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

**Theory:** BFS is an uninformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it. It does not use a heuristic algorithm.

From the standpoint of the algorithm, all child nodes obtained by expanding a node are added to a FIFO (i.e., First In, First Out) queue. In typical implementations, nodes that have not yet been examined for their neighbors are placed in some container (such as a queue or linked list) called "open" and then once examined are placed in the container "closed".

**Algorithm:**

If the element sought is found in this node, quit the search and return a result, otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.

procedure BFS(Graph,source):

create a queue Q

enqueue source onto Q

mark source

while Q is not empty:

dequeue an item from Q into v

for each edge e incident on v in Graph:

let w be the other end of e

if w is not marked:

mark w

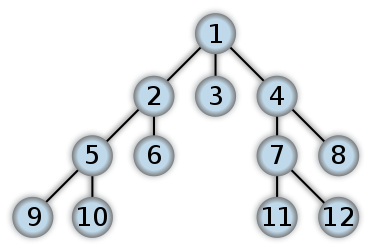
enqueue w onto Q

**Complexity:**

The time complexity can be expressed as O(|V|+|E|) [1] since every vertex and every edge will be explored in the worst case. Note: O(|E|) may vary between O(1) and O(|V|^2), depending on how sparse the input graph is.

When the number of vertices in the graph is known ahead of time, and additional data structures are used to determine which vertices have already been added to the queue, the space complexity can be expressed as O(|V|) where |V| is the cardinality of the set of vertices. If the graph is represented by an Adjacency list it occupies \Theta(|V|+|E|)[2] space in memory, while an Adjacency matrix representation occupies \Theta(|V|^2)

**Block Diagram:**



**Code:**

edge(frankfurt, mannheim).

edge(frankfurt, wurzburg).

edge(frankfurt, kassel).

edge(mannheim, karlsruhe).

edge(wurzburg, erfurt).

edge(wurzburg, nurnberg).

edge(kassel, munchen).

edge(karlsruhe, augsburg).

edge(nurnberg, stuttgart).

edge(nurnberg, munchen).

edge(munchen, augsburg).

link(A, B) :- edge(A, B).

link(A, B) :- edge(B, A).

next(A, B, C) :- link(A, B), not(member(B, C)).

heads([], []).

heads([[H|\_]|S], [H|T]) :- heads(S, T).

bfs(\_, [[Node|\_]|\_], \_, \_) :- write(Node), write('\n'), fail.

bfs(\_, [[Goal|Breadcrumbs]|\_], Goal, Path) :- reverse([Goal|Breadcrumbs], Path), !.

bfs(Discovered, [[Node|Breadcrumbs]|Paths], Goal, Path) :-

findall([Next,Node|Breadcrumbs], next(Node, Next, Discovered), Trails),

heads(Trails, Neighbors), append(Discovered, Neighbors, Rediscovered),

append(Paths, Trails, Crumbs), bfs(Rediscovered, Crumbs, Goal, Path).

search(Start, Goal, Path) :- bfs([Start], [[Start]], Goal, Path).

**Output:**

?- search(frankfurt, stuttgart, Path).

frankfurt

mannheim

wurzburg

kassel

karlsruhe

erfurt

nurnberg

munchen

augsburg

stuttgart

Path = [frankfurt, wurzburg, nurnberg, stuttgart].

?- search(karlsruhe, erfurt, Path).

karlsruhe

augsburg

mannheim

munchen

frankfurt

kassel

nurnberg

wurzburg

stuttgart

erfurt

Path = [karlsruhe, mannheim, frankfurt, wurzburg, erfurt].

**Conclusion:** We have implemented breadth first search in prolog using the basic prolog functions and rules that helps in implementation.