19CSE302 Design and Analysis of Algorithms

Lab Sheet 8

S Abhishek

AM.EN.U4CSE19147

Colab

Prims Algorithm

```
from typing import List, Dict

class Node :

    def __init__(self, arg_id) :

        self._id = arg_id

class Graph :

    def __init__(self, source : int, adj_list : Dict[int, List[int]]):

        self.source = source

        self.adjlist = adj_list

    def PrimsMST (self) -> int:

        priority_queue = { Node(self.source) : 0 }

        added = [False] * len(self.adjlist)

        min_span_tree_cost = 0
        while priority_queue :

            node = min (priority_queue, key=priority_queue.get)

            cost = priority_queue[node]

            del priority_queue[node]

            if added[node._id] == False:

                min_span_tree_cost += cost

                added[node._id] = True

                print("Node : " + str(node._id) + ", Cost : "+str(min_span_tree_cost))

                for item in self.adjlist[node._id] :

                    adjnode = item[0]

                    adjcost = item[1]

                    if added[adjnode] == False :

                        priority_queue[Node(adjnode)] = adjcost

        return min_span_tree_cost

def main() :

    # Graph 1

    G1 = {}

    # G1[0] = [ (adjacent_node, cost) ]

    G1[0] = [ (1,1), (2,2), (3,1), (4,1), (5,2), (6,1) ]
    G1[1] = [ (0,1), (2,2), (6,2) ]
    G1[2] = [ (0,2), (1,2), (3,1) ]
    G1[3] = [ (0,1), (2,1), (4,2) ]
    G1[4] = [ (0,1), (3,2), (5,2) ]
    G1[5] = [ (0,2), (4,2), (6,1) ]
    G1[6] = [ (0,1), (2,2), (5,1) ]

    g1 = Graph(0, G1)

    cost = g1.PrimsMST()

    print("\nCost of the MST : " + str(cost) +"\n")
    # Graph 2

    G2 = {}

    # G2[0] = [ (adjacent_node, cost) ]

    G2[0] = [ (1,4), (2,1), (3,5) ]
    G2[1] = [ (0,4), (3,2), (4,3), (5,3) ]
    G2[2] = [ (0,1), (3,2), (4,8) ]
    G2[3] = [ (0,5), (1,2), (2,2), (4,1) ]
    G2[4] = [ (1,3), (2,8), (3,1), (5,3) ]
    G2[5] = [ (1,3), (4,3) ]

    g2 = Graph(0, G2)

    cost = g2.PrimsMST()

    print("\nCost of the MST : " + str(cost))


if __name__ == "__main__" :

    main()
```

```
Node : 0, Cost : 0
Node : 1, Cost : 1
Node : 3, Cost : 2
Node : 4, Cost : 3
Node : 6, Cost : 4
Node : 2, Cost : 5
Node : 5, Cost : 6

Cost of the MST : 6
```

```
Node : 0, Cost : 0
Node : 2, Cost : 1
Node : 3, Cost : 3
Node : 4, Cost : 4
Node : 1, Cost : 6
Node : 5, Cost : 9

Cost of the MST : 9
```

Kruskal's Algorithm

```
from collections import defaultdict

class Graph:

    def __init__(self, vertices):

        self.V=vertices

        self.graph=[]

    def addEdge(self, u, v, w):

        self.graph.append([u, v, w])

    def find(self, parent, i):

        if(parent[i]==i):

          return i

        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):

        xroot = self.find(parent, x)

        yroot = self.find(parent, y)

        if(rank[xroot] < rank[yroot]):

          parent[xroot] = yroot

        elif(rank[xroot] > rank[yroot]):

          parent[yroot] = xroot

        else:

          parent[yroot] = xroot

          rank[xroot] += 1

    def KruskalMST(self):

      result = []
      i = 0
      e = 0
      self.graph = sorted(self.graph, key=lambda item:item[2])
      parent, rank = [], []
      for node in range(self.V):
        parent.append(node)
        rank.append(0)
      while(e < self.V-1):
        u, v, w = self.graph[i]
        i += 1
        x = self.find(parent, u)
        y = self.find(parent, v)
        if(x != y):
          e+=1
          result.append([u, v, w])
          self.union(parent, rank, x, y)
      print("The Edges in the MST :", end = " ")
      for u, v, weight in result:
        print("(%d, %d) = %d" %(u ,v , weight), end = "    ")
      print()

if __name__ == "__main__" :
  # Graph 1
  g = Graph(5)
  g.addEdge(0, 1, 5)
  g.addEdge(0, 2, 13)
```

```
g.addEdge(0, 4, 15)
g.addEdge(1, 0, 5)
g.addEdge(1, 2, 10)
g.addEdge(1, 3, 8)
g.addEdge(2, 0, 13)
g.addEdge(2, 1, 10)
g.addEdge(2, 4, 20)
g.addEdge(2, 3, 6)
g.addEdge(3, 1, 8)
g.addEdge(3, 2, 6)
g.addEdge(4, 0, 15)
g.addEdge(4, 2, 20)

g.KruskalMST()

# Graph 2

g = Graph(4)

g.addEdge(0,1,10)
g.addEdge(0,2,6)
g.addEdge(0,3,5)
g.addEdge(1,3,15)
g.addEdge(2,3,4)

g.KruskalMST()
```

```
The Edges in the MST : (0, 1) = 5   (2, 3) = 6   (1, 3) = 8   (0, 4) = 15
The Edges in the MST : (2, 3) = 4   (0, 3) = 5   (0, 1) = 10
```

Thankyou!!