RAILWAY TRACK ALLOCATION

ALGORITHM AND PROBLEM-SOLVING LAB (15B17CI471)

PROJECT SYNOPSIS

Submitted To:
Dr. Anita Sahoo
Dr. Dhanalekshmi G

TEAM MEMBERS:

1) Ayush Gupta	21103016
2) Shreya Agrawal	21103028
3) Kanishk Raj Mittal	21103015
4) Harshit Singh	21103013

▪ PROBLEM STATEMENT:-

This project aims to give an efficient railway route grid of a given area that currently uses roads and present the routes visually.

▪ INTUITION OF THIS PROBLEM-

We are given a map of an area which includes lists of cities, geographical features such as water bodies, forests, savannah, seashores, etc represented by a 2D Array. Our task is to *connect the above-mentioned places through railway lines using the minimum resources* for final implementation by the government of the area

▪ NEED OF IMPLEMENTATION-

While a need for further expansion of railway networks is strongly desired around the globe, the most common solution would be an expansion of the railway network by constructing new stations. However, constructing new infrastructure is generally decided by policymakers and involves a long-term decision process requiring large investments.

Due to high construction costs and budget constraints in India, it is more efficient to increase connectivity of all major metropolitan cities using minimum resources.

▪ CONCEPTS AND PLATFORMS USED:-

1) **Language:** C++
2) **Data structures:** STL, 2D Array, stack-recursion, queue
3) **Algorithms:** A* algorithm, some self-made algorithms, Dynamic Programming, Backtracking, Memoization, Depth first search(DFS)
4) Shortest distance from a point to a line
5) File handling
6) Graphics (OPTIONAL)
7) Version control systems: Git
8) Code collaboration platforms: GitHub

github.com/skywalker290/Track_laying

SOURCE CODE:-

```
#include <bits/stdc++.h>
using namespace std;
void bfs(int sx,int sy,vector<array<int,2>> &markedPath);
const int matmax1=500;
const int matmax2=200;
const int MAXN = 10000;
float dist[100000];
float parent[100000];
// int mat[matmax1][matmax2];
vector<vector<int>>mat;
vector<int>x_cr={5,92,150,195,241};
vector<int>y_cr={7,20,167,158,15};

void makecities()
{
    int i;
    for(i=0;i<x_cr.size();i++)
    {
        if(mat[x_cr[i]][y_cr[i]]=='*' || mat[x_cr[i]][y_cr[i]]=='@')
        {
            mat[x_cr[i]][y_cr[i]]='C';
        }
    }
}



bool isValid(int x, int y, int m, int n,int var) {
    return (x >= 0 && x < m && y >= 0 && y < n && (mat[x][y] == 0 || mat[x][y] == '*' || mat[x][y] == var
||mat[x][y]=='@'));
}






bool isBlocked(int x, int y) {
    return mat[x][y] == 0;
}
```

```
int getNodeId(int x, int y, int n) {
    return x * n + y;
}




priority_queue<pair<float, int>, vector<pair<float, int>>, greater<pair<float, int>>> pq;
void dijkstra(int startX, int startY, int endX, int endY, int m, int n,int var) {
    for (int i = 0; i < m * n; i++) {
        dist[i] = numeric_limits<float>::max();
        parent[i] = -1;
    }
    dist[getNodeId(startX, startY, n)] = 0;
    pq.push(make_pair(0, getNodeId(startX, startY, n)));
    while (!pq.empty()){
        int u = pq.top().second;
        pq.pop();
        int x = u / n;
        int y = u % n;
        if (x == endX && y == endY) {
            // cout<<dist[u]<<endl;
            return;
        }
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0) continue;
                int newX = x + dx;
                int newY = y + dy;
                if (isValid(newX, newY, m, n,var) && !isBlocked(newX, newY)) {
                    int v = getNodeId(newX, newY, n);
                    float weight = dx != 0 && dy != 0 ? 1.414 : 1.0;
                    if (dist[u] + weight < dist[v]) {
                        dist[v] = dist[u] + weight;
                        parent[v] = u;
                        pq.push(make_pair(dist[v], v));
                    }
                }
            }
        }
    }
}
```

```
float SDist(int startX, int startY, int endX, int endY, int m, int n,int var) {
    for (int i = 0; i < m * n; i++) {
        dist[i] = numeric_limits<float>::max();
        parent[i] = -1;
    }
    dist[getNodeId(startX, startY, n)] = 0;
    pq.push(make_pair(0, getNodeId(startX, startY, n)));
    while (!pq.empty()){
        int u = pq.top().second;
        pq.pop();
        int x = u / n;
        int y = u % n;
        if (x == endX && y == endY) {
            // cout<<dist[u]<<endl;
            return dist[u];
        }
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0) continue;
                int newX = x + dx;
                int newY = y + dy;
                if (isValid(newX, newY, m, n,var) && !isBlocked(newX, newY)) {
                    int v = getNodeId(newX, newY, n);
                    float weight = dx != 0 && dy != 0 ? 1.414 : 1.0;
                    if (dist[u] + weight < dist[v]) {
                        dist[v] = dist[u] + weight;
                        parent[v] = u;
                        pq.push(make_pair(dist[v], v));
                    }
                }
            }
        }
    }
}
```

```
vector<array<int,2>> shortestPath(int startX, int startY, int endX, int endY, int m, int n,int var) {
    dijkstra( startX, startY, endX, endY, m, n,var);
    vector<float> path;
    int curr = getNodeId(endX, endY, n);
    while (curr != -1) {
```

```cpp
        path.push_back(curr);
        curr = parent[curr];
    }
    reverse(path.begin(), path.end());
    array<int,2>arr;
    vector<array<int,2>>pp;
    for (int i = 0; i < path.size(); i++) {
        arr[0] = path[i] / n;
        arr[1] = int(path[i]) % n;
        // cout << "(" << x << "," << y << ")";

        pp.push_back(arr);
    }
    return pp;
}
```

```cpp
float Dist(vector<array<int,2>>path){
    float dist=0.00;
    int x=0,y=0;
    int x1=0,y1=0;
    for (int i = 0; i < path.size(); i++) {

        x = path[i][0];
        y = path[i][1];

        if(x1+1==x && y1+1==y){
            dist+=1.414;
        }
        else{
            dist+=1;
        }

        x1=x;y1=y;
    }
    return dist;
}
```

```
void algo1()
{
    //vector<int>x_cr;
    //vector<int>y_cr;
    int m=mat.size()-1;
    int n=mat[0].size();
    int n1=x_cr.size();
    vector<array<int,2>>t;
    vector<array<int,2>>lp;
    int i,k=INT_MIN,st,ed,j,d;
    for(i=0;i<n1;i++)
    {
        for(j=i;j<n1;j++)
        {
            t=shortestPath(x_cr[i],y_cr[j],x_cr[j],y_cr[j],m,n,'*');
            d=Dist(t);
            if(k<d)
            {
                st=i;
                ed=j;
                k=d;
                lp=t;
            }
        }
    }
    for(i=0;i<lp.size();i++)
    {
        mat[lp[i][0]][lp[i][1]]='@';
    }


    for(i=0;i<n1;i++)
    {
        if(i!=st && i!=ed )
        {
            bfs(x_cr[i],y_cr[i],lp);
        }
    }

}


void bfs(int sx,int sy,vector<array<int,2>> &markedPath)
{
    int m=mat.size()-1, n=mat[0].size(); // dimensions of the matrix
```

```
bool visited[MAXN][MAXN]; // keep track of visited cells
int dist[MAXN][MAXN]; // distance from starting point to each cell
pair<int, int> prev[MAXN][MAXN]; // previous cell in the shortest path
memset(visited, false, sizeof visited);
memset(dist, -1, sizeof dist);


// initialize starting point
queue<pair<int, int>> q;
q.push({sx, sy});
visited[sx][sy] = true;
dist[sx][sy] = 0;

// perform BFS
while (!q.empty()) {
   int x = q.front().first, y = q.front().second;
   q.pop();
   int flag=0;
   // check if current cell is on the marked path
   for (auto p : markedPath) {
      int px = p[0], py = p[1];
      if (x == px && y == py)
      {
         flag=1;
         break;

      }

   }
   if(flag)break;

   // explore neighbors
   for (int dx = -1; dx <= 1; dx++) {
      for (int dy = -1; dy <= 1; dy++) {
         if (dx == 0 && dy == 0) continue;
         int nx = x + dx, ny = y + dy;
         if (isValid(nx, ny,m,n,'*') && !visited[nx][ny]) {
            visited[nx][ny] = true;
            dist[nx][ny] = dist[x][y] + 1;
            prev[nx][ny] = {x, y}; // update previous cell
            q.push({nx, ny});
         }
      }
   }
}
// if no path found, return -1
```

```
    //vector<pair<int, int>> path;
    int px,py;
    int x = markedPath[markedPath.size()-1][0], y =  markedPath[markedPath.size()-1][1];
    //path.push_back({x, y});
    while (x != sx || y != sy){
        tie(px, py) = prev[x][y];
        mat[x][y]='@';
        // path.push_back({px, py});
        x = px;
        y = py;
    }
    //reverse(path.begin(), path.end());
    return;
}




void inputmatrix(){
    ifstream infile("kanishkjob.txt");

    char ch;
    int i=0;
    infile.get(ch);
    mat.push_back({});
    while(!infile.eof()){
        mat[i].push_back(ch);
        if(ch=='\n'){
            mat.push_back({});
            i++;
        }
        infile.get(ch);

    }
}






void outputmatrix(){
    ofstream outfile("output.txt");
    int m=mat.size()-1;
    int n=mat[0].size();
    char ch='*';
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            outfile.put(char(mat[i][j]));
        }
    }

}




void setpath(vector<array<int,2>> path){
    int x,y;
    for (int i = 0; i < path.size(); i++) {
        x = path[i][0];
        y = path[i][1];
        mat[x][y]='@';
    }
}



void Algo2(){
    int m=mat.size()-1;
    int n=mat[0].size();
    int I=0;
    int J=0;
    int d=0;
    int counter=1;
    float progress=0.00;
    vector<float>value(x_cr.size(),INT_MAX);
    vector<array<int,2>> path;


    // for(int k=0;k<x_cr.size();k++){
    //     I+=x_cr[k];
    //     J+=y_cr[k];

    //     I=I/x_cr.size();
    //     J=J/x_cr.size();
    // }
    int error=20;
    int dsum=INT_MAX;
```

```
for(int i=0;i<m;i+=5){
    for(int j=0;j<n;j+=error){
        if(mat[i][j]=='*'){
            // dsum=0;
            int sm=0;
            counter+=error;
            // progress=((counter*100)/(m*n));
            progress=counter;
            cout<<i<<" "<<j<<endl;
            for(int k=0;k<x_cr.size();k++){

                // cout<<"("<<progress<<"/"<<(m*n*x_cr.size())/error<<")"<<endl;
                 d=Dist(shortestPath( i, j, x_cr[k], x_cr[k], m, n,'*'));
                //  cout<<d<<endl;


                sm+=d;



            }
            cout<<sm<<endl;
            if(dsum>sm){
                dsum=sm;
                I=i;
                J=j;
            }
        }
    }
}

// for(int k=0;k<x_cr.size();k++){
//     I+=x_cr[k];
//     J+=y_cr[k];

//     I=I/x_cr.size();
//     J=J/x_cr.size();
// }



    cout<<I<<" "<<J<<endl;
for(int k=0;k<x_cr.size();k++){
    vector<array<int,2>> path = shortestPath( I, J, x_cr[k], x_cr[k], m, n,'*');
    int d=Dist(path);
    setpath(path);
    }
    mat[I][J]='#';
    makecities();
```

```
}
```

```
int main() {
    cout<<"\n=============================\n";

    ifstream infile("kanishkjob.txt");
    ofstream outfile("output.txt");
    inputmatrix();


    int m, n;
    m=mat.size()-1;
    n=mat[0].size();
    char ch;

    // for (int i = 0; i < m; i++) {
    //     for (int j = 0; j < n; j++) {
    //         infile.get(ch); mat[i][j]=ch;
    //     }
    // }



    int startX, startY, endX, endY;
    startX=0;
    startY=0;
    endX=160;
    endY=0;
    int par='*';
    // for(int var=1;var<4;var++){
    //     if(var==1){
    //         par='*';
//         cout<<"Normal Path:\n";
//     }
//     else if(var==2){
//         par='W';
//         cout<<"River Path:\n";
//     }
//     else if(var==3){
//         par='X';
//         cout<<"Mountai Path:\n";
//     }
    //  vector<array<int,2>> path = shortestPath( startX, startY, endX, endY, m, n,par);
//     setpath(path);
//     outputmatrix();
//     cout<<Dist(path)<<endl;
// }
// return 0;

// makecities();



Algo2();
int d;
d= SDist( 0, 0, 160, 0, m, n,'*');
cout<<d;




outputmatrix();
// for (int i = 0; i < m; i++) {
//     for (int j = 0; j < n; j++) {
//         outfile.put(char(mat[i][j]));
//     }
// }

cout<<"\n============================\n";


return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
void bfs(int sx,int sy,vector<array<int,2>> markedPath);
const int matmax1=500;
const int matmax2=200;
const int MAXN = 10000;
float dist[100000];
float parent[100000];
// int mat[matmax1][matmax2];
vector<vector<int>>mat;
vector<int>x_cr={5,92,150,195,241};
vector<int>y_cr={7,20,147,158,15};

void makecities()
{
    int i;
    for(i=0;i<x_cr.size();i++)
    {
        if(mat[x_cr[i]][y_cr[i]]=='*')
        {
            mat[x_cr[i]][y_cr[i]]='C';
        }
    }
}

bool isValid(int x, int y, int m, int n,int var) {
    return (x >= 0 && x < m && y >= 0 && y < n && (mat[x][y] == 0 || mat[x][y] == '*' || mat[x][y] == var
||mat[x][y]=='@' ||mat[x][y]=='C'));
}

bool isValid1(int x, int y, int m, int n,int var) {
    return (x >= 0 && x < m && y >= 0 && y < n && (mat[x][y] == 0 || mat[x][y] == '*' || mat[x][y] == var
||mat[x][y]=='@' ));
}

bool isBlocked(int x, int y) {
    return mat[x][y] == 0;
}
int getNodeId(int x, int y, int n) {
    return x * n + y;
}
```

```
void dijkstra(int startX, int startY, int endX, int endY, int m, int n,int var) {
    for (int i = 0; i < m * n; i++) {
        dist[i] = numeric_limits<float>::max();
        parent[i] = -1;
    }
    dist[getNodeId(startX, startY, n)] = 0;
    priority_queue<pair<float, int>, vector<pair<float, int>>, greater<pair<float, int>>> pq;
    pq.push(make_pair(0, getNodeId(startX, startY, n)));
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        int x = u / n;
        int y = u % n;
        if (x == endX && y == endY) {
            return;
        }
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0) continue;
                int newX = x + dx;
                int newY = y + dy;
                if (isValid(newX, newY, m, n,var) && !isBlocked(newX, newY)) {
                    int v = getNodeId(newX, newY, n);
                    float weight = dx != 0 && dy != 0 ? 1.414 : 1.0;
                    if (dist[u] + weight < dist[v]) {
                        dist[v] = dist[u] + weight;
                        parent[v] = u;
                        pq.push(make_pair(dist[v], v));
                    }
                }
            }
        }
    }
}




vector<array<int,2>> shortestPath(int startX, int startY, int endX, int endY, int m, int n,int var) {
    dijkstra( startX, startY, endX, endY, m, n,var);
    vector<float> path;
    int curr = getNodeId(endX, endY, n);
    while (curr != -1) {
        path.push_back(curr);
        curr = parent[curr];
    }
    reverse(path.begin(), path.end());
    array<int,2>arr;
    vector<array<int,2>>pp;
    for (int i = 0; i < path.size(); i++) {
        arr[0] = path[i] / n;
        arr[1] = int(path[i]) % n;
        // cout << "(" << x << "," << y << ")";

        pp.push_back(arr);
    }
    return pp;
}








float Dist(vector<array<int,2>>path){
    float dist=0.00;
    int x=0,y=0;
    int x1=0,y1=0;
    for (int i = 0; i < path.size(); i++) {

        x = path[i][0];
        y = path[i][1];

        if(x1+1==x && y1+1==y){
            dist+=1.414;
        }
        else{
            dist+=1;
        }

        x1=x;y1=y;
    }
    return dist;
}




void algo1()
```

```
{
    //vector<int>x_cr;
    //vector<int>y_cr;
    int m=mat.size()-1;
    int n=mat[0].size();
    int n1=x_cr.size();
    vector<array<int,2>>t;
    vector<array<int,2>>lp;
    int i,k=INT_MIN,st,ed,j,d;
    for(i=0;i<n1;i++)
    {
        for(j=i+1;j<n1;j++)
        {
            t=shortestPath(x_cr[i],y_cr[i],x_cr[j],y_cr[j],m,n,'*');
            d=Dist(t);
            if(k<d)
            {
                st=i;
                ed=j;
                k=d;
                lp=t;
            }
        }
    }

    // for(i=0;i<lp.size();i++)
    // {
    //     cout<<lp[i][0]<<lp[i][1]
    // }

    for(i=0;i<lp.size();i++)
    {
        if( mat[lp[i][0]][lp[i][1]]!='C')
        {
            mat[lp[i][0]][lp[i][1]]='@';
        }
    }
    //cout<<st;
    //cout<<ed;

    for(i=0;i<n1;i++)
     {
        if(i!=st && i!=ed )
         {
            bfs(x_cr[i],y_cr[i],lp);
         }
     }

}

void setpath(vector<array<int,2>> path){
    int x,y;
    for (int i = 0; i < path.size(); i++) {
        x = path[i][0];
        y = path[i][1];
        if(mat[x][y]!='C')
            mat[x][y]='@';
    }
}

void bfs(int sx,int sy,vector<array<int,2>> markedPath)
{
    int m=mat.size()-1, n=mat[0].size(); // dimensions of the matrix
    bool visited[m][n]; // keep track of visited cells
    int dist[m][n]; // distance from starting point to each cell
    pair<int, int> prev[m][n]; // previous cell in the shortest path
    memset(visited, false, sizeof visited);
    memset(dist, -1, sizeof dist);


    // initialize starting point
    queue<pair<int, int>> q;
    q.push({sx, sy});
    visited[sx][sy] = true;
    dist[sx][sy] = 0;
    int flag=0;
    // cout<<markedPath.size()<<" ";
    int I=0;
    int J=0;
    //perform BFS

    while (!q.empty()) {
        int x = q.front().first, y = q.front().second;
        q.pop();

        //cout<<"rs";
        // check if current cell is on the marked path
        // for (auto p : markedPath) {
        //     int px = p[0], py = p[1];
        //     cout<<px<<" "<<py<<endl;
        //     if (x == px && y == py)
        //     {
```

```
    //        flag=1;

    //        break;

    //    }

    // }
    // if(flag)break;
    //cout<<"fdsv";
    if(mat[x][y] =='@')
    {
        I=x;
        J=y;
        break;
    }
    // explore neighbors
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            if (dx == 0 && dy == 0) continue;
            int nx = x + dx, ny = y + dy;
            //cout<<nx<<","<<ny<<endl;
            if (isValid(nx, ny,m,n,'*') && !visited[nx][ny]) {
                visited[nx][ny] = true;
                dist[nx][ny] = dist[x][y] + 1;
                prev[nx][ny] = {x, y}; // update previous cell
                q.push({nx, ny});
            }
        }
    }
}
// if no path found, return -1
//vector<pair<int, int>> path;
int px,py;
int x = markedPath[markedPath.size()-1][0], y =  markedPath[markedPath.size()-1][1];

vector<array<int,2>> path = shortestPath( I, J, sx, sy, m, n,'*');
setpath(path);

//path.push_back({x, y});

// while (x != sx || y != sy){
//    tie(px, py) = prev[x][y];
//    cout<<px<<py
//    if(mat[x][y]!='C')
//    {
//        mat[x][y]='@';
```

```
    //     }

    //     // path.push_back({px, py});
    //     x = px;
    //     y = py;
    // }
    //reverse(path.begin(), path.end());
    return;
}




void inputmatrix(){
    ifstream infile("kanishkjob.txt");

    char ch;
    int i=0;
    infile.get(ch);
    mat.push_back({});
    while(!infile.eof()){
        mat[i].push_back(ch);
        if(ch=='\n'){
            mat.push_back({});
            i++;
        }
        infile.get(ch);

    }
}





void outputmatrix(){
    ofstream outfile("output.txt");
    int m=mat.size()-1;
    int n=mat[0].size();
    char ch='*';


    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            outfile.put(char(mat[i][j]));
        }
    }

}









// void Algo2(){
//     int m=mat.size()-1;
//     int n=mat[0].size();
//     vector<double>value(x_cr.size(),INT_MAX);
//     for(int i=0;i<m;i++){
//         for(int j=0;j<n;j++){
//             for(int k=0;k<x_cr.size();k++){
//                 vector<array<int,2>> path = shortestPath( i, j, x_cr[k], x_cr[k], m, n,'*');
//                 value[k]=min(Dist(path),value[k]);
//             }

//         }
//     }





// }






int main() {
    cout<<"\n==============================\n";

    ifstream infile("kanishkjob.txt");
    ofstream outfile("output.txt");
    inputmatrix();


    int m, n;
    m=mat.size()-1;
```

```
n=mat[0].size();
char ch;

// for (int i = 0; i < m; i++) {
//     for (int j = 0; j < n; j++) {
//         infile.get(ch); mat[i][j]=ch;
//     }
// }



int startX, startY, endX, endY;
startX=0;
startY=160;
endX=234;
endY=76;
int par='*';
// for(int var=1;var<4;var++){
//     if(var==1){
//         par='*';
//         cout<<"Normal Path:\n";
//     }
//     else if(var==2){
//         par='W';
//         cout<<"River Path:\n";
//     }
//     else if(var==3){
//         par='X';
//         cout<<"Mountai Path:\n";
//     }
//     vector<array<int,2>> path = shortestPath( startX, startY, endX, endY, m, n,par);
//     setpath(path);
//     outputmatrix();
//     cout<<Dist(path)<<endl;
// }
// return 0;

makecities();
algo1();

outputmatrix();
// for (int i = 0; i < m; i++) {
//     for (int j = 0; j < n; j++) {
//         outfile.put(char(mat[i][j]));
//     }
// }
```

```
    cout<<"\n=============================\n";


    return 0;
}
```

OUTPUTS:-