Assignment - 3

Name: Md. Minhazul Mowla

ID: 23201390

Section: 13

Course: CSE221

Submission Date: 11/5/25

Ans. to the Ques. No. - 1

Python

```
import heapq
def dijkstra(g, s):
    d = {}
    p = {}
    for i in g:
        d[i] = float('inf')
        p[i] = None
    d[s] = 0
    pq = []
    heapq.heappush(pq, (0, s))
    while len(pq) > 0:
        a, b = heapq.heappop(pq)
        if a > d[b]:
            continue
        for i in g[b]:
            w = g[b][i]
            j = a + w
            if j < d[i]:
                d[i] = j
                p[i] = b
                heapq.heappush(pq, (j, i))
    return d, p
```

```
def fun(g, x, y):
    d1 = dijkstra(g, x)
    d2 = dijkstra(g, y)
    min_d = float('inf')
    m = None
    for i in g:
        z = d1[i] + d2[i]
        if z < min_d:
            min_d = z
            m = i
    return m, min_d

city = {1: {2:10, 3:15}, 2: {1:10, 3:5, 4:20},
        3: {1:15, 2:5, 4:10}, 4: {2:20, 3:10, 50:5}
        50: {4:5}}

my_house = 1
benji_house = 50
meeting_point, total_distance = fun(city, my_house, benji_house)
print(meeting_point)
print(total_distance)
```

Ans. to the Ques. No. - 2

Python

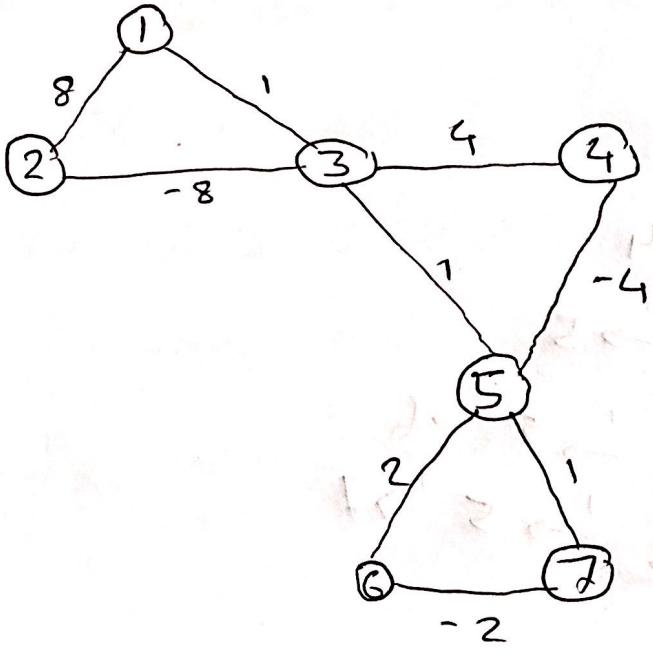
```
def kfc(g, kfc_loc, x, y):
    d1, r = dijkstra(g, x)
    ~~d2, s = dijkstra(g, y)~~
    rev = reverse-graph(g)
    d2, s = dijkstra(rev, y)
    min_d = float('inf')
    m = None
    for i in kfc_loc:
        z = d1[i] + d2[i]
        if z < min_d
            min_d = z
            m = i
    return m, min_d

def reverse-graph(g):
    rev = {}
    for i in g:
        rev[i] = {}
    for i in g:
        e = g[i]
        for j in e:
            w = e[j]
            rev[j][i] = w
    return rev
```

```
city = {1: {2: 10, 3: 15}, 2: {4: 20}, 3: {4: 10,
        5: 5}, 4: {50: 5}, 5: {50: 20}, 50: {}}

kfc_loc = [3, 4]
my_house = 1
benji_house = 50
ans, time = kfc(city, kfc_loc, my_house, benji_house)
print(ans)
print(time)
```

Ans. to the Que. No. - 3



Vertex	Cost	Parent
1	0	None
2	∞ 8	1
3	∞ 1	1
4	∞ 5	3
5	∞ 2	3
6	∞ 4	5
7	∞ 3	5

2: $1 \xrightarrow{8} 2$

3: $1 \xrightarrow{1} 3$

4: $1 \xrightarrow{1} 3 \xrightarrow{4} 4$

5: $1 \xrightarrow{1} 3 \xrightarrow{1} 5$

6: $1 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{2} 6$

7: $1 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{1} 7$

b

Dijkstra's algorithm fails to find the correct shortest paths in this graph because it assumes that once a vertex is extracted from the priority queue, its shortest path is finalized. This assumptions holds only when all edge weights are non-negative. In the graph, negative (example: (2, 3, -8)) allow for paths that can further reduce the distance of already processed vertices. For example: The correct shortest path to vertex 3 is $1 \rightarrow 2 \rightarrow 3$ with $8 + (-8) = 0$ But the algorithm incorrectly finalizes the distance to 3 as 2 (from $1 \rightarrow 3$) and ignores the cheaper path because vertex 3 was already ~~po~~ processed

c

Vertex	Cost	Parent
1	0	None
2	∞ 8	1
3	∞ ~~7~~ 0	~~7~~ 2
4	∞ ~~5~~ 4	3
5	∞ ~~2~~ ~~7~~ 0	~~3~~ 4
6	∞ ~~4~~ ~~3~~ 2	5
7	∞ ~~3~~ ~~2~~ ~~7~~ 0	~~5~~ 6

$q = [(0,1)]$ (d, v)

$q = [(1,3), (8,2)]$

$q = [(2,5), (5,4), (8,2)]$

$q = [(3,7), (4,6), (5,4), (8,2)]$

$q = [(4,6), (5,4), (8,2)]$

$q = [(2,7), (5,4), (8,2)]$

$q = [(5,4),(8,2)]$

$q = [(1,5),(8,2)]$

$q = [(3,6),(8,2)]$

$q = [(1,7),(8,2)]$

$I = [(8,2)]$

$q = [(0,3)]$

$q = [[4,4)]$

$I = [(0,5)]$

$q = [(2,6)]$

$q = [(0,7)]$

$q = []$

2 : $1 \xrightarrow{8} 2$

3 : $1 \xrightarrow{8} 2 \xrightarrow{-8} 3$

4 : $1 \xrightarrow{8} 2 \xrightarrow{-8} 3 \xrightarrow{4} 4$

5 : $1 \xrightarrow{8} 2 \xrightarrow{-8} 3 \xrightarrow{4} 4 \xrightarrow{-4} 5$

6 : $1 \xrightarrow{8} 2 \xrightarrow{-8} 3 \xrightarrow{4} 4 \xrightarrow{-4} 5 \xrightarrow{2} 6$

7 : $1 \xrightarrow{8} 2 \xrightarrow{-8} 3 \xrightarrow{4} 4 \xrightarrow{-4} 5 \xrightarrow{2} 6 \xrightarrow{-2} 7$

E = Edge
V = vertices

Python

```
import heapq
def dijkstra(g, s):
    dist = [float('inf')] * (n+1)  ... O(n)
    p = [-1] * (n+1)  ... O(n)
    dist[s] = 0
    q = [(0, s)]
    while q:
        d1, n = heapq.heappop(q)  ..... O(E log V)
        if d1 <= dist[n]
            for v, w in g[n]:  ..... O(E)
                if dist[v] > dist[n] + w:
                    dist[v] = dist[n] + w
                    p[v] = n
                    heapq.heappush(q,
                        (dist[v], v))
                    .....O(E log V)
    print(distance)

n, m = map(int, input().split())
g = [[] for i in range(n+1)]
for i in range(m):
    u, v, w = map(int, input().split())
    g[u] += [(v, w)]
dijkstra(g, 1)
```

Time complexity

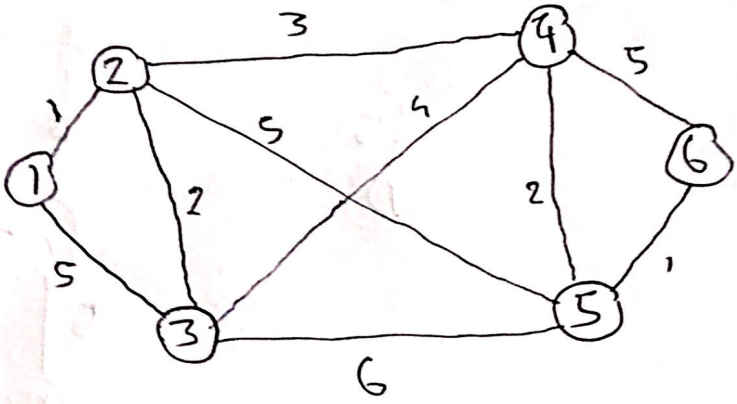
$= O(n + n + E \log V + E + E \log V)$

$= O(E \log V)$

$[\because E > V-1]$

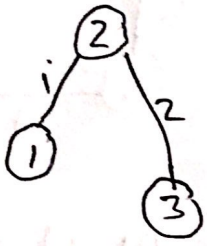
Ans. to the Ques No. 4

i.

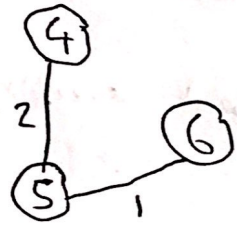


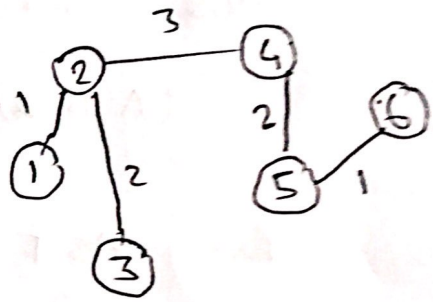
Ans 1

Mst_{left}



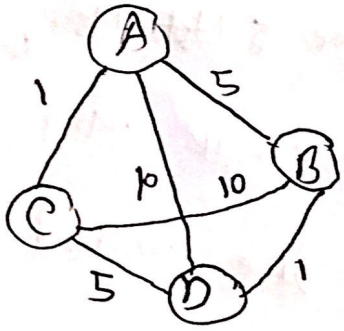
Mst_{right}



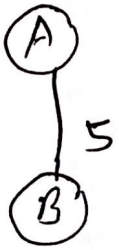


In the above example, the algorithm works for MST

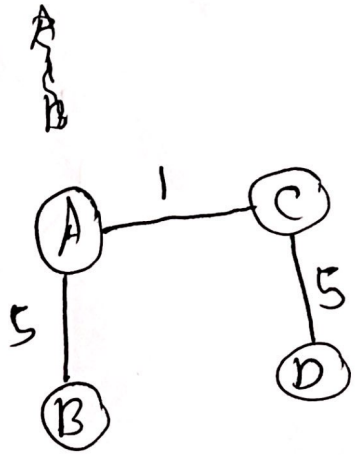
ii.



MST_{left} MST_{right}



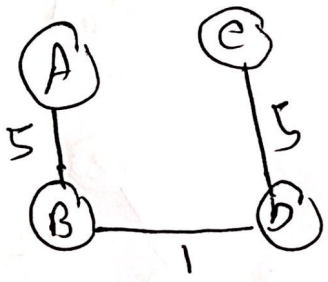
Ans 2



on

Here, final MST cost is 11.

$\{(A-B, 5), (C-D, 5), (A-C, 1)\}$



But MST is about avoiding heavy edges.

Ans should here been $= \{(A-C, 1), (B-D), 1), (A-B, 5)\}$

on $\{(A-C, 1), (B-D, 1), (C-D, 5)\}$.

The algorithm fails because ~~MST~~ MST_{left} and MST_{right} don't consider tighten overall MST. Thus, the algorithm won't always give connect answer.

[Proved]

Ans. to the Ques No-5

Python

```
from collections import deque
def fun(notes, k):
    notes = sorted(notes)
    d = [float('inf')] * (k+1)
    q = deque([0])
    c = [-1] * (k+1)
    while q:
        t = q.popleft()
        for i in notes:
            n = t + i
            if n <= k and d[n] > d[t] + 1:
                d[n] = d[t] + 1
                c[n] = i
                q += [n]
    if d[k] = float('inf'):
        return 'Impossible'
    p = []
    x = k
    while x > 0:
        y = c[x]
        p += [y]
        x -= y
    path.reverse()
    return f"{d[k]} notes added: {', '.join(map(str,p))}"
```

1. Breadth-first search(BFS) is the ~~best~~ relevant graph theory algorithm because we needed minimum number of notes to complete the target amount

2. Vertices: Each possible amount from 0 to the target amount(k) represents a vertex

Edges: A directed ~~eg~~ edge exists from vertex 'i' to vertex 'j' if there's a note with value 'v' such that j = i+v.
The weight of each edge is 1. ~~or what~~