Koushik Sahu
118CS0597
Artificial Intelligence Lab-II
8th Sept 2021, Wednesday

Problem 1:
Code:

```
import math, statistics, collections, copy

INF = 1e9+5

def take_input():
    print('Enter the number of cities: ', end='')
    n = int(input())
    print('Enter the number of edges: ', end='')
    m = int(input())
    g = list()
    for i in range(m):
        g.append([])
    for i in range(m):
        print('Enter src dest wt: ', end='')
        u, v, wt = map(int, input().split())
        g[u].append((v, wt))
    return n, m, g

res = INF

def dfs(u, seen, cost, g, curr_path, optimal_path):
    seen[u] = True
    global res
    curr_path.append(u)
    for v, wt in g[u]:
        if v==0:
            seen_all = True
            for i in seen:
                seen_all &= i
            if seen_all:
                present_path = copy.deepcopy(curr_path)
                present_path.append(0)
                print(f'path considered: {present_path} cost: {cost+wt}')
                if cost+wt < res:
                    res = cost+wt
                    optimal_path.clear()
                    for city in curr_path: optimal_path.append(city)
```

```
        else:
            if not seen[v]:
                child_res = dfs(v, seen, cost+wt, g, curr_path, optimal_path)
                if child_res < res:
                    res = child_res
    seen[u] = False
    curr_path.pop()
    return res

if __name__ == '__main__':
    n, m, g = take_input()
    cost = 0
    seen = [False]*n
    curr_path = list()
    optimal_path = list()
    print(f'****** Brute forcing all the paths ******')
    min_cost = dfs(0, seen, cost, g, curr_path, optimal_path)
    optimal_path.append(0)
    print(f'Minimum cost: {min_cost}')
    print(f'Optimal path: {optimal_path}')
```

Input:

```
5
20
0 1 1
0 2 6
0 3 8
0 4 4
1 0 7
1 2 8
1 3 5
1 4 6
2 0 6
2 1 8
2 3 9
2 4 7
3 0 8
3 1 5
3 2 9
3 4 8
4 0 4
4 1 6
4 2 7
4 3 8
```

Output:

```
****** Brute forcing all the paths ******
path considered: [0, 1, 2, 3, 4, 0] cost: 30
path considered: [0, 1, 2, 4, 3, 0] cost: 32
path considered: [0, 1, 3, 2, 4, 0] cost: 26
path considered: [0, 1, 3, 4, 2, 0] cost: 27
path considered: [0, 1, 4, 2, 3, 0] cost: 31
path considered: [0, 1, 4, 3, 2, 0] cost: 30
path considered: [0, 2, 1, 3, 4, 0] cost: 31
path considered: [0, 2, 1, 4, 3, 0] cost: 36
path considered: [0, 2, 3, 1, 4, 0] cost: 30
path considered: [0, 2, 3, 4, 1, 0] cost: 36
path considered: [0, 2, 4, 1, 3, 0] cost: 32
path considered: [0, 2, 4, 3, 1, 0] cost: 33
path considered: [0, 3, 1, 2, 4, 0] cost: 32
path considered: [0, 3, 1, 4, 2, 0] cost: 32
path considered: [0, 3, 2, 1, 4, 0] cost: 35
path considered: [0, 3, 2, 4, 1, 0] cost: 37
path considered: [0, 3, 4, 1, 2, 0] cost: 36
path considered: [0, 3, 4, 2, 1, 0] cost: 38
path considered: [0, 4, 1, 2, 3, 0] cost: 35
path considered: [0, 4, 1, 3, 2, 0] cost: 30
path considered: [0, 4, 2, 1, 3, 0] cost: 32
path considered: [0, 4, 2, 3, 1, 0] cost: 32
path considered: [0, 4, 3, 1, 2, 0] cost: 31
path considered: [0, 4, 3, 2, 1, 0] cost: 36
Minimum cost: 26
Optimal path: [0, 1, 3, 2, 4, 0]
```

Problem 2:
Code:

```
import math, statistics, collections, copy

INF = int(1e9+5)

def take_input():
    print('Enter the number of cities: ', end='')
    n = int(input())
    print('Enter the number of edges: ', end='')
    m = int(input())
    global INF
    g = [[INF for i in range(n)] for i in range(n)]
    for i in range(m):
        print('Enter src dest wt: ', end='')
        u, v, wt = map(int, input().split())
        g[u][v] = wt
    return n, m, g
```

```
def pretty_print(x):
    for i in range(len(x)):
        print(" ".join(str(x[i])))

class BranchBound:
    def __init__(self, n, g):
        self.n = n
        self.g = g

    def run(self, src):
        optimal_path = list()
        gl = list()
        cst, g = BranchBound.eval_cost(self.g)
        for i in range(self.n): g[i][0] = INF
        gl.append((cst, src, g))
        while len(gl) > 0:
            print('available cities at this level:')
            for i in gl:
                print(f'vertex: {i[1]} cost: {i[0]}')
            cst, u, g = min(gl, key=lambda x: x[0])
            print('city choosen:')
            print(f'vertex: {u} cost: {cst}')
            if cst < INF:
                optimal_path.append(u)
                gl.clear()
                for i in range(self.n):
                    edge_cst = g[u][i]
                    if edge_cst < INF:
                        new_g = BranchBound.add_edge(g, u, i)
                        cst1, new_g = BranchBound.eval_cost(new_g)
                        gl.append((cst+edge_cst+cst1, i, new_g))
        return optimal_path

    @staticmethod
    def eval_cost(g_in):
        g = copy.deepcopy(g_in)
        n = len(g)
        cst = 0
        for i in range(n):
            cst += min(g[i][:])
            cst += min(g[:][i])
        for i in range(n):
            mn = min(g[i][:])
```

```
            for j in range(n):
                if g[i][j] < INF:
                    g[i][j] -= mn
        for j in range(n):
            mn = min(g[:][j])
            for i in range(n):
                if g[i][j] < INF:
                    g[j][i] -= mn
        if cst >= INF: cst = 0
        return cst, g

    @staticmethod
    def add_edge(g_in, u, v):
        g = copy.deepcopy(g_in)
        n = len(g)
        for i in range(n):
            g[u][i] = INF
            g[i][v] = INF
        g[v][u] = INF
        return g

if __name__ == '__main__':
    n, m, g = take_input()
    bb = BranchBound(n, g)
    optimal_path = bb.run(0)
    optimal_path.append(0)
    print(f'Optimal path: {optimal_path}')
```

Input:

```
5
20
0 1 1
0 2 6
0 3 8
0 4 4
1 0 7
1 2 8
1 3 5
1 4 6
2 0 6
2 1 8
2 3 9
2 4 7
3 0 8
```

```
3 1 5
3 2 9
3 4 8
4 0 4
4 1 6
4 2 7
4 3 8
```

Output:

```
available cities at this level:
vertex: 0 cost: 42
city choosen:
vertex: 0 cost: 42
available cities at this level:
vertex: 1 cost: 42
vertex: 2 cost: 47
vertex: 3 cost: 49
vertex: 4 cost: 45
city choosen:
vertex: 1 cost: 42
available cities at this level:
vertex: 2 cost: 45
vertex: 3 cost: 42
vertex: 4 cost: 43
city choosen:
vertex: 3 cost: 42
available cities at this level:
vertex: 2 cost: 43
vertex: 4 cost: 42
city choosen:
vertex: 4 cost: 42
available cities at this level:
vertex: 2 cost: 42
city choosen:
vertex: 2 cost: 42
Optimal path: [0, 1, 3, 4, 2, 0]
```