THE UNIVERSITY OF LAHORE
ISLAMABAD CAMPUS

Artificial Intelligence (CS13217)

Lab Report

Name:	Sameea Naeem
Registration #:	CSU-XS16-139
Lab Report #:	07
Submitted To:	Sir. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 7
Implementation of Kruskal's algorithm

Objective
To implement the Kruskal's algorithm.

Software Tool
1. windows 10
2. sublime text
3. Python
4. Latex

1 Theory

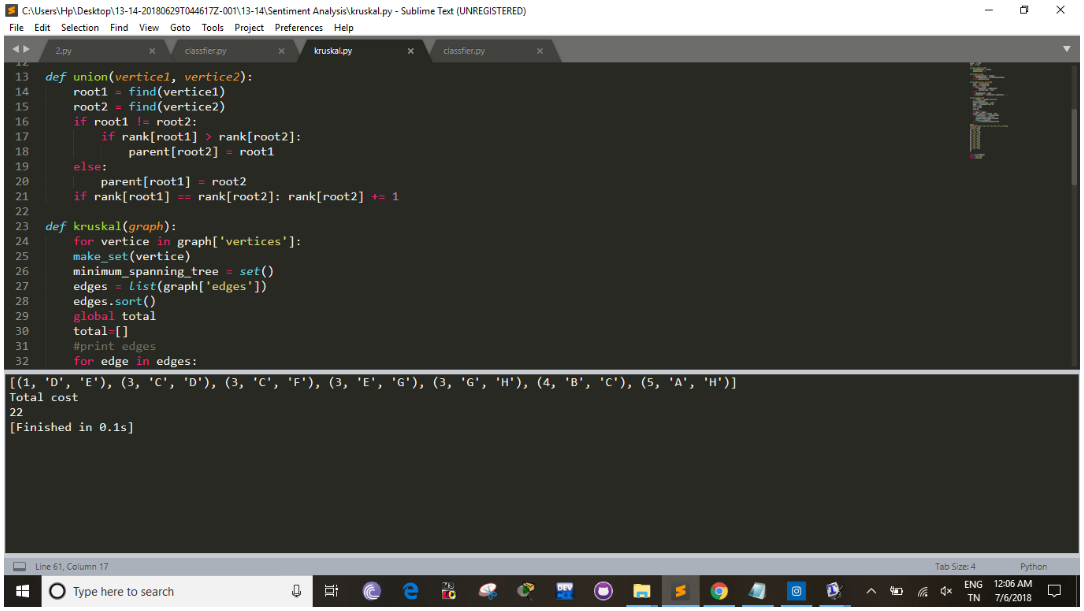
Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties. The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

2 Task

```
parent = dict()
rank = dict()

def make_set(vertice):
    parent[vertice] = vertice
    rank[vertice] = 0
```


C:\Users\Hp\Desktop\13-14-20180629T044617Z-001\13-14\Sentiment Analysis\kruskal.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2.py
classfier.py
kruskal.py
classfier.py
Line 61, Column 17
Tab Size: 4
Python
Type here to search
ENG
TN
12:06 AM
7/6/2018

```python
def union(vertice1, vertice2):
    root1 = find(vertice1)
    root2 = find(vertice2)
    if root1 != root2:
        if rank[root1] > rank[root2]:
            parent[root2] = root1
    else:
        parent[root1] = root2
    if rank[root1] == rank[root2]: rank[root2] += 1

def kruskal(graph):
    for vertice in graph['vertices']:
    make_set(vertice)
    minimum_spanning_tree = set()
    edges = list(graph['edges'])
    edges.sort()
    global total
    total=[]
    #print edges
    for edge in edges:
```

```
[(1, 'D', 'E'), (3, 'C', 'D'), (3, 'C', 'F'), (3, 'E', 'G'), (3, 'G', 'H'), (4, 'B', 'C'), (5, 'A', 'H')]
Total cost
22
[Finished in 0.1s]
```

Figure 1: Implementation of kruskals algorithm

```python
def find(vertice):
    if parent[vertice] != vertice:
        parent[vertice] = find(parent[vertice])
    return parent[vertice]

def union(vertice1, vertice2):
    root1 = find(vertice1)
    root2 = find(vertice2)
    if root1 != root2:
        if rank[root1] > rank[root2]:
            parent[root2] = root1
        else:
            parent[root1] = root2
        if rank[root1] == rank[root2]: rank[root2] += 1

def kruskal(graph):
    for vertice in graph['vertices']:
        make_set(vertice)
        minimum_spanning_tree = set()
        edges = list(graph['edges'])
        edges.sort()
        global total
        total=[]
        #print edges
    for edge in edges:
        weight, vertice1, vertice2 = edge
        if find(vertice1) != find(vertice2):
            union(vertice1, vertice2)
            minimum_spanning_tree.add(edge)
            total.append(weight)
    return sorted(minimum_spanning_tree)

graph = {
'vertices': ['A', 'B', 'C', 'D', 'E', 'F', 'G','H'],
'edges': set([
(8, 'A', 'B'),
(5, 'A', 'H'),
(10, 'A', 'F'),
(4, 'B', 'C'),
(4, 'B', 'F'),
(4, 'B', 'H'),
(4, 'B', 'E'),
(3, 'C', 'F'),
(3, 'C', 'D'),
(1, 'D', 'E'),
(6, 'D', 'F'),
(3, 'E', 'G'),
(3, 'G', 'H'),
])
}

print kruskal(graph)
print 'Total cost'
print sum(total)
```

3 Conclusion

Out put of algorithm is
(1, 'D', 'E'), (3, 'C', 'D'), (3, 'C', 'F'), (3, 'E', 'G'), (3, 'G', 'H'), (4, 'B', 'C'), (5, 'A', 'H')