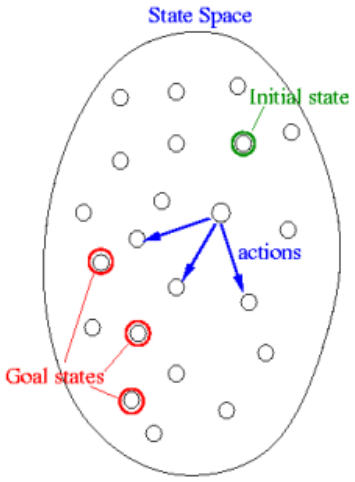
UNIT II

AI SEARCH ALGORITHMS

Unit II: AI Search Algorithms	2a. Explain key issues of search strategies. 2b. Explain various	2.1 Search Problem-Representation of search problems, Illustration of Search process 2.2 Basic search algorithm, Key issues, Evaluating various search strategies.
	search strategies 2c. Illustrate Breadth First Search for a graph. 2d. List different uninformed search strategies. 2e. List different informed search strategies.	2.3 Uninformed search-Various types of Uninformed search, Breadth First Search-Algorithm, Illustration of BFS for a graph 2.4 Informed search-Various types of informed search

Search Problem

A search problem consists of the following:
• S: the full set of states
• s0 : the initial state
• A:S→S is a set of operators
• G is the set of final states. Note that G ⊆S



The search problem is to find a sequence of actions which transforms the agent from the

initial state to a goal state g∈G. A search problem is represented by a 4-tuple {S, s0, A,G}.

S: set of states
s0 ∈ S : initial state
A: SÆ S operators/ actions that transform one state to another state
G : goal, a set of states. G ⊆ S

This sequence of actions is called a solution plan. It is a path from the initial state to a goal state. A plan P is a sequence of actions.

P = {a0, a1, ... , aN} which leads to traversing a number of states {s0, s1, ... , sN+1∈G}.

A sequence of states is called a path. The cost of a path is a positive number. In many cases the path cost is computed by taking the sum of the costs of each action.
Representation of search problems
A search problem is represented using a directed graph.
• The states are represented as nodes.
• The allowed actions are represented as arcs.

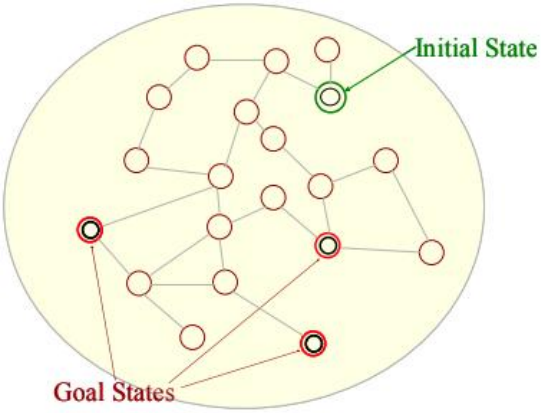
Searching process

The generic searching process can be very simply described in terms of the following steps:

Do until a solution is found or the state space is exhausted.
1. Check the current state
2. Execute allowable actions to find the successor states.
3. Pick one of the new states.
4. Check if the new state is a solution state
   If it is not, the new state becomes the current state and the process is repeated

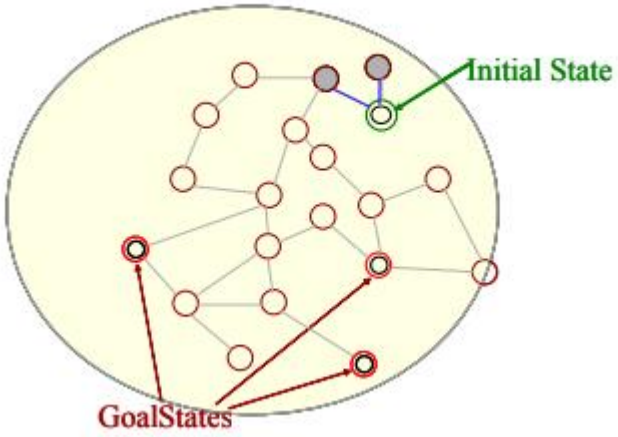
Illustration of a search process

We will now illustrate the searching process with the help of an example. Consider the problem depicted in Figure.

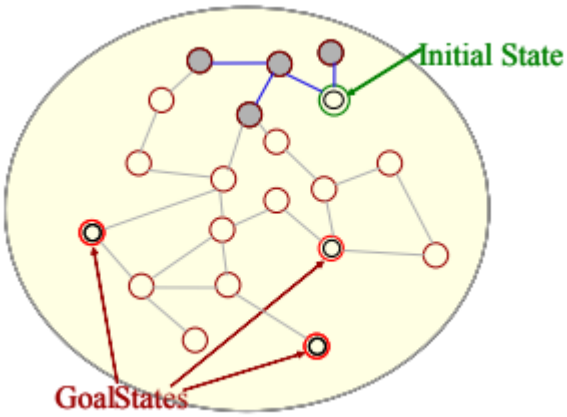


s0 is the initial state.

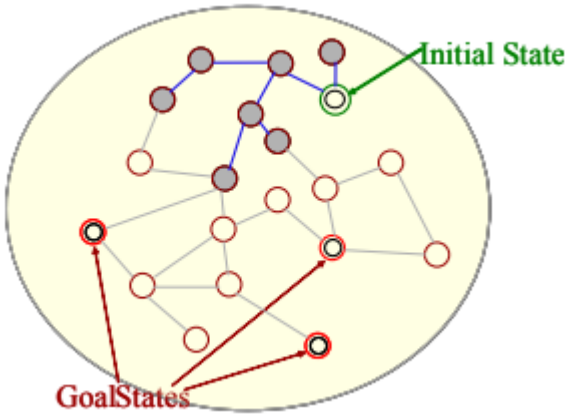
The successor states are the adjacent states in the graph.
There are three goal states.



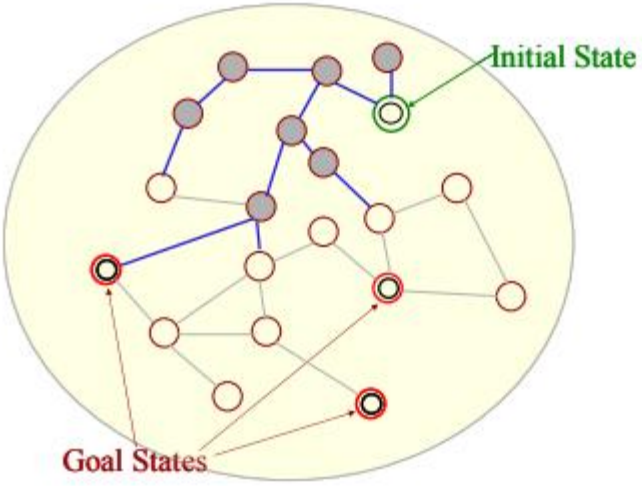
The two successor states of the initial state are generated.



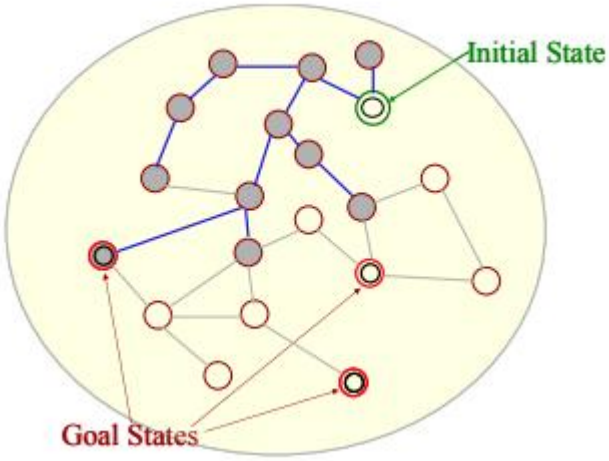
The successors of these states are picked and their successors are generated.



Successors of all these states are generated.



The successors are generated.

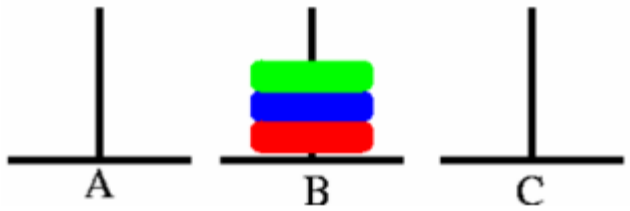


A goal state has been found.

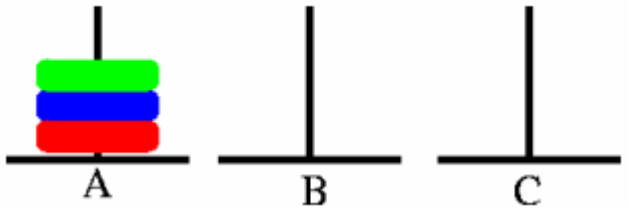
The above example illustrates how we can start from a given state and follow the successors, and be able to find solution paths that lead to a goal state. The grey nodes define the search tree. Usually the search tree is extended one node at a time. The order in which the search tree is extended depends on the search strategy.

Example problem: Pegs and Disks problem

Consider the following problem. We have 3 pegs and 3 disks.
Operators: one may move the topmost disk on any needle to the topmost position to any other needle
In the goal state all the pegs are in the needle B as shown in the figure below.

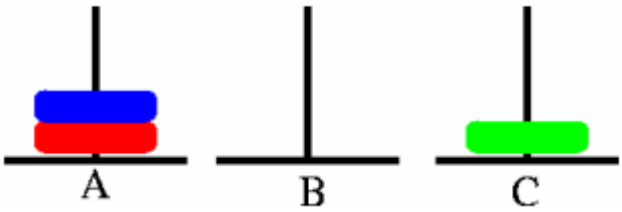


The initial state is illustrated below.

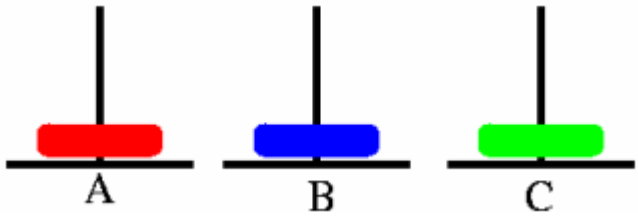


Now we will describe a sequence of actions that can be applied on the initial state.

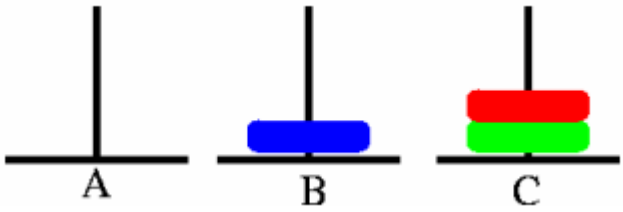
Step 1: Move A → C



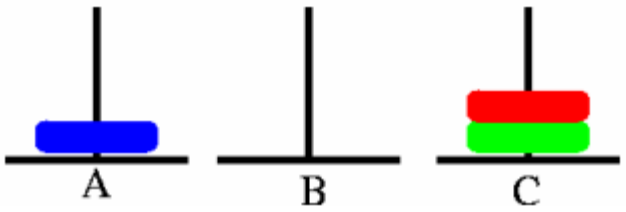
Step 2: Move A → B



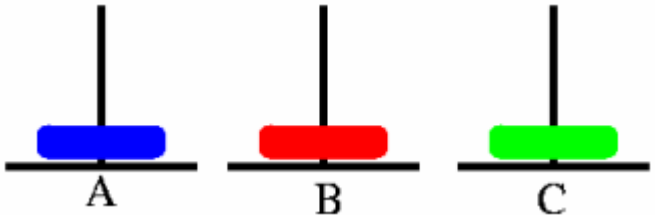
Step 3: Move A → C



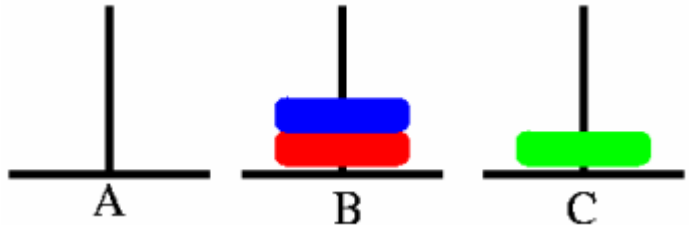
Step 4: Move B→ A



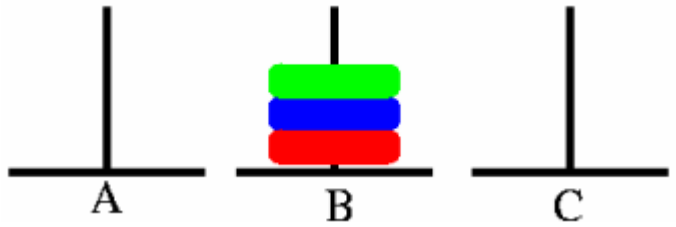
Step 5: Move C → B



Step 6: Move A → B



Step 7: Move C→ B



Example:8 queens problem

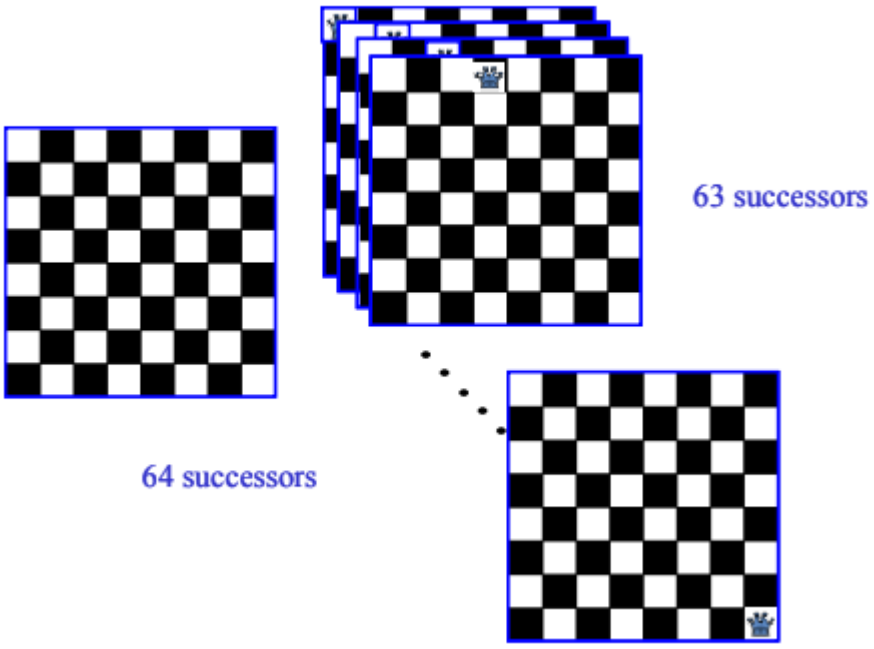
The problem is to place 8 queens on a chessboard so that no two queens are in the same row, column or diagonal

The picture below on the left shows a solution of the 8-queens problem. The picture on the right is not a correct solution, because some of the queens are attacking each other.

How do we formulate this in terms of a state space search problem? The problem formulation involves deciding the representation of the states, selecting the initial state representation, the description of the operators, and the successor states. We will now show that we can formulate the search problem in several different ways for this problem.

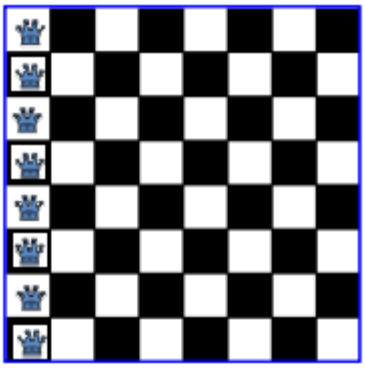
N queens problem formulation 1

• States: Any arrangement of 0 to 8 queens on the board
• Initial state: 0 queens on the board
• Successor function: Add a queen in any square
• Goal test: 8 queens on the board, none are attacked



N queens problem formulation 2

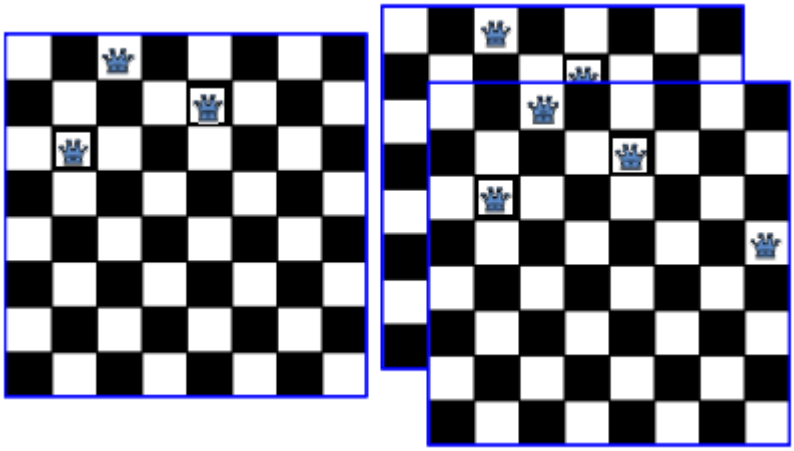
• States: Any arrangement of 8 queens on the board
• Initial state: All queens are at column 1
• Successor function: Change the position of any one queen
• Goal test: 8 queens on the board, none are attacked



If we consider moving the queen at column 1, it may move to any of the seven remaining columns.

N queens problem formulation 3

• States: Any arrangement of k queens in the first k rows such that none are attacked
• Initial state: 0 queens on the board
• Successor function: Add a queen to the (k+1)th row so that none are attacked.
• Goal test : 8 queens on the board, none are attacked.



Search

Searching through a state space involves the following:

• A set of states
• Operators and their costs
• Start state
• A test to check for goal state

The basic search algorithm

```
Let L be a list containing the initial state (L= the fringe)
Loop
        if L is empty return failure
        Node ← select (L)
         if Node is a goal
                then return Node
                    (the path from initial state to Node)
        else generate all successors of Node, and
             merge the newly generated states into L
End Loop
```

We need to denote the states that have been generated. We will call these as nodes. The data structure for a node will keep track of not only the state, but also the parent state or the operator that was applied to get this state. In addition the search algorithm maintains a list of nodes called the fringe. The fringe keeps track of the nodes that have been generated but are yet to be explored. The fringe represents the frontier of the search tree generated.

Initially, the fringe contains a single node corresponding to the start state. In this version we use only the OPEN list or fringe. The algorithm always picks the first node from fringe for expansion. If the node contains a goal state, the path to the goal is returned. The path corresponding to a goal node can be found by following the parent pointers. Otherwise all the successor nodes are generated and they are added to the fringe.

Search algorithm: Key issues

Corresponding to a search algorithm, we get a search tree which contains the generated and the explored nodes. The search tree may be unbounded. This may happen if the state space is infinite. This can also happen if there are loops in the search space. **How can we handle loops?**

Corresponding to a search algorithm, **should we return a path or a node?** The answer to this depends on the problem. For problems like N-queens we are only interested in the goal state. For problems like 15-puzzle, we are interested in the solution path.

We see that in the basic search algorithm, we have to select a node for expansion. **Which node should we select? Alternatively, how would we place the newly generated nodes in the fringe?** We will subsequently explore various search strategies and discuss their properties,

Depending on the search problem, we will have different cases. The search graph may be weighted or unweighted. In some cases we may have some knowledge about the quality of intermediate states and this can perhaps be exploited by the search algorithm. Also depending on the problem, our aim may be to find a minimal cost path or any to find path as soon as possible.

Evaluating Search strategies

We will look at various search strategies and evaluate their problem solving performance. What are the characteristics of the different search algorithms and what is their efficiency? We will look at the following three factors to measure this.

1. **Completeness**: Is the strategy guaranteed to find a solution if one exists?
2**. Optimality**: Does the solution have low cost or the minimal cost?
3. What is the search cost associated with the time and memory required to find a solution?

a**. Time complexity**: Time taken (number of nodes expanded) (worst or average case) to find a solution.

b. Space complexity: Space used by the algorithm measured in terms of the maximum size of fringe

The different search strategies that we will consider include the following:

1**. Blind Search strategies or Uninformed search**

   a. Depth first search
   b. Breadth first search
   c. Iterative deepening search
   d. Iterative broadening search

2. Informed Search
3. Constraint Satisfaction Search
4. Adversary Search

Search Tree – Terminology

• Root Node: The node from which the search starts.

• Leaf Node: A node in the search tree having no children.

• Ancestor/Descendant: X is an ancestor of Y is either X is Y's parent or X is an ancestor of the parent of Y. If S is an ancestor of Y, Y is said to be a descendant of X.

• Branching factor: the maximum number of children of a non-leaf node in the search tree

• Path: A path in the search tree is a complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.

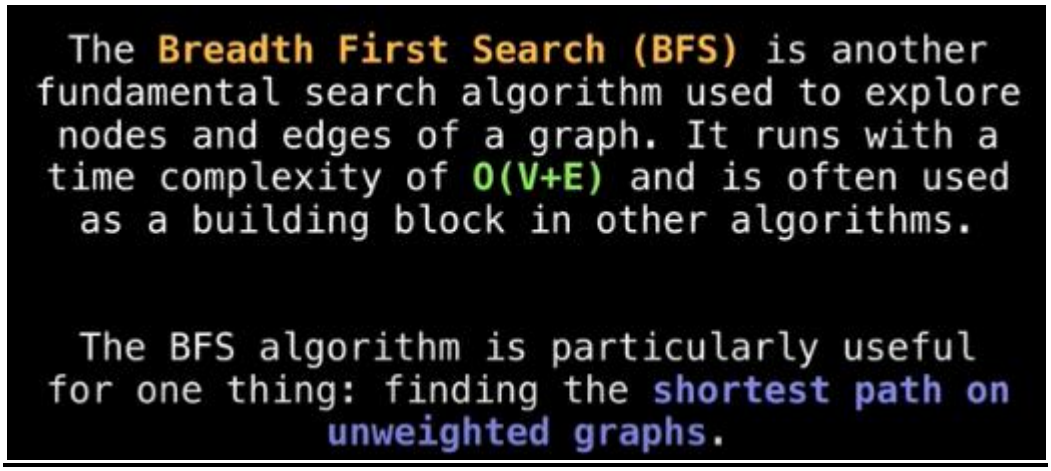
We also need to introduce some data structures that will be used in the search algorithms.

Node data structure

A node used in the search algorithm is a data structure which contains the following:

1. A state description
2. A pointer to the parent of the node
3. Depth of the node
4. The operator that generated this node
5. Cost of this path (sum of operator costs) from the start state

Breadth First Search Algorithm:



Breadth first search
Let *fringe* be a list containing the initial state
Loop
    if *fringe* is empty return failure
    Node ← remove-first (fringe)
     if Node is a goal
        then return the path from initial state to Node
    else generate all successors of Node, and
        (merge the newly generated nodes into *fringe*)
        add generated nodes to the back of *fringe*
End Loop

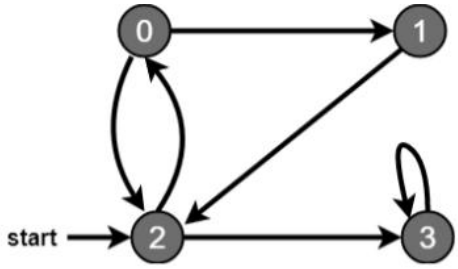
Breadth-First Traversal (or Search) for a graph is similar to Breadth-First Traversal of a tree
The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- Visited and
- Not visited.

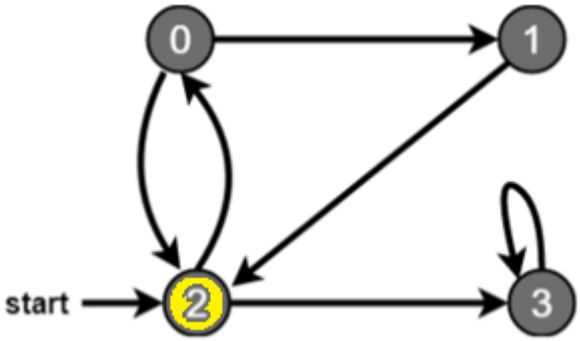
Implementation of BFS traversal:
Follow the below method to implement BFS traversal.

- Declare a queue and insert the starting vertex.
- Initialize a visited array and mark the starting vertex as visited.
- Follow the below process till the queue becomes empty:
    - Remove the first vertex of the queue.
    - Mark that vertex as visited.
    - Insert all the unvisited neighbours of the vertex into the queue.

Example 1:

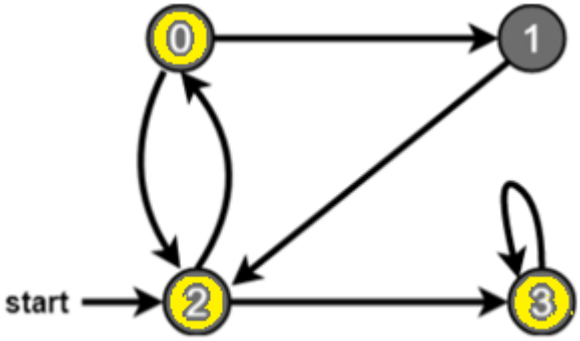


Step 1:
Initially fringe contains only one node corresponding to the source state 2.



FRINGE: 2

Step 2:
2 is removed from fringe. The node is expanded, and its children 0 and 3 are generated. They are placed at the back of fringe.



FRINGE: 2,0,3

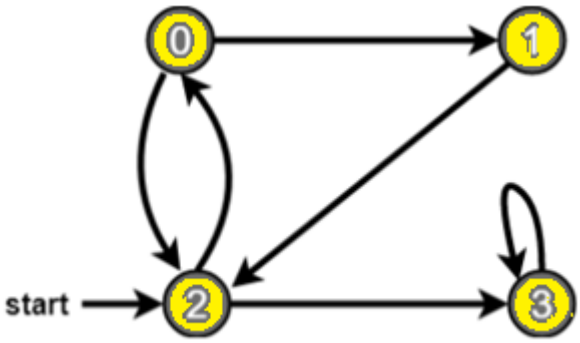
Node 0 is removed from fringe and is expanded. Its children 1, 2 are generated and put at the back of fringe.
2 is also an adjacent vertex of 0.
2 is already marked as visited vertices, then 2 will not be processed again.
So only 1 is placed at the back of fringe.

OR

FRINGE: 2,3,0
Node 3 is removed from fringe and is expanded. Its neighbour is 2
2 is already marked as visited vertices, then 2 will not be processed again.

Node 0 is removed from fringe and is expanded. Its children 1, 2 are generated
and put at the back of fringe.
2 is also an adjacent vertex of 0.
2 is already marked as visited vertices, then 2 will not be processed again.
So only 1 is placed at the back of fringe.

Step 3:



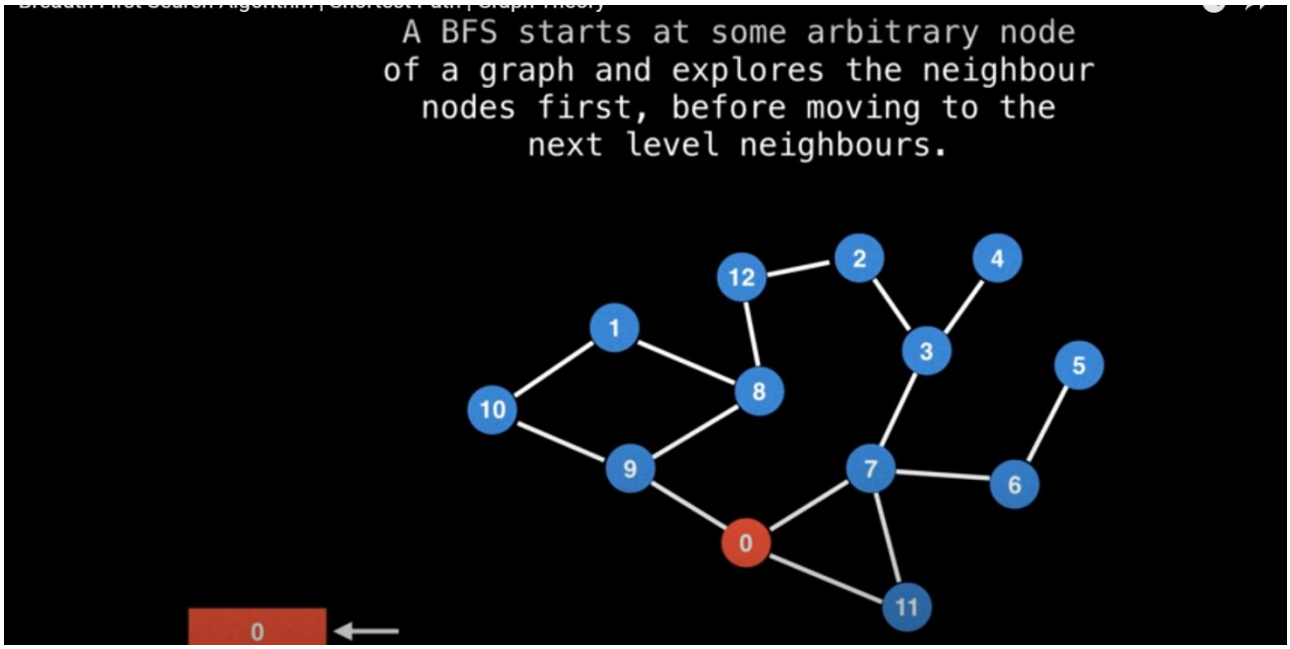
FRINGE: 2,0,3,1.
OR
FRINGE: 2,3,0,1

Node 1 is removed from fringe and is expanded.
Node 3 is placed in the back of fringe,
As node 3 has neighbours 2 and 3 itself,node 2 abd 3 is already visited, algorithm terminates.

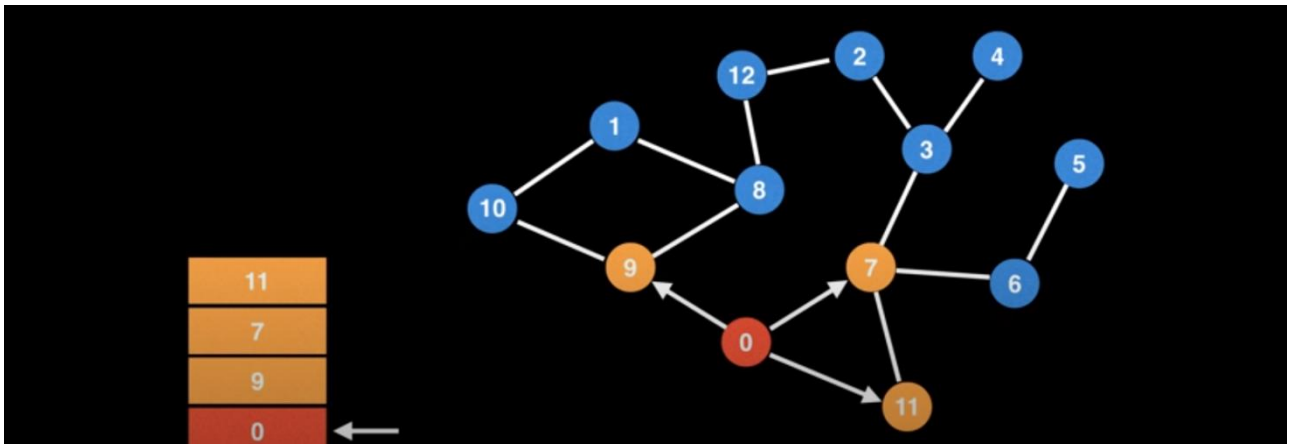
There can be multiple BFS traversals for a graph. Different BFS traversals for
the above graph :
2, 3, 0, 1
2, 0, 3, 1

EXAMPLE 2:

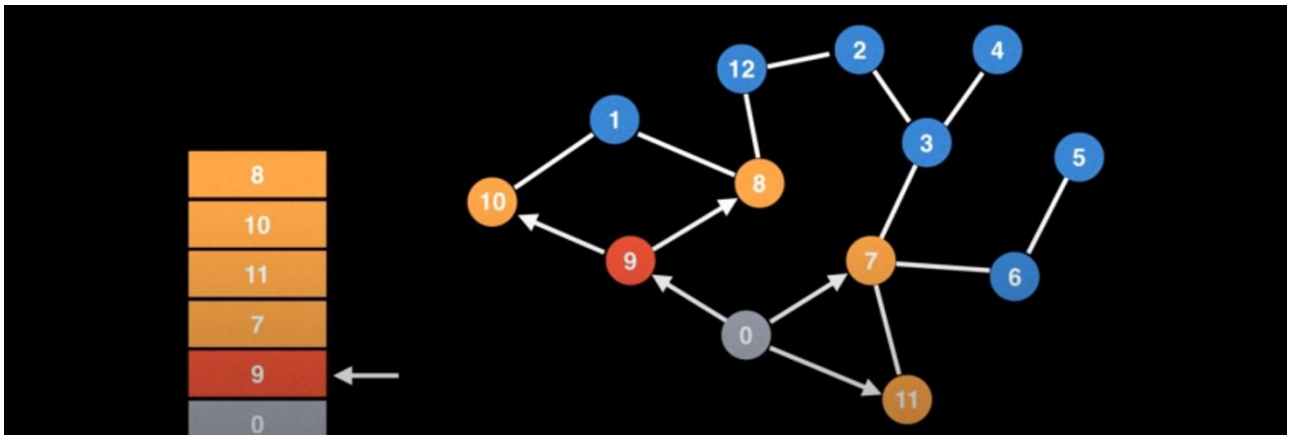
Start is from 0



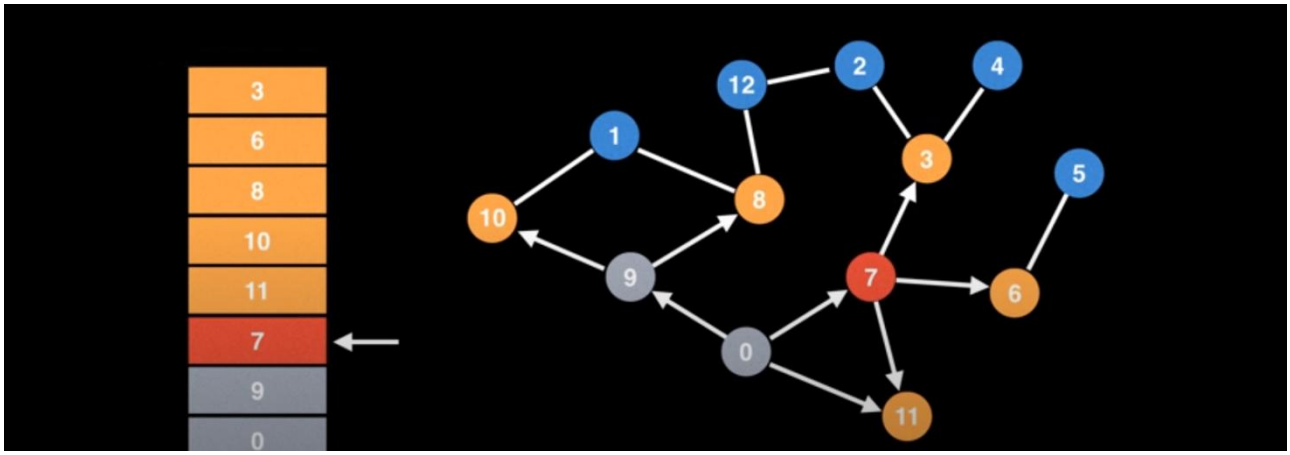
FRINGE:0



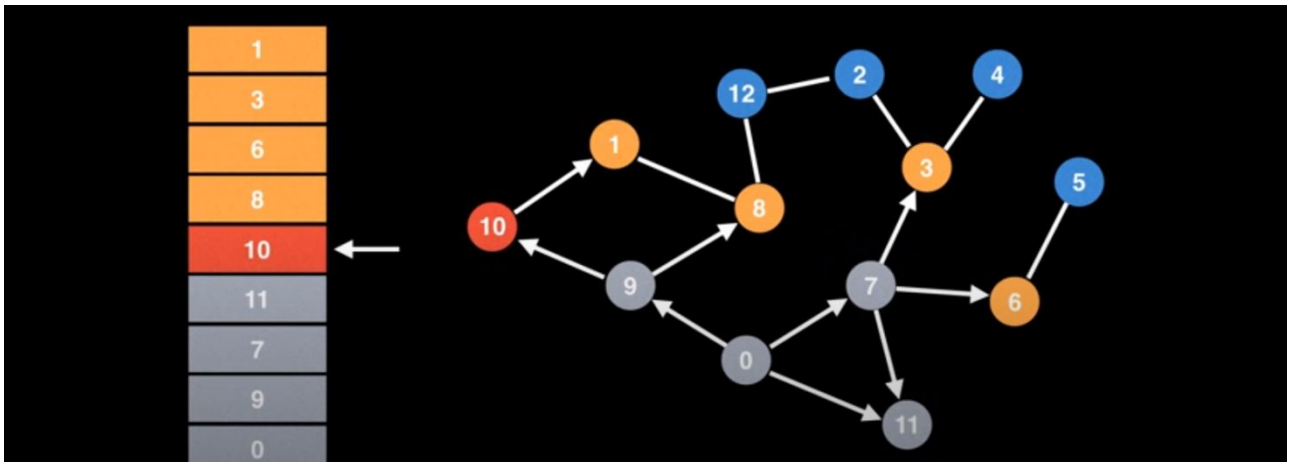
FRINGE:0,9,7,11



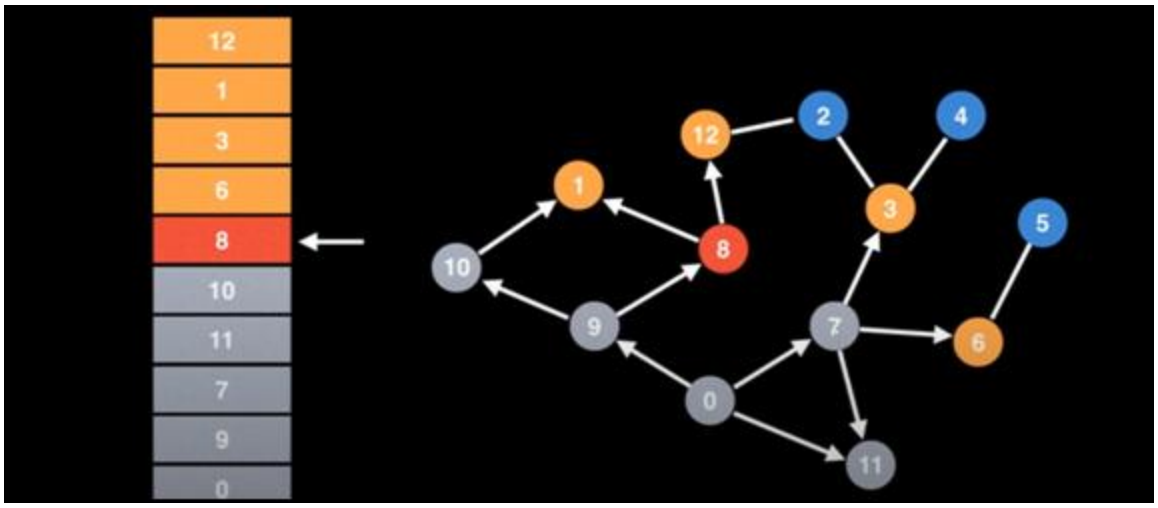
FRINGE: 0,9,7,11,10,8



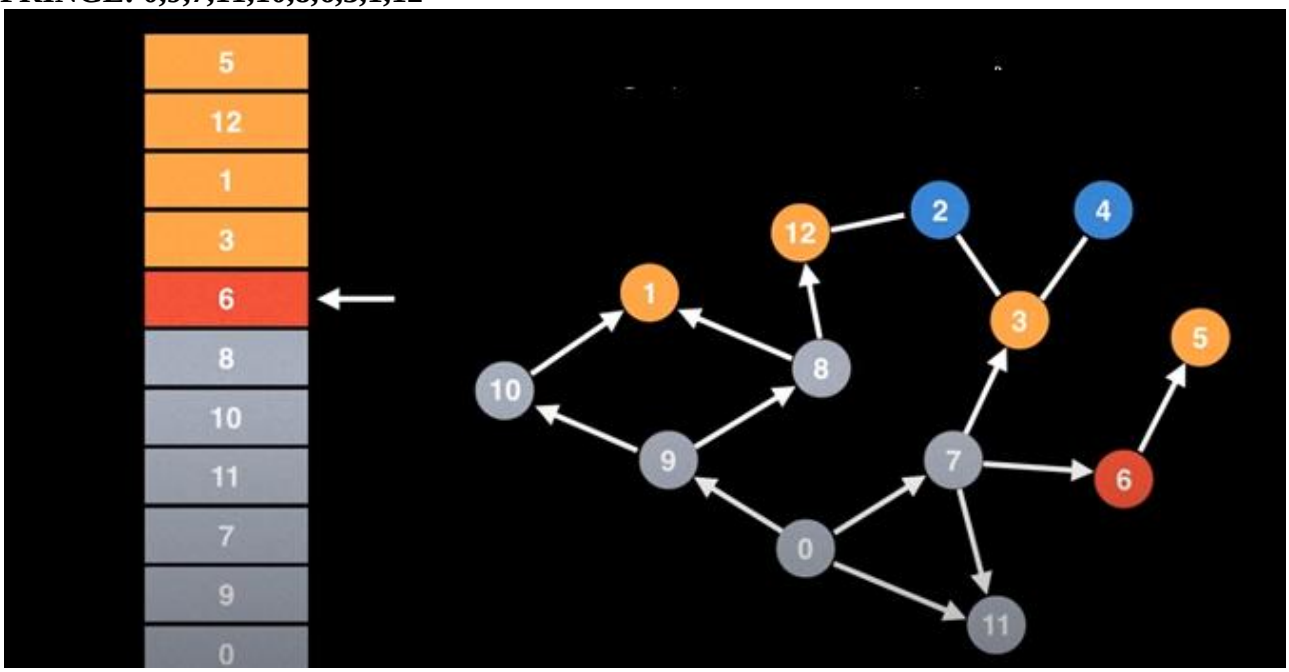
FRINGE: 0,9,7,11,10,8,6,3



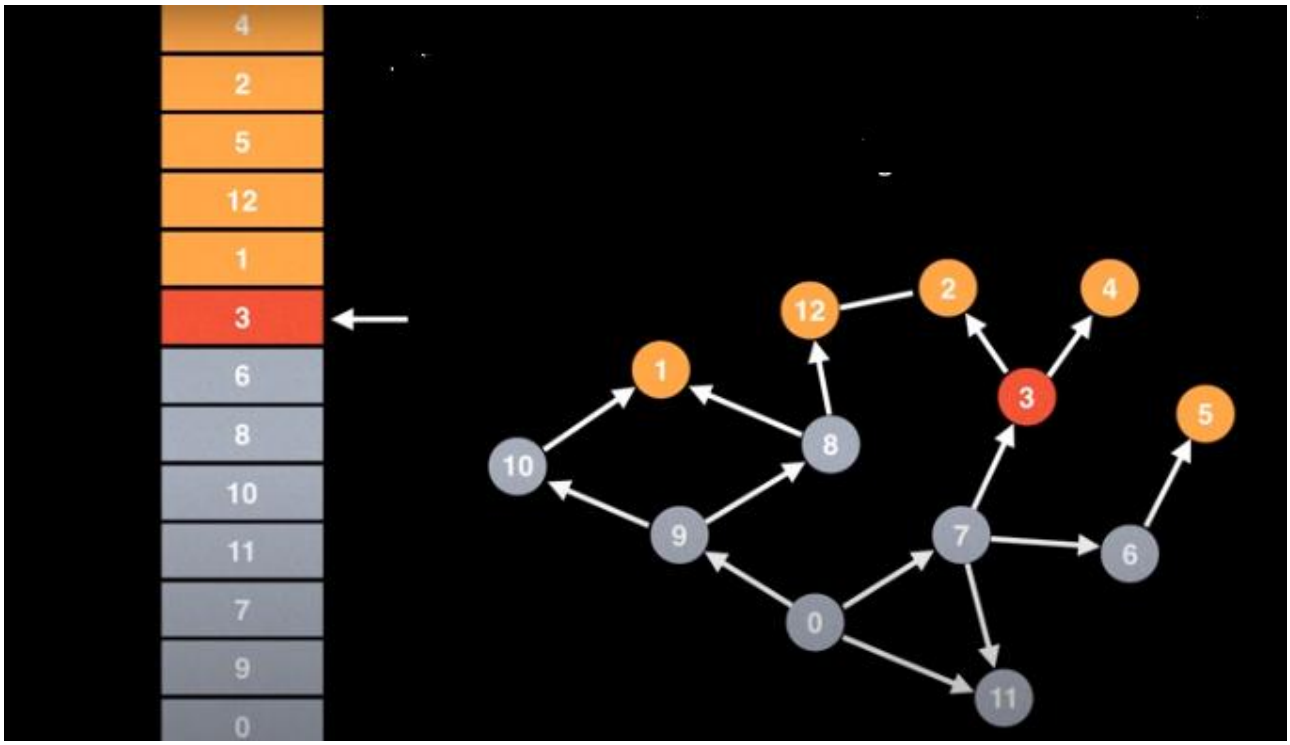
FRINGE: 0,9,7,11,10,8,6,3,1
(As 11 is already visited,it is not added in fringe again)



FRINGE: 0,9,7,11,10,8,6,3,1,12



FRINGE: 0,9,7,11,10,8,6,3,1,12,5



FRINGE: 0,9,7,11,10,8,6,3,1,12,5,2,4
After 3 is visited,1,12,5,2,4 have no neighboure.
So 4 is goal state. BFS algorithm terminates.

Advantages of BFS

- BFS will never be trapped in any unwanted nodes.
- If the graph has more than one solution, then BFS will return the optimal solution which provides the shortest path.

Disadvantages of BFS

- BFS stores all the nodes in the current level and then go to the next level. It requires a lot of memory to store the nodes.
- BFS takes more time
  to reach the goal state which is far away.

Informed search algorithms:
Types:

1. **Greedy best-first search algorithm**

2. **A* search algorithm**

Comparison of uninformed and informed search algorithms

Uninformed search	Informed search
Uninformed search is also known as blind search	informed search is also called heuristics search.
Uniformed search does not require much information	Informed search requires domain-specific details.
time complexity of uninformed search strategies is more	informed search strategies are more efficient
It cannot handle problems easily.	Handles problems more better than uninformed search