In [10]:
```
r,c = map(int,input().split())
matrix =[]
for i in range(r):
    a=[]
    for j in range(c):
        a.append(int(input()))
    matrix.append(a)


for i in range(r):
    for j in range(c):
        print(matrix[i][j],end=" ")
```

```
2 3
1
2
3
4
5
6
1 2 3 4 5 6
```

In [12]:
```
from collections import deque

class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.graph = {i: [] for i in range(vertices)}

    def add_edge(self, u, v):
        self.graph[u].append(v)

    def bfs(self, start):
        visited = set()
        queue = deque([start])
        visited.add(start)
        print("BFS Traversal starting from node", start)
        while queue:
            vertex = queue.popleft()
            print(vertex, end=" ")
            for neighbor in self.graph[vertex]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append(neighbor)

if __name__ == "__main__":
    g = Graph(6)
    g.add_edge(0, 1)
    g.add_edge(0, 2)
    g.add_edge(1, 3)
    g.add_edge(1, 4)
    g.add_edge(2, 5)
    g.bfs(0)
```

```
BFS Traversal starting from node 0
0 1 2 3 4 5
```

In [ ]:

In [ ]:

In [11]:
```
import heapq

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v, weight):
        if u not in self.graph:
            self.graph[u] = []
        self.graph[u].append((v, weight))

    def best_first_search(self, start, goal, heuristic):
        open_list = []
        heapq.heappush(open_list, (heuristic[start], start))
        parent = {start: None}
        visited = set()

        while open_list:
            current_heuristic, current_node = heapq.heappop(open_list)
            if current_node == goal:
                path = []
                while current_node is not None:
                    path.append(current_node)
                    current_node = parent[current_node]
                path.reverse()
                print("Path from start to goal:", path)
                return
            visited.add(current_node)
            for neighbor, weight in self.graph.get(current_node, []):
                if neighbor not in visited:
                    heuristic_value = heuristic.get(neighbor, float('inf'))
                    heapq.heappush(open_list, (heuristic_value, neighbor))
                    parent[neighbor] = current_node

        print("No path found to the goal.")

if __name__ == "__main__":
    g = Graph()
    g.add_edge('A', 'B', 1)
    g.add_edge('A', 'C', 4)
    g.add_edge('B', 'D', 2)
    g.add_edge('C', 'D', 5)
    g.add_edge('D', 'E', 3)

    heuristic = {'A': 4, 'B': 2, 'C': 1, 'D': 0, 'E': 3}

    g.best_first_search('A', 'E', heuristic)
```

```
Path from start to goal: ['A', 'C', 'D', 'E']
```

In [ ]: