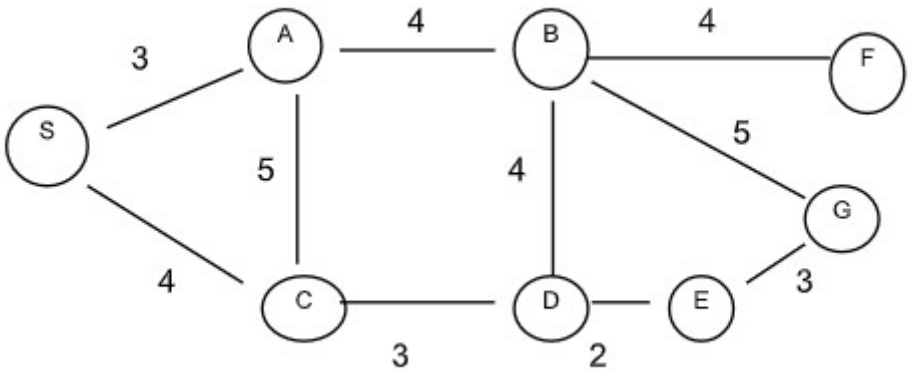
Experiment No : 4

Aim : Write a C/C++/Java program to obtain the sequence of nodes expanded if an A* Search was applied to the diagram below. Assume that the heuristic estimation of distances from each of the states to G are: (G is a goal state)
1. S=18.5 2. A=10.5 3. B=6 4. C=9.2 5. D=6.2 6. E=4.5 7. F=infinity.



Theory : A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied. The search is based on the formula
f(n) = g(n) + h(n)

Code :
```
#include<iostream>
#include<bits/stdc++.h>
#define INF 100000
using namespace std;
typedef pair<char,char> edgePair ;
typedef struct node{
char name;
char parent;
float f;
float g;
bool operator<(const struct node& rhs)const{
return f < rhs.f;
}
}node;
vector<node> open;
map<char,bool> closed;
vector<node> path;
map<char,list<char>>edges;
map<edgePair,float> weights;
map<char,float> hCost;
bool edgeExists(char src,char dst){
auto end=edges[src].end();
auto start=edges[src].begin();
```

```
return find(start,end,dst)!=end;
}
void addEdge(char src,char dst,float weight,float h_src,float h_dst){
// Check if edge already added
if(edgeExists(src,dst)){
return;
}
// Add to edges list
edges[src].push_back(dst);
edges[dst].push_back(src);
// Add to weights list
weights[edgePair(src,dst)]=weight;
weights[edgePair(dst,src)]=weight;
// Add to hCost array
if(src!='G')
hCost[src]=h_src;
if(dst!='G')
hCost[dst]=h_dst;
}
void generateGraph(){
addEdge('S','A',3,18.5,10.5);
addEdge('S','C',4,18.5,9.2);
addEdge('A','C',5,10.5,9.2);
addEdge('A','B',4,10.5,6);
addEdge('C','D',3,9.2,6.2);
addEdge('D','E',2,6.2,4.5);
addEdge('E','G',3,4.5,0);
addEdge('B','G',5,6,0);
addEdge('B','F',4,6,INF);
}
void addNode(vector<node> &l,node &n){
l.insert(upper_bound(l.begin(),l.end(),n),n);
}
vector<node>::iterator findNode(vector<node> &l,char name){
vector<node>::iterator i;
for(i=l.begin();i!=l.end();i++){
if((*i).name==name){
break;
}
}
return i;
}
void updateNode(vector<node> &l,char name,char parent,float f,float g){
auto i=findNode(l,name);
node n = node{
name:name,
parent:parent,
f:f,
g:g,};
if(i==l.end()){
addNode(l,n);
```

```
return;
}
else if(f<(*i).f){
l.erase(i);
addNode(l,n);
}
return;
}
float tracePath(char last){
vector<node>::iterator it;
it=findNode(path,last);
if((*it).parent=='-'){
cout<<last;
return 0;
}
float cost=tracePath((*it).parent);
cout<<" "<<last<<" ";
return cost + weights[edgePair((*it).parent,last)];
}
void aStarSearch(char src,char dst){
for(auto i:edges){
closed[i.first]=false;
}
node start=node{
name:src,
parent:'-',
f:0,
g:0,
};
addNode(open,start);
while(open.size()!=0){
node curr=*open.begin();
open.erase(open.begin());
closed[curr.name]=true;
path.push_back(curr);
float f,g,h;
for(auto i:edges[curr.name]){
if(i==dst){
cout<<"Optimal path found:";
float costAtEnd=weights[edgePair(curr.name,dst)];
float cost=tracePath(curr.name)+costAtEnd;
cout<<dst<<endl;
cout<<"Total cost is: "<<cost<<endl;
return;
}
if(closed[i]){
continue;
}
g=curr.g + weights[edgePair(curr.name,i)];
h=hCost[i];
f=g+hCost[i];
```

```
updateNode(open, i,curr.name,f,g);
}
}
cout<<"Failed"<<endl;
}
int main(){
generateGraph();
aStarSearch('S','G');
}
```

Output :
Optimal path found:S A B G
Total cost is: 12

Conclusion : C/C++/Java program to obtain the sequence of nodes expanded if an A* Search is applied was written and executed successfully.

Deepraj Bhosale
181105016