Trabalho prático 1 – Projeto e Analise de Algoritmos

Mateus V. Souza

Departamento de Ciências da Computação – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos, 6627 – Prédio do ICEx – Pampulha, Belo Horizonte, Minas Gerais, Brasil CEP: 31270-90

mateus.vilela@dcc.ufmg.br

1. Descrição do problema

O propósito deste trabalho é dado um grafo em que cada nó é uma pessoa e as arestas representam que existe uma relação entre as duas pessoas, o arco ainda tem duas ponderações como o grau de amizade e a distância entre essas duas pessoas, é encontrar o sub-grafo que seja conexo e que possua o maior ValorResposta para a equação (1) onde E é o conjunto de arcos deste sub-grafo. A Figura.1 ilustra como é os nós e os arcos do grafo.

$$ValorResposta = \frac{\sum_{i=1}^{E} Amizade_{\ i}}{\sum_{i=1}^{E} Distancia_{\ i}} \quad (1)$$

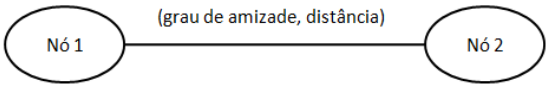

Figura 1. Ilustração dos nós e arcos do grafo

2. Estratégia de resolução do problema

O problema se demonstrou muito difícil no início, mas a solução foi pegar o grafo inicial e reponderar suas arestas com um Novo Peso através da equação (2) onde o ValorResposta é a resposta do ValorResposta do sub-grafo desejado.

$$NovoPeso\ = Amizade - Valor \times Ditância \quad (2)$$

Um melhor entendimento desta reponderação pode ser visto através das equações (3), (4), (5) e (6) onde se demonstra o principio da reponderação.

$$ValorResposta = \frac{\sum_{i=1}^{E} Amizade_{\ i}}{\sum_{i=1}^{E} Distancia_{\ i}} \quad (3)$$

$$ValorResposta \times \sum_{i=1}^{E} Distancia_i\ = \sum_{i=1}^{E} Amizade_i \quad (4)$$

$$0\ = \sum_{i=1}^{E} Amizade_i - ValorResposta \times \sum_{i=1}^{E} Distancia_i \quad (5)$$

$$0\ = \sum_{i=1}^{E} (Amizade_i - ValorResposta \times Distância_i) \quad (6)$$

Com isso se estipulava um valor para o ValorResposta e rodava o algoritmo Prim com os valores dos NovoPeso das arestas com um valor negativo. Este procedimento gerava uma árvore geradora máxima para o grafo (ao decorrer deste documento se ira mencionar que o algoritmo Prim ira gerar a arvore geradora máxima por pegar os arcos com seus valores de pesos invertido), se pegava os arcos dessa árvore geradora e os

arcos que não estavam na árvore geradora máxima e tem pesos positivos. Caso a soma destes arcos fosse zero se chegou no sub-grafo desejado. Caso a soma não desse zero se tentava um novo valor para ValorResposta e se repetia o procedimento.

A busca do ValorResposta se deu por uma busca binária onde se pegou o Lower Bound como zero e o Upper Bound como a soma de todos os valores das Amizades dos arcos no grafo inicial. Através da busca binária se busca encontrar o valor para ValorResposta entre o Lower Boun e o Upper Bound que faça o somatório dos NovosPesos da árvore geradora máxima mais os arcos com valor para o NovoPeso positivos e não contidos na árvore criada. Este ValorResposta que zera o somatório citado é a resposta para o problema.

Vale lembrar que se roda uma busca em profundidade (DFS) para todo grafo de entrada para ver se ele é conexo. Caso ele não for conexo ele não possui uma sub-grafo conexo para ser encontrado e a resposta retornada pelo programa é -1.000.

3. Programa criado

O primeiro desafio foi ler os dados. Foi criada a estratégia de pegar os dados entrados pela entrada e os colocar em um arquivo texto temporário que é criado no início do programa e destruído no final de sua execução. Estes dados contidos agora no arquivo texto temporário eram lidos e a cada grafo a ser analisado as estruturas criadas para armazenar o grafo eram preenchidas com os dados iniciais.

Irei analisar daqui para frente à execução apenas de um grafo. Par a um grafo as estruturas abaixo são iniciadas:

- Matriz de três dimensões: Número de nós do Grafo X Número de nós do grafo X 3. Pode se considerar que para cada grafo se tem três matrizes de Número de nós do Grafo X Número de nós do grafo. A primeira é uma matriz de adjacência do grafo para representar os arcos, segunda contendo o valor da amizade para cada arco e a terceira para a distância para cada arco.
- Um vetor com o tamanho do Número de nós do Grafo. Este é para conter os estados dos nós na realização da busca em profundidade.

Após esta etapa era realizada uma busca em profundidade para verifica se o grafo era conexo ou não. Caso o grafo fosse desconexo, o programa retornava o valor de -1.000 para o grafo. Caso contrário, ele seguia o procedimento do programa.

Caso o grafo for conexo, se iniciava as variáveis Lower Bound e Upper Bound como descritas acima para se realizar a busca binária (Lower Bound como zero e o Upper Bound como a soma de todos os valores das Amizades dos arcos no grafo inicial). Para cada ValorResposta se realizava a rotina de criação dos valores para os NovoPeso para cada arco:

- MatrizNovosPesos: Número de nós do Grafo X Número de nós do grafo. Esta guardava os novos valores para as arestas do grafo.

A matriz com os arcos do sub-grafo que se quer encontrar, que será citada mais abaixo neste documento, também é inicializada com todos os seus valores como zero.

Após está etapa se executa o algoritmo Prim para a geração da árvore geradora máxima. Para execução do Prim se utiliza as seguintes estruturas:

- Um vetor com o tamanho do Número de nós do Grafo para gravar os predecessores que iram indicar a árvore geradora máxima calculada.
- Um vetor com o tamanho do Número de nós do Grafo para ser utilizado com os valores de Chave que são utilizados no algoritmo Prim como pode ser encontrado em (Cormen, Leiserson, Rivest, & Stein, 2012).
- Uma estrutura para ser utilizada como uma lista (mas em meu programa utilizei um tipo vector para exercer esta função) que tem o tamanho inicial no Prim de Número de nós do Grafo e no fim do Prim ela se encontra vazia.

Após a execução do Prim. É executado um processo que completa a estrutura abaixo com os arcos da árvore geradora máxima gerada pelo Prim.

- Matriz com os arcos do sub-grafo que se quer encontrar: Número de nós do Grafo X Número de nós do grafo.

Além disso, se roda um procedimento para colocar na matriz acima os arcos de valor do NovoPeso positivos que não estão da árvore geradora máxima encontrada. Com está matriz com os arcos do sub-grafo que se quer encontrar se calcula o valor do somatório dos NovoPeso dos arcos neste sub-grafo e se guarda este valor da soma em uma variável criada.

Este valor da soma encontrada é usada na busca binária para se ver se o valor do ValorResposta testado até o momento é a resposta que se quer encontra. Caso o valor da soma for zero se para a busca binária, pois chegou no ValorResposta que se quer. Caso contrário se continua a busca binária atualizando o LowerBound e o Upper Bound.

Uma melhor visualização do algoritmo esta abaixo em pseudo-codigo:

Resposta(ValorResposta)

Repondera Pesos (ValorResposta);

Prim ();

Preenche Matrix Conectividade Do ValorResposta Com Arvore Geradora Maxima do Prim();

Preenche Matrix Conectividade Do ValorResposta Com Arcos Positivos restantes ();

Soma os Arcos da Matrix Conectividade Do ValorResposta e coloca na variável SomatorioDosArcos();

Fim

BuscaBinaria()

Calcula LowerBound e UpperBound();
Resposta(UpperBound)
Se (SomatorioDosArcos = 0)
    Resposta encontrada, retorna UpperBound;
Resposta(UpperBound)
Se (SomatorioDosArcos = 0)
    Resposta encontrada, retorna LowerBound;
Enquanto SomatorioDosArcos diferente de zero faz:
    Intermediario = ((UpperBound - LowerBound) / 2) + LowerBound;
    Resposta(Intermediario)
    Se (SomatorioDosArcos < 0)
        UpperBound = Intermediario;
    Se (SomatorioDosArcos < 0)
        LowerBound = Intermediario;
Fim enquanto;
Retorna Intermediario;

Fim

Algoritmo()

Le dados de um grafo e inicia estruturas do grafo lido();
Realiza busca em profundidade no grafo;
Verifica se grafo é conexo();
Caso for conexo, faz:
    BuscaBinaria();
Caso não for conexo, faz:
    Retorna -1.000;

Fim

4. Analises de complexidade

A complexidade da leitura dos dados foi de Θ(Númro de grafor lidos+Número de arcos lidos). Isto se deu pelo número de linhas que se tinha que ler da entrada.

A complexidade abaixo será calculada se analisando a entrada de apenas um grafo.

A complexidade de memória é de Θ(Número de Nós do grafo X Número de nós do grafo). Caso utilizarmos a notação de que V é igual ao Número de Nós do grafo (que iremos utilizar no restante do documento). Ela fica Θ(V*V). Isto se deve ao fato de a maior estrutura criada para o armazenamento de dados ser uma matriz V por V.

A complexidade de tempo será realizada em termos de atribuição e mudança do valor de variáveis.

No melhor caso só se roda a busca em profundidade e se verifica que o grafo é desconexo. Como a complexidade de se iniciar as matrizes de conectividade é O(V*V) e a da busca em profundidade é de O(V+E), onde E é o número total de arcos do grafo, a complexidade neste caso é de $O(V^2)$.

Nos outros casos onde ele é um grafo conexo se irá realizar a busca binária e o algoritmo Prim em todas as iterações da busca binária. Como o algoritmo Prim tem uma complexidade $O(V^2)$ que domina as outras operações dentro da busca binária, ela é tomada como base. No melhor caso da busca binária (O(1)) a complexidade fica como $O(V^2)$. No caso médio e pior caso da busca binária, sendo a complexidade da busca binária como O(UpperBound - LowerBound), a complexidade nesses casos fica como $O((UpperBound - LowerBound) * (V^2))$.

5. Implementação e Testes realizados

O programa foi implementado em C++ e o código está junto com está documentação. As entradas utilizadas para teste, InstaciaA.in e InstanciaB.in, que foram passadas na documentação e no grupo de discussão do trabalho, foram passadas junto com a documentação também.

Os testes foram realizados na maquina com Sistema Operacional Ubunto 14,04 LTS 64-bit, que possui 3,6 RAM, Intel Core i7-3517U CPU 1,90 GHz X 4.

O compilador usado foi g++ (Ubunto 4.8.2-19ubunto1) 4.8.2.

O resultado obtido para cada grafo em cada instância está na Tabela 1 e o resultado para os tempos de execução para as instâncias estão na Tabela 2. Os resultados dos tempos de execução foram à média do tempo de execução de 10 execuções do programa.

Tabela 1. Resultados obtidos

Instâncias	Grafo	Resultado
InstanciaA.in	Grafo 1	1.400
	Grafo 2	-1.000
	Grafo 3	0.635
InstanciaB.in	Grafo 1	1.009
	Grafo 2	0.001
	Grafo 3	1.345

Tabela 2. Tempos de execução

Instância	Tempo (segundos)
InstanciaA.in	0,0049
InstanciaB.in	0,0339

6. Observações importantes

O algoritmo pode ter um desempenho melhor caso se obtenha melhores UpperBound e LowerBound para os grafos.

A comprovação de certas complexidades não foram comprovadas a fundo, pois podem ser encontradas no livro (Cormen, Leiserson, Rivest, & Stein, 2012) e na internet. Também pelo fato para esta documentação não se tornar muito extensa.

O programa foi testado na maquina cipo.grad.dcc.ufmg.br atravez dos comendos ssh e scp no computador Porsche (150.164.11.38) que se encontra no laboratório LAPO da UFMG.

O maior UpperBound aceitável no meu programa (a soma de todas as amizades dos arcos do grafo) não pode ultrapassar o valor de 1.8e+307. Caso contrário, o programa ira falhar.

Os arquivos compile.sh e execute.sh estão juntos com está documentação.

Coloquei uma quebra de linha antes de imprimir as respostas para melhorar a visualização delas.

Utilizei largamente a estrutura Vector neste trabalho. Pois ainda não a tinha utilizado antes e gostaria de me familiarizar com sua utilização.

Caso a pessoa que corrigiu está documentação tenha alguma recomendação sobre como faze-lá melhor, ou o programa, fique a vontade para me enviar um e-mail para mateusvilelasouza@gmail.com com as recomendações. Isto se da pelo fato que sou engenheiro de produção e não estou acostumado a fazer documentação de programas.

7. Ressalva

Realizei este trabalho e o postei no dia 5 como estipulado. Mas estou o reenviando com está ressalva para ressaltar um ponto que só me foi descoberto ao discutir com colegas de PAA na terça, dia 7 de outubro. Eu fiz meu trabalho considerando que não se terá entradas com as mesmas pessoas com níveis de amizade e distancia distintos. Pois considerei que o nível de amizade e distância não varia de X para Y e de Y para X. Com isso não tratei caso se tiver duas entradas com as mesmas pessoas na mesma linha. Meu programa sempre ira considerar a ultima relação entre os pares passada na entrada.

Exemplo:

3 4 3 1

4 3 2 1

Ou

3 4 3 1

3 4 2 1

Isto se deu pela descrição do problema pela parte no documento da descrição que fala: " The first line defines the number of users in the network. The following lines represent pairs of friends and consists of 4 integers separeted by spaces, where the first two numbers are indentifiers of two users that are friends, the third number defines the level of friendship and the fourth the distance between them.". Que no meu entendimento, um par X e Y é o mesmo caso ele for apresentado como X e depois Y ou Y e depois X. Um exemplo é você pegar um par de pessoas. Caso eu te apresentar a você como João e depois Jose, será o mesmo par caso eu te apresentar como Jose e depois João, o par sempre será João e José a meu ver.

E não considero que se entre com o mesmo par duas vezes, pois não se pode ter duas relações de amizade com a mesma pessoa. É ilógico alguém ser muito amigo e ao mesmo tempo pouco amigo da mesma pessoa no meu entendimento. Ou estar perto e longe neste contexto da mesma pessoa.

Com isso considerei a terceira entrada na documentação como invalida e a atualizei no meu trabalho para:

4

1 2 1 10

2 3 3 3

2 4 3 2

3 4 3 1

Na qual é qualificada como uma entrada valida para mim.

Caso tenha entradas com o mesmo par, ele só ira considerar a ultima entrada e por ventura ira gerar um resultado diferente caso ele considerasse 3 4 2 1 e 4 3 2 1 como pares diferentes.

Mencionei isto no fórum e o Hector Ignacio Azpurua Perez Imaz (que acredito ser um dos monitores) considerou essa minha interpretação valida e mencionou tentar nos testes não repetir pares nas instancias.

Eu poderia corrigir este problema. Mas teria que mudar toda estrutura de dados do problema e a tática de solução, mas estou fazendo o TP2 desta disciplina e estou sem tempo para isso. Logo espero que levem está consideração ao corrigirem meu TP1.

8. Conclusão

Este trabalho se mostrou bem desafiador. Principalmente na área de leitura dos dados e no desenvolvimento de uma tática de resolução do problema. O problema se demonstrou muito interessante e a solução proposta se verificou eficaz.

Bibliografia

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2012). *Algoritmos - teoria e Prática.* Rio de Janeiro: Campus.