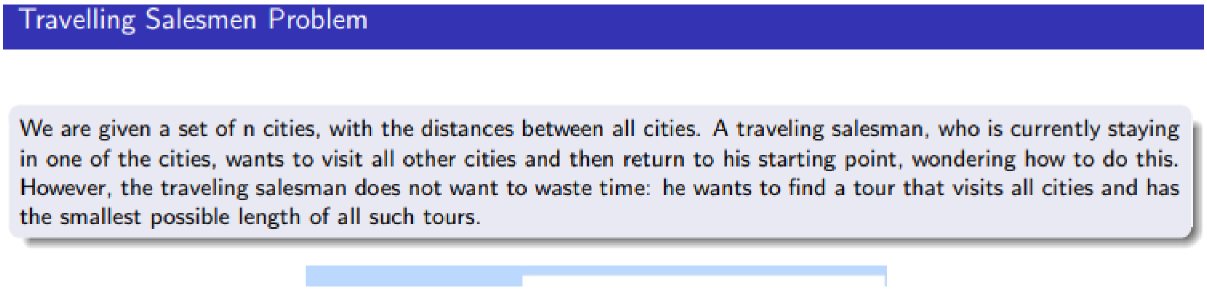
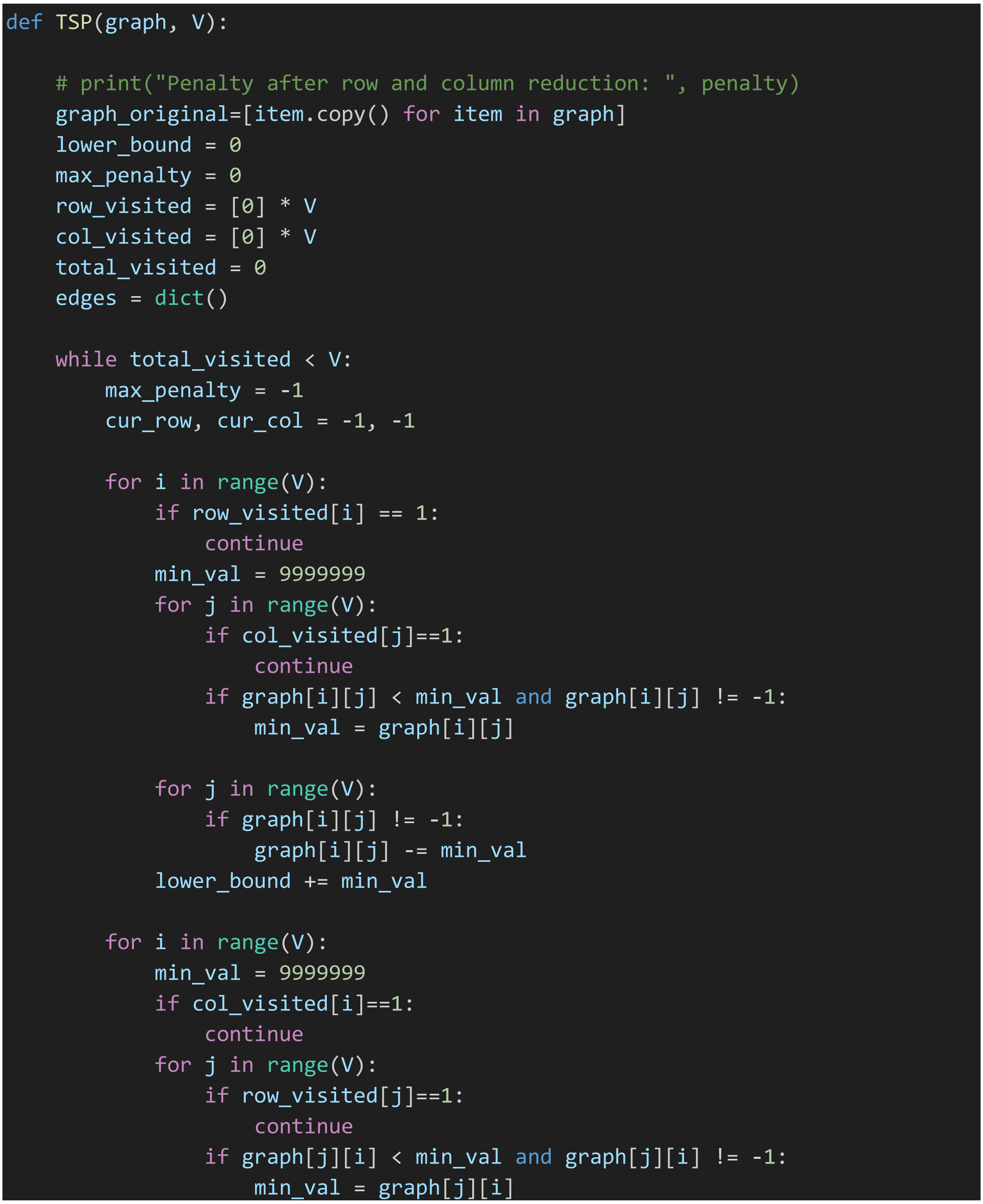


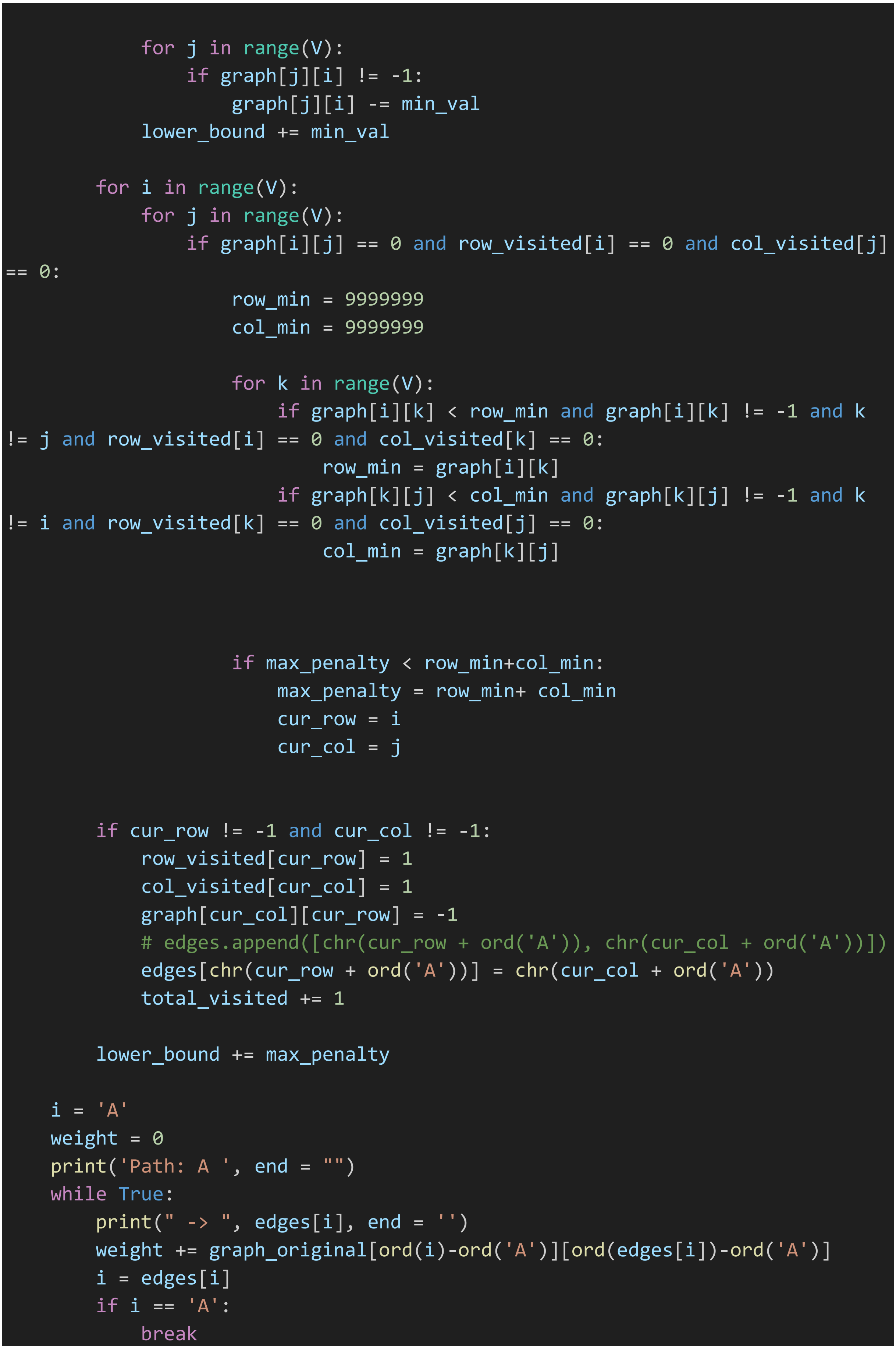
Travelling Salesmen Problem

We are given a set of n cities, with the distances between all cities. A traveling salesman, who is currently staying in one of the cities, wants to visit all other cities and then return to his starting point, wondering how to do this. However, the traveling salesman does not want to waste time: he wants to find a tour that visits all cities and has the smallest possible length of all such tours.

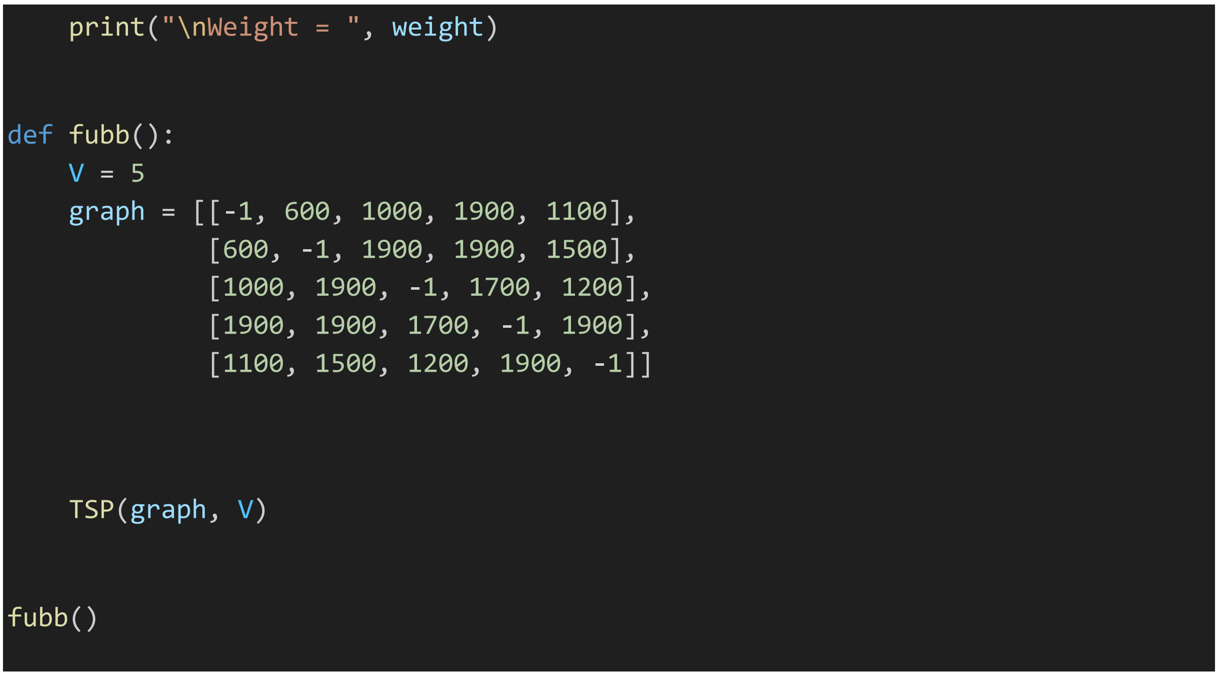
Code:



```python
def TSP(graph, V):

    # print("Penalty after row and column reduction: ", penalty)
    graph_original=[item.copy() for item in graph]
    lower_bound = 0
    max_penalty = 0
    row_visited = [0] * V
    col_visited = [0] * V
    total_visited = 0
    edges = dict()

    while total_visited < V:
        max_penalty = -1
        cur_row, cur_col = -1, -1

        for i in range(V):
            if row_visited[i] == 1:
                continue
            min_val = 9999999
            for j in range(V):
                if col_visited[j]==1:
                    continue
                if graph[i][j] < min_val and graph[i][j] != -1:
                    min_val = graph[i][j]

            for j in range(V):
                if graph[i][j] != -1:
                    graph[i][j] -= min_val
            lower_bound += min_val

        for i in range(V):
            min_val = 9999999
            if col_visited[i]==1:
                continue
            for j in range(V):
                if row_visited[j]==1:
                    continue
                if graph[j][i] < min_val and graph[j][i] != -1:
                    min_val = graph[j][i]
```

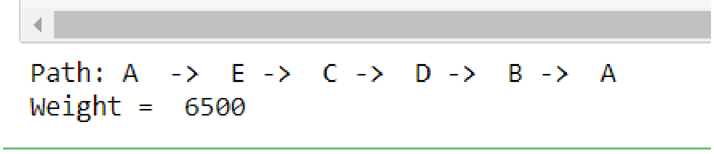


```
            for j in range(V):
                if graph[j][i] != -1:
                    graph[j][i] -= min_val
            lower_bound += min_val

        for i in range(V):
            for j in range(V):
                if graph[i][j] == 0 and row_visited[i] == 0 and col_visited[j]
== 0:
                    row_min = 9999999
                    col_min = 9999999

                    for k in range(V):
                        if graph[i][k] < row_min and graph[i][k] != -1 and k
!= j and row_visited[i] == 0 and col_visited[k] == 0:
                            row_min = graph[i][k]
                        if graph[k][j] < col_min and graph[k][j] != -1 and k
!= i and row_visited[k] == 0 and col_visited[j] == 0:
                            col_min = graph[k][j]



                    if max_penalty < row_min+col_min:
                        max_penalty = row_min+ col_min
                        cur_row = i
                        cur_col = j


        if cur_row != -1 and cur_col != -1:
            row_visited[cur_row] = 1
            col_visited[cur_col] = 1
            graph[cur_col][cur_row] = -1
            # edges.append([chr(cur_row + ord('A')), chr(cur_col + ord('A'))])
            edges[chr(cur_row + ord('A'))] = chr(cur_col + ord('A'))
            total_visited += 1

        lower_bound += max_penalty

    i = 'A'
    weight = 0
    print('Path: A ', end = "")
    while True:
        print(" -> ", edges[i], end = '')
        weight += graph_original[ord(i)-ord('A')][ord(edges[i])-ord('A')]
        i = edges[i]
        if i == 'A':
            break
```



```
    print("\nWeight = ", weight)


def fubb():
    V = 5
    graph = [[-1, 600, 1000, 1900, 1100],
             [600, -1, 1900, 1900, 1500],
             [1000, 1900, -1, 1700, 1200],
             [1900, 1900, 1700, -1, 1900],
             [1100, 1500, 1200, 1900, -1]]



    TSP(graph, V)


fubb()
```

Output:



```
Path: A  ->  E ->  C ->  D ->  B ->  A
Weight =  6500
```