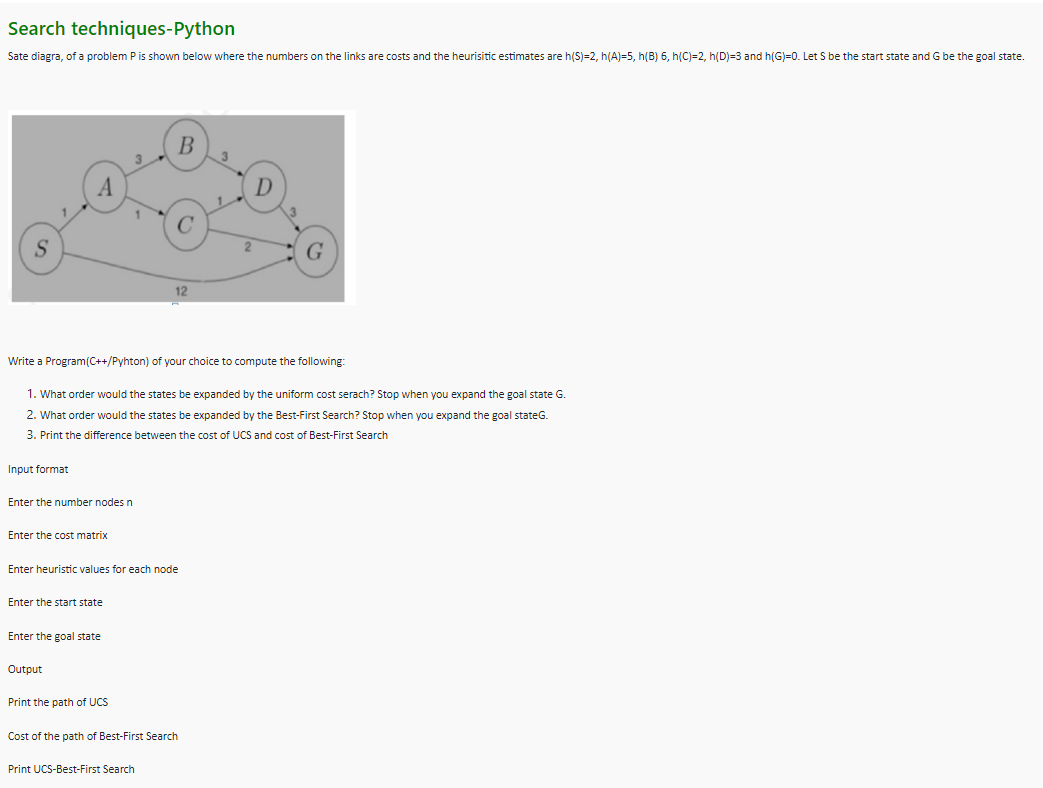
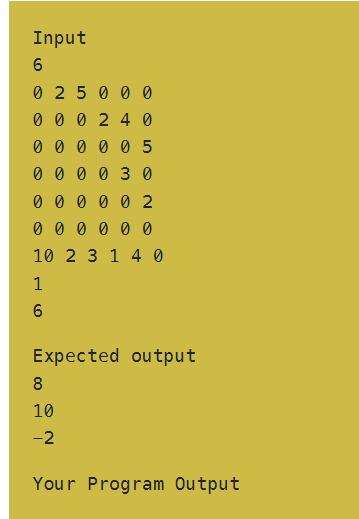
Uniform Cost Search & Best First Search

Question



Public Test Case



Code:

#include "iostream"

#include "vector"

#include "algorithm"

#define inf 10e6

using namespace std;

struct link{

  int dest,cost;

};

void pushChild(vector<link> &q,int \*\*g,int n,int index,int \*cost,int \*visited,int \*parent){

  link child;

  for(int i=0;i<n;++i){

    if(g[index][i] && !visited[i] && cost[i] > cost[index] + g[index][i]){

      child.dest = i;

      cost[i] = cost[index] + g[index][i];

      child.cost = cost[i];

      q.push\_back(child);

      parent[i] = index;

    }

  }

}

bool higher(link a,link b){

  return a.cost > b.cost;

}

void print(vector<link> q){

  for(int i=0;i<q.size();++i)

    cout<<q[i].dest+1<<" ";

  cout<<"\n";

}

void trace(vector<int> &ar,int \*parent,int i){

  if(parent[i] == i){

    ar.push\_back(i);

    return;

  }

  else{

    ar.push\_back(i);

    trace(ar,parent,parent[i]);

  }

}

int uniCostSearch(int \*\*g,int start,int n,int dest){

  int cur = start,key;

  int \*parent = new int[n];

  int \*cost = new int[n];

  vector<int> path;

  vector<link> queue;

  int \*visited = new int[n];

  for(int i=0;i<n;++i){

    visited[i] = 0;

    parent[i] = i;

    cost[i] = inf;

  }

  cost[start] = 0;

  visited[start] = 1;

  pushChild(queue,g,n,start,cost,visited,parent);

  while(1){

    if(queue.empty()){

      cout<<"Path Not Possible\n";

      break;

    }

    sort(queue.begin(),queue.end(),higher);

    key = queue[queue.size() - 1].dest;

    queue.pop\_back();

    if(key == dest){

      parent[key] = cur;

      break;

    }

    cur = key;

    visited[key] = 1;

    pushChild(queue,g,n,key,cost,visited,parent);

  }

  trace(path,parent,dest);

  reverse(path.begin(),path.end());

  return cost[path[path.size() - 1]];

}

void pushChild2(vector<link> &q,int \*\*g,int n,int index,int \*cost,int \*visited,int \*parent,int \*h){

  link child;

  for(int i=0;i<n;++i){

    if(g[index][i] && !visited[i] && cost[i] > cost[index] + g[index][i]){

      child.dest = i;

      cost[i] = cost[index] + g[index][i];

      child.cost = h[i];

      q.push\_back(child);

      parent[i] = index;

    }

  }

}

int bestFirstSearch(int \*\*g,int start,int n,int dest,int \*h){

  int cur = start,key;

  vector<int> path;

  vector<link> queue;

  int \*visited = new int[n];

  int \*parent = new int[n];

  int \*cost = new int[n];

  for(int i=0;i<n;++i){

    visited[i] = 0;

    parent[i] = i;

    cost[i] = inf;

  }

  cost[start] = 0;

  visited[start] = 1;

  pushChild2(queue,g,n,start,cost,visited,parent,h);

  while(1){

    if(queue.empty()){

      cout<<"Path Not Possible\n";

      break;

    }

    sort(queue.begin(),queue.end(),higher);

    key = queue[queue.size() - 1].dest;

    queue.pop\_back();

    if(key == dest){

      parent[key] = cur;

      break;

    }

    cur = key;

    visited[key] = 1;

    pushChild2(queue,g,n,key,cost,visited,parent,h);

  }

  trace(path,parent,dest);

  reverse(path.begin(),path.end());

  return cost[path[path.size() - 1]];

}

int main(){

  int n,start,end,UCS,BFS;

  cin>>n;

  int \*\*g = new int\*[n];

  int \*h = new int[n];

  for(int i=0;i<n;++i){

    g[i] = new int[n];

    for(int j=0;j<n;++j)

      cin>>g[i][j];

  }

  for(int i=0;i<n;++i)

    cin>>h[i];

  cin>>start>>end;

  start--;end--;

  UCS = uniCostSearch(g,start,n,end);

  BFS = bestFirstSearch(g,start,n,end,h);

  cout<<UCS<<"\n"<<BFS<<"\n"<<UCS-BFS<<"\n";

}