Kruskal's Algorithm for MST

Mahesh Bharadwaj K - 185001089

February 11, 2020

Class Edge

```
import numpy as np

class edge(object):
    __slots__ = ['_edge']

    def __init__(self,l=-1,r=-1,w=-1):
        self._edge = (l,r,w)

    def __lt__(self,other):
        return self._edge[2] < other._edge[2]

    def __gt__(self,other):
        return self._edge[2] > other._edge[2]

    def getEdge(self):
        return self._edge

    def __str__(self):
        return str(self._edge)
```

Min Heap Implementation

```
class minheap(object):
    __slots__ = ['_size','_array']

    def __init__(self,firstOb):
        self._size = 0
        self._array = [firstOb]*20

    def __len__(self):
        return self._size

    def display(self):
        i = 1
        while i <= self._size:
            print(str(self._array[i]))
            i = i + 1

    def insert(self,newOb):
        self._size = self._size + 1
        i = self._size

        while self._array[i//2] > newOb:
            self._array[i] = self._array[i//2]
            i = i // 2

        self._array[i] = newOb

    def delete(self):
        if self._size == 0:
```

```
            return None

        min = self._array[1]
        last = self._array[self._size]
        self._size = self._size - 1

        i = 1
        while i * 2 <= self._size:
            child = i * 2

            if child != self._size and self._array[child + 1] < self._array[child]:
                child += 1

            if last > self._array[child]:
                self._array[i] = self._array[child]
            else:
                break

            i = child

        self._array[i] = last
        return min
```

Program

```
class disjoint_set(object):
    __slots__ = ['_arr','_size']

    def __init__(self):
        self._arr = np.array([-1 for i in range(100)])
        self._size = 0

    def initialise(self,size):
        assert(type(size) == int and size <= 100)
        self._size = size - 1

    def findClass(self,a):
        assert(type(a) == int)
        assert(a-1 <= self._size)

        a = a - 1
        while self._arr[a] > -1:
            a=self._arr[a]

        return a

    def merge(self,a,b):
        assert(type(a) == int and type(b) == int)
        assert(a-1 <= self._size and b-1 <= self._size)

        class_a = self.findClass(a)
        class_b = self.findClass(b)


        #Signs swapped since comparing negative numbers
        if self._arr[class_a] > self._arr[class_b]:
            self._arr[class_a] = class_b
        elif self._arr[class_a] < self._arr[class_b]:
            self._arr[class_b] = class_a
        else:
            self._arr[class_a] -= 1
            self._arr[class_b] = class_a

    def __str__(self):
        string = ''
```

```
        for x in range(self._size+1):
            string += ( str(self._arr[x]) + ' ' )
        return string




#ip = [ tuple(map(int,edge.split())) for edge in input().split('\n')]
#print(ip)

'''
n = 12
e = 20
edges = [(1,2,3),(1,3,5),(1,4,4),(2,5,3),(2,6,6),(3,4,2),(3,7,4),(4,5,1),(4,8,5),\
      (5,6,2),(5,9,4),(6,10,5),(7,8,3),(7,11,6),(8,9,6),(8,11,7),(9,10,3),(9,12,5)\
      (10,12,9),(11,12,8)]
'''

n = 7
e = 12

edges = [edge(1,2,2),edge(1,3,4),edge(1,4,1),edge(2,4,3),edge(2,5,10),edge(3,4,2),
    edge(3,6,5),          edge(4,5,7),edge(4,6,8),edge(4,7,4),edge(5,7,6),edge(6,7,1)]


heap = minheap(edge(-1,-1,-1))

for edge_obj in edges:
    heap.insert(edge_obj)

ds = disjoint_set()
ds.initialise(n)

selected = []
count=0
length=0

while len(heap):

    e = heap.delete().getEdge()

    class_l = ds.findClass(e[0])
    class_r = ds.findClass(e[1])

    if class_l != class_r:
        selected.append(e)
        ds.merge(e[0],e[1])
        count += 1
        length += e[2]
        if count == n - 1:
            break

print('The selected edges are: ',selected)
print('Length of the path is : ',length)
```