Assignment No: 8

**Title:**

Use heuristic search techniques to implement best first search and A\* algorithm.

**Theory:**

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function

Best first search algorithm:

* **Step 1:** Place the starting node into the OPEN list.
* **Step 2:** If the OPEN list is empty, Stop and return failure.
* **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
* **Step 4:** Expand the node n, and generate the successors of node n.
* **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
* **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
* **Step 7:** Return to Step 2.

**Program:**

**BFS.java:**

import java.util.\*;

public class Bfs {

static HashMap<String,Integer> hm=new HashMap();

static ArrayList<String> close=new ArrayList();

static HashMap<String,Integer> open=new HashMap();

static Scanner sc=new Scanner(System.in);

static String arr[][]=new String[20][20];

static int getVertex=0;

public static void accept(){

System.out.println("enter the number of vertes");

getVertex=sc.nextInt();

for(int i=0;i<getVertex;i++){

System.out.println("Enter the vertex and heursitic value");

String ver=sc.next();

int heu=sc.nextInt();

hm.put(ver,heu);

arr[i][0]=ver;

System.out.println("Enter no of childrens");

int child=sc.nextInt();

for(int k=0;k<child;k++){

System.out.println("Enter the child for "+ver);

arr[i][k+1]=sc.next();

}

}

for(Map.Entry m:hm.entrySet())

System.out.println(m.getKey()+" "+m.getValue());

for(int j=0;j<getVertex;j++){

System.out.println();

for(int k=0;k<getVertex;k++) {

if(arr[j][k]==null)

continue;

System.out.print(" "+arr[j][k]);}

}

}

public static void bfs(){

int i=1,j=0;

System.out.println("Enter the starting and end node");

String start=sc.next();

String end=sc.next();

close.add(start);

while(!(start.equals(end))){

for(int k=0;k<getVertex;k++){

if(arr[k][0].equals(start) && arr[k][0]!=null){

for(int h=1;h<=getVertex;h++){

if(arr[k][h]!=null) {

open.put(arr[k][h],hm.get(arr[k][h]));

}

}

break;

}

}

start=min();

}

}

public static String min(){

String start="";

int k=0;

int min=1000;

for(Map.Entry m:open.entrySet()){

System.out.print(m.getKey()+" "+m.getValue());

if((int)m.getValue() < min ){

min=(int)m.getValue();

start=(String)m.getKey();}

}

close.add(start);

open.remove(start);

System.out.println();

System.out.println("min and start"+min+start);

System.out.println("Open after removing");

System.out.println();

for(Map.Entry m:open.entrySet())

System.out.print(m.getKey()+" "+m.getValue());

System.out.println("closed after removing");

System.out.print(close);

return start;

}

public static void main(String[] args) {

accept();

bfs();

System.out.println();

System.out.print(close);

}

}

**Astar.java:**

import java.util.\*;

public class astar {

static HashMap<String,Integer> hm=new HashMap();

static ArrayList<String> close=new ArrayList();

static HashMap<String,Integer> open=new HashMap();

static int mn=0;

static Scanner sc=new Scanner(System.in);

static String arr[][]=new String[20][20];

static int getVertex=0;

public static void accept(){

System.out.println("enter the number of vertices");

getVertex=sc.nextInt();

for(int i=0;i<getVertex;i++){

System.out.println("Enter the vertex and heursitic value");

String ver=sc.next();

int heu=sc.nextInt();

hm.put(ver,heu);

arr[i][0]=ver;

System.out.println("Enter no of children");

int child=sc.nextInt();

for(int k=1;k<=child\*2;k++){

System.out.println("Enter the child for "+ver);

arr[i][k++]=sc.next();

System.out.println("Enter the value for "+ver);

arr[i][k]=sc.next();

}

}

for(Map.Entry m:hm.entrySet())

System.out.println(m.getKey()+" "+m.getValue());

for(int j=0;j<getVertex;j++){

System.out.println();

for(int k=0;k<getVertex\*2;k++) {

if(arr[j][k]==null)

continue;

System.out.print(" "+arr[j][k]);}

}

}

public static void bfs(){

int g=0,f=0,flag=0;

System.out.println("Enter the starting and end node");

String start=sc.next();

String end=sc.next();

f=g+hm.get(start);

close.add(start);

// open.put(start,f);

while(!(start.equals(end))){

for(int k=0;k<getVertex;k++){

//System.out.println("outer loop "+k);

if(arr[k][0].equals(start) && arr[k][0]!=null){

for(int h=1;h<=getVertex\*2;h=h+2){

if(arr[k][h]!=null) {

g=mn+Integer.parseInt(arr[k][h+1]);

flag=1;

System.out.println(g);

f=g+hm.get(arr[k][h]);

System.out.println(f);

open.put(arr[k][h],f);

}

}

break;

}

}

if(flag==0){

System.out.println("The "+start+" node has no child ...Plz reenter");

break;

}

start=min(start);

}

}

public static String min(String ma){

String start="";

int min=1000;

for(Map.Entry m:open.entrySet()){

System.out.print(m.getKey()+" "+m.getValue());

if((int)m.getValue() < min ){

min=(int)m.getValue();

start=(String)m.getKey();}

}

open.clear();

for(int i=0;i<getVertex;i++){

if(arr[i][0]!=null && arr[i][0].equals(ma)){

for(int j=1;j<=getVertex\*2;j=j+2){

if( arr[i][j]!=null && arr[i][j].equals(start)){

mn=mn+Integer.parseInt(arr[i][j+1]);}

}

}

}

System.out.println();

System.out.println("mn is"+mn);

close.add(start);

//open.remove(start);

System.out.println();

System.out.println("min and start"+min+start);

System.out.println("Open after removing");

System.out.println();

for(Map.Entry m:open.entrySet())

System.out.print(m.getKey()+" "+m.getValue());

System.out.println("closed after removing");

System.out.print(close);

return start;

}

public static void main(String[] args) {

accept();

bfs();

System.out.println();

System.out.print(close);

}

}

**Output:**

E:\4TH Year\Assignment no 9> java astar

enter the number of vertices

4

Enter the vertex and heursitic value

a

3

Enter no of children

2

Enter the child for a

d

Enter the value for a

4

Enter the child for a

b

Enter the value for a

7

Enter the vertex and heursitic value

b

6

Enter no of children

1

Enter the child for b

f

Enter the value for b

3

Enter the vertex and heursitic value

c

5

Enter no of children

0

Enter the vertex and heursitic value

d

8

Enter no of children

2

Enter the child for d

v

Enter the value for d

3

Enter the child for d

q

Enter the value for d

3

a 3

b 6

c 5

d 8

a d 4 b 7

b f 3

c

d v 3 q 3Enter the starting and end node

a

d

4

12

7

13

b 13d 12

mn is4

min and start12d

Open after removing

closed after removing

[a, d]

[a, d]

E:\4TH Year\Assignment no 9> java Bfs

enter the number of vertes

4

Enter the vertex and heursitic value

a

4

Enter no of childrens

3

Enter the child for a

b

Enter the child for a

c

Enter the child for a

d

Enter the vertex and heursitic value

b

3

Enter no of childrens

1

Enter the child for b

g

Enter the vertex and heursitic value

c

5

Enter no of childrens

0

Enter the vertex and heursitic value

d

1

Enter no of childrens

1

Enter the child for d

h

a 4

b 3

c 5

d 1

a b c d

b g

c

d hEnter the starting and end node

a

d

b 3c 5d 1

min and start1d

Open after removing

b 3c 5closed after removing

[a, d]

[a, d]