NAME: GUNPUTH Lokesh Kumar

LECTURER: Mr. BEEHARRY Shiam

MODULE: AI

TP4

Task1:

Task 1 – Breadth First Search

I. Create a graph with at least 10 nodes.

```
/* Facts */
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).
edge(d, h).
edge(e, h).
edge(f, i).
edge(g, i).

/* Rules */
connected(X, Y) :-
  edge(X, Y).
connected(X, Y) :-
  edge(Y, X).
```

```
| ?- connected(a, X).

X = b ? ;

X = c ? ;

no
| ?- connected(b, X).

X = d ? ;

X = e ? ;

X = a ? ;

no
| ?- connected(c, X).

X = f ? ;

X = g ? ;

X = a ? ;

(16 ms) no
| ?- connected(d, X).

X = h ? ;

X = b ? ;

no
```

```
| ?- connected(e, X).

X = i ? ;

X = b ? ;

no
| ?- connected(f, X).

X = i ? ;

X = c ? ;

no
| ?- connected(g, X).

X = i ? ;

X = c ? ;
```

```
| ?- connected(h, X).

X = d ? ;

X = e ? ;

no
| ?- connected(i, X).

X = f ? ;

X = g

(15 ms) yes
| ?- connected(j, X).

no
| ?-
```

II. Write a program in Java or Python or Prolog to perform a search of a specific node in the graph.

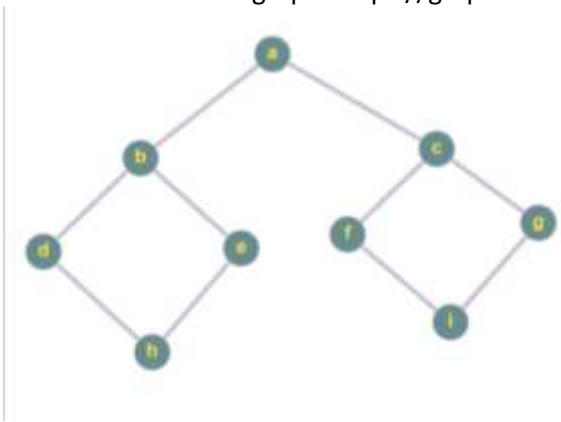
Facts and rules:

```
/* Facts */
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).
edge(d, h).
edge(e, h).
edge(f, i).
edge(g, i).

/* Rule */
connected(X, Y) :-
  edge(X, Y).

/* Breadth-First Search */
bfs(Node, Node, [Node], _).
bfs(Start, Node, [Start|Path], Visited) :-
  connected(Start, Next),
  \+ member(Next, Visited), /* Avoid revisiting visited nodes */
  append(Visited, [Next], UpdatedVisited),
  bfs(Next, Node, Path, UpdatedVisited).

/* Search for a specific node in the graph using breadth-first search */
search_bfs(Node, Path) :-
  bfs(a, Node, Path, [a]). /* Start the search from node 'a' */
```

Execution:

```
GNU Prolog console
File  Edit  Terminal  Prolog  Help
| ?- search_bfs(a, Path).

Path = [a] ? ;

no
| ?- search_bfs(b, Path).

Path = [a,b] ? ;

(16 ms) no
| ?- search_bfs(c, Path).

Path = [a,c] ? ;

(16 ms) no
| ?- search_bfs(d, Path).

Path = [a,b,d] ? ;

no
| ?- search_bfs(e, Path).

Path = [a,b,e] ? ;

(16 ms) no
| ?- search_bfs(f, Path).

Path = [a,c,f] ? ;

no
| ?- search_bfs(g, Path).

Path = [a,c,g] ? ;

no
| ?- search_bfs(h, Path).

Path = [a,b,d,h] ? ;

Path = [a,b,e,h] ? ;

no
| ?- search_bfs(i, Path).

Path = [a,c,f,i] ? ;

Path = [a,c,g,i] ? ;

(16 ms) no
| ?- search_bfs(j, Path).
```

III. Describe the graph, the algorithm and generate proper output for the results.
Source to create the graph: https://graphonline.ru/en/



The program implements a depth-first search (DFS) algorithm to search for a specific node in the graph. The DFS algorithm starts from a specified starting node and explores as far as possible along each branch before backtracking. It uses recursion to traverse the graph and keeps track of the visited nodes to avoid cycles.

Let's consider an example query to search for node 'i':

The program will perform a DFS starting from node 'a' and search for a path to node 'i'. It will output a valid path from 'a' to 'i' if one exists. For example, the output might be:

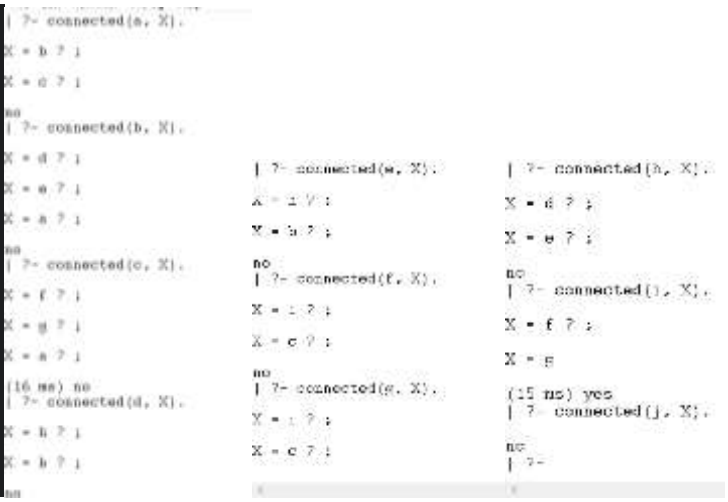
Path = [a, c, g, i].

This indicates that there is a path from 'a' to 'i' in the graph, and the path traverses' nodes 'a', 'c', 'g', and finally 'i'.

Task2: Depth First Search

I. Create a graph with at least 10 nodes.

```
/* Facts */
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).
edge(d, h).
edge(e, h).
edge(f, i).
edge(g, i).

/* Rules */
connected(X, Y) :-
  edge(X, Y).
connected(X, Y) :-
  edge(Y, X).
```



II. Write a program in Java or Python or Prolog to perform a search of a specific node in the graph.

Facts and Rules:

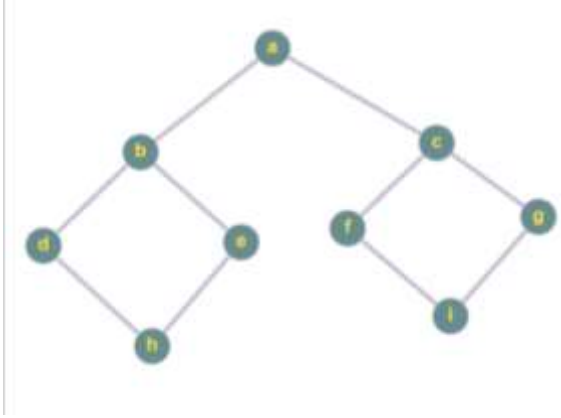
```
/* Facts */
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).
edge(d, h).
edge(e, h).
edge(f, i).
edge(g, i).

/* Rule */
connected(X, Y) :-
  edge(X, Y).

/* Depth-First Search */
dfs(Node, Node, [Node], _).
dfs(Start, Node, [Start|Path], Visited) :-
  connected(Start, Next),
  \+ member(Next, Visited), /* Avoid revisiting visited nodes */
  dfs(Next, Node, Path, [Next|Visited]).

/* Search for a specific node in the graph using depth-first search */
search_dfs(Node, Path) :-
  dfs(a, Node, Path, [a]). /* Start the search from node 'a' */

```

Execution:

```
| ?- search_dfs(a, Path).

Path = [a] ? ;

(16 ms) no
| ?- search_dfs(b, Path).

Path = [a,b] ? ;

no
| ?- search_dfs(c, Path).

Path = [a,c] ? ;

(16 ms) no
| ?- search_dfs(d, Path).

Path = [a,b,d] ? ;

no
| ?- search_dfs(e, Path).

Path = [a,b,e] ? ;

no
| ?- search_dfs(f, Path).

Path = [a,c,f] ? ;

no
| ?- search_dfs(g, Path).

Path = [a,c,g] ? ;

no
```

```
| ?- search_dfs(h, Path).

Path = [a,b,d,h] ? ;

Path = [a,b,e,h] ? ;

no
| ?- search_dfs(i, Path).

Path = [a,c,f,i] ? ;

Path = [a,c,g,i] ? ;

(16 ms) no
| ?- search_dfs(j, Path).
```

III. Describe the graph, the algorithm and generate proper output for the results.
Source to create the graph: https://graphonline.ru/en/



The program implements a depth-first search (DFS) algorithm to search for a specific node in the graph. The DFS algorithm starts from a specified starting node and explores as far as possible along each branch before backtracking. It uses recursion to traverse the graph and keeps track of the visited nodes to avoid cycles.

Let's consider an example query to perform a depth-first search for node 'i':

The program will perform a DFS starting from node 'a' and search for a path to node 'i'. It will output a valid path from 'a' to 'i' if one exists. For example, the output might be:

Path = [a, c, g, i].

This indicates that there is a path from 'a' to 'i' in the graph, and the path traverses' nodes 'a', 'c', 'g', and finally 'i'.