MO417 – Complexidade de Algoritmos

Nathana Facion - RA 191079 - Lista 9

01/06/2017

Questão 1.

O algoritmo tem suas arestas ordenadas de acordo com o seu custo, Essa ordenação é não decrescente. Então, seja G um grafo, seja E o conjunto de arestas, E = $\{e_1, e_2 ... e_k\}$. Seja C os custos dessas arestas, C = $\{c_e1, c_e2 ... c_ek\}$. Seja T a árvore geradora mínima de G.

Os custos dessas arestas então estão ordenados da seguinte forma:

$$c_{e1} \leq c_{e2} \leq ... \leq c_{ek}$$

Seja T' uma árvore geradora mínima com custo de c_{ei}^2 em cada aresta (i representa qualquer aresta).

Os custos dessas arestas devem se manter ordenados:

$$c_{e1}^2 \leq c_{e2}^2 \leq ... \leq c_{ek}^2$$

Vamos demonstrar que para qualquer 2 arestas em sequência seguida, a desigualdade deve continuar válida. Se ela continua válida então: a aresta com o maior custo, continua com o maior custo, a segunda aresta com maior custo continua com o segundo maior custo ... a de menor custo continua com o menor custo.

Então em T temos:

$$c_{ei} \leq c_{e(i+1)}$$

E em T' :

$$c_{ei}^2 \leq c_{e(i+1)}^2$$

Demonstração

Pegamos então a aresta original e multiplicamos os dois lados por c_{ei} :

$$c_{ei} * c_{ei} \leq c_{e(i+1)} * c_{ei}$$

Resultado :

$$c_{ei}^2 \leq c_{e(i+1)} * c_{ei}$$

Pegamos então a aresta original e multiplicamos os dois lados por $c_{e(i+1)}$:

$$c_{ei} * c_{e(i+1)} \leq c_{e(i+1)} * c_{e(i+1)}$$

Resultado :

$$c_{ei} * c_{e(i+1)} \leq c^2_{e(i+1)}$$

Então podemos juntar os dois resultados anteriores:

$$c^2_{ei} \leq c_{e(i+1)} * c_{ei} \leq c^2_{e(i+1)}$$

Concluimos que :

$$c^2_{ei} \leq c^2_{e(i+1)}$$

Logo, T' ainda é uma árvore geradora de G mesmo com a alteração dos custos, já que as arestas aparecerão na mesma ordem.

Entretanto, suponha que a afirmação acima seja falsa, isto é, T' não é uma árvore geradora mínima de G. Sendo assim, G possui arestas u e v, tal que $u \in T'$ e $v \notin T'$ e $c^2(v) < c^2(u)$, logo poderíamos melhorar T' trocando u por v, formando uma árvore T'' que tem custo menor.

Mas então poderíamos realizar a mesma troca em T, já que se $c^2(v) < c^2(u)$ e, como demonstramos acima, $c(v) < c(u)$, logo teríamos que T não é uma árvore gerador mínima de G, uma contradição. $\square$

Com isso provamos o porque concordamos com a afirmação do enunciado.

Questão 2.

Primeiro, rodamos o Kruskal Máximo para encontrar uma árvore geradora máxima para o grafo, isso é, encontraremos o caminho de maior velocidade entre os vértices. Com isso temos o retorno da variável A, que conterá as arestas da árvore geradora máxima. Em A aplicaremos o MergeSort, para ordenar da menor velocidade para a maior. Com isso, podemos substituir a quantidade f de arestas de menor velocidade por fibra, melhorando a velocidade da rede geral.

O Algoritmo não realiza a troca de arestas em G, apenas retorna as arestas que devem ser trocadas (variável arestasTrocar) . Poderíamos ter trocado as arestas de G por meio de um For também.

Algoritmo

Algorithm 1 Otimiza a rede para melhor conexão

```
function MELHORCONEXAO(f, G, w)
    // G é um grafo conexo (a rede)
    // f é a quantidade de fibra
    // w são as velocidades.
    A ← AGM − Kruskal(G, w)
    A ← MergeSort(A)                      ▷ Ordene do menor para maior
    arestasTrocar = []
    for i=0 até i = f do
        arestasTrocar.insere(A[i])
    // retorna a menor conexão e as arestas que devem ser trocadas
    devolva A[i + 1].w, arestasTrocar

function AGM − Kruskal(G, w)
    A ← ∅
    Ordene as arestas em ordem não crescente de peso
    for cada (u, v) ∈ E nessa ordem do
        if u e v estão em componentes distintos de (V, A) then
            A ← A ∪ (u, v)
    devolva A
```

Corretude

Kruskal Máximo: Seja G um grafo. A cada iteração, o grafo parcial formado pelos vértices de G e as arestas em T consistem de várias componentes conexas. Para construir componentes conexas de forma que cada vez fiquem maiores, examinam-se as arestas de G em ordem não crescente de velocidade. Se uma aresta junta dois nós em uma diferente componente conexa, adiciona-se ela a T e, consequentemente, as duas componente conexas transformam-se em uma. Caso contrário, a aresta é rejeitada, pois ela uniria dois nós da mesma componente conexa e não pode ser adicionada a T sem formar um ciclo. O algoritmo finaliza quando tivermos uma única componente conexa. Como as arestas são adicionadas das de maior velocidade para a de menor velocidade, as de maior sempre estarão na solução a menos que seja uma aresta rejeitada como explicado acima. Com isso temos as maiores velocidades na árvore e isso faz com que nossa conexão seja otimizada.

Algoritmo geral: Com o algoritmo de Kruskal Máximo, nós removemos as arestas de menor velocidade que estavam contida em um ciclo e retornamos então as arestas da árvore geradora máxima para a váriavel A. Sabemos que a árvore encontrada é a correta pois o algoritmo de Kruskal já foi provado em sala de aula. A única alteração é que a ordem de nossas arestas está da maior para a menor, não queremos uma árvore geradora mínima e sim máxima, por isso a alteração.

Após atribuirmos as arestas da árvore geradora maxima para a variável A, precisamos ordenar em ordem crescente, assim saberemos quais arestas contêm a menor velocidade.

Depois de ordenado, podemos escolher dois vértices u e v ligados por uma aresta (u, v), que tem a menor velocidade em A. Então podemos substituí-la por uma fibra que aumentará a velocidade entre u e v em A, e consequentemente em G. Isso faz com que qualquer caminho entre dois vértices que use a aresta (u, v) tenha sua velocidade melhorada. Como em uma árvore qualquer aresta é de corte, apenas a alteração desta mesma aresta melhora a conexão. Outra opção de escolha de aresta fará com que velocidade mínima da rede não mude.

Como recebemos como parâmetro uma quantidade f de fibra que pode ser usada para otimizar a rede, se trocarmos as arestas de menor velocidade da árvore por elas, teremos uma velocidade maior tanto na árvore T, quanto no grafo G. Logo, a nossa rede final terá uma maior velocidade entre qualquer duas filiais.