Project of COMP6651

ANALYSIS OF PROJECT PROBLEM:

Problem 1:

Algorithm:

1. Dictionary D stores the input with
   meadow name : **keys** and [1]
   their neighbor and weights : **values**.
2. Set S contains capital letter meadows:
3. From s in S:
   Call shortestPath(D, s, "z"(destination node)) function (Dijikstra implemented)
   Compare the cost and store the smallest value
4. Print meadow name with shortest path and its cost

Time constraint:

The time complexity of input and the output is O(V + E).
Time complexity of the Dijkstra algorithm is O(E + V lgV) when using min heap. As the graph is not a simple graph, i.e. we have multiple edges.
In case all the vertices are caps, worst case analysis, it will run for V number of times.
Which gives us the complexity of :
T(n) = O(V+E) + V(O(E + VlogV))
In the worst case graph will be a complete graph i.e total edges E = V(V-1)/2, which is approx V^2.
Max value of P(edges) ~~ 10^4,
So max value of V ~~ 10^2.
Which gives us time complexity ~ O(V^2logV), running in less than 1 sec.

Memory constraint:

Python uses 28bytes for int(distance range given 1 to 1000), and 50bytes for string, and its memory behaviour is not fixed. So as per allowed by the TA, i have used C++ data size, which uses 1byte for char and 4byte for int

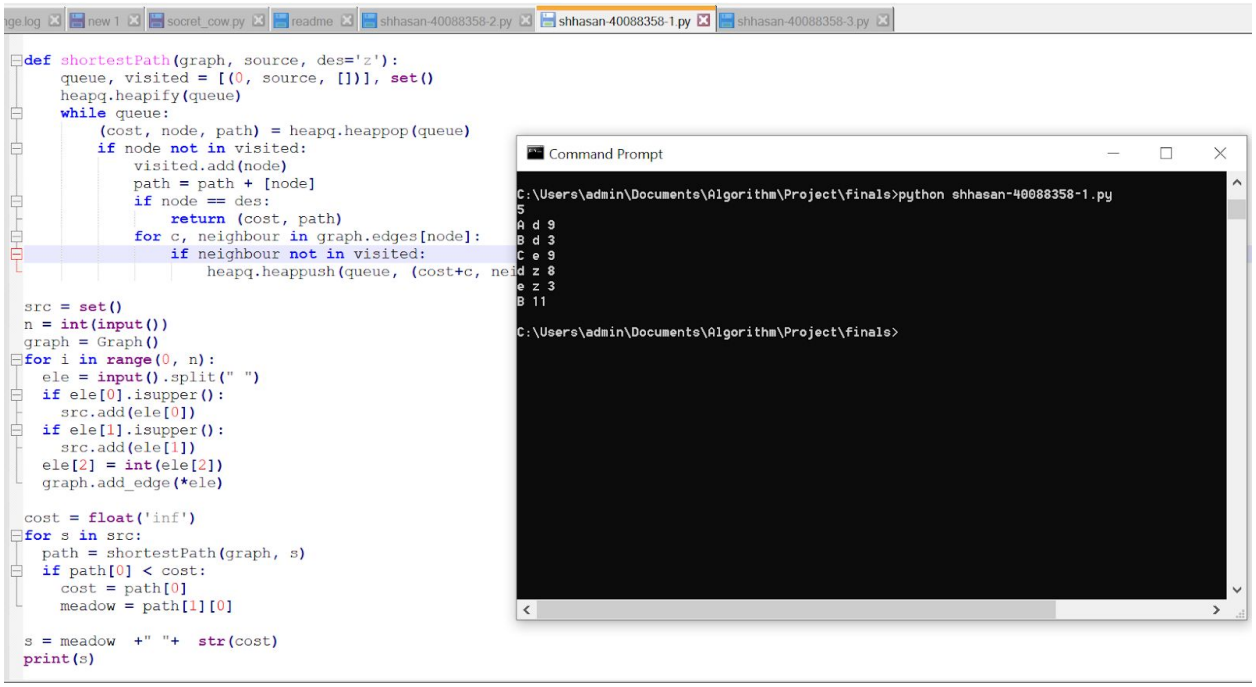
- Used set to store capital letter meadows..In total it can be (10,000*1)bytes ~ 9.7kB
- Other variables that are used to do the calculation and updation use almost negligible memory.

[1]

- For final graph that uses a dictionary to store undirected graph, its (2*10,000*4*1*1)bytes ~ 80kB

So in total it has used ~90kB, which is less than 256MB(the given limit)

Runtime screenshot:



Problem 3:

Algorithm:

1. Cust_list = Sorted(list of customers)
2. Counter = 0
3. While Cust_list :
    a. C = Get the first value which is min
    b. Q = get the queue number where C belong,
    c. C_Q = pop the first customer of Q
    d. If C_Q is same as C
        i. Remove C from cust_list
        ii. Remove C_Q from Q
    e. Else
        i. Remove C_Q from cust_list
    f. If patience number greater than counter
        i. Increase counter
    g. Else

i. Return the counter

Time constraint:

Step 1 took O(nlogn)

Step 2 takes O(1)

Step 3 runs for n number of time , taking O(n) time.

Giving the runtime as O(nlogn).

Given n <= 10^5, so nlogn will run in less than 1 sec time.

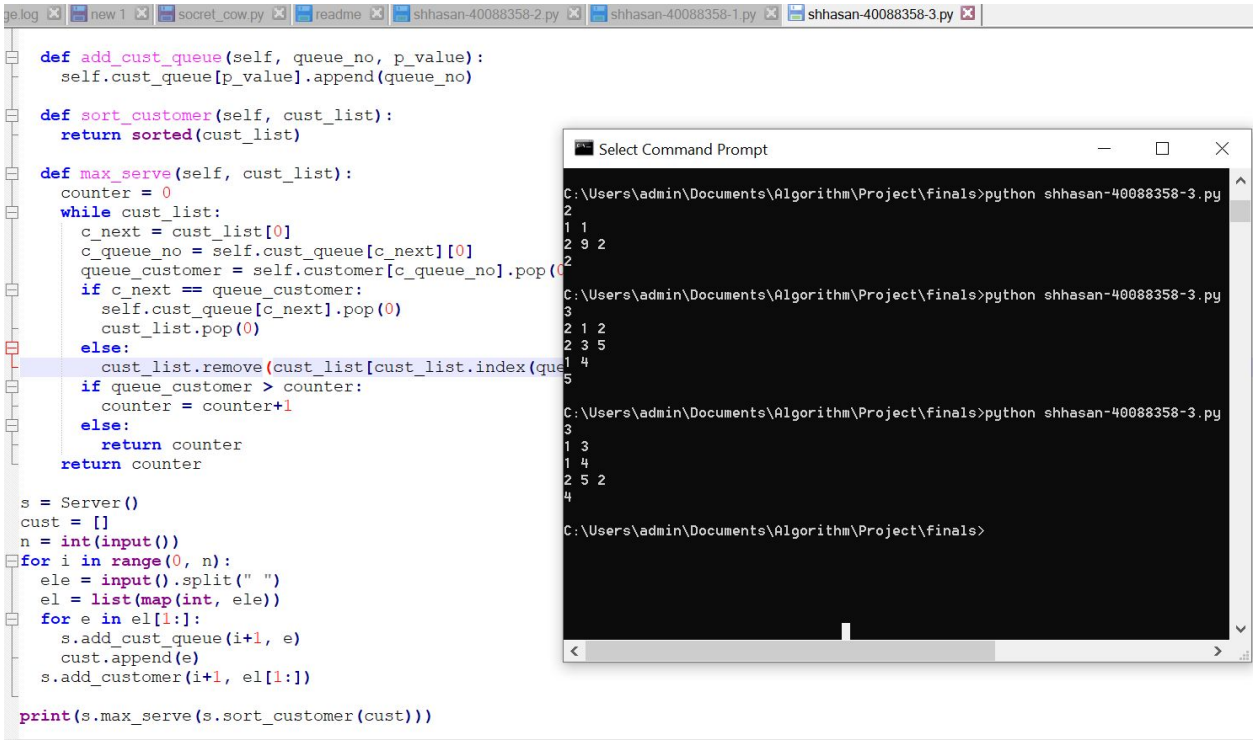
Memory constraint:

Python uses 28bytes for int(distance range given 1 to 1000), and 50bytes for string, and its memory behaviour is not fixed. So as per allowed by the TA, i have used C++ data size, which uses 1byte for char and 4byte for int

1. Used 2 dictionaries to store the value one is for customer with queue value, and another queue with customer values,so it took (2*10,000*4*4)bytes ~ 312kB
2. Used a list of customers that took (10,000*4)bytes ~ 40kB
3. Other variables that are used to do the calculation and updation use almost negligible memory.
4. Another sorted list of customers uses ~40kB.

So in total it took 400kB, which is less than 256MB

Runtime Screenshot:



```
def add_cust_queue(self, queue_no, p_value):
    self.cust_queue[p_value].append(queue_no)

def sort_customer(self, cust_list):
    return sorted(cust_list)

def max_serve(self, cust_list):
    counter = 0
    while cust_list:
        c_next = cust_list[0]
        c_queue_no = self.cust_queue[c_next][0]
        queue_customer = self.customer[c_queue_no].pop(0
        if c_next == queue_customer:
            self.cust_queue[c_next].pop(0)
            cust_list.pop(0)
        else:
            cust_list.remove(cust_list[cust_list.index(que
        if queue_customer > counter:
            counter = counter+1
        else:
            return counter
    return counter

s = Server()
cust = []
n = int(input())
for i in range(0, n):
    ele = input().split(" ")
    el = list(map(int, ele))
    for e in el[1:]:
        s.add_cust_queue(i+1, e)
        cust.append(e)
    s.add_customer(i+1, el[1:])

print(s.max_serve(s.sort_customer(cust)))
```

```
Select Command Prompt

C:\Users\admin\Documents\Algorithm\Project\finals>python shhasan-40088358-3.py
2
1 1
2 9 2
2

C:\Users\admin\Documents\Algorithm\Project\finals>python shhasan-40088358-3.py
3
2 1 2
2 3 5
1 4
5

C:\Users\admin\Documents\Algorithm\Project\finals>python shhasan-40088358-3.py
3
1 3
1 4
2 5 2
4

C:\Users\admin\Documents\Algorithm\Project\finals>
```

Problem 2:

Algorithm :

1. Add the edges in the graph, after doing the calculation to know which edges are adjacent.
2. Counter = 0
3. I = 1 to no. of vertices in the graph
    a. create subsets of length I.
    b. while subsets:
        i. S = subsets.pop(0)
            1. Create 2 different list with all vertices of S as A1 and all vertices of entire graph as A2
            2. Start removing vertices from A1 and its neighbor from A2
            3. if A2 becomes empty, then its a dominating set
            4. Increase the counter by 1
4. Return the counter

Time constraint:

Step 1 will take O(V+E) time, additional calculation took negligible time
Step 2 will take O(1)
Step 4 will take O(2^n)*n.
So the total run time will be O(n*2^n)

Memory constraint:
Python uses 28bytes for int(distance range given 1 to 1000), and 50bytes for string, and its memory behaviour is not fixed. So as per allowed by the TA, i have used C++ data size, which uses 1byte for char and 4byte for int.
No of intervals <= 5000

1. Used dictionary to store the graph data which took (2*5000*4) ~40kB
2. Other variables that are used to do the calculation and updation use almost negligible memory.
3. As per the requirement, 512MB storage is the peak storage which the algorithm will use, as I am not storing everything in a single shot but keeps on deleting the subsets of the previous level.
4. As we know ,

Number of subsets of size 0 = $^{n}C_0$

Number of subsets of size 1 = $^{n}C_1$

Number of subsets of size 2 = $^{n}C_2$

5.
6. For n input, we have max subsets at level n/2
7. So for 5000, we will have (5000/2 = 2500) level with max elements, number of subsets of size 2500 = 5000 C 2500 = 1.593718685 E+1503
8. So we require (2500*4*1.593718685 E+1503)bytes
9. This algorithm will stay in the memory limit for n=30-35. Taking 150-400MB

Runtime screenshots: