Justin Li
4/13/23

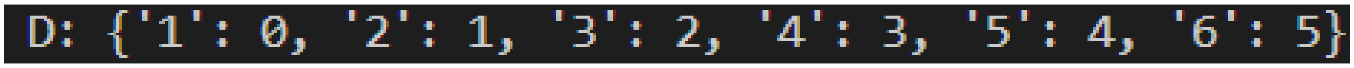
COEN 242 HW 1: Dijkstra's Algorithm

Code:

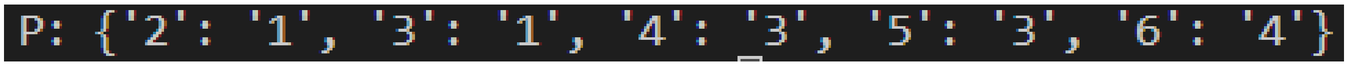
```
def Dijkstra(G, start, end=None):
    D = {}  # Distance vec
    P = {}  # Predecessor Vec
    Q = priorityDictionary()  # estimated distances of non-final vertices
    Q[start] = 0

    for v in Q:
        D[v] = Q[v]
        if v == end:
            break

        for x in G[v]:
            length = D[v] + G[v][x]
            if x in D:
                if length < D[x]:
                    raise ValueError("Already found better path to final
vertex")
            elif x not in Q or length < Q[x]:
                Q[x] = length
                P[x] = v

    return (f'D: {D}, P: {P}')

# Code for Priority Dictionary
# Source: https://code.activestate.com/recipes/117228/
class priorityDictionary(dict):
    def __init__(self):
        self.__heap = []
        dict.__init__(self)

    def smallest(self):
        if len(self) == 0:
            raise IndexError
        heap = self.__heap
```

```python
        while heap[0][1] not in self or self[heap[0][1]] != heap[0][0]:
            lastItem = heap.pop()
            insertionPoint = 0
            while 1:
                smallChild = 2*insertionPoint+1
                if smallChild+1 < len(heap) and \
                        heap[smallChild] > heap[smallChild+1]:
                    smallChild += 1
                if smallChild >= len(heap) or lastItem <=
heap[smallChild]:
                    heap[insertionPoint] = lastItem
                    break
                heap[insertionPoint] = heap[smallChild]
                insertionPoint = smallChild
        return heap[0][1]

    def __iter__(self):
        def iterfn():
            while len(self) > 0:
                x = self.smallest()
                yield x
                del self[x]
        return iterfn()

    def __setitem__(self,key,val):
        dict.__setitem__(self,key,val)
        heap = self.__heap
        if len(heap) > 2 * len(self):
            self.__heap = [(v,k) for k,v in self.iteritems()]
            self.__heap.sort()
        else:
            newPair = (val,key)
            insertionPoint = len(heap)
            heap.append(None)
            while insertionPoint > 0 and \
                    newPair < heap[(insertionPoint-1)//2]:
                heap[insertionPoint] = heap[(insertionPoint-1)//2]
                insertionPoint = (insertionPoint-1)//2
            heap[insertionPoint] = newPair
    def setdefault(self,key,val):
        if key not in self:
            self[key] = val
        return self[key]

# Output
G = {'1': {'2':1, '3':2},
    '2': {'3':5, '4':7},
    '3': {'4':1, '5':2},
    '4': {'6':2},
    '5': {'3':8,'4':3, '6':5},
    '6': {'4': 9}}

print(Dijkstra(G,'1'))
```

Output:



```
D: {'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5}
```



```
P: {'2': '1', '3': '1', '4': '3', '5': '3', '6': '4'}
```