Session 7: Dijkstra's Algorithm

Mahesh Bharadwaj K - 185001089

February 6, 2020

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
```

Dijkstra Code

```
from MinHeap import minheap
import math

class entry(object):
    __slots__ = ['_vertex','_dist','_prev']

    def __init__(self,vertex=None,dist=None,prev=None):
```

```
        self._vertex = vertex
        self._dist = dist
        self._prev = prev

    def __lt__(self,other):
        if self._dist < other._dist:
            return True
        return False

    def __gt__(self,other):
        if self._dist > other._dist:
            return True
        return False

    def __str__(self):
        return '|    {0:2d}    | {1:3d} |   {2:1s}    |'.format(chr(65 +self._vertex),self._dist,
    self._prev)


def print_path(v):
    global table
    global graph
    if v._prev == '-':
        return graph[v._vertex]['vertex']
    return print_path(table[int(v._prev)]) + ' --> ' + graph[v._vertex]['vertex']



graph = [{'vertex':'A','adj':[(1,3),(2,5),(3,4)]},{'vertex':'B','adj':[(0,3),(4,3),(5,6)]},{'
    vertex':'C','adj':[(0,5),(3,2),(6,4)]},\
        {'vertex':'D','adj':[(0,4),(2,2),(4,1),(7,5)]},{'vertex':'E','adj':[(1,3),(3,1),(5,2)
    ,(8,4)]},{'vertex':'F','adj':[(1,6),(4,2),(9,5)]},\
        {'vertex':'G','adj':[(2,4),(7,3),(10,6)]},{'vertex':'H','adj':[(3,5),(6,3),(8,6)
    ,(10,7)]},{'vertex':'I','adj':[(4,4),(7,6),(9,3),(11,5)]},\
        {'vertex':'J','adj':[(5,5),(8,6),(11,9)]},{'vertex':'K','adj':[(6,6),(7,7),(11,8)]},{'
    vertex':'L','adj':[(8,5),(9,9),(10,8)]}]

table = [None] * 12

for i in range(1,12):
    table[i] = entry(i,math.inf,str(' '))

#Source Vertex
table[0]=entry(0,0,str('-'))

#Min heap declared
vertex_heap = minheap(entry('*',-1,'-'))
vertex_heap.insert(table[0])

while len(vertex_heap):
    curr_vertex = vertex_heap.delete()
    adj = graph[curr_vertex._vertex]['adj']

    for edge in adj:
        if curr_vertex._dist + edge[1] < table[edge[0]]._dist:
            table[edge[0]]._dist = curr_vertex._dist + edge[1]
            table[edge[0]]._prev = str(curr_vertex._vertex)
            vertex_heap.insert(table[edge[0]])

print('+--------+-----+------+')
print('| Vertex | DIS | Prev |')
print('+--------+-----+------+')

for entry in table:
    print('|    {0:2s}    | {1:3d} |   {2:1s}    |'.format(graph[entry._vertex]['vertex'],entry.
    _dist,entry._prev))

print('+--------+-----+------+')

print('\n\nThe paths are:\n')
for entry in table:
    print(print_path(entry))
```

Output

Vertex	DIS	Prev
A	0	-
B	3	0
C	5	0
D	4	0
E	5	3
F	7	4
G	9	2
H	9	3
I	9	4
J	12	5
K	15	6
L	14	8

The paths are:

```
A
A --> B
A --> C
A --> D
A --> D --> E
A --> D --> E --> F
A --> C --> G
A --> D --> H
A --> D --> E --> I
A --> D --> E --> F --> J
A --> C --> G --> K
A --> D --> E --> I --> L
```
