**N Queen Problem with Genetic Algorithm**

#include<stdio.h>

int N,Q,B[10][10],P[100][100],n,F[100];

float Fit[100];

main()

{

int i,j,sum=0;

printf("Total Population =");

scanf("%d",&N);

printf("Total Queens = ");

scanf("%d",&Q);

for(i=1;i<=N;i++)

{

for(j=1;j<=Q;j++)

{

n=random();

B[j][n]=1;

P[i][j]=n;

printf("%d",P[i][j]);

}

printf("\n\n");

Board();

F[i] = 28 - fit();

sum = sum + F[i];

initial();

printf("F[%d]=%d\n",i,F[i]);

}

printf("Sum = %d\n",sum);

for(i = 1;i <= N;i++)

{

printf("%d %d\n",F[i],sum);

Fit[i] = F[i] / sum \* 100;

printf("Fit[%d] = %.2f\n",i,Fit[i]);

}

}

initial()

{

int i,j;

for(i = 1;i <= 8; i++)

{

for (j = 1; j <= 8; j++)

{

B[i][j]=0;

}

}

}

int random()

{

int random = rand() % Q + 1;

return random;

}

Board()

{

int i,j;

for(i=1;i<=Q;i++)

{

for(j=1;j<=Q;j++)

{

printf("%d ",B[i][j]);

}

printf("\n\n");

}

}

fit()

{

int count=0;

int i,j;

for(i = 1;i <= Q; i++)

{

for (j = 1; j <= Q; j++)

{

if((B[i][j]==1) && (attack(i,j)==1))

{

count++;

}

}

}

return count;

}

attack(int i,int j,int count)

{

int k,l;

for(k=i+1;k<=Q;k++)

{

if(B[k][j] == 1)

{

printf("[%d,%d],[%d,%d] Attack side\n",i,j,k,j);

return 1;

}

}

for(k=i+1;k<=Q;k++)

{

for(l=1;l<=Q;l++)

{

if(((k+l) == (i+j)) || ((k-l) == (i-j)))

{

if(B[k][l] == 1)

{

printf("[%d,%d],[%d,%d] Attack diagonal\n",i,j,k,j);

return 1;

}

}

}

}

return 0;

}

**A\* Search For Romania Problem**

#include<bits/stdc++.h>

using namespace std;

#define DISCONNECTED -1

int num\_of\_node,num\_of\_edge,graph[30][30],pathCost[30];

void init\_Heuristic();

struct Node{

int from,to;

int cost;

};

struct CompareNode{

bool operator()(Node& n1, Node& n2){

if(n1.cost > n2.cost)return true;

return false;

}

};

map<int,int>Heuristic;

priority\_queue<Node, vector<Node>, CompareNode> PQ;

vector<Node>path;

void AStarSearch(int source,int destination){

init\_Heuristic();

for(int i = 1; i <= num\_of\_node; i++){

if(graph[source][i] != DISCONNECTED){

Node n;

n.from = source;

n.to = i;

n.cost = graph[source][i] + Heuristic[i];

pathCost[i] = graph[source][i];

PQ.push(n);

}

}

while(!PQ.empty()){

Node tmp = PQ.top();

path.push\_back(tmp);

if(tmp.to == destination)break;

PQ.pop();

for(int i = 1; i <= num\_of\_node; i++){

if(graph[tmp.to][i] != DISCONNECTED){

Node n;

n.from = tmp.to;

n.to = i;

n.cost = pathCost[tmp.to] + graph[tmp.to][i] + Heuristic[i];

pathCost[i] = pathCost[tmp.to] + graph[tmp.to][i];

PQ.push(n);

}

}

}

}

int main(){

int a,b,c,source,destination;

cout << "Enter Node: " << endl;

cin >> num\_of\_node;

cout << "Enter Edge: " << endl;

cin >> num\_of\_edge;

for(int i=1; i<=num\_of\_node; i++)

for(int j=1; j<=num\_of\_node; j++)

graph[i][j] = DISCONNECTED;

for(int i = 0; i < num\_of\_edge; i++){

cin >> a >> b >> c;

graph[a][b] = graph[b][a] = c;

}

cout << "Enter source: " << endl;

cin >> source;

cout << "Enter destination: " << endl;

cin >> destination;

AStarSearch(source,destination);

for(int i = 0; i < path.size(); i++)

cout << path.at(i).from << " -> " << path.at(i).to << " = " << path.at(i).cost << endl;

return 0;

}

void init\_Heuristic(){

///straight line distance ///

Heuristic[1] = 380; Heuristic[13] = 0;

Heuristic[2] = 374; Heuristic[14] = 77;

Heuristic[3] = 253; Heuristic[15] = 80;

Heuristic[4] = 366; Heuristic[16] = 151;

Heuristic[5] = 329; Heuristic[17] = 161;

Heuristic[6] = 193; Heuristic[18] = 199;

Heuristic[7] = 176; Heuristic[19] = 226;

Heuristic[8] = 244; Heuristic[20] = 234;

Heuristic[9] = 241;

Heuristic[10] = 242;

Heuristic[11] = 160;

Heuristic[12] = 100;

}

///INPUT DATA FOR ROMANIA MAP///

/\*

20 23

1 2 71

1 3 151

2 4 75

3 6 80

3 7 99

3 4 140

4 5 118

5 8 111

6 11 146

6 12 97

7 13 211

8 9 70

9 10 75

10 11 120

11 12 138

12 13 101

13 14 90

13 15 85

15 16 98

15 18 142

16 17 86

18 19 92

19 20 87

\*/