Kruskal's Algorithm for Minimum Spanning Tree Using Disjoint Sets

Mahesh Bharadwaj K - 185001089

February 12, 2020

Min Heap Implementation

Array Implementation of MinHeap to Fetch shortest edges first

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

    def __init__(self,firstOb):
        self._size = 0
        self._array = [firstOb]*1005

    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
```

Edge Class Declaration

The MinHeap uses the < and > operators and hence, a class for edge is implemented

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

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

    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)
```

Disjoint Set Implementation

Numpy arrays used to improve access times [approx O(1)]

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

    def __init__(self):
        self._arr = np.array([-1 for i in range(200)])
        self._size = 0

    def initialise(self, size):
        assert(type(size) == int and size <= 200)
        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 we are 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
```

Kruskal′s Program

*The edges of the graph are stored in **graph_ip.txt***

```
n = 100
e = 1000

with open('graph_ip.txt','r') as input_file:
    ip = input_file.read().split('\n')

    edges = []

    for i in range(e):
        edges.append(edge(tuple(map(int,ip[i].split(' ')))))


    heap = minheap(edge())

    for i in range(e):
        heap.insert(edges[i])

    ds = disjoint_set()
    ds.initialise(n)

    selected = []
    count = 0
    length = 0

    while len(heap):
        edgeTuple = heap.delete().getEdge()
        class_l = ds.findClass(edgeTuple[0])
        class_r = ds.findClass(edgeTuple[1])

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

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

"""
#Testing for Graph given in PPT

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())

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)

OUTPUT:
The selected edges are:  [(1, 4, 1), (6, 7, 1), (3, 4, 2), (1, 2, 2), (4, 7, 4), (5, 7,
    6)]
Length of the path is :  16

"""
```

Output

```
python3 Kruskals.py
```

```
The selected edges are:  [(48, 91, 7), (8, 98, 8), (23, 50, 29), (29, 33, 32), (13, 48, 43),
(2, 98, 61), (5, 55, 72), (6, 60, 86), (39, 86, 136), (4, 98, 137), (12, 58, 148),
(73, 83, 152), (11, 98, 154), (30, 46, 156), (60, 70, 177), (2, 94, 183), (16, 45, 188),
(28, 97, 190), (20, 58, 191), (60, 92, 199), (36, 62, 233), (22, 58, 237), (53, 89, 238),
(3, 45, 276), (40, 87, 278), (21, 73, 279), (5, 6, 302), (30, 36, 304), (54, 81, 318),
(68, 71, 326), (13, 61, 332), (7, 82, 339), (20, 99, 352), (57, 89, 361), (50, 84, 366),
(64, 82, 374), (23, 79, 390), (15, 28, 393), (75, 91, 395), (83, 91, 415), (13, 98, 420),
(3, 42, 423), (43, 81, 445), (36, 95, 456), (16, 87, 468), (48, 66, 472), (7, 87, 481),
(4, 12, 484), (48, 74, 485), (48, 71, 492), (40, 51, 496), (57, 73, 499), (26, 43, 501),
(10, 49, 509), (42, 84, 517), (1, 80, 522), (25, 31, 573), (79, 81, 585), (83, 85, 587),
(39, 52, 590), (64, 91, 606), (5, 9, 608), (34, 37, 652), (25, 42, 656), (34, 65, 663),
(1, 82, 675), (56, 90, 676), (39, 63, 714), (60, 94, 721), (30, 88, 729), (18, 86, 772),
(45, 47, 787), (36, 63, 788), (45, 77, 814), (17, 80, 820), (14, 100, 841), (69, 83, 842),
(19, 95, 878), (9, 86, 891), (20, 96, 912), (25, 35, 920), (24, 97, 943), (56, 93, 946),
(37, 66, 947), (89, 100, 968), (41, 98, 969), (15, 42, 1045), (54, 90, 1049), (15, 38, 1093),
(72, 78, 1155), (7, 59, 1183), (44, 64, 1218), (29, 52, 1219), (78, 92, 1307), (49, 82, 1333),
(67, 78, 1448), (32, 62, 1637), (27, 94, 2120), (25, 76, 3285)]
Length of the path is :  58692
```

```
cat graph_ip.txt | head
```

```
68 88 1599
21 28 8268
45 70 9041
```

```
34 90 1126
10 28 8492
42 91 2760
83 98 603
25 99 2097
19 90 2595
60 64 2858
```