**Aim:** To implement Depth First Search in Prolog.

**Theory:**

The first strategy is depth-first search. In depth-first search, the frontier acts like a last-in first-out stack. The elements are added to the stack one at a time. The one selected and taken off the frontier at any time is the last element that was added.

**Algorithm:**

procedure DFS-iterative(G,v):

let S be a stack

S.push(v)

while S is not empty

v ← S.pop()

if v is not labeled as discovered:

label v as discovered

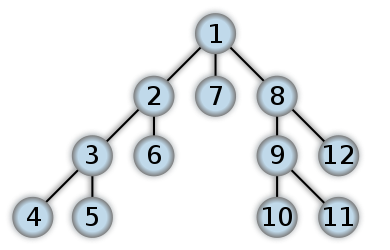
for all edges from v to w in G.adjacentEdges(v) do

S.push(w)

**Complexity:**

The time and space analysis of DFS differs according to its application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time O(|V| + |E|), linear in the size of the graph. In these applications it also uses space O(|V|) in the worst case to store the stack of vertices on the current search path as well as the set of already-visited vertices.

**Block Diagram:**



**Code:**

Experiment2.pl

link(a,e).

link(a,b).

link(e,d).

link(e,f).

link(f,c).

link(b,f).

link(b,c).

path(X, X, [X]).

path(X, Y, [X | T]):- link(X, Z), path(Z, Y, T).

Experiment2(b).pl

link(a,b).

link(a,d).

link(b,c).

link(b,e).

link(d,e).

link(d,f).

link(d,j).

link(e,g).

link(c,i).

link(g,i).

link(e,h).

link(f,h).

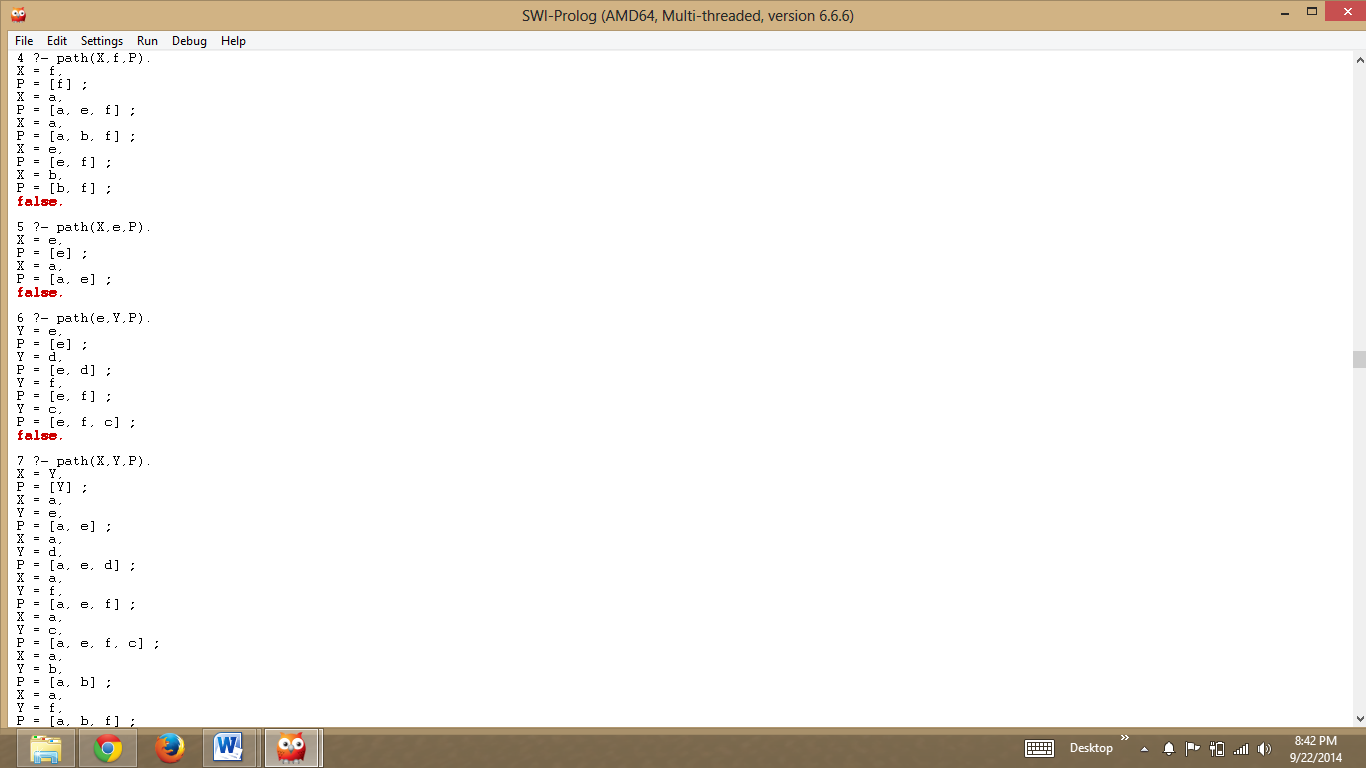
link(j,k).

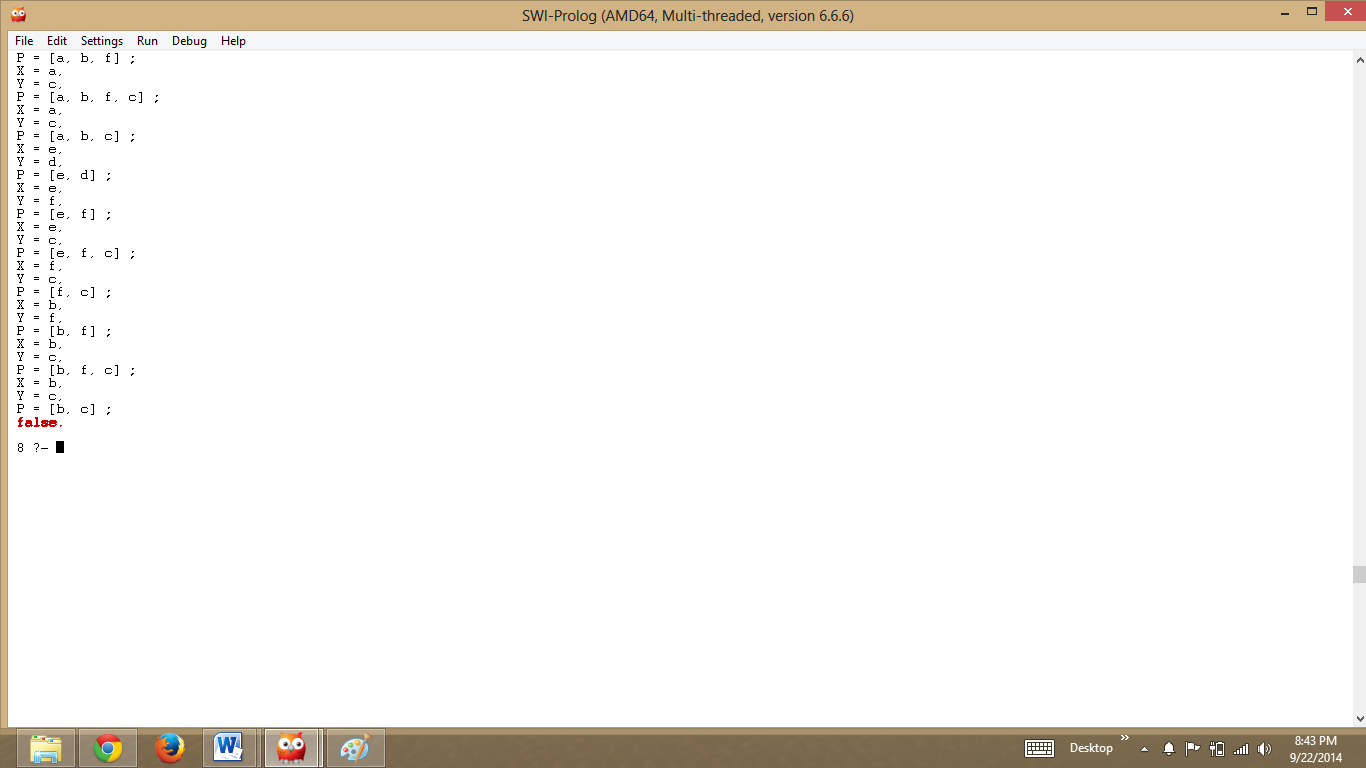
link(k,h).

path(X, X, [X]).

path(X, Y, [X | T]):- link(X, Z), path(Z, Y, T).

**Output:**





**Conclusion:** We have implemented depth first search in prolog using the basic prolog functions and rules that helps in implementation. We have also understood how DFS works using prolog.