

**Department of Computer Science and Engineering**

**Project Report**

**Course Title**: Artificial Intelligence

**Course Code**: CSE366

**Semester**: Fall-2020

**Section**: 01

**Project Title**: Comparison between A\* Tree Search and A\* Graph Search

**Submitted To**

Amit Kumar Das

Senior Lecturer

Department of Computer Science and Engineering

**Submitted By**

Syeda Tamanna Sheme

ID: 2018-2-60-010

**Date of Submission:** 01-12-2021

**Project Title**: Comparison between A\* Tree Search and A\* Graph Search

**Project Description:** A\* search is a graph traversal and path finding algorithm. It is mainly used in real life situations because of its efficiency. It finds the path from one node to another with the lowest cost possible. A\* search uses a heuristic value, which is an estimated cost of reaching the destination point.

A\* uses a function *f*(n), which can be written as

*f*(n) = *g*(n) + *h*(n), where *f*(n) = total estimated path cost

*g*(n) = cost to reach to a node

*h*(n) = heuristic value of a node

**A\* Tree VS A\* Graph:**

A\* tree search traverses the whole graph to find a path with the lowest cost possible. It can expand/ traverse the same node multiple times to get the lowest cost path. As it has to traverse each node possibly multiple times, the time to calculate a path takes much longer depending on the size of the graph.

A\* graph search traverses a node only one time, if a node is already expanded, that node will not be visited again. Thus the final path might not give the lowest cost, but it is much faster.

**A\* Tree Search Code:**

#include<bits/stdc++.h>

using namespace std;

#define MAX 1000000

int n,e,dest;

int traverse\_count = 0;

vector <int> adj[MAX];

vector <int> adjw[MAX];

int dis[MAX];

int H[MAX];

int F[MAX];

int par[MAX];

int path[MAX];

queue <int> q;

void a\_star\_tree(int s)

{

dis[s] = 0;

F[s] = 0+H[s];

par[s] = s;

traverse\_count++;

q.push(s);

int u,v;

while(!q.empty())

{

u = q.front();

q.pop();

for (int i=0; i<adj[u].size(); i++)

{

v = adj[u][i];

if(v == dest)

{

dis[v] = dis[u]+adjw[u][i];

if(F[v]>dis[v]+H[v])

{

F[v] = dis[v]+H[v];

par[v] = u;

}

traverse\_count++;}

else

{

q.push(v);

dis[v] = dis[u]+adjw[u][i];

if(F[v]>dis[v]+H[v])

{

F[v] = dis[v]+H[v];

par[v] = u;

}

traverse\_count++;

}

}

}

}

int main()

{

char a[100], b[100],c[100],node[100],edge[100],h[100];

fstream file;

file.open("nodeandedges.txt",ios::in);

while(!file.eof())

{

file.getline(node,100,'|');

file.getline(edge,100);

stringstream int1(node);

stringstream int2(edge);

int1>>n;

int2>>e;

}

file.close();

int u, v, w;

file.open("input.txt",ios::in);

while(!file.eof())

{

file.getline(a,100,'|');

file.getline(b,100,'|');

file.getline(c,100);

stringstream int1(a);

stringstream int2(b);

stringstream int3(c);

int1>>u;

int2>>v;

int3>>w;

adj[u].push\_back(v);

adjw[u].push\_back(w);

}

file.close();

file.open("heuristic.txt",ios::in);

int i = 1;

while(!file.eof())

{

file.getline(h,100);

stringstream int1(h);

int1>>H[i];

i++;

}

file.close();

for(int i=0; i<=n; i++)

{

dis[i] = -1;

F[i] = 99999;

}

int s;

cout<<"Enter source:";

cin>>s;

cout<<"Enter destination:";

cin>>dest;

a\_star\_tree(s);

cout<<"\nCost from source using A-star Tree search: "<<F[dest]<<endl;

cout<<endl;

int current = dest;

int cnt = 0;

while(current!=s)

{

path[cnt] = current;

current = par[current];

cnt++;

}

path[cnt] = current;

cout<<"Path from "<<s<<" to "<<dest<<": ";

for(int i=cnt;i>0;i--)

{

cout<<path[i]<<"->";

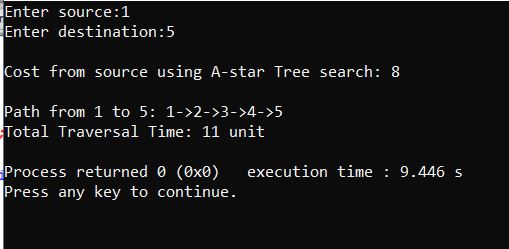
}

cout<<path[0]<<endl;

cout<<"Total Traversal Time: "<<traverse\_count<<" unit"<<endl;

}

**A\* Tree Search Output:**



**A\* Graph Search Code:**

#include<bits/stdc++.h>

using namespace std;

#define MAX 1000000

int n,e,dest;

int traverse\_count = 0;

vector <int> adj[MAX];

vector <int> adjw[MAX];

int dis[MAX];

int H[MAX];

int F[MAX];

int par[MAX];

int path[MAX];

queue <int> q;

void a\_star\_graph(int s)

{

dis[s] = 0;

F[s] = 0+H[s];

par[s] = s;

traverse\_count++;

q.push(s);

int u,v;

while(!q.empty())

{

u = q.front();

q.pop();

for (int i=0; i<adj[u].size(); i++)

{

v = adj[u][i];

if(v == dest)

{

dis[v] = dis[u]+adjw[u][i];

if(F[v]>dis[v]+H[v])

{

F[v] = dis[v]+H[v];

par[v] = u;

}

traverse\_count++;

}

else

{

if(dis[v] == -1)

{

q.push(v);

dis[v] = dis[u]+adjw[u][i];

if(F[v]>dis[v]+H[v])

{

F[v] = dis[v]+H[v];

par[v] = u;

}

traverse\_count++;

}

}

}

}

}

int main()

{

char a[100], b[100],c[100],node[100],edge[100],h[100];

fstream file;

file.open("nodeandedges.txt",ios::in);

while(!file.eof())

{

file.getline(node,100,'|');

file.getline(edge,100);

stringstream int1(node);

stringstream int2(edge);

int1>>n;

int2>>e;

}

file.close();

int u, v, w;

file.open("input.txt",ios::in);

while(!file.eof())

{

file.getline(a,100,'|');

file.getline(b,100,'|');

file.getline(c,100);

stringstream int1(a);

stringstream int2(b);

stringstream int3(c);

int1>>u;

int2>>v;

int3>>w;

adj[u].push\_back(v);

adjw[u].push\_back(w);

}

file.close();

file.open("heuristic.txt",ios::in);

int i = 1;

while(!file.eof())

{

file.getline(h,100);

stringstream int1(h);

int1>>H[i];

i++;

}

file.close();

for(int i=0; i<=n; i++)

{

dis[i] = -1;

F[i] = 99999;

}

int s;

cout<<"Enter source:";

cin>>s;

cout<<"Enter destination:";

cin>>dest;

a\_star\_graph(s);

cout<<"\nCost from source using A-star Graph search: "<<F[dest]<<endl;

cout<<endl;

int current = dest;

int cnt = 0;

while(current!=s)

{

path[cnt] = current;

current = par[current];

cnt++;

}

path[cnt] = current;

cout<<"Path from "<<s<<" to "<<dest<<": ";

for(int i=cnt;i>0;i--)

{

cout<<path[i]<<"->";

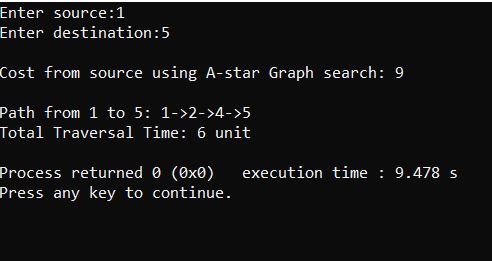
}

cout<<path[0]<<endl;

cout<<"Total Traversal Time: "<<traverse\_count<<" unit"<<endl;

}

**A\* Graph Search Output:**



**Comparison:**

In A\* tree, the lowest cost found from path 1 to 5 is 8, where it took 11 units of time (if we consider visiting 1 node costs 1 unit time), where in A\* graph search, the lowest cost found is 9 but the time it took is only 6 units.

**Conclusion:**

Between A\* tree and A\* graph, there is no exact better than another conclusion. Rather the efficiency depends on the demand of the user. For accuracy, A\* tree is better. For time efficiency, A\* graph is better.