MST part of Lecture 10 : Thu 25th Aug 2016, EE-677-2016

Sachin B. Patkar

26th Aug 2016

1 Python script for Prim's algorithm for finding

Caution : This is a "first-cut" and "inefficient" implementation of Prim's algorithm for finding MST (minimum spanning tree).

For simplicity we also assume that the input graph is a connected graph.

Let us prepare the script. Let's name this function after Prim who invented this algorithm for computing MST (minimum spanning tree).

Within this function **PrimMST**, we will work with the list of nodes V. For simplicity we will assume it to be **V** = [0,1,2,....,N-1].

This greedy algorithm would maintain the set of nodes of the partial MST, using an indicator list, namely, **isNodeInPartialMst**.

Let $V(partialMST)$ denote the set of nodes of this partial MST, and let $E(partialMST)$ denote set of its edges.

Then $v \in V(partialMST)$ if and only if **isNodeInPartialMst[v] == True**.

Furthermore to represent $E(partialMST)$, we will use the idea that in each iteration, we add an edge to the $partialMST$, and this edge is between the newly added node, say **nextNode** and another node that was already in the $partialMST$. This already existing node in $partialMST$ is regarded as the **parent[nextNode]**.

In this manner, we represent $E(partialMST)$ with the help of the list **parent**, such that $(u, v) \in E(partialMST)$ if and only if **u == parent[v]** or **v == parent[u]** .

```
############## PrimMST_lec10_Thu25Aug16_ee677_2016.py ####################
```

```
MAX=99999999

def PrimMST ( neighbors , costOfEdges  ) :

  V = range ( len ( neighbors ) )

  numV = len( V )
  isNodeInPartialMst = [ False for v in V ]
  parent = [ None for v in V ]

  isNodeInPartialMst[0] = True

  for i in range( numV - 1 ) :

    nextNode, parentOfNextNode = \
           findNodeNotInPartialMstThatIsNearestToPartialMst (  \
               neighbors, costOfEdges, isNodeInPartialMst )
    if ( nextNode == None ) :
      raise Exception( "Graph is disconnected " )

    isNodeInPartialMst [ nextNode ] = True
    parent [ nextNode ] = parentOfNextNode

  return parent


def findNodeNotInPartialMstThatIsNearestToPartialMst \
    ( nbrs , cost, partialMstIndicator ) :
  V = range ( len ( nbrs ) )
  numV = len ( V )

  tmp = MAX
  bestNode = None
  parentOfBestNode = None
  for v in V :
    if ( partialMstIndicator [ v ] == False ) :
      for u in nbrs[ v ] :
        if (partialMstIndicator[u]==True) and (cost[v][u] <= tmp) :
            tmp = cost[v][u]
            bestNode = v
```

```
            parentOfBestNode = u
  return bestNode, parentOfBestNode


G_neighbors = [[1, 2, 3, 4, 5, 6, 7, 8, 9], \
  [0, 2, 3, 4, 5, 6, 7, 8, 9], \
  [0, 1, 3, 4, 5, 6, 7, 8, 9], \
  [0, 1, 2, 4, 5, 6, 7, 8, 9], \
  [0, 1, 2, 3, 5, 6, 7, 8, 9], \
  [0, 1, 2, 3, 4, 6, 7, 8, 9], \
  [0, 1, 2, 3, 4, 5, 7, 8, 9], \
  [0, 1, 2, 3, 4, 5, 6, 8, 9], \
  [0, 1, 2, 3, 4, 5, 6, 7, 9], \
  [0, 1, 2, 3, 4, 5, 6, 7, 8]]

cost_edges = [[0, 25, 65, 49, 55, 4, 59, 21, 98, 8], \
  [25, 0, 39, 27, 35, 3, 5, 21, 45, 89], \
  [65, 39, 0, 22, 77, 91, 93, 15, 8, 7], \
  [49, 27, 22, 0, 36, 89, 45, 13, 3, 2], \
  [55, 35, 77, 36, 0, 79, 78, 77, 15, 62], \
  [4, 3, 91, 89, 79, 0, 70, 64, 11, 99], \
  [59, 5, 93, 45, 78, 70, 0, 73, 41, 57], \
  [21, 21, 15, 13, 77, 64, 73, 0, 35, 20], \
  [98, 45, 8, 3, 15, 11, 41, 35, 0, 40], \
  [8, 89, 7, 2, 62, 99, 57, 20, 40, 0]]

print "parents of nodes in mst obtained by Prim's algo are \n", \
    PrimMST( G_neighbors, cost_edges )
#output : parents of nodes in mst obtained by Prim's algo are
#output : [None, 5, 9, 9, 8, 0, 1, 3, 3, 0]
```