ALGORITHM PROBLEM SOLVING LAB

Even Sem - 2023

REPORT

Administration Ally

Team members:

Astha Raghuwanshi-21103042

Prerna-21103043

Princi Agrawal-21103048

TABLE OF CONTENTS:

Objective of the project:

The objective of creating an administrative ally software is to help college administration overcome common challenges they face by leveraging technology to improve their efficiency, accuracy, and productivity.

The program assists the Administration in the following domains:

1.Ensuring CCTV surveillance at the campus.

2.Efficient Induction Schedule.

3.Games to be organised in FSM.

4. Wing Allocation as per preferences.

5.Emergency Exit Guide.

Task to be performed:

1. Ensuring CCTV surveillance at the campus:

Creating a program that helps find the minimum number of cameras required to cover the entire campus can significantly benefit the college administration, providing a cost-effective, efficient, and secure environment for students, staff, and visitors.

By determining the minimum number of cameras required, the administration can save on unnecessary camera installations, leading to cost savings. This way, they can optimize the use of resources and allocate funds to other essential areas of the institution.

2. Efficient Induction Schedule:

The program can save time for both visitors and the administration by suggesting the most efficient path for touring the campus. This would enable visitors to cover the entire campus in the minimum amount of time, making their visit more efficient and reducing the time required for the administration to provide tours.

By providing a more efficient tour experience, the program can enhance the visitor experience and leave a positive impression of the college. Visitors would appreciate the convenience of an organized tour, and a good experience would lead to better word-of-mouth promotion.

3. Games to be organised in FSM:

Creating a program that helps determine the popular games to be included in a sports meet can provide several benefits to the college administration:

By selecting popular games that are well-liked by the students, the administration can increase participation in the sports meet. This can lead to more students being involved in sports activities, promoting a healthier and more active lifestyle.

By selecting popular games, the administration can attract more spectators to the event. This can enhance the atmosphere and make the sports meet more exciting, leading to increased engagement from the audience.

By selecting popular games, the administration can provide the students with an opportunity to showcase their skills in games that are well-known and recognized. This can give the college a competitive edge in intercollegiate competitions and help in building a reputation for excellence in sports.

4. Hostel wing allocation:

The program can efficiently allocate hostel wings to students based on their preferences, reducing the workload of the administration. This would lead to a faster and more efficient allocation process, saving time and resources.

The program can also help minimize conflicts that may arise from students being assigned to wings they do not prefer. By ensuring that students are assigned to wings they prefer, the program can reduce the likelihood of conflicts between students.

The program can also reduce the workload of the administration in terms of handling complaints and requests for room changes. With the efficient allocation of rooms, there would be fewer complaints and requests for changes, freeing up the administration to focus on other important tasks.

By ensuring that students are assigned to hostel wings based on their preferences, the program can improve student satisfaction with the hostel allocation process. This would lead to a better overall experience for students and a more positive reputation for the college.

5. Emergency Exit Guide:

By providing a clear location of the nearest exit, the program can improve safety for students, staff, and visitors. In case of an emergency, quick and safe evacuation is critical, and the program can help ensure that this is done efficiently.

In case of an emergency, people tend to panic, making it challenging to locate the nearest exit. The program helps in reducing panic and making it easier for people to evacuate the building safely.

Algorithms used:

1.Graph coloring using greedy:

Graph coloring can be used to determine the minimum number of CCTV cameras required to cover the entire college campus. The idea is to represent the college campus as a graph, where the vertices represent different locations in the campus, and the edges represent the connections between these locations. Each vertex will be assigned a color, representing whether or not it is being monitored by a CCTV camera.

To apply graph coloring to this problem, we first need to construct the graph representing the college campus. We can then use a graph coloring algorithm, such as the Greedy algorithm to assign colors to the vertices such that no adjacent vertices have the same color.

In the context of CCTV installation, each color corresponds to a CCTV camera that monitors all the vertices of that color. The minimum number of colors required to cover the entire graph will then give us the minimum number of CCTV cameras required to cover the entire campus.

Here's a step-by-step algorithm to determine the minimum number of CCTV cameras required:

1. Represent the college campus as a graph G=(V,E), where V is the set of vertices representing locations in the campus, and E is the set of edges representing connections between these locations.
2. Apply a graph coloring algorithm, such as the Greedy algorithm or the Welsh-Powell algorithm, to the graph G. The output of this algorithm will be a set of colors, where each color represents a CCTV camera that monitors all the vertices of that color.
3. The minimum number of CCTV cameras required to cover the entire campus is equal to the number of colors used in step 2.

2. MST using Kruskal’s:

Kruskal's algorithm is a well-known algorithm for finding the minimum spanning tree of a weighted undirected graph. The minimum spanning tree is a subset of edges that connect all the vertices of the graph with the minimum total weight. This algorithm can be used to

determine the path that should be followed during a campus visit to ensure that each point is covered in minimum time.

Here's how we have used Kruskal's algorithm to determine the path for a campus visit:

1. Represent the campus as a graph G = (V, E), where V is the set of points of interest, and E is the set of connections between these points. Each connection has a weight representing the time required to travel between the two points.

2. Apply Kruskal's algorithm to the graph G to obtain the minimum spanning tree. The minimum spanning tree will be a subset of the edges that connect all the vertices of the graph with the minimum total weight.

3. Traverse the minimum spanning tree in a way that visits each vertex exactly once. This can be done using any tree traversal algorithm, such as depth-first search or breadth-first search. The order of traversal will be the path that should be followed during the campus visit to ensure that each point is covered in minimum time.

4. The total time required for the campus visit will be the sum of the weights of the edges in the minimum spanning tree.

3. 0/1 Knapsack using DP:

Here, 0/1 Knapsack algorithm have been used to determine the optimal selection of sports/games that should be included in the sports meet, based on their popularity and time required to conduct each game.

The `weight` attribute of each sport/game represents the time required to conduct the game, which is analogous to the weight of an item in a traditional knapsack problem. Similarly, the `value` attribute of each sport/game represents the popularity or excitement factor of the game, which is analogous to the value of an item in a knapsack problem.

The goal of the knapsack algorithm in this scenario is to select a subset of sports/games that can be included in the sports meet, such that the total time required to conduct the selected games does not exceed the given capacity, and the total excitement generated by the selected games is maximized.

The `knapsack` function in the program implements the knapsack algorithm to find the maximum excitement that can be generated by including the selected sports/games within the given capacity. The function calculates the maximum excitement using dynamic programming technique, tabulation.

Here are the steps in which the knapsack algorithm can be implemented in this scenario:

1. Define a structure to represent each sport/game, with attributes such as name, weight (time required to conduct the game), and value (popularity or excitement factor of the game).

2. Initialize a vector of sports/games with their respective duration and popularity values.

3. Set the capacity of the knapsack to the maximum available time duration for the sports meet.

4. Implement the knapsack algorithm using dynamic programming technique, tabulation. The algorithm should take into account the weight (time required to conduct each game) and value (popularity or excitement factor) of each sport/game, and should determine the maximum excitement that can be generated by including a subset of the available sports/games in the sports meet, without exceeding the given capacity.

5. Return the maximum excitement generated by including the selected sports/games in the event, and also display the list of sports/games to be included.

4. Wing Allocation as per preferences:

Ford-Fulkerson algorithm and Bipartite Matching can indeed be used to assist in the allocation of hostel wings to students based on their preferences. Here's how:

The students can be represented as vertices on one side of a bipartite graph, and the hostel wings can be represented as vertices on the other side. We will create an edge between a student and a hostel wing if the student prefers that wing.

In this problem, the capacity of an edge represents the number of students who can be allocated to that particular wing. This number will depend on the maximum occupancy of each wing.

Once the graph is constructed, we can use the bipartite matching algorithm to find a maximum matching in the graph. This matching will assign each student to a wing they prefer, subject to the capacity constraints.

If the maximum matching has used up all the capacities of the edges, then we have a valid allocation. If not, we need to either increase the capacity of some edges or inform some students that they cannot be allocated to their preferred wing.

Once we have a valid allocation, we can implement it by assigning students to their allocated wings.

5. Emergency Exit Guide:

Dijkstra's algorithm can be used to find the shortest path to the nearest emergency exit in case of an emergency in a college campus.

Each building or landmark in the campus can be represented as a vertex, and the paths between them can be represented as edges. The edges can be weighted based on the distance between the buildings.

The algorithm needs to know the current location of the person who needs to find the nearest emergency exit. This can be input as a starting vertex in the graph.

The algorithm continues until it reaches the emergency exit, which is one of the vertices in the graph.

Dijkstra's algorithm can be used to find the nearest exit in a college campus by modelling the campus as a weighted graph, where the buildings are nodes and the paths between them are edges with weights representing the distance or time required to travel from one building to another.

To apply Dijkstra's algorithm, we have followed these steps:

1. Create a graph representation of the campus: Identify all the prominent locations and paths between them. Assign weights to the edges based on the distance or time required to travel between them.

2. Identify the current location: Determine the location where the person is currently located.

3. Run Dijkstra's algorithm: Use Dijkstra's algorithm to find the shortest path from the current location to each exit. This will give you the distance or time required to reach each exit from the current location.

4. Select the nearest exit: Once you have the distances to each exit, select the exit with the shortest distance. This will be the nearest exit.

By using Dijkstra's algorithm in this way, we can efficiently find the nearest exit for a person in a college campus. This can be useful in emergency situations or for visitors who are unfamiliar with the campus layout.

CODE:

```
#include <bits/stdc++.h>
#include <iostream>
#include <conio.h>
#include<graphics.h>
using namespace std;
const int MAXN = 13;

// CCTV CAMERA
class Graph
{
   int V;          // No. of vertices
   list<int> *adj; // Pointer to an array containing adjacency lists
public:
   Graph(int V);               // Constructor
   void addEdge(int v, int w); // function to add an edge to graph
   void printVertexCover();    // prints vertex cover
};

Graph::Graph(int V)
{
   this->V = V;
   adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
```

```
    adj[v].push_back(w); // Add w to v's list.
    adj[w].push_back(v); // Since the graph is undirected
}

// The function to print vertex cover
void Graph::printVertexCover()
{
    // Initialize all vertices as not visited.
    bool visited[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    list<int>::iterator i;

    // Consider all edges one by one
    for (int u = 0; u < V; u++)
    {
        // An edge is only picked when both visited[u] and visited[v]
        // are false
        if (visited[u] == false)
        {
            // Go through all adjacents of u and pick the first not
            // yet visited vertex (We are basically picking an edge
            // (u, v) from remaining edges.
            for (i = adj[u].begin(); i != adj[u].end(); ++i)
            {
                int v = *i;
                if (visited[v] == false)
                {
                    // Add the vertices (u, v) to the result set.
```

```
                // We make the vertex u and v visited so that
                // all edges from/to them would be ignored
                visited[v] = true;
                visited[u] = true;
                break;
            }
        }
    }
}
int ccount = 0;
// Print the vertex cover
cout << "Cameras to be put up at: ";
for (int i = 0; i < V; i++)

    if (visited[i])
    {
        cout << i << " ";

        ccount++;
    }
cout << endl;
cout << "Total count of cameras needed: " << ccount << endl;
}

// INDUCTION PROGRAM
#define t 15
int parent[t];

int find(int i)
{
```

```
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void union1(int i, int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

void kruskalMST(int cost[][t])
{
    int mincost = 0;

    vector<string> vec;
    vec.push_back("Gate1");
    vec.push_back("Gate2");
    vec.push_back("Gate3");
    vec.push_back("Ground");
    vec.push_back("ABB1");
    vec.push_back("Mess");
    vec.push_back("JBS");
    vec.push_back("Sarojini");
    vec.push_back("OAT");
    vec.push_back("BoysHostel");
    vec.push_back("Library");
    vec.push_back("GBpanth");
    vec.push_back("H3");
```

```
vec.push_back("MaulanaAzad");
vec.push_back("Cafe");

for (int i = 0; i < t; i++)
{
   parent[i] = i;
}

int edge_count = 0;
while (edge_count < t - 1)
{
   int min = INT_MAX, a = -1, b = -1;
   for (int i = 0; i < t; i++)
   {
      for (int j = 0; j < t; j++)
      {
         if (find(i) != find(j) && cost[i][j] < min)
         {
            min = cost[i][j];
            a = i;
            b = j;
         }
      }
   }

   union1(a, b);

   cout << "Edge " << edge_count++ << ": (" << a << " " << b << ")"
        << "===>>" << vec[a] << "-" << vec[b] << endl;
   mincost += min;
```

```
    }
    cout<<endl;
    cout<<"Minimum cost to cover JP college in meters:"<<mincost<<endl;
}

// FSM GAMES
struct Sport
{
    string name;
    int weight; // duration of each game in minutes
    int value;  // popularity or excitement factor of each game
};

int knapsack(vector<Sport> &sports, int n, int capacity)
{
    int dp[n + 1][capacity + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int w = 0; w <= capacity; w++)
        {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (sports[i - 1].weight <= w)
                dp[i][w] = max(sports[i - 1].value + dp[i - 1][w - sports[i - 1].weight], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    int result = dp[n][capacity];
```

```
    int w = capacity;
    vector<string> included_sports;
    for (int i = n; i > 0 && result > 0; i--)
    {
        if (result == dp[i - 1][w])
            continue;
        else
        {
            included_sports.push_back(sports[i - 1].name);
            result -= sports[i - 1].value;
            w -= sports[i - 1].weight;
        }
    }
    cout << "Sports to include in the event: " << endl;
    for (int i = included_sports.size() - 1; i >= 0; i--)
        cout << included_sports[i] << endl;
    return dp[n][capacity];
}

// Wing Allocation
const int Maxn = 105;
const int INF = 1e9;

int source, sink;
int num_students, num_wings;

int adj_matrix[Maxn][Maxn];
int capacity[Maxn][Maxn];
int flow[Maxn][Maxn];
```

```
int parent1[Maxn];
bool visited[Maxn];

bool bfs()
{
    memset(visited, false, sizeof(visited));
    queue<int> q;
    q.push(source);
    visited[source] = true;
    parent1[source] = -1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int v = 1; v <= num_students + num_wings + 1; v++)
        {
            if (!visited[v] && capacity[u][v] - flow[u][v] > 0)
            {
                visited[v] = true;
                parent1[v] = u;
                q.push(v);
            }
        }
    }
    return visited[sink];
}

int max_flow()
{
    int max_flow = 0;
```

```
    while (bfs())
    {
        int path_flow = INF;
        for (int v = sink; v != source; v = parent1[v])
        {
            int u = parent1[v];
            path_flow = min(path_flow, capacity[u][v] - flow[u][v]);
        }
        for (int v = sink; v != source; v = parent1[v])
        {
            int u = parent1[v];
            flow[u][v] += path_flow;
            flow[v][u] -= path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;
}

void add_edge(int u, int v, int c)
{
    adj_matrix[u][v] = c;
    adj_matrix[v][u] = c;
    capacity[u][v] = c;
}

void assign_wings(vector<vector<int>> &preferences)
{
    source = 0;
    sink = num_students + num_wings + 1;
```

```
    for (int i = 1; i <= num_students; i++)
    {
        add_edge(source, i, 8);
        int wing = i + num_students;
        for (int j = 0; j < 8; j++)
        {
            add_edge(i, wing++, preferences[i - 1][j]);
        }
    }
    for (int i = num_students + 1; i <= num_students + num_wings; i++)
    {
        add_edge(i, sink, 16);
    }

    int max_matching = max_flow();
    // Print the assignments
    cout << "Number of students assigned a wing: " << max_matching << endl;
    for (int i = 1; i <= num_students; i++)
    {
        int wing = i + num_students;
        for (int j = 0; j < 8; j++)
        {
            if (flow[i][wing++] == 8)
            {
                cout << "Set of Students " << i << " assigned to wing " << (wing++) -
num_students << endl;
                break;
            }
        }
    }
}
```

```
// Emergency Gate
//  Number of vertices in the graph
#define z 15
int dist[z];
// Function to find the vertex with minimum distance value
int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < z; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}

// Function to print the final distances and the path to be followed
void printSolution(int dist[], int parent[], int dest, string temp,int source)
{
   vector<string> vec1;
   vec1.push_back("Gate1");
   vec1.push_back("Gate2");
   vec1.push_back("Gate3");
   vec1.push_back("Ground");
   vec1.push_back("ABB1");
   vec1.push_back("Mess");
   vec1.push_back("JBS");
   vec1.push_back("Sarojini");
   vec1.push_back("OAT");
   vec1.push_back("BoysHostel");
   vec1.push_back("Library");
   vec1.push_back("GBpanth");
```

```
    vec1.push_back("H3");
    vec1.push_back("MaulanaAzad");
    vec1.push_back("Cafe");
    cout << "Minimum distance from "<<vec1[source]<<" to " << temp << " : " << dist[dest]
<< endl;


    // Print the path to be followed
    cout << "Path: ";
    int path[z];
    int path_index = 0;
    int i = dest;
    while (i != -1)
    {
        path[path_index] = i;
        path_index++;
        i = parent[i];
    }
    for (int j = path_index - 1; j >= 0; j--)
        cout <<" "<<vec1[path[j]]<< " ";
    cout << endl;
}


// Function that implements Dijkstra's single source shortest path algorithm
void dijkstra(int graph[z][z], int src, int dest, string temp)
{
    // The output array. dist[i] will hold the shortest distance from src to i


    int parent[z]; // parent[i] stores the vertex that is just before i in the shortest path tree


    bool sptSet[z]; // sptSet[i] will be true if vertex i is included in shortest path tree or shortest
distance from src to i is finalized
```

```
   // Initialize all distances as INFINITE and sptSet[] as false
   for (int i = 0; i < z; i++)
      dist[i] = INT_MAX, sptSet[i] = false;

   // Distance of source vertex from itself is always 0
   dist[src] = 0;
   parent[src] = -1;

   // Find shortest path for all vertices
   for (int count = 0; count < z - 1; count++)
   {
      int u = minDistance(dist, sptSet); // Pick the minimum distance vertex from the set of vertices not yet processed
      sptSet[u] = true;                  // Mark the picked vertex as processed

      // Update dist value of the adjacent vertices of the picked vertex
      for (int v = 0; v < z; v++)
      {
         // Update dist[v] only if it is not in sptSet, there is an edge from u to v, and total weight of path from src to v through u is smaller than current value of dist[v]
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
         {
            dist[v] = dist[u] + graph[u][v];
            parent[v] = u;
         }
      }
   }

   printSolution(dist, parent, dest, temp,src);
```

```
}
int main()
{

    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << "                !!----------------------------------WELCOME TO ADMINISTRATION
ALLY----------------------------!!" << endl;
    system("cls");
    cout << endl;
    cout << endl;
    int menu;
Menu:
    cout << endl;
    cout << endl;
    cout << "                 !!*  Select a Task : *!!" << endl;
    cout << endl
         << endl;
    cout << "                 1.  Ensuring CCTV surveillance at the Campus" << endl;
    cout << "                 2.  Efficient Induction Schedule" << endl;
    cout << "                 3.  Games to be organised in FSM" << endl;
```

```
    cout << "                4.  Wing Allocation as per preferences" << endl;
    cout << "                5.  Emergency Exit Guide" << endl;
    cout << "                  Press any key to continue....." ;
    _getch();
    cout << endl;
    int choice;
    cout << endl;
    cout << endl;
    cout << endl;
    cout << "Enter your choice : ";
    cin >> menu;

    switch (menu)
    {
    case 1: // CCTV camera
    {
CCTV:
        Graph g(13);
        g.addEdge(0, 4);
        g.addEdge(0, 10);
        g.addEdge(0, 11);
        g.addEdge(1, 5);
        g.addEdge(1, 10);
        g.addEdge(1, 12);
        g.addEdge(2, 6);
        g.addEdge(2, 9);
        g.addEdge(2, 11);
        g.addEdge(3, 0);
        g.addEdge(3, 4);
        g.addEdge(3, 7);
```

```
g.addEdge(3, 9);
g.addEdge(3, 10);
g.addEdge(4, 3);
g.addEdge(4, 8);
g.addEdge(4, 10);
g.addEdge(5, 1);
g.addEdge(5, 6);
g.addEdge(5, 8);
g.addEdge(5, 12);
g.addEdge(6, 2);
g.addEdge(6, 5);
g.addEdge(6, 8);
g.addEdge(6, 9);
g.addEdge(7, 3);
g.addEdge(7, 8);
g.addEdge(7, 9);
g.addEdge(8, 4);
g.addEdge(8, 5);
g.addEdge(8, 6);
g.addEdge(8, 7);
g.addEdge(9, 0);
g.addEdge(9, 2);
g.addEdge(9, 3);
g.addEdge(9, 6);
g.addEdge(9, 7);
g.addEdge(10, 0);
g.addEdge(10, 1);
g.addEdge(10, 3);
g.addEdge(10, 5);
g.addEdge(11, 2);
```

```
    g.addEdge(11, 12);
    g.addEdge(12, 1);
    g.addEdge(12, 5);
    g.addEdge(12, 11);
    g.printVertexCover();

    cout << endl;
    cout << endl;
    int choice1;
    cout << "Press 1 to go to Main Menu or Press 0 to exit the Program: ";
    cin >> choice1;
    system("cls");
    if (choice1 == 0)
    {
        break;
    }
    else if (choice1 == 1)
    {
        goto Menu;
    }
    else
    {
        cout << "Invalid input";
    }

    return 0;
}
break;
case 2: // Induction program
{
```

```
    initwindow(700,600,"Image Window");
    readimagefile("jp_gr1.jpg", 0,0,700,600);


    _getch();
```

Induction:

```
    int time_to_cover[t][t] = {{999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999,
999, 999, 61},
        {999, 999, 999, 999, 86, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 260, 999, 999, 999},
        {999, 80, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 97, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 139, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 105, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 116, 999, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 77, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 69, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 73, 999, 999, 999, 999, 999, 999},
        {999, 999, 249, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 160, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 999, 237, 999, 999, 999},
        {999, 999, 999, 999, 999, 999, 999, 999, 999, 65, 999, 999, 999, 999, 999}
    };


    kruskalMST(time_to_cover);
    int choice1;
    cout << "Press 1 to go to Main Menu or Press 0 to exit the Program: ";
    cin >> choice1;
    system("cls");
    if (choice1 == 0)
    {
       break;
```

```
        }
        else if (choice1 == 1)
        {
            goto Menu;
        }
        else
        {
            cout << "Invalid input";
        }

        return 0;
    }
    break;

    case 3:  //FSM Games
    {
        _getch();
FSM:
        vector<Sport> sports =
        {
            {"Badminton", 60, 4},
            {"Cricket", 240, 10},
            {"Football", 200, 8},
            {"Squash", 40, 3},
            {"Table Tennis", 30, 2},
            {"Chess", 20, 1},
            {"Billiards", 10, 1},
            {"Basketball", 100, 5},
            {"Volleyball", 100, 5},
            {"Swimming", 45, 2},
```

```
        {"Skating", 20, 1},
        {"Carrom", 60, 3},
        {"Kabaddi", 50, 2},
        {"Kho-Kho", 35, 3}
    };
    int s = sports.size();
    int capacity = 720; // maximum available time in minutes for the sports meet
    int max_value = knapsack(sports, s, capacity);
    cout << "Maximum excitement generated in the sports meet: " << max_value << endl;
    int choice1;
    cout << "Press 1 to go to Main Menu or Press 0 to exit the Program: ";
    cin >> choice1;
    system("cls");
    if (choice1 == 0)
    {
       break;
    }
    else if (choice1 == 1)
    {
       goto Menu;
    }
    else
    {
       cout << "Invalid input";
    }

    return 0;
}
break;
case 4: //Wing Allocation
```

```
    {
        initwindow(400,300,"Image Window");
        readimagefile("bipar1.jpg", 0,0,400,300);
        _getch();
Wing:
        num_students = 16;
        num_wings = 8;
        vector<vector<int>> preferences = {{0, 0, 0, 8, 8, 0, 0, 8},
            {8, 0, 0, 0, 0, 0, 8, 0},
            {0, 8, 0, 8, 8, 0, 0, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {8, 0, 0, 8, 8, 0, 8, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {0, 0, 0, 8, 8, 0, 0, 8},
            {8, 0, 0, 0, 0, 8, 0, 0},
            {0, 8, 0, 0, 8, 0, 0, 8},
            {8, 8, 0, 0, 0, 8, 0, 0},
            {0, 8, 8, 8, 8, 8, 0, 8},
            {0, 0, 0, 8, 0, 8, 0, 0},
            {8, 0, 8, 0, 0, 0, 8, 0}
        };
        assign_wings(preferences);
        int choice1;
        cout << "Press 1 to go to Main Menu or Press 0 to exit the Program: ";
        cin >> choice1;
        system("cls");
        if (choice1 == 0)
```

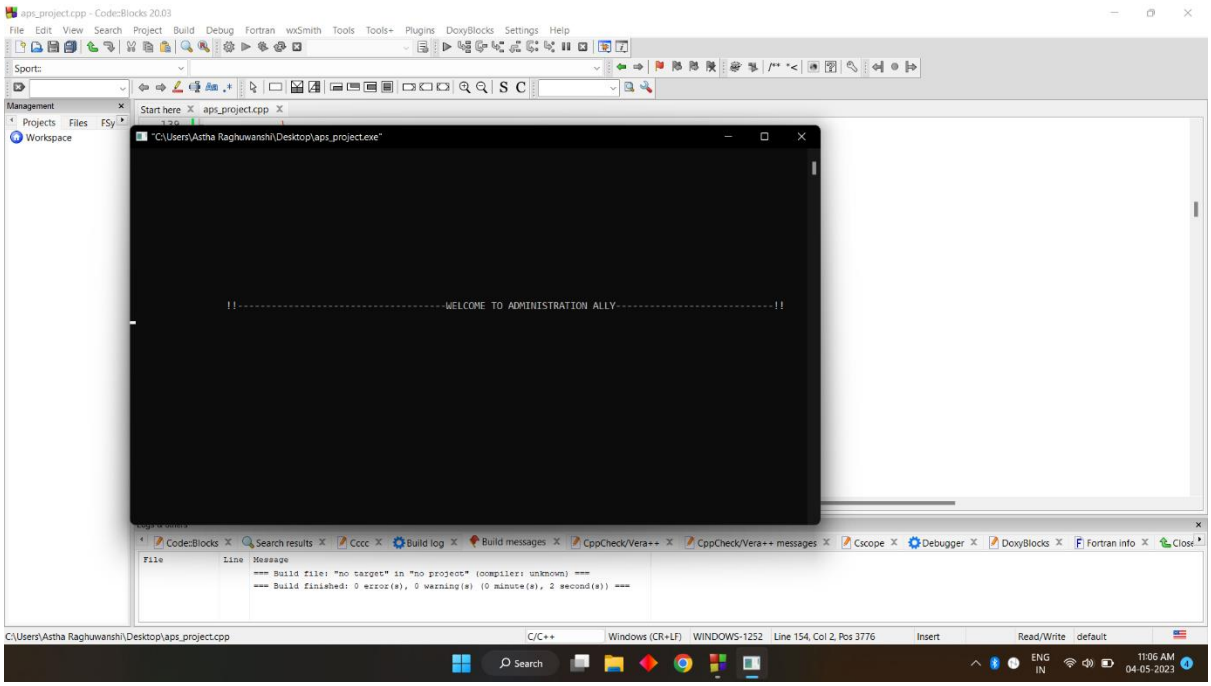
```
        {
            break;
        }
        else if (choice1 == 1)
        {
            goto Menu;
        }
        else
        {
            cout << "Invalid input";
        }

        return 0;
    }
    break;
    case 5: // Emergency gate exit

    {
        initwindow(700,600,"Image Window");
        readimagefile("jp_gr1.jpg", 0,0,700,600);
        _getch();
Emergency_exit:
        int graph[z][z] = {{0, 170, 999, 999, 999, 999, 999, 999, 100, 70, 90, 999, 999, 999, 75},
            {170, 0, 999, 999, 50, 999, 999, 999, 95, 115, 126, 999, 999, 999, 118},
            {999, 999, 0, 999, 999, 90, 198, 167, 999, 999, 999, 40, 72, 66, 999},
            {999, 999, 999, 0, 27, 999, 83, 74, 10, 32, 48, 999, 999, 999, 44},
            {999, 50, 999, 27, 0, 999, 999, 17, 38, 999, 999, 999, 999, 999, 999},
            {999, 999, 90, 999, 999, 0, 125, 85, 999, 999, 999, 90, 72, 66, 999},
            {999, 999, 198, 83, 999, 125, 0, 36, 999, 62, 72, 999, 999, 69},
            {999, 999, 167, 74, 17, 85, 36, 0, 999, 999, 999, 999, 999, 999, 999},
```

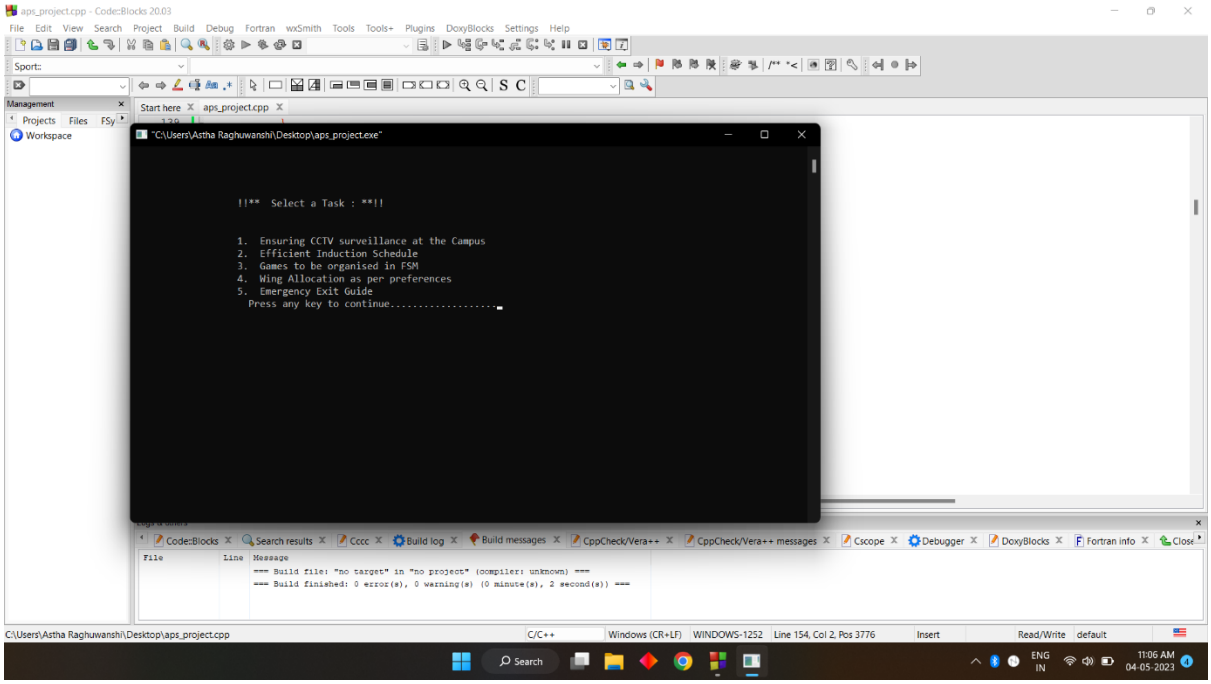
```
    {100, 95, 999, 10, 38, 999, 999, 999, 0, 45, 55, 999, 999, 999, 50},
    {70, 115, 999, 32, 999, 999, 62, 999, 45, 0, 13, 999, 999, 999, 22},
    {90, 126, 999, 48, 999, 999, 72, 999, 55, 13, 0, 999, 999, 999, 8},
    {999, 999, 40, 999, 999, 90, 999, 999, 999, 999, 999, 0, 999, 15, 999},
    {999, 999, 72, 999, 999, 72, 999, 999, 999, 999, 999, 999, 0, 27, 999},
    {999, 999, 66, 999, 999, 66, 999, 999, 999, 999, 999, 15, 27, 0, 999},
    {75, 118, 999, 44, 999, 999, 69, 999, 50, 22, 8, 999, 999, 999, 0}
};
int source;
cout<<endl;
cout<<"Use 0 for Gate1"<<endl;
cout<<"Use 1 for Gate2"<<endl;
cout<<"Use 2 for Gate3"<<endl;
cout<<"Use 3 for Ground"<<endl;
cout<<"Use 4 for ABB1"<<endl;
cout<<"Use 5 for Mess"<<endl;
cout<<"Use 6 for JBS"<<endl;
cout<<"Use 7 for Sarojini"<<endl;
cout<<"Use 8 for OAT"<<endl;
cout<<"Use 9 for BoysHostel"<<endl;
cout<<"Use 10 for Library"<<endl;
cout<<"Use 11 for GBpanth"<<endl;
cout<<"Use 12 for H3"<<endl;
cout<<"Use 13 for MaulanaAzad"<<endl;
cout<<"Use 14 for Cafe"<<endl;
cout<<endl;
cout << "Enter the source : ";
cin >> source;
cout<<endl;
int arr[4];
```

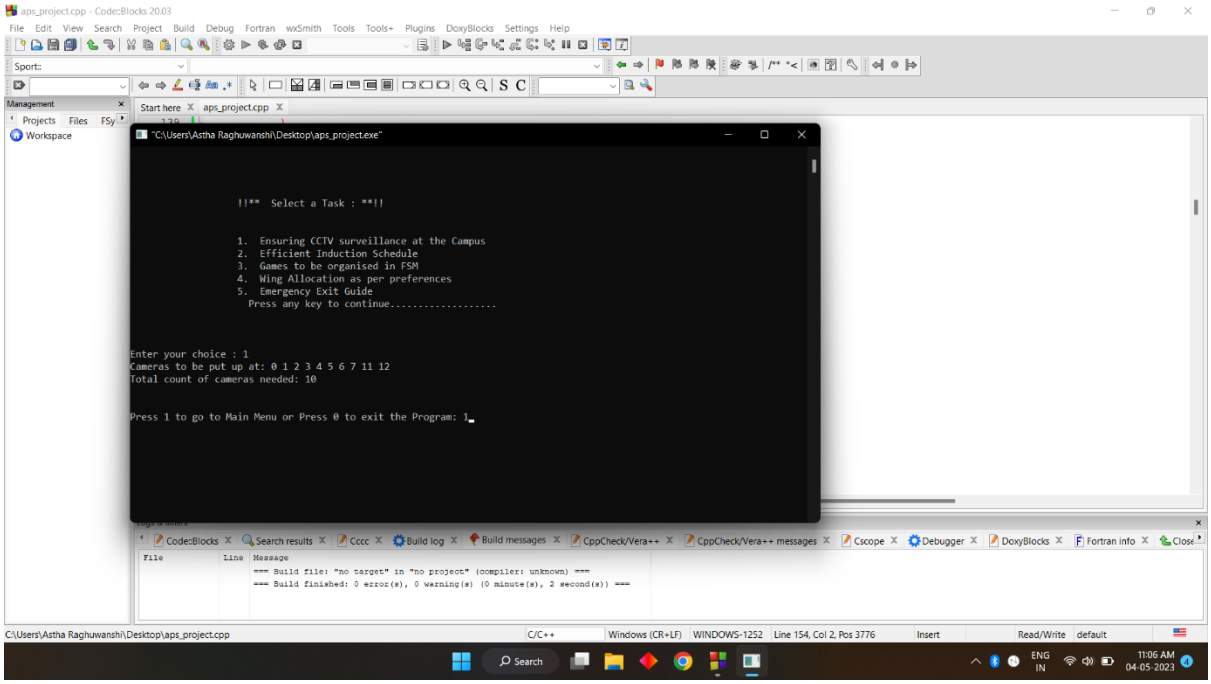
```
    dijkstra(graph, source, 0, "GATE1");
    arr[0] = dist[0];
    dijkstra(graph, source, 1, "GATE2");
    arr[1] = dist[1];
    dijkstra(graph, source, 2, "GATE3");
    arr[2] = dist[2];
    dijkstra(graph, source, 3, "GROUND");
    arr[3] = dist[3];

    int ans = INT_MAX;
    for (int i = 0; i < 4; i++)
    {
       if (ans > arr[i])
       {
          ans = arr[i];
       }
    }
    cout<<endl;
    cout << "The path that u should follow is the the one with minimum cost : " << ans << endl;
    int choice1;
    cout << endl;
    cout << endl;
    cout << "Press 1 to go to Main Menu or Press 0 to exit the Program: ";
    cin >> choice1;
    system("cls");
    if (choice1 == 0)
    {
       break;
    }
    else if (choice1 == 1)
```

```
        {
            goto Menu;
        }
        else
        {
            cout << "Invalid input";
        }

        return 0;
    }
    }
    return 0;
}
```

OUTPUT:




"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....


"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice : 1
Cameras to be put up at: 0 1 2 3 4 5 6 7 11 12
Total count of cameras needed: 10
Press 1 to go to Main Menu or Press 0 to exit the Program: 1

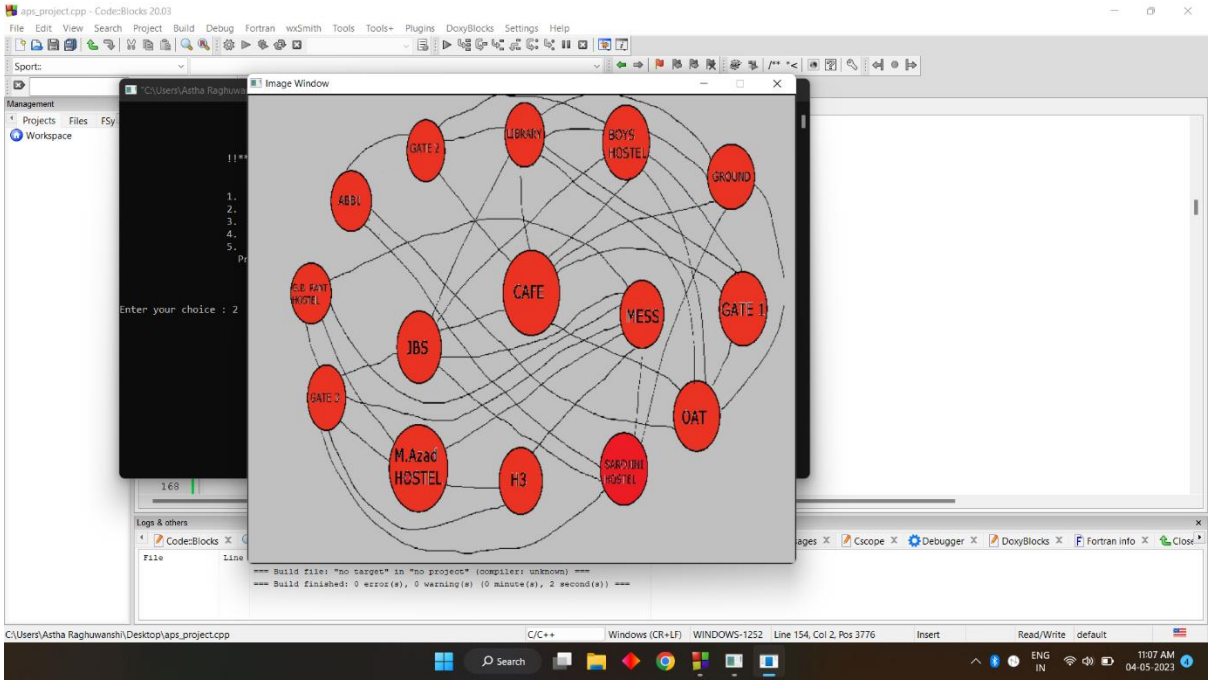
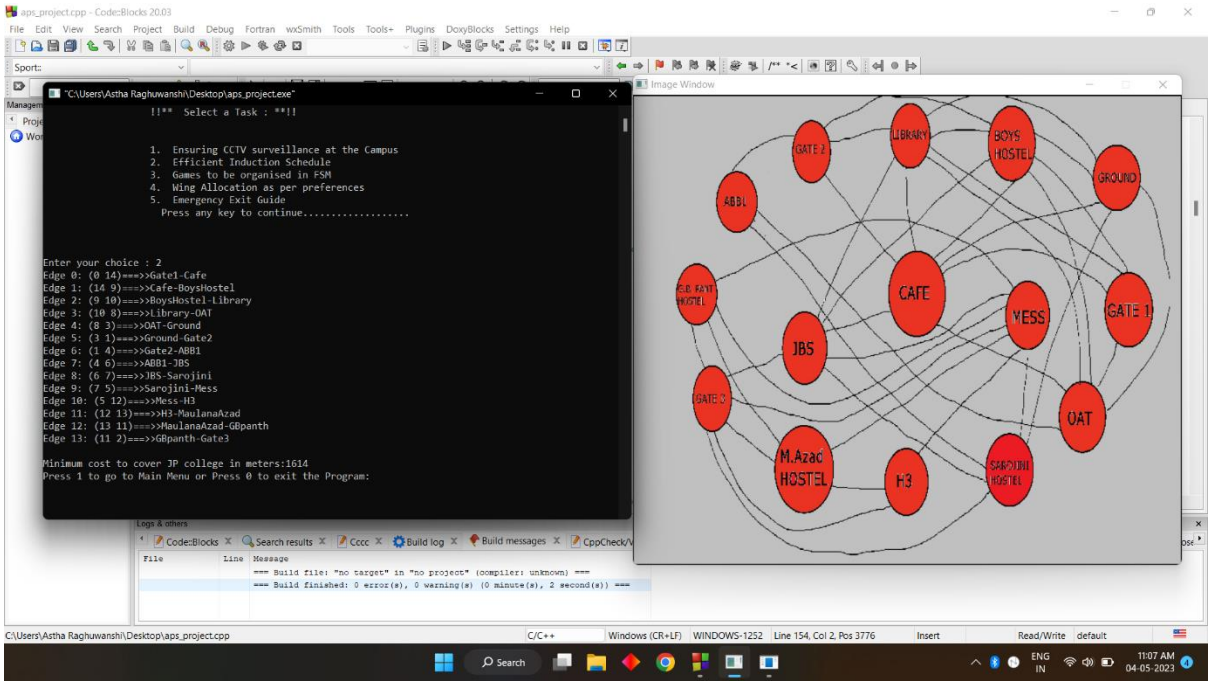
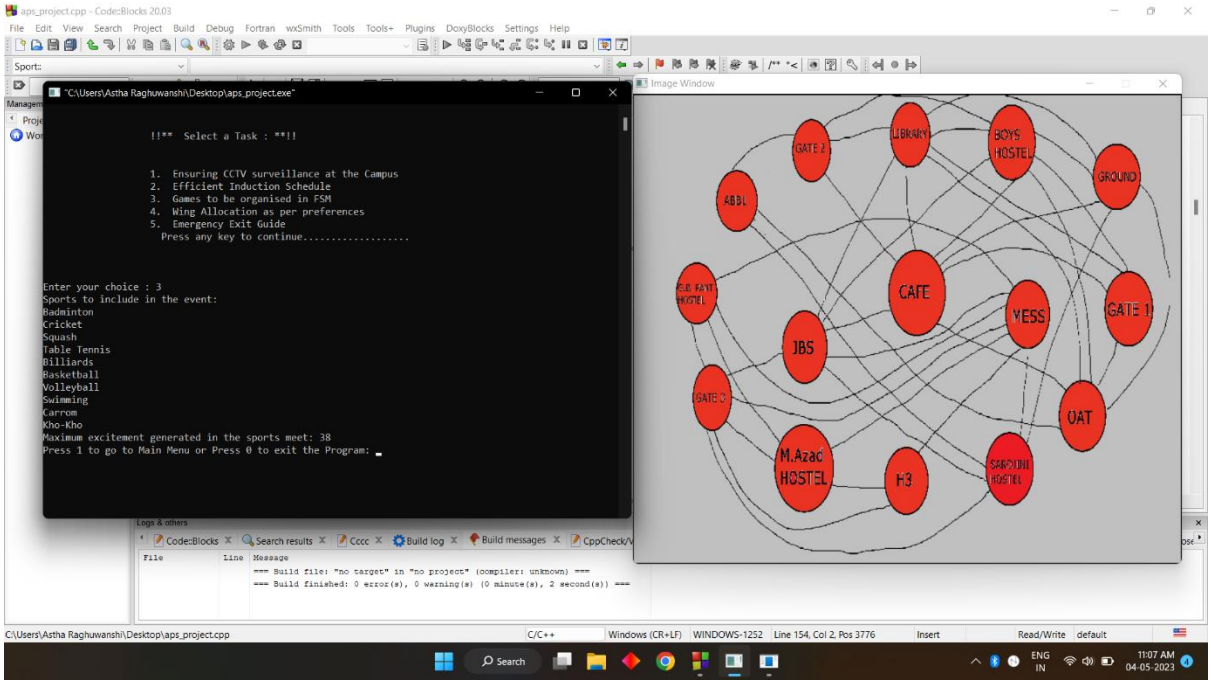
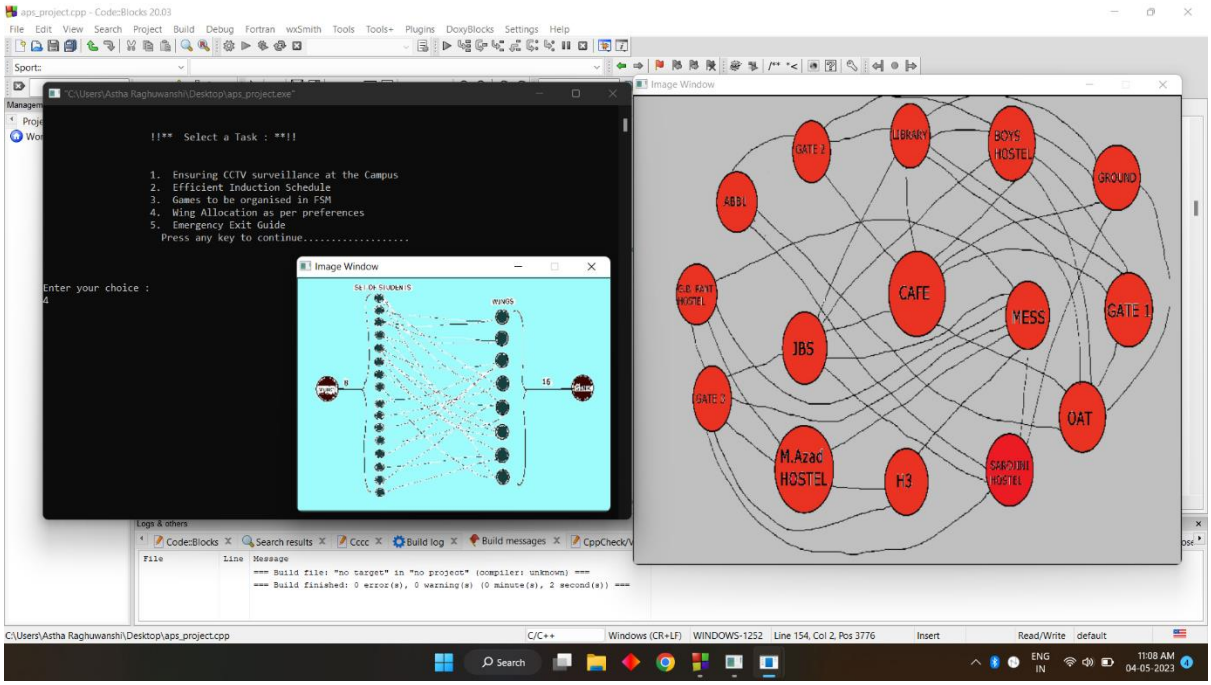
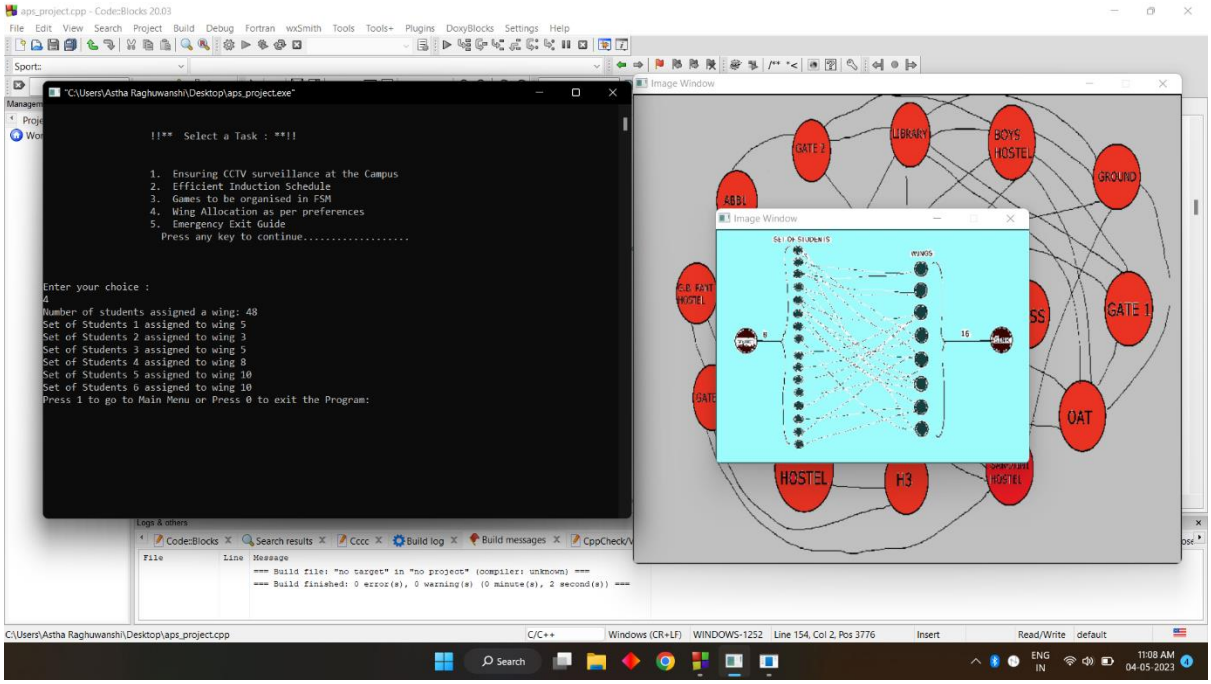


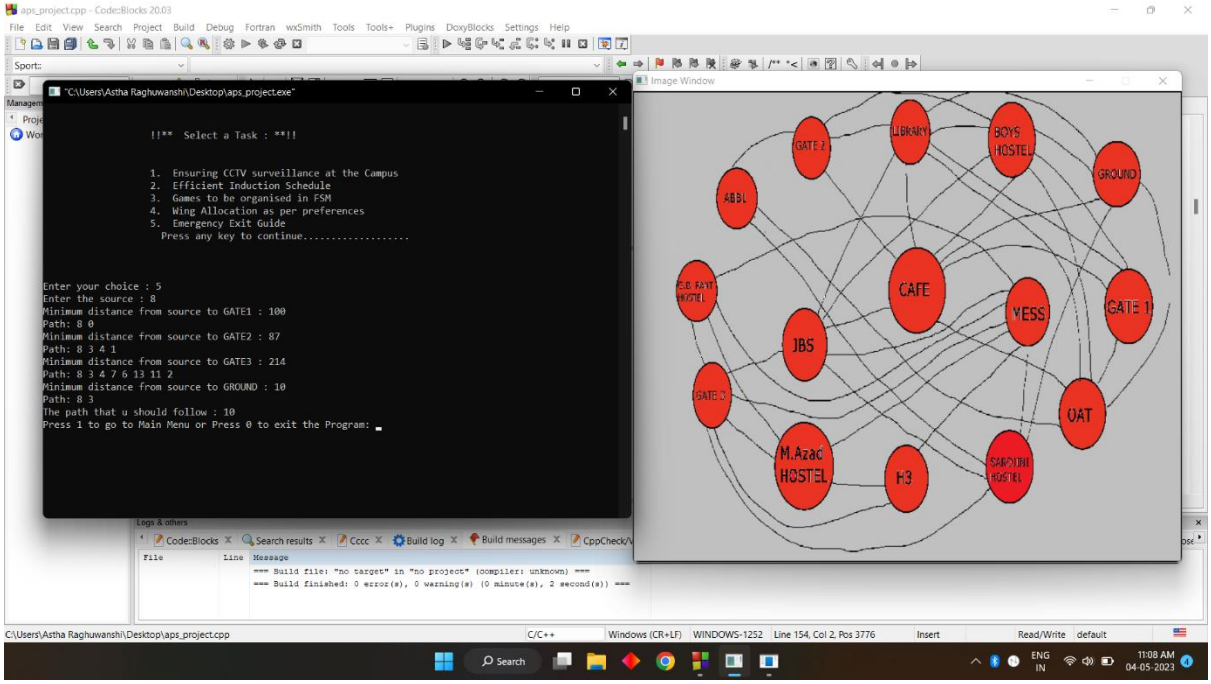
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
CAFE
MESS
GATE 1
JBS
OAT
M.Azad HOSTEL
H3
SAROJINI HOSTEL
Enter your choice : 2


"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice : 2
Edge 0: (0 14)===>>Gate1-Cafe
Edge 1: (14 9)===>>Cafe-BoysHostel
Edge 2: (9 10)===>>BoysHostel-Library
Edge 3: (10 8)===>>Library-OAT
Edge 4: (8 3)===>>OAT-Ground
Edge 5: (3 1)===>>Ground-Gate2
Edge 6: (1 4)===>>Gate2-ABB1
Edge 7: (4 6)===>>ABB1-JBS
Edge 8: (6 7)===>>JBS-Sarojini
Edge 9: (7 5)===>>Sarojini-Mess
Edge 10: (5 12)===>>Mess-H3
Edge 11: (12 13)===>>H3-MaulanaAzad
Edge 12: (13 11)===>>MaulanaAzad-GBpanth
Edge 13: (11 2)===>>GBpanth-Gate3
Minimum cost to cover JP college in meters:1614
Press 1 to go to Main Menu or Press 0 to exit the Program:
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
CAFE
MESS
GATE 1
JBS
OAT
M.Azad HOSTEL
H3
SAROJINI HOSTEL


"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice : 3
Sports to include in the event:
Badminton
Cricket
Squash
Table Tennis
Billiards
Basketball
Volleyball
Swimming
Carrom
Kho-Kho
Maximum excitement generated in the sports meet: 38
Press 1 to go to Main Menu or Press 0 to exit the Program:
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
CAFE
MESS
GATE 1
JBS
GATE 3
OAT
M.Azad HOSTEL
H3
SAROJINI HOSTEL


"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice :
4
Image Window
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
CAFE
MESS
GATE 1
JBS
GATE 3
OAT
M.Azad HOSTEL
H3
SAROJINI HOSTEL


aps_project.cpp - Code::Blocks 20.03
"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice :
4
Number of students assigned a wing: 48
Set of Students 1 assigned to wing 5
Set of Students 2 assigned to wing 3
Set of Students 3 assigned to wing 5
Set of Students 4 assigned to wing 8
Set of Students 5 assigned to wing 10
Set of Students 6 assigned to wing 10
Press 1 to go to Main Menu or Press 0 to exit the Program:
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
GATE 1
OAT
HOSTEL
H3
Code::Blocks
Search results
Build log
Build messages
CppCheck
=== Build file: "no target" in "no project" (compiler: unknown) ===
=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 2 second(s)) ===
C:\Users\Astha Raghuwanshi\Desktop\aps_project.cpp
Windows (CR+LF)
WINDOWS-1252
Line 154, Col 2, Pos 3776
Insert
Read/Write
default
Search
ENG
IN
11:08 AM
04-05-2023


aps_project.cpp - Code::Blocks 20.03
"C:\Users\Astha Raghuwanshi\Desktop\aps_project.exe"
!!** Select a Task : **!!
1. Ensuring CCTV surveillance at the Campus
2. Efficient Induction Schedule
3. Games to be organised in FSM
4. Wing Allocation as per preferences
5. Emergency Exit Guide
Press any key to continue.....
Enter your choice : 5
Enter the source : 8
Minimum distance from source to GATE1 : 100
Path: 8 0
Minimum distance from source to GATE2 : 87
Path: 8 3 4 1
Minimum distance from source to GATE3 : 214
Path: 8 3 4 7 6 13 11 2
Minimum distance from source to GROUND : 10
Path: 8 3
The path that u should follow : 10
Press 1 to go to Main Menu or Press 0 to exit the Program:
Image Window
GATE 2
LIBRARY
BOYS HOSTEL
GROUND
ABBL
CAFE
MESS
GATE 1
JBS
OAT
M.Azad HOSTEL
H3
Code::Blocks
Search results
Build log
Build messages
CppCheck
=== Build file: "no target" in "no project" (compiler: unknown) ===
=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 2 second(s)) ===
C:\Users\Astha Raghuwanshi\Desktop\aps_project.cpp
Windows (CR+LF)
WINDOWS-1252
Line 154, Col 2, Pos 3776
Insert
Read/Write
default
Search
ENG
IN
11:08 AM
04-05-2023