PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

KRUSKALS ALGORITHM :

Kruskal's algorithm is a greedy algorithm. Where every time searches for the minimum edge from the all the edges in the graph.

Algorithm:

Step 1:

Sort all the edges with respect to the weight of each edge in decreasing order

Step 2:

Pick the edge whose weight is minimum among the all the edges of the graph and if it forms a cycle then discard the edge and pick the next smallest edge

Step 3:

Repeat the steps util V-1 edges in spanning tree

Code:

```
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,a):
        if parent[a] == a:
            return a
        return self.find(parent, parent[a])
    def union(self, parent, rank,x,y):
        xadj = self.find(parent, x)
        yadj = self.find(parent, y)
        if rank[xadj]<rank[yadj]:
            parent[xadj] = yadj
        elif rank[xadj]>rank[yadj]:
            parent[yadj]=xadj
        else:
            parent[yadj] = xadj
            rank[xadj]+=1
    def kMST(self):
        mst_res = []
        se,fe=0,0 #se is used to iterate sorted edges,fe is used to iterate final edges of minimum spanning
        self.graph = sorted(self.graph,key=lambda item: item[2],reverse = False)
        p =[] #parent array
        r =[] #rank array
        for n in range(self.V):
            p.append(n)
            r.append(0)
        while fe< self.V-1:
            h=self.graph[se]
            u=h[0]
            v=h[1]
            w=h[2]
            se+=1
            x = self.find(p, u)
            y = self.find(p, v)
            if x!=y:
                fe+=1
                mst_res.append([u,v,w])
                self.union(p,r,x,y)
        min=0
        print("mst is:")
        for u, v, weight in mst_res:
            min+=weight
            print("%d -- %d == %d" %(u,v,weight))
        print("cost:"+str(min))
```

PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

Driver's code:

```
g = Graph(4)
g.addEdge(0,1,5)
g.addEdge(0,2,6)
g.addEdge(0,3,5)
g.addEdge(1,1,10)
g.addEdge(1,3,15)
g.addEdge(2,3,4)
g.addEdge(3,3,1)
g.kMST()
```

Output:

```
========== RESTART:
edge    weight
0---1   2
1---2   3
0---3   6
1---4   5
>>>
```

Time complexity:

Here firstly for sorting i used Timsort fuction in python which is on any case complexity is O(ElogE) or O(ElogV)

Time complexity varies with Input Size:

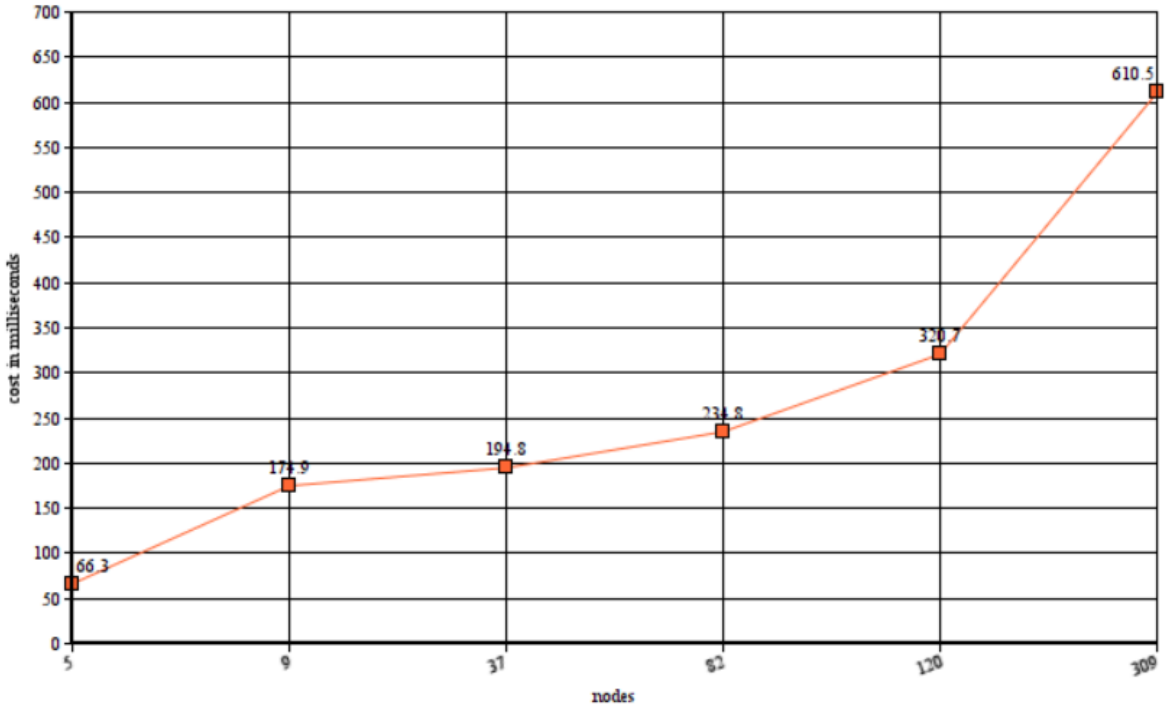
nodes	Time taken to process(in milli seconds)
• 5 nodes • 9 nodes • 37 nodes • 82 nodes • 120 nodes • 309 nodes	• 66.3 • 174.9 • 194.8 • 234.8 • 320.7 • 587.6

PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

Graphical representations :



PRIMS ALGORITHM :

Prim's algorithm also follows a greedy approach. It takes the edge whose weight is minimum from a vertex, and it does not form any cycle.

Approach:

Prim's algorithm is also using greedy approach

1. Remove all the loop edges and parallel edges
2. Choose any arbitrary node as root node and start from that node
3. Check all the outgoing edges and pick the edge with minimum cost
4. Do the process until all the nodes are included

PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

Algorithm:

def prims(self):

key ->initialize all the elements in the array with max value
Parent->initialize every element to none means not visted
key[0]->0 # which means making first node as rootnode
mst_set->initialize every element in the array with false
parent[0]=-1
for cout in range(self.v):
U->minimum value edge in the graph
mst_set[u] ->True #mark the node u is visited
for v in range(self.v):
if graph[u][v]>0 and mst_set[v] == False and key[v]>self.graph[u][v]:
key[v]=self.graph[u][v]
parent[v] = u

Code:

```
import sys
class Graph():
    def __init__(self,vertices):
        self.v=vertices
        self.graph=[[0 for column in range(vertices)]
                    for row in range(vertices)]
    def printMST(self,parent):
        print("edge  \tweight")
        for i in range(1,self.v):
            print(str(parent[i])+"---"+str(i)+"\t"+str(self.graph[i][parent[i]]))
    def minkey(self,key,mst_set):
        min=sys.maxsize
        for v in range(self.v):
            if key[v]<min and mst_set[v] == False:
                min = key[v]
                min_index=v
        return min_index
    def prims(self):
        key = [sys.maxsize]* self.v
        parent=[None]*self.v
        key[0]=0
        mst_set=[False]*self.v
        parent[0]=-1
        for cout in range(self.v):
            u=self.minkey(key,mst_set)
            mst_set[u] = True
            for v in range(self.v):
                if self.graph[u][v]>0 and mst_set[v] == False and key[v]>self.graph[u][v]:
                    key[v]=self.graph[u][v]
                    parent[v] = u
        self.printMST(parent)
g=Graph(5)
g.graph=[[0,2,0,6,0],
         [2,0,3,8,5],
         [0,3,0,0,7],
         [6,8,0,0,9],
         [0,5,7,9,0]]
g.prims()
```

PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

Time Complexity:

Time complexity of above algorithm is O(V^2) which is not good so we should implement another algorithm to implement it.

Method -2:

Step-1:

Create a minheap of size of v (vertices)every node in minheap contains vertex and its key value

Step-2:

Initialize first value is 0 and all the other vertices key value with infinity

Step-3:

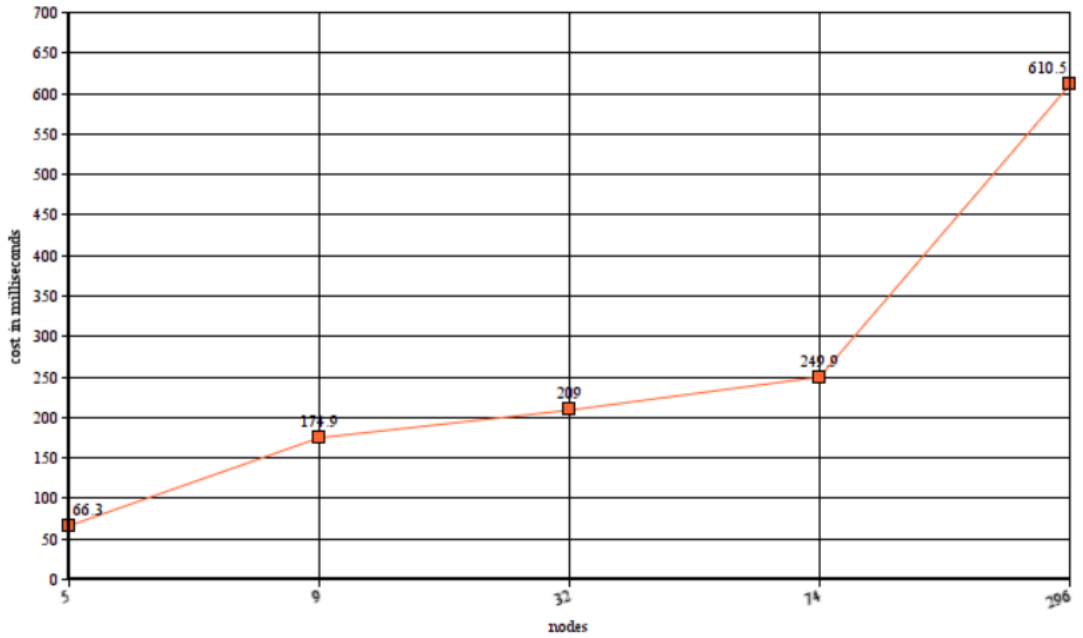
While minheap is empty

- Extract the minvalue from the heap
- Adjacent vertices of extracted min value node check adjacent vertices are presented in the min heap or not.if vertex is in minheap and key value of the node is greater than weight of the edge then update the key value of vertex with edge weight

Here the complexity is O(ElogV) which is faster than previous brute force method

Input nodes	Time taken to process in milliseconds
• 5 nodes • 9 nodes • 32 nodes • 74 nodes • 296 nodes	• 66.3 • 174.9 • 209 • 249.9 • 610.5

Graphical representation:



PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

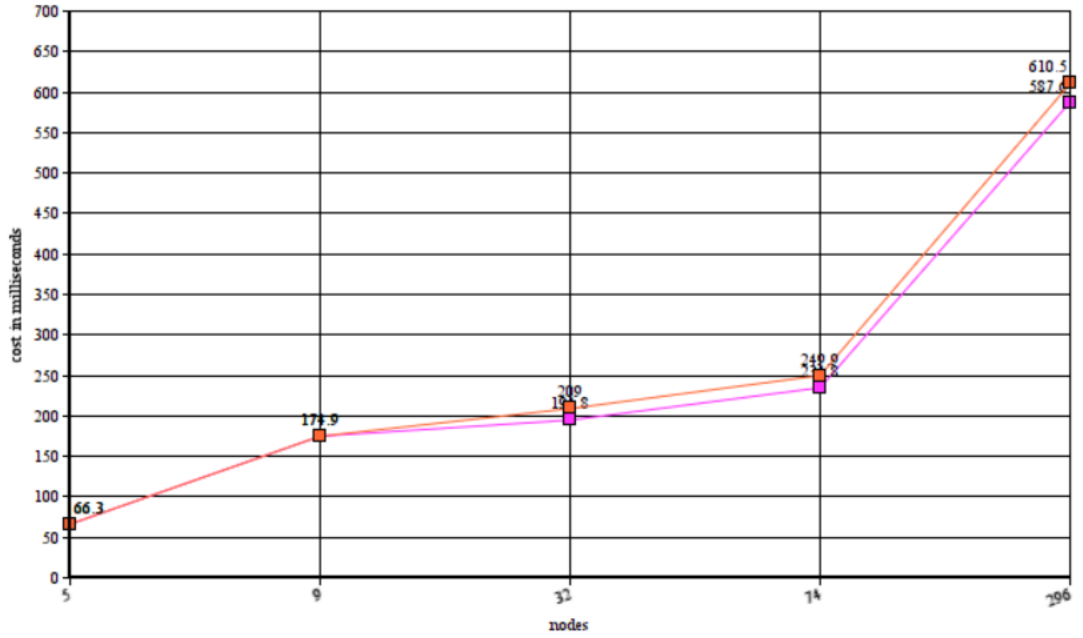
COMPARISION BETWEEN KRUSKALS AND PRIMS :

Kruskal's Algorithm	Prims Algorithm
• It starts with the edge which is carrying minimum weight • It visits a node not more than once • It works on disconnected graphs • It may result to produce disconnected graph • Time complexity is 0(ElogV) • It works faster on a sparse graph	• It starts with any vertex in the graph • It visits the node more than once • It works on only connected graphs • It always results in producing a connected graph • Time complexity is O(V^2) which can be reduced to O(ElogV)with fibanocci heaps • It works fast on dense graph

Real Time comparision of both algorithms:

Nodes	Time taken by Kruskal's	Time taken by Prims
5 nodes	66.3	66.3
9 nodes	174.9	174.9
32 nodes	194.8	209
74 nodes	234.8	249.9
296 nodes	587.6	610.5

Graphical comparision:



PROJECT REPORT:DESIGN AND ANALYSIS OF ALGORITHM

PRIM'S AND KRUSKAL'S ALGORITHM

CHANDU PAVAN BUDDA(1001977117)

Report:

We can improve prims algorithm performance by using different algorithms for picking minimum edge like priority queue which makes the algorithm much better in terms of time complexity while it may be bad in terms of space complexity, but we can modify it with different algorithms in future
In the case of Kruskal's algorithm we can use better sorting algorithms to increase the time complexity like heap sort whose time complexity O(nlogn) on any case except in worst case O (1). We also know that Tim sort algorithm which uses both merge sort and insertion sort to increase performance of the function.So, like we can improve the performance of the algorithm by merging different algorithms

Summary:

In this study, the time taken by the Kruskal's algorithm to give the minimum spanning tree containing 74 nodes in 234.8 whereas prims algorithm taken 249.9. Furthermore, if you took more than 500 nodes prims algorithm time increases more than the Kruskal's algorithm. Both the algorithm produces the same minimum spanning tree with same cost.

Supplementary Materials:

https://github.com/chandu000/Algorithms