAtual.adiciona(origem)
```

```
11:   origem.Visitado ← verdadeiro
12:     Enquanto (ListaAtual <> Vazia) Faça
13:         se (ListaAtual não contem origem) então
14:             atual ← atual + 1
15:         fim se
16:         se (atual = distancia) então
17:              retorna tamanho da ListaAtual
18:         fim se
19:         Para cada Vertice v de ListaAtual Faça
20:             v.Visitado ← verdadeiro
21:                 Para cada Aresta adj de v.listaAdj Faça
22:                     se (adj.Vertice não foi visitado) então
23:                         ListaAuxiliar.adiciona(adj.Vertice)
24:                         adj.Vertice.Visitado ← verdadeiro
25:                     fim se
26:                  fim para
27:             fim para
28:         ListaAtual ← ListaAuxiliar
29:         ListaAuxiliar ← nova Lista Vazia
30:     fim enquanto
31:   retorna -1
32: fim método
```

Temos uma variável chamada *atual* que ira acumulando a cada loop até chegarmos na distancia solicitada, quando achamos simplesmente retornamos o tamanho da *ListaAtual.*

4 Resultados

Alguns casos de teste para o primeiro problema, numero de Bacon de um Ator qualquer:

Burt Reynolds: 2
Tom Hanks: 1
Vivean Gray: 4
Robert De Niro: 1
Jenna Elfman: 3
Peter McDonald: 3
Jim Henson: 4
Wendy Lyon: 4
Brad Renfro: 5
Linda Harrison: 4

Resultados do segundo problema, informar o nome do ator, ou atores que possuem o maior número de Bacon encontrado, são eles:

Richard Arlen, Clara Bow, Jobyna Ralston, Bernhard Goetzke, John F. Hamilton, Nita Naldi, John Longden, Charles Paton, Sara Allgood

Resultados do terceiro problema, apresentar o número de atores que possuem um determinado número de bacon (entre 1 e o maior valor possivel):

Nº de Bacon 1: 80
Nº de Bacon 2: 1173
Nº de Bacon 3: 3574
Nº de Bacon 4: 2057
Nº de Bacon 5: 485
Nº de Bacon 6: 90
Nº de Bacon 7: 25
Nº de Bacon 8: 9

Alguns testes informando a distancia (graus de separação) entre dois atores quaisquer:

Yvan Attal e Richard Johnson: 5
Telly Savalas e Tom Hanks: 2
Nestor Paiva e Vivean Gray: 6
Arthur Byron e Robert De Niro: 4
Jean Simmons e Bruce Vilanch: 4
Louise Lasser e Cynthia Rhodes: 3
Boyan Milushev e Henry Fonda: 3
Olivia Negron e Roy Roberts: 7
Laurie Metcalf e Arthur Lowe: 4
Alan Howard e John Belushi: 4

5 Conclusão

A busca em largura (BFS) se mostrou muito satisfatória para ajudar a resolver os problemas mostrados, pois as questões relacionadas ao número de Bacon nada mais são do que os Graus de separação entre dois vértices quaisquer em um grafo. Analisando o algoritmo de busca em BFS, percebemos que fazendo certas alterações mas mantendo seu estilo de busca conseguiríamos chegar aos resultados necessários. Com isso, não houve maiores dificuldades em desenvolver o algoritmo.

A criação da estrutura para criar o grafo inicialmente foi um problema, mas após analisarmos mais a fundo percebemos que criando uma estrutura baseada em HashMaps (chave,valor) conseguiríamos partindo dos três arquivos recebidos extrair as informações necessárias para montarmos o grafo.