PERÍODO DE 2017.1
PROF. MARCUS VINICIUS S. POGGI DE ARAGÃO
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Estruturas Discretas - Segundo Trabalho

Gabriel Barbosa Diniz 1511211 | Lucas Rodrigues 1510848 | Mateus Ribeiro de Castro 1213068

2 DE JULHO DE 2017

Observação$_1$: Os códigos fontes dos algoritmos referentes aos teoremas provados seguirá em anexo em um arquivo Jupyter Notebook para melhor entendimento, compilação, execução, testes, etc.

1 PRIMEIRO TEOREMA

ENUNCIADO DO TEOREMA : Sabe-se encontrar a árvore de peso máximo de $G = (V, E)$ que contém o vértice 1 e possui K vértices.

- Denomina-se A_k a árvore obtida com certo valor de k.
- Denomina-se V_k e E_k as listas de vértices e arestas, respectivamente, que compõem a árvore A_k.

Caso Base: Provando por indução simples em K, temos para o caso base $k = 1$, e assim haverá somente o vértice 1 e nenhuma aresta; o peso total será 0. Esta é a única árvore possível de 1 vértice e que contém v_1. Está definida por $V_1 = \{v_1\}$ e $E_1 = \emptyset$.

Hipótese Indutiva: Pela hipótese indutiva, temos que o teorema é válido para k vértices e desejamos provar, portanto, que é válido também para $k+1$ vértices. Portanto, conhecemos V_k e E_k, e deseja-se determinar V_{k+1} e E_{k+1}.

Passo Indutivo: Considere o grafo B_k formado pelos vértices pertencentes a $V - V_k$ e por todas as arestas formadas por vértices $(b_1, b_2) \in (V - V_k)$. Considere o conjunto R de arestas do tipo (a, b) em que $a \in A_k$ e $b \in B_k$. Necessariamente, A_{k+1} tem seu conjunto de vértices

definido por $V_k \cup \{b\}$ e seu conjunto de arestas definido por $E_k \cup \{(a,b)\}$. Determinando a e b, portanto, determinamos inteiramente A_{k+1}, onde a e b são os vértices da aresta de maior peso entre as arestas R. Com isso, está determinado A_{k+1}.

Com isso então podemos, através da prova indutiva resolvida derivar um algoritmo genérico que corresponde a prova deste teorema. Segue abaixo então o algoritmo em ***python***:

```python
import numpy as np

################################################################################
#
# Algoritmo que recebe uma matriz de adjascencias com os pesos das arestas
#
# OBS: Ao inves de usar varios parametros, procure usar o retorno multiplo,
#      retornando o visited e a arvore. No caso, base tenta retornar uma matriz
#      vazia com o visited incluindo o primeiro vertice.
#
################################################################################

def HeavyTree(M,k):

    n = len(M)

    if(k > n or k < 0):
        print("K invalido")
        return -1

    # Caso Base
    if(k == 1):
        visited = [0]
        tree = np.zeros((n,n))
        return tree, visited

    tree, visited = HeavyTree(M,k-1)
    B = [] # Vertices vizinhos a vertices visitados

    for i in range(n):
        if(i not in visited): # Aqui temos a avaliacao de todos os vertices
            for j in visited: # nao visitados que sao vizinhos de vertices
                if(M[i,j] != 0 and j not in B): # visitados.
                    B = B + [i]
                    break

    biggest = 0
    new_v = 0
    old_v = 0

    for i in B:
        #new = np.max(M[i])
        for j in visited:
            if (M[i,j] > biggest):
                biggest = M[i,j]
```

```
                new_v = i
                old_v = j

    visited = visited + [new_v]

    #for i in range(n):
    #    if(M[new_v,i] == biggest):
    #        tree[i][new_v] = biggest
    #        tree[new_v][i] = biggest
    #        break

    tree[new_v][old_v] = biggest
    tree[old_v][new_v] = biggest

    return tree, visited
```

Listing 1: Python algorithm

OBSERVAÇÃO SOBRE O ALGORITMO REALIZADO : Por questão prática do algoritmo em si, podemos dizer que foi realizado como passo indutivo:

- Procura-se $max(T(k))$.
- Soma os pesos das arestas de maior peso de cada uma das folhas.
- Para todos os nós, adicionam-se as arestas de maior peso com os vértices restantes, ou seja, $max(p_bt.qb \in (V - V_k))$.

Dessa forma, saberemos que adicionamos as arestas de maior peso, e à medida que se aumenta k basta incluirmos mais arestas de maior peso.

Testes do Algoritmo: A tabela abaixo ilustra o tempo de execução do algoritmo para diferentes instâncias e com diferentes valores do parâmetro k. O tempo foi medido executando o algoritmo por 5 segundos e contando o número de execuções.

1.1 TEOREMA BÔNUS

ENUNCIADO DO TEOREMA BÔNUS : Sabe-se encontrar a floresta de peso mínimo de de $G = (V, E)$ onde os componentes conexos possuem pelo menos k vértices.

Caso Base: Por indução simples em k. Para o caso base $k = 1$, a floresta F_1 conterá todos os vértices de V, porém nenhuma aresta. Assim, haverá $|V|$ componentes conexas e a soma dos pesos será mínima.

Hipótese Indutiva: Pela hipótese indutiva, temos que o teorema é válido para k vértices e desejamos provar, portanto, que é válido também para $k+1$ vértices. Podemos definir componente conexo como qualquer árvore $A = (V', E')$ tal que $V' \subset V$, $E' \subset E$ e $|E'| > 0$. Pela hipótese indutiva, um componente conexo de F_k possuirá pelo menos k vértices.

Passo Indutivo: Sendo assim, o único modo de garantir que este componente passe a conter pelo menos $k+1$ vértices é adicionando um novo vértice a este componente. Então, enquanto houverem componentes conexos em F_{k+1} com número de vértices menores que $k+1$, devemos, do conjunto de arestas de G ainda não utilizadas em F_{k+1} (*i.e.,* $S_k = E - E_{k+1}$), para um componente conexo A de F_{k+1}, escolher a aresta de menor peso $s = (v_1, v_2) \in S_k$ tal que $v_1 \in A$ e $v_2 \notin A$. Após a inclusão dessa aresta, o conjunto de componentes conexos deve ser re-avaliado. Desta maneira, todo componente conexo contará com pelo menos $k+1$ vértices e, assim, obteremos F_{k+1}, provando o teorema.

O algoritmo derivado desta prova indutiva é conhecido como ***Algoritmo de Borůvka*** e é utilizado para se obter a Árvore Geradora Mínima de grafos ponderados cujos pesos das arestas são distintos. Segue abaixo o algoritmo em ***Python***:

```
from pygraph.classes.graph import graph
from pygraph.algorithms.accessibility import connected_components

def teo_2(g, k):

  # Salvaguarda
  if k > len(g.nodes()):
    raise ValueError('FORBIDDEN: K > |V|')

  if k <= 0:
    raise ValueError('FORBIDDEN: K <= 0')

  # Caso base
  if k == 1:
    forest = graph()

    for node in g.nodes():
      forest.add_node(node)

    return forest

```

```
  # Hipotese indutiva
  forest = teo_2(g, k-1)

  # Enquanto ainda houverem componentes conexos
  # que nao satisfazem a condicao

  while True:
    # Atualiza a lista de componentes, pois pode ter
    # mudado durante a adicao
    cc = _transform_cc(connected_components(forest))

    # Seleciona um que tenha comprimento < k
    selected_component = None
    for component in cc:
      if len(component) < k:
        selected_component = component
        break

    # Se nao conseguiu selecionar, significa que todos
    # satisfazem comprimento >= k, e podemos parar o while
    if selected_component == None:
      break

    # Caso haja um selecionado, selecionar a aresta de menor
    # peso que tenha somente um dos vertices em selected_component
    edges = g.edges()
    used_edges = forest.edges()
    unused_edges = [e for e in edges if e not in used_edges]
    neighbor_edges = [e for e in unused_edges if e[0] in selected_component]

    min_edge = min(neighbor_edges, key=lambda e: g.edge_weight(e))
    forest.add_edge(min_edge)

  return forest


def _transform_cc(cc):

  inv_map = {}

  for k, v in cc.iteritems():
    inv_map[v] = inv_map.get(v, [])
    inv_map[v].append(k)

  return inv_map.values()
```

Listing 2: Python algorithm

Testes do Algoritmo: A tabela abaixo ilustra o tempo de execução do algoritmo, em milissegundos, para diferentes instâncias e com diferentes valores do parâmetro k.

- O tempo foi medido executando o algoritmo por 5 segundos e contando o número de execuções.

OBSERVAÇÃO SOBRE OS TESTES : No anexo enviado estão figuras que ilustram o passo-a-passo dos vértices e arestas escolhidos em cada etapa do algoritmo quando aplicado na instância ulysses16.

entrada	$k=1$	$k=2$	$k=3$	$k=5$	$k=10$	$k=15$	$k=20$	$k=30$	$k=40$	$k=50$
ulysses16	0.01	1.41	1.99	2.31	3.35	3.64	–	–	–	–
ulysses22	0.01	3.09	4.66	6.36	23.57	24.95	25.02	–	–	–
bays29	0.01	7.37	11.36	14.04	21.32	21.58	21.86	–	–	–
eil51	0.02	59.49	98.04	148.13	163.11	170.35	173.74	175.71	175.82	183.72
eil76	0.03	284	393	539	610	770	818	855	861	865
bier127	0.06	1861	3127	3917	6735	8142	10210	10352	10996	13077