“Minimum Cost to Connect two Groups of Points”

**A PROJECT REPORT**

***Submitted by***

***K.L.V Jayaram (192211200)***

*Under the guidance of*

**Dr.R.DHANALAKSHMI**

*in partial fulfilment for the completion of course*

**CSA0650-Design and Analysis of Algorithms for Amortized Analysis**



**SIMATS ENGINEERING**

**THANDALAM**

**JUNE 2024**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **S.NO** | **CONTENTS** |
| **1** | **Abstarct** |
| **2** | **Introduction** |
| **3** | **C Code Implementation** |
| **4** | **Test Cases** |
| **5** | **Time Complexity Analysis** |
| **6** | **Conclusion** |

**Minimum Cost to Connect Two Groups of Points**

**ABSTRACT:**

**This problem addresses the task of connecting two groups of points, where each group has a specified number of points and the cost of connection between any two points is provided in a cost matrix. The goal is to connect all points in the first group to at least one point in the second group and vice versa, while minimizing the total connection cost.**

**We propose a dynamic programming (DP) approach using bitmasking to represent the state of connections. The DP table is used to keep track of the minimum cost for each possible state of connections between the groups. By iterating through each point in the first group and exploring all possible connections to points in the second group, we update the DP table to reflect the minimum cost for each state. The final solution is obtained by finding the minimum cost among all states that ensure every point in the second group is connected.**

**INTRODUCTION:**

**The objective is to ensure that each point in both groups is connected to at least one point in the opposite group, thereby forming a fully connected bipartite graph. This problem can be approached using dynamic programming with bit masking. This approach systematically explores all possible ways of connecting the points while keeping track of the minimum costs associated with each possible state of connections. By leveraging the properties of bitmasks to represent connection states efficiently, we can devise a solution that balances complexity and computational feasibility.**

**CODING:**

**#include <iostream>**

**#include <vector>**

**#include <cstring>**

**#include <algorithm>**

**using namespace std;**

**class Solution {**

**public:**

**int connectTwoGroups(vector<vector<int>>& cost) {**

**int m = cost.size(), n = cost[0].size();**

**vector<vector<int>> f(m + 1, vector<int>(1 << n, 0x3f3f3f3f));**

**f[0][0] = 0;**

**for (int i = 1; i <= m; ++i) {**

**for (int j = 0; j < (1 << n); ++j) {**

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

**if (j & (1 << k)) {**

**int c = cost[i - 1][k];**

**int x = min({f[i][j ^ (1 << k)], f[i - 1][j], f[i - 1][j ^ (1 << k)]}) + c;**

**f[i][j] = min(f[i][j], x);**

**}**

**}**

**}**

**}**

**return f[m][(1 << n) - 1];**

**}**

**};**

**int main() {**

**Solution sol;**

**vector<vector<int>> cost = {{15, 96}, {36, 2}};**

**cout << "Minimum cost to connect two groups: " << sol.connectTwoGroups(cost) << endl;**

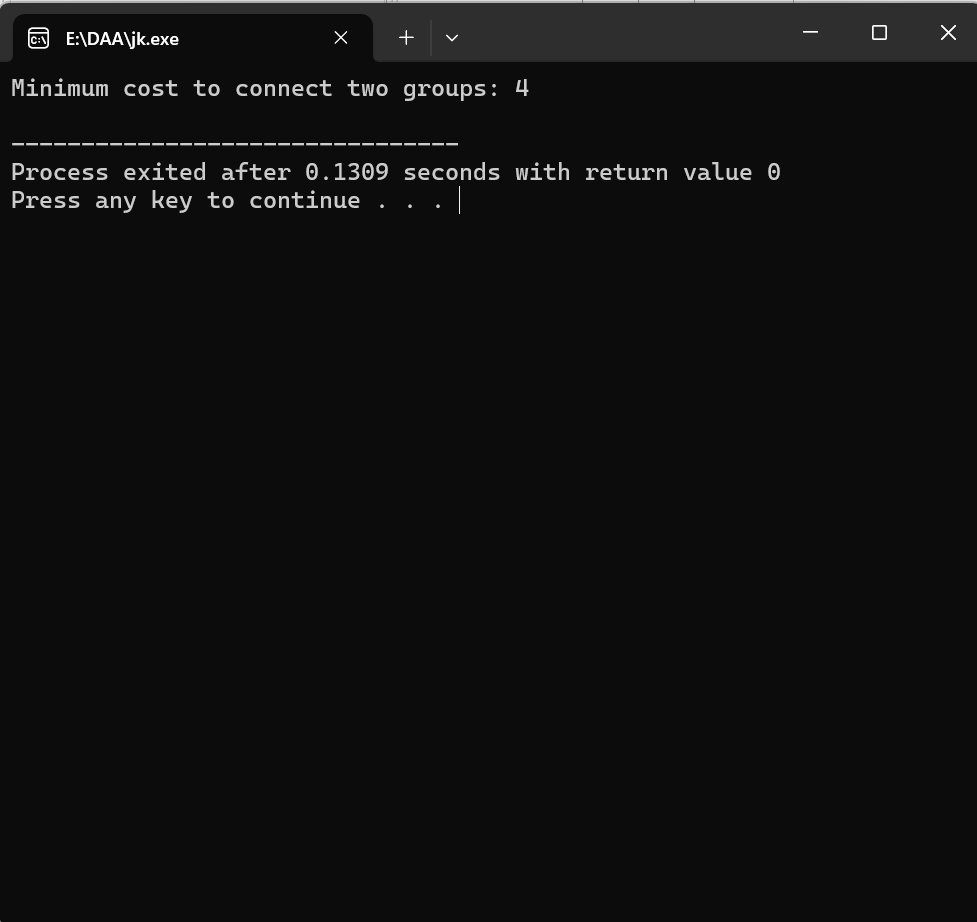
**return 0;**

**}**

**RESULT SCREENSHOT:**

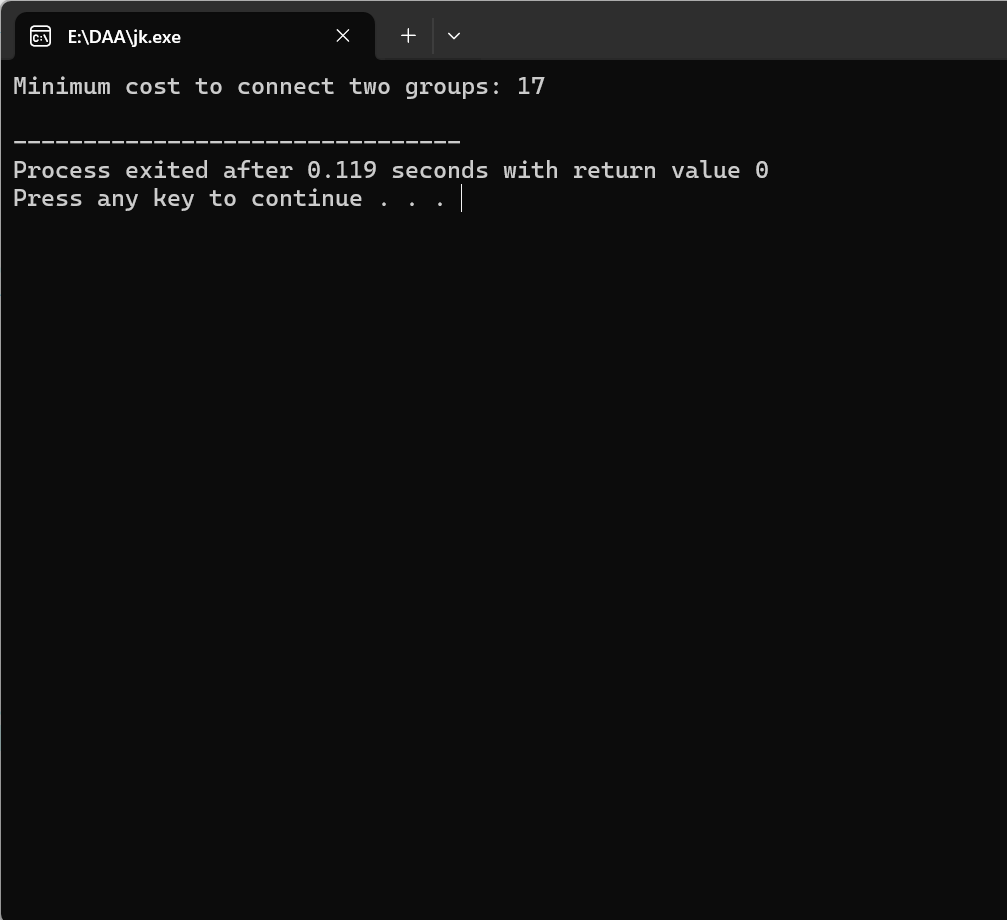
**Test Case – 1:**

****

****

**Test Case – 2:**

****

****

**COMPLEXITY ANALYSIS:**

**Time Complexity:**

**There is an outer loop that runs m times, where m is the length of the first dimension of cost. Inside this loop, there is another loop that runs for all subsets of the second group, which is 2n times, where n is the length of the second dimension of cost. Finally, within the second loop, there is an innermost loop that iterates over all n elements of the second group. The innermost loop checks and updates the best cost to connect elements of the first group to a subset of elements of the second group. Therefore, the time complexity is O(m \* n \* 2n), as for each of the m elements in the first group and for each of the 2n subsets of the second group, we perform n checks and updates.**

**Space Complexity:**

**As for the space complexity, there are two main data structures to consider: the f and cost arrays. Each of these arrays has a length of 2n, as they store subsets of the second group. Hence, the space complexity is O(2n), as this is the largest amount of space used by the algorithm at any point in time.**

**BEST CASE:**

**The best-case scenario in terms of computational complexity occurs when we can quickly determine the minimum costs without needing to explore many states. However, due to the nature of the problem, we still need to iterate through all possible states for correctness, leading to the following complexity:**

**Time Complexity: 𝑂(m×2n×n)**

**Space Complexity: 𝑂(2n)**

**WORST CASE:**

**The worst case occurs when every possible state needs to be explored without any shortcuts, which is essentially what the algorithm already accounts for. This leads to the same complexity as derived:**

**Time Complexity: 𝑂(m×2n×n)**

**Space Complexity: 𝑂(2n)**

**AVERAGE CASE:**

**In the average case, the complexity remains the same as we still need to consider all possible states to ensure correctness. The actual run-time may vary slightly based on specific cost matrix values and their distribution, but asymptotically, the complexity doesn't change:**

**Time Complexity: 𝑂(m×2n×n)**

**Space Complexity: 𝑂(2n)**

**CONCLUSION:**

**For this problem and the given solution, the best case, worst case, and average case time and space complexities are essentially the same due to the necessity of exploring all potential states to guarantee the minimum connection cost. The specific values and distribution of the costs in the matrix may affect the constant factors and actual run-time but do not change the overall asymptotic complexity.**

**Time Complexity (Best, Worst, and Average): 𝑂(m×2n×n)**

**Space Complexity (Best, Worst, and Average): 𝑂(2n)**

**This uniform complexity across different cases highlights the inherent computational intensity of the problem when approached using dynamic programming and bit-masking techniques.**