MC558 - Projeto e análise de algoritmos II - Unicamp

Laboratório 3 - Cabos

Erik Perillo, RA135582

16 de outubro de 2016

1 Abordagem

O desafio é determinar um modo de se conectar todos os pontos de comunicação com a mínima quantidade de cabos. A ideia é fazer uma conversão para um problema de se achar a árvore geradora mínima de um grafo cujas arestas representam um *link* entre dois pontos de conexão, com peso igual à distância entre eles. Após obtida a árvore, basta contar as distâncias de cabo trançado usadas e as distâncias de cabo de fibra.

O pseudo-algoritmo a seguir descreve de forma abstraída a ideia.

Algorithm 1

```
1:  procedure CABOS(points, fiber-thresh)
2:      for p1, p2 ∈ points : p1 ≠ p2 do
3:          (u, v).weight = dist(p1, p2)
4:          G = G ∪ (u, v)
5:      s ← any-vertex(G)
6:      π ← PRIM-ALG(G, s)
7:      fiber-cost ← 0
8:      normal-cost ← 0
9:      for u ∈ G : π[u] ≠ NULL do
10:         cost ← (π[u], u).weight
11:         if cost > fiber-thresh then
12:             fiber-cost ← fiber-cost + cost
13:         else
14:             normal-cost ← normal-cost + cost
        return (normal-cost, fiber-cost)
```

2 Complexidade

Seja V o número de vértices do grafo e E o número de arestas do mesmo.

As linhas 2 a 4 do algoritmo iteram sobre todas as combinações de pontos a fim de formar um grafo fortemente conexo. Assim, tem V^2 iterações. O algoritmo para se obter uma árvore geradora mínima é o algoritmo de *Prim* [1], na linha 6. Sua complexidade pode ser $O(E \log V)$, mas a biblioteca `STL` de `C++` não se dispõe do método `DECREASE-KEY` para `min-heaps`, o que fez com que se criasse uma nova `min-heap` a cada iteração sobre os vértices ainda na `min-heap`. Um vetor que guardava se certo vértice ainda estava na `min-heap` foi usado para ajudar nas construções. Isso não é tão grave, visto que a criação de uma `min-heap` leva tempo $O(V)$. Assim, a complexidade do algoritmo de *Prim* implementada foi: $O(V)$ chamadas a `build-min-heap`, mantendo as $O(V)$ chamadas a `extract-min` e $O(E)$ *updates* das *keys*. Assim, seu tempo total é $O(V^2 + V \log V + E)$. O *loop* das linhas 9-14 leva $O(V + E)$ (o termo E se devendo ao fato que há uma busca pelo peso). O tempo total então é:

$$O(V^2 + V^2 + V \log V + E + V + E) = O(V^2 + V \log V)$$

.

Referências

[1] Cormen et al. *Introduction to Algorithms, 3th ed.* The MIT Press, 2009, pp. 634–636.