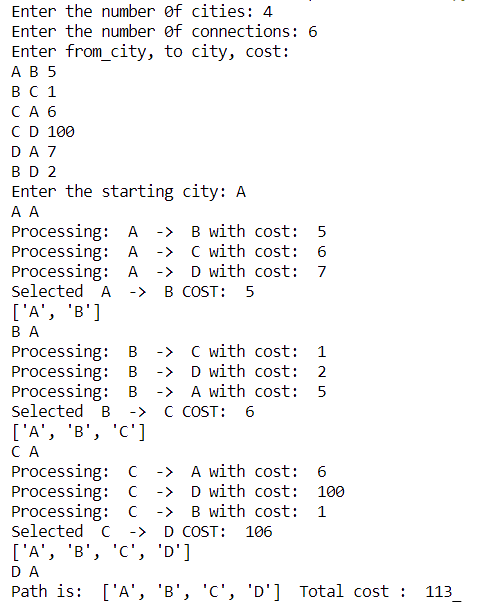
***Program 01: Travelling Salesperson***

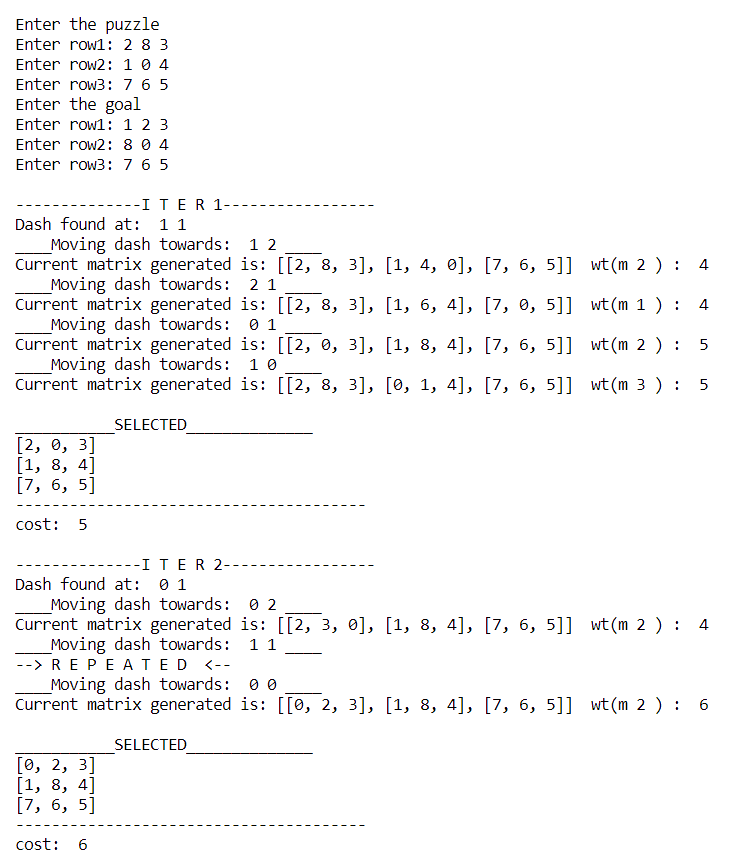
nodes\_count = int(input("Enter the number 0f cities: "))  # nodes a re cities  
  
edges\_count = int(input("Enter the number 0f connections: ")  
                  )  # connection are edges  
  
# Take connectiopns as i/p from the user  
connection\_list = []  
print("Enter from\_city, to city, cost: ")  
for i in range(edges\_count):  
    from\_city, to\_city,cost = input("").split(" ")  
    cost = int(cost)  
    connection\_list.append([from\_city, to\_city, cost])  
  
# nodes\_count = 4  
# connection\_list = [['A', 'B', 10], ['A', 'C', 20], [  
#     'A', 'D', 15], ['B', 'C', 25], ['C', 'D', 30], ['B', 'D', 35]]  
  
# Undirected graph hence reverse the edges  
final\_list = []  
for connectionL in connection\_list:  
    from\_node = connectionL[0]  
    to\_node = connectionL[1]  
    cost = connectionL[2]  
    final\_list.append([to\_node, from\_node, cost])  
  
for connectionL in final\_list:  
    connection\_list.append(connectionL)  
  
start = input("Enter the starting city: ")  
  
  
def TSP(connection\_list, current, start, visited, total\_cost, nodes\_count):  
    print(current, start)  
    if len(visited) == nodes\_count:  
        for connectionL in connection\_list:  
            if (connectionL[0] == current and connectionL[1] == start):  
                total\_cost = total\_cost+connectionL[2]  
                break  
  
        print("Path is: ", visited, " Total cost : ", total\_cost)   
        return  
    selected\_node = ""  
    selected\_cost = 100000  
    # Explore path from start to other node and take the best one not visited  
    for connectionL in connection\_list:  
        from\_city = connectionL[0]  
        if from\_city is not current:  
            continue  
        to\_city = connectionL[1]  
        cost = connectionL[2]  
        print("Processing: ", from\_city, " -> ", to\_city, "with cost: ", cost)  
        if to\_city not in visited and cost < selected\_cost:  
            selected\_cost = cost  
            selected\_node = to\_city  
    # As we found the next node add it to visited and add the cost to the total\_cost  
    if selected\_node == "":  
        print("No node selected")  
        return  
    visited.append(selected\_node)  
    print("Selected ", current, " -> ", selected\_node,  
          "COST: ", total\_cost+selected\_cost)  
    print(visited)  
    TSP(connection\_list, selected\_node, start,  
        visited, total\_cost+selected\_cost, nodes\_count)  
  
  
visited = [start]  
TOTAL\_COST = 0  
TSP(connection\_list, start, start, visited, TOTAL\_COST, nodes\_count)

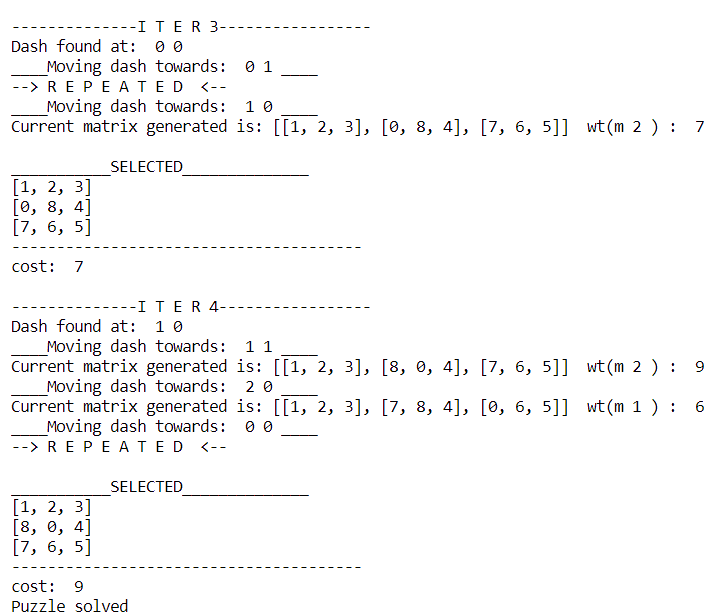
***Output:***

******

***Program 02: 8 - Puzzle Problem***  
  
max\_count = 0  
  
def display(puzzle):  
    for row in puzzle:  
        print(row)  
    print("---------------------------------------")      
def compare(puzzle, goal):  
    match = 0  
    for i in range(3):  
        for j in range(3):  
            if puzzle[i][j] == goal[i][j]:  
                match = match+1  
    return match  
def move\_dash(iter, puzzle, goal, generated\_states,x,y):  
    if iter==10:  
        return  
    print(  
        f"\n--------------I T E R {iter}-----------------")  
  
    print("Dash found at: ", x, y)  
    i = 0  
      
    # Move dash  
    d\_X = [0, 1, -1, 0]  
    d\_y = [1, 0, 0, -1]  
      
    # To have the next puzzle with maximum weight  
    max\_match\_count = 0  
      
    next\_puzzle = []  
      
    selected\_x, selected\_y = -1, -1  
      
    for i in range(4):  
        # get new positions for dash  
        new\_x = x+d\_X[i]  
        new\_y = y+d\_y[i]  
        # validate positions  
        if new\_x < 0 or new\_x >= 3 or new\_y < 0 or new\_y >= 3:  
            continue  
          
        # print position selected  
        print("\_\_\_\_Moving dash towards: ", new\_x, new\_y,"\_\_\_\_")  
          
        # update the puzzle  
        num = puzzle[new\_x][new\_y]  
        puzzle[new\_x][new\_y] = 0  
        puzzle[x][y] = num  
          
        #Generating puzzle string  
        puzzle\_str=str(puzzle)  
          
        #If this is repeated state then dont consider it  
        if puzzle\_str in generated\_states:  
            print("--> R E P E A T E D  <--")  
              
            # restore puzzle matrix  
            num = puzzle[x][y]  
            puzzle[new\_x][new\_y] = num  
            puzzle[x][y] = 0  
            continue;  
        else:  
            generated\_states.append(puzzle\_str)  
      
        # get weight  
        weight\_cost = compare(puzzle, goal)  
  
        # ckeep track of matrix with max weight  
        if weight\_cost > max\_match\_count:  
            max\_match\_count = weight\_cost  
            next\_puzzle = []  
            for i in range(3):  
                row = []  
                for j in range(3):  
                    row.append(puzzle[i][j])  
                next\_puzzle.append(row)  
            selected\_x = new\_x  
            selected\_y = new\_y  
              
        print(  
            f"Current matrix generated is: {puzzle}  wt(m {i} ) : ", weight\_cost)  
  
        # restore puzzle matrix  
        num = puzzle[x][y]  
        puzzle[new\_x][new\_y] = num  
        puzzle[x][y] = 0  
  
        i = i+1  
  
    # Otherise print the matrix selected  
    print("\n\_\_\_\_\_\_\_\_\_\_\_SELECTED\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
    display(next\_puzzle)  
    print("cost: ",max\_match\_count)  
      
    # Update the puzzle to undergo next iteration  
    puzzle = next\_puzzle  
      
    # match\_count is < 9 proceed next iter  
    if max\_match\_count < 9:  
        move\_dash(iter+1,puzzle, goal, generated\_states,selected\_x,selected\_y)  
    elif max\_match\_count==9:  
        print("Puzzle solved")  
  
#Take input  
puzzle=[]  
goal=[]  
print("Enter the puzzle")  
for i in range(3):  
    row=[int(item) for item in input(f"Enter row{i+1}: ").split(" ")]  
    puzzle.append((row))  
      
print("Enter the goal")  
for i in range(3):  
    row=[int(item) for item in input(f"Enter row{i+1}: ").split(" ")]  
    goal.append((row))  
      
#To keep all generated sattes  
generated\_states=[]  
generated\_states.append(str(puzzle))  
  
x,y,i=0,0,0  
for row in puzzle:  
    x=i  
    if 0 in row:  
        y=row.index(0)  
        break  
    i=i+1   
  
move\_dash(1, puzzle, goal, generated\_states,x,y)

***Output:***

******

******