**PRACTICAL NO. 6**

**Name**: Zoya Ghachi

**Roll No**: A1\_B2\_23

**Aim-1:** A traveling salesman is getting ready for a big sales tour. Starting at his hometown, suitcase in hand, he will conduct a journey in which each of his target cities is visited exactly once before he returns home. Given the pairwise distances between cities, what is the best order in which to visit them, so as to minimize the overall distance traveled?

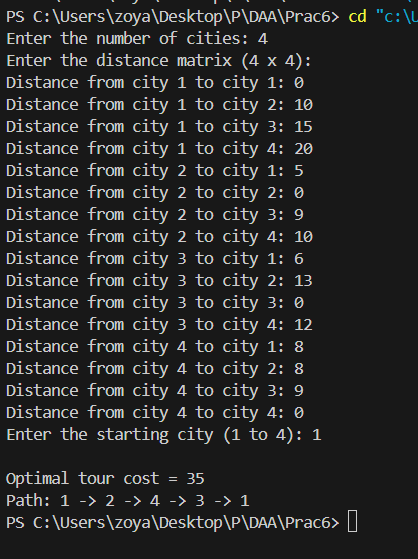
**Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ver** | **1** | **2** | **3** | **4** |
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 9 | 10 |
| 3 | 6 | 13 | 0 | 12 |
| 4 | 8 | 8 | 9 | 0 |

**Consider different starting vertex:** 1, 2, 3, 4

**Sample Output:**

Path: 1🡪2🡪4🡪3🡪1, Cost of travelling is: 35



**LEETCODE SUBMISSION:**

<https://leetcode.com/problems/find-the-shortest-superstring/solutions/194932/Travelling-Salesman-Problem/>

class Solution {

    public String shortestSuperstring(String[] A) {

        int n = A.length;

        int[][] graph = new int[n][n];

        // build the graph

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                graph[i][j] = calc(A[i], A[j]);

                graph[j][i] = calc(A[j], A[i]);

            }

        }

        int[][] dp = new int[1 << n][n];

        int[][] path = new int[1 << n][n];

        int last = -1, min = Integer.MAX\_VALUE;

        // start TSP DP

        for (int i = 1; i < (1 << n); i++) {

            Arrays.fill(dp[i], Integer.MAX\_VALUE);

            for (int j = 0; j < n; j++) {

                if ((i & (1 << j)) > 0) {

                    int prev = i - (1 << j);

                    if (prev == 0) {

                        dp[i][j] = A[j].length();

                    } else {

                        for (int k = 0; k < n; k++) {

                            if (dp[prev][k] < Integer.MAX\_VALUE && dp[prev][k] + graph[k][j] < dp[i][j]) {

                                dp[i][j] = dp[prev][k] + graph[k][j];

                                path[i][j] = k;

                            }

                        }

                    }

                }

                if (i == (1 << n) - 1 && dp[i][j] < min) {

                    min = dp[i][j];

                    last = j;

                }

            }

        }

        // build the path

        StringBuilder sb = new StringBuilder();

        int cur = (1 << n) - 1;

        Stack<Integer> stack = new Stack<>();

        while (cur > 0) {

            stack.push(last);

            int temp = cur;

            cur -= (1 << last);

            last = path[temp][last];

        }

        // build the result

        int i = stack.pop();

        sb.append(A[i]);

        while (!stack.isEmpty()) {

            int j = stack.pop();

            sb.append(A[j].substring(A[j].length() - graph[i][j]));

            i = j;

        }

        return sb.toString();

    }

    private int calc(String a, String b) {

        for (int i = 1; i < a.length(); i++) {

            if (b.startsWith(a.substring(i))) {

                return b.length() - a.length() + i;

            }

        }

        return b.length();

    }

}

