AEDs III – Trabalho Prático 1

Discente...: José Flávio Lopes
Matrícula: 2019.1.08.045

Descrição do Problema

Dado um grafo ponderado como entrada, computar sua Árvore Geradora Mínima e, partir dela, calcular o subgrafo mais custoso dentre todos os subgrafos induzidos com três vértices.

Abordagem Utilizada

Para solucionar o problema, foi utilizada uma lista de adjacência para representar computacionalmente o grafo de entrada; o algoritmo de Kruskal foi utilizado para computar a Árvore Geradora Mínima do grafo; a MST em questão foi tratada como um grafo e, também, modelada computacionalmente com o uso de uma lista de adjacência. O algoritmo utilizado para calcular o subgrafo mais custoso de três vértices recebe a MST, modelada, como entrada.

A ideia principal do algoritmo é: para cada vértice V da MST, calcular todos subgrafos de três vértices que tenha V como vértice inicial.

Pseudocódigos

Algoritmo de Kruskal para o cálculo da MST:

```
MST-KRUSKAL(grafo):
        i = 0
        j = 0
        mst = novoGrafo()
        iniciaUnionFind(grafo)
        arestas = ordena(grafo->arestas)
        enquanto j < grafo->numeroVertices - 1 faca:
           aresta = arestas[i]
           findV1 = find(aresta->vertice1)
           findV2 = find(aresta->vertice2)
           se findV1 != findV2 entao:
                adicionaAresta(mst, aresta->vertice1, aresta->vertice2, aresta->peso)
                union(findV1, findV2)
                j = j + 1
           fimse
           i = i + 1
        fimenquanto

        retorna mst
```

Union-Find utilizado:

```
INITIALIZE-3 ( )
para cada vértice v
    chefe[v] := v
    alt[v] := 0
```

```
UNION-3 (r, s)   ▷ union-by-rank; r ≠ s
se alt[r] > alt[s]
    chefe[s] := r
senão chefe[r] := s
      se alt[r] = alt[s]
          alt[s] := alt[r] + 1
```

```
FIND-2 (v)
enquanto chefe[v] ≠ v
    v := chefe[v]
devolva v
```

Algoritmo para calcular o subgrafo mais custoso dentre todos o subgrafos de três vértices:

```
SubGrafoMaisCustoso(MST):
    maiorPeso = 0
    maiorV1
    maiorV2
    maiorV3

    para cada v em MST->V:
      para cada v2 em MST->Adj[v]:
        para cada v3 em MST->Adj[v2]:
          se v3 > v & peso(v, v2) + peso(v2, v3) > maiorPeso:
            maiorPeso = peso(v, v2) + peso(v2, v3)
            maiorV1 = v
            maiorV2 = v2
            maiorV3 = v3
          fimse
        fimpara
      fimpara
    fimpara
```

Resultado do Algoritmo para as instâncias:

Instância 1 (burma14.txt):

```
burma14.txt
4 5 6
5.40
```

Instância 2 (a28.txt):

```
a28.txt
6 5 275
31.24
```

Instância 3 (att48.txt):

```
att48.txt
25 9 34
2232.58
```

Instância 4 (berlin52.txt):

```
berlin52.txt
14 42 32
606.87
```

Instância 5 (eil101.txt):

```
eil101.txt
35 48 63
21.67
```

Instância 6 (bier127.txt):

```
bier127.txt
97 96 122
7739.53
```

Instância 7 (att532.txt):

```
att532.txt
397 431 470
1125.84
```

Instância 8 (brd14051.txt):

```
brd14051.txt
12493 13824 13854
1358.43
```

Instância 9 (d15112.txt):

```
d15112.txt
5370 13959 12535
1619.92
```

Conclusão

Após certa reflexão a cerca do trabalho, concluí que não fiquei satisfeito com o algoritmo desenvolvido, e estou certo de que há maneiras mais eficientes para solucionar o problema. Dentre as dificuldades que tive, a que mais me amarrou foi a complexidade de espaço que os grafos com muitos vértices demandavam da modelagem que minha solução utiliza; confesso que não explorei outra forma de modelagem, me limitando apenas à lista de adjacências. Ademais, estou feliz com o conhecimento adquirido, embora certo de que com mais esforço ele poderia ser ainda mais amplo.