Relatório da implementação do algoritmo de Prim e algoritmo de Kruskal

Átila Bernardo Mota Sousa

Departamento de Computação
Universidade Federal de Sergipe (UFS) – Aracaju, SE – Brasil

Resumo. *Este relatório descreve a implementação do algoritmo de Prim e algoritmo de Kruskal em Python e suas validações. Também são mostrados os testes com diferentes grafos e suas respectivas saídas.*

1. Informações Gerais

Tanto o algoritmo de Prim quanto o de Kruskal são utilizados para o mesmo objetivo, que é encontrar árvores geradoras de custo mínimo para um determinado grafo. Para essa implementação foi escolhida a linguagem Python e foram usados grafos apresentados nos slides de aula para validar os algoritmos. Além disso, como o algoritmo de Kruskal tem a necessidade de detectar ciclos, foi utilizado o código da *busca em profundidade (dfs)* da implementação 1 para detectá-los.

2. Resultados

2.1- Códigos em Python

2.1.1- Algoritmo de Prim

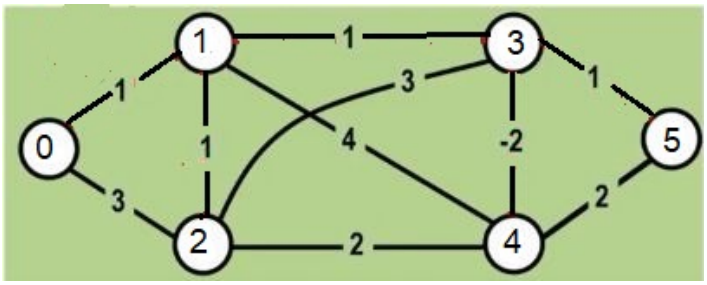
```python
def prim(self, i):
    T = []
    T.append(i)
    N = []
    for j in range(len(self.li)):
        N.append(j)
    N.remove(i)
    Tmin = []
    n = len(N)
    while len(T) != n + 1:
        min = float("inf")
        aresta = {}
        for j in T:
            for k in N:
                try:
                    for ar in self.li[j]:
                        if ar.x == k and ar.peso < min:
                            min = ar.peso
                            aresta = {'x': j, 'y': k, 'peso': ar.peso}
                except:
                    pass
        T.append(aresta['y'])
        N.remove(aresta['y'])
        Tmin.append(aresta)
```

2.1.2- Algoritmo de Kruskal

```python
def kruskal(self):
    H=[]
    arestas = []
    for vertice in range(len(self.li)):  #Construindo o vetor de arestas H
        for aresta in self.li[vertice]:
            aresta = {'x': vertice, 'y': aresta.x, "peso": aresta.peso}
            a1 = str(str(aresta['x']) + str(aresta['y']))
            a2 = str(str(aresta['y']) + str(aresta['x']))
            if str(a1) not in arestas and str(a2) not in arestas:
                H.append(aresta)
                arestas.append(str(a1))
    H = sorted(H, key = lambda x: x['peso']) #Ordenando crescente
    T = Grafo()
    for i in range(len(self.li)):
        T.insere_vertice()
    T.insere_aresta(H[0]['x'], H[0]['y'], H[0]['peso'])
    i=1
    j=0
    while j < len(self.li) - 2 and i < len(H) - 1:
        grafo_aux = deepcopy(T)
        grafo_aux.insere_aresta(H[i]['x'], H[i]['y'], H[i]['peso'])
        if grafo_aux.dfs2_recursivo(H[i]['x']) != -1:   #verificando a existencia de ciclos com o dfs
            T.insere_aresta(H[i]['x'], H[i]['y'], H[i]['peso'])
            j += 1
        i +=1
```

2.2- Grafos testados

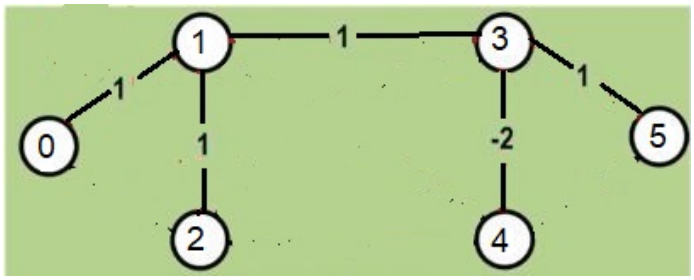
2.2.1- Grafo 1



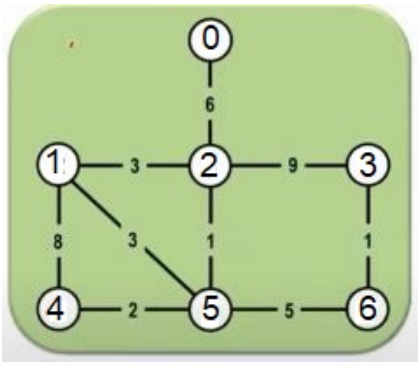
Saída:

```
Grafo 1:
0 ->  1 ->  2 ->  NULL
1 ->  0 ->  2 ->  3 ->  4 ->  NULL
2 ->  0 ->  1 ->  3 ->  4 ->  NULL
3 ->  1 ->  2 ->  4 ->  5 ->  NULL
4 ->  1 ->  2 ->  3 ->  5 ->  NULL
5 ->  3 ->  4 ->  NULL
Arvore minima do algoritmo de Prim:
0 ->  1 ->  NULL
1 ->  0 ->  2 ->  3 ->  NULL
2 ->  1 ->  NULL
3 ->  1 ->  4 ->  5 ->  NULL
4 ->  3 ->  NULL
5 ->  3 ->  NULL
Arvore minima do algoritmo de Kruskal:
0 ->  1 ->  NULL
1 ->  0 ->  2 ->  3 ->  NULL
2 ->  1 ->  NULL
3 ->  4 ->  1 ->  5 ->  NULL
4 ->  3 ->  NULL
5 ->  3 ->  NULL
```

Árvore geradora mínima:



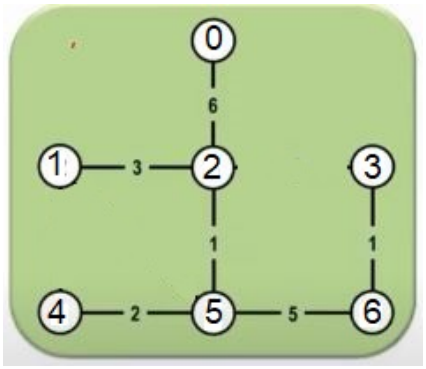
2.2.2- Grafo 2



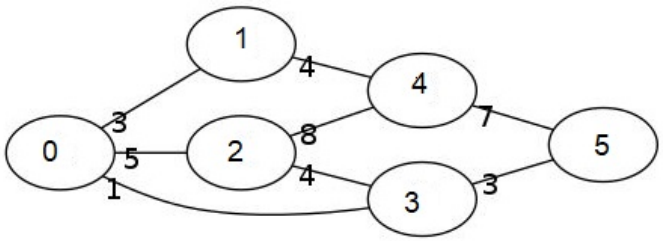
Saída:

```
Grafo 2:
0 ->  2 ->  NULL
1 ->  2 ->  4 ->  5 ->  NULL
2 ->  0 ->  1 ->  3 ->  5 ->  NULL
3 ->  2 ->  6 ->  NULL
4 ->  1 ->  5 ->  NULL
5 ->  1 ->  4 ->  6 ->  2 ->  NULL
6 ->  5 ->  3 ->  NULL
Arvore minima do algoritmo de Prim:
0 ->  2 ->  NULL
1 ->  2 ->  NULL
2 ->  0 ->  5 ->  1 ->  NULL
3 ->  6 ->  NULL
4 ->  5 ->  NULL
5 ->  2 ->  4 ->  6 ->  NULL
6 ->  5 ->  3 ->  NULL
Arvore minima do algoritmo de Kruskal:
0 ->  2 ->  NULL
1 ->  2 ->  NULL
2 ->  5 ->  1 ->  0 ->  NULL
3 ->  6 ->  NULL
4 ->  5 ->  NULL
5 ->  2 ->  4 ->  6 ->  NULL
6 ->  3 ->  5 ->  NULL
```

Árvore geradora mínima:



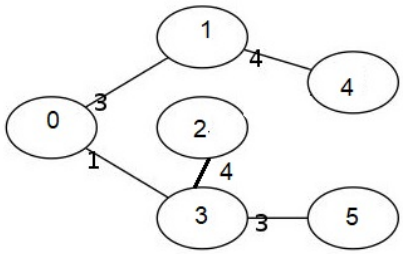
2.2.3- Grafo 3



Saída:

```
Grafo 3:
0 ->  1 ->  2 ->  3 ->  NULL
1 ->  0 ->  4 ->  NULL
2 ->  0 ->  4 ->  3 ->  NULL
3 ->  0 ->  2 ->  5 ->  NULL
4 ->  1 ->  2 ->  5 ->  NULL
5 ->  3 ->  4 ->  NULL
Arvore minima do algoritmo de Prim:
0 ->  3 ->  1 ->  NULL
1 ->  0 ->  4 ->  NULL
2 ->  3 ->  NULL
3 ->  0 ->  5 ->  2 ->  NULL
4 ->  1 ->  NULL
5 ->  3 ->  NULL
Arvore minima do algoritmo de Kruskal:
0 ->  3 ->  1 ->  NULL
1 ->  0 ->  4 ->  NULL
2 ->  3 ->  NULL
3 ->  0 ->  5 ->  2 ->  NULL
4 ->  1 ->  NULL
5 ->  3 ->  NULL
```

Árvore geradora mínima:



Referências

Vídeo-aulas disponíveis no youtube do professor Renê Gusmão do Departamento de Computação da UFS. Disponível em:

<https://www.youtube.com/watch?v=54oAIViZChA&list=PLFmoA27GocLc87HfGQtGfsQveHlUIErXc&index=2&ab_channel=Prof.Ren%C3%AAGusm%C3%A3oProf.Ren%C3%AAGusm%C3%A3o>