Chapter 2

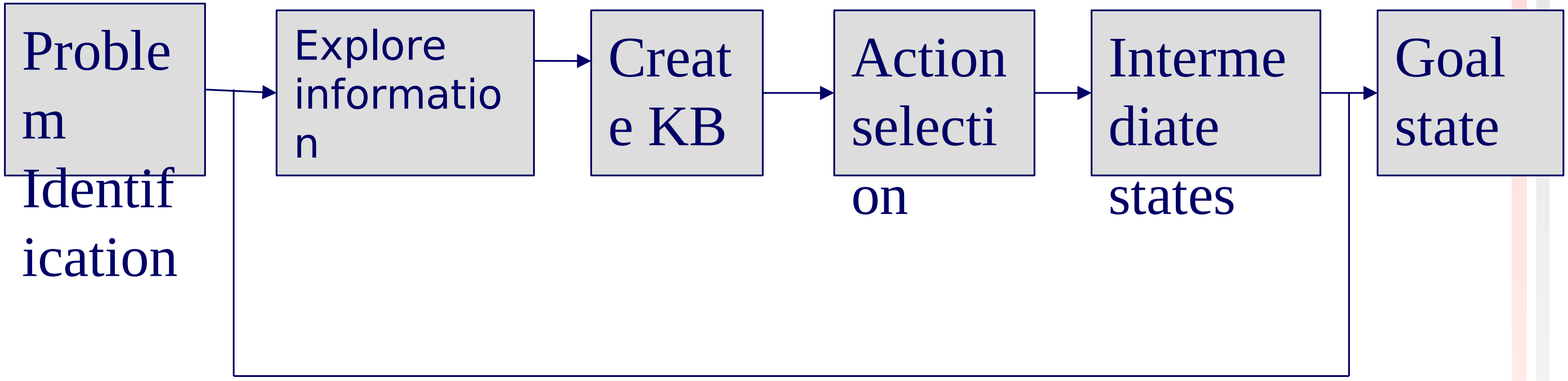
Problems, Problem Spaces, and Search?

Dr. Latesh Malik

Problem solving Process

A systematic approach to defining the problem (question or situation that presents uncertainty, perplexity or difficulty) and creating a vast number of possible solutions without judging these solutions

Problem solving Process



Problem solving
process

Problem solving Process

Problem solving techniques involve the following:

- Problem identification
- Problem analysis and representation
- Planning
- Execution
- Evaluating solution
- Consolidating gains

State Space

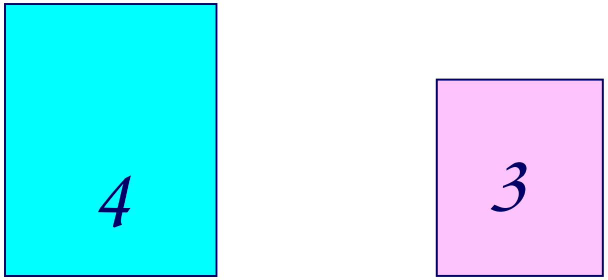
- Before an AI problem can be solved it must be represented as a state space. The state space is then searched to find a solution to the problem.

- A state space essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state and a goal state.

- Each state space takes the form of a tree or a graph.

Defining the problem

- *A water jug problem: 4-gallon and 3-gallon*



- *no marker on the bottle*
- *pump to fill the water into the jug*
- *How can you get exactly 2 gallons of water into the 4-gallons jug?*

A state space search

(x,y) : order pair

x : water in 4-gallons ➔ *x = 0,1,2,3,4*

y : water in 3-gallons ➔ *y = 0,1,2,3*

start state : (0,0)

goal state : (2,n) where n = any value

Rules :

1. *Fill the 4 gallon-jug (4,-)*
2. *Fill the 3 gallon-jug (-,3)*
3. *Empty the 4 gallon-jug (0,-)*
4. *Empty the 3 gallon-jug (-,0)*

Water jug rules

1	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow (x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow (x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full

Water jug rules

8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

A water jug solution

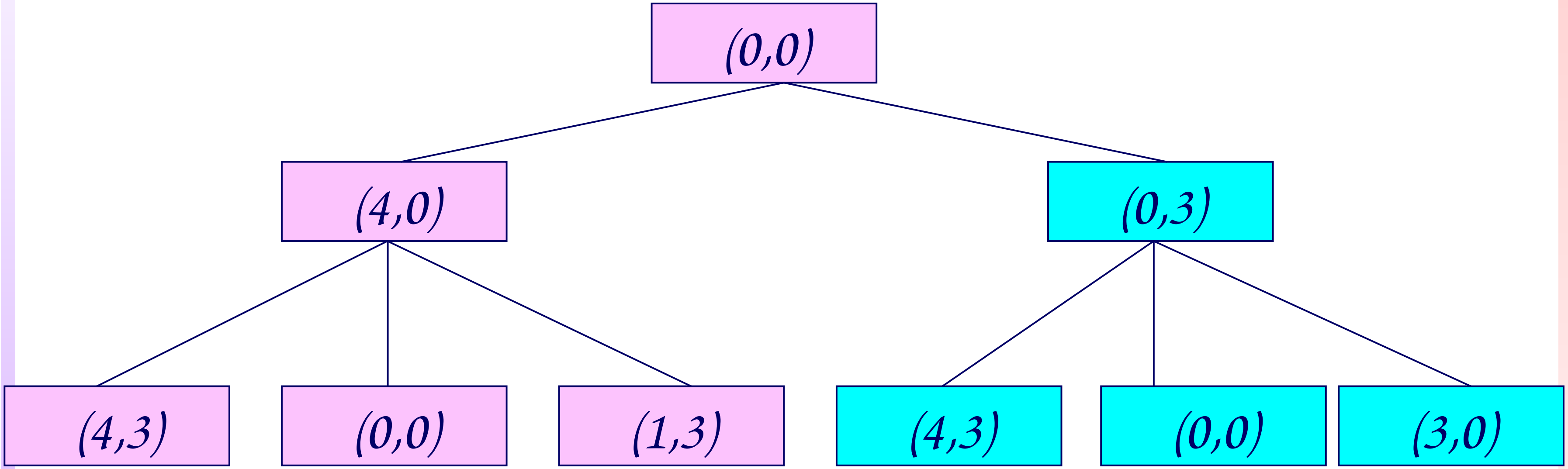
4-Gallon Jug	3-Gallon Jug	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Solution : path / plan

Formal description of a problem

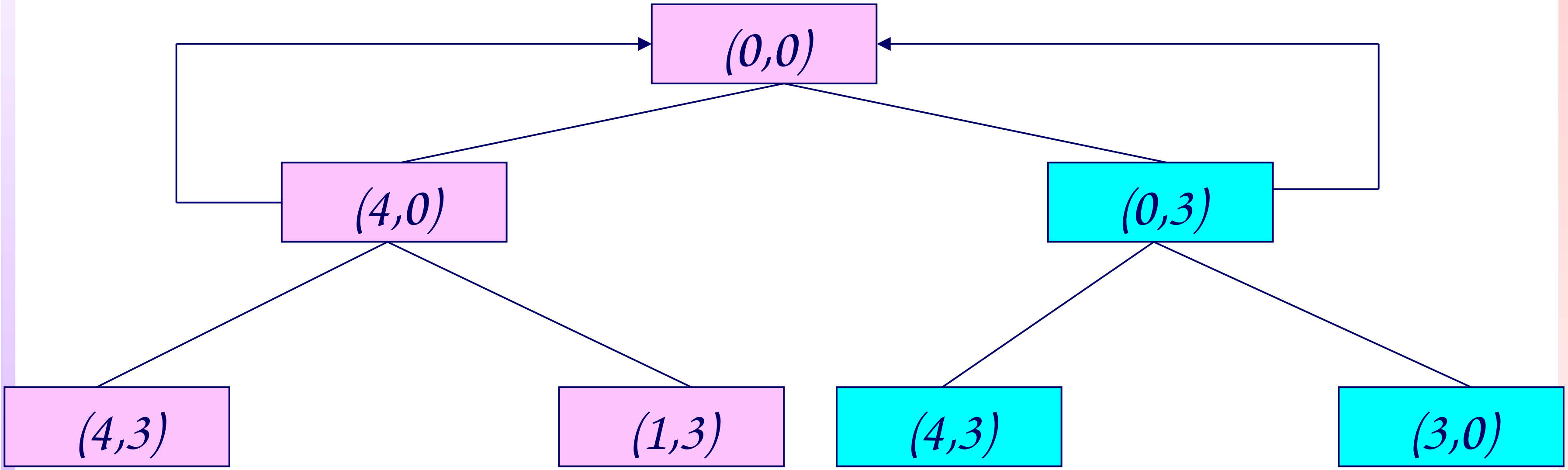
1. *Define a state space that contains all the possible configurations of the relevant objects.*
2. *Specify state/states that describes the situation of start state.*
3. *Specify state/states that describes the situation of goal state.*
4. *Specify the set of rules.*
    - assumption, generalization

Search Tree



- *Water jug problem.*

Search Graph



- *Water jug problem.*
  - *Cycle: good control strategy that causes motion (BFS/DFS)*
  - *When will the search terminate?*

Water Jug Problem

- States: *Amount of water in jugs*
- Goal: *To get the specified amount of water in big jug .*
- Path cost: *Number of actions applied.(minimum the no of actions better is the solution)*
- Actions:

1. Empty the big jug

2. Empty the small jug

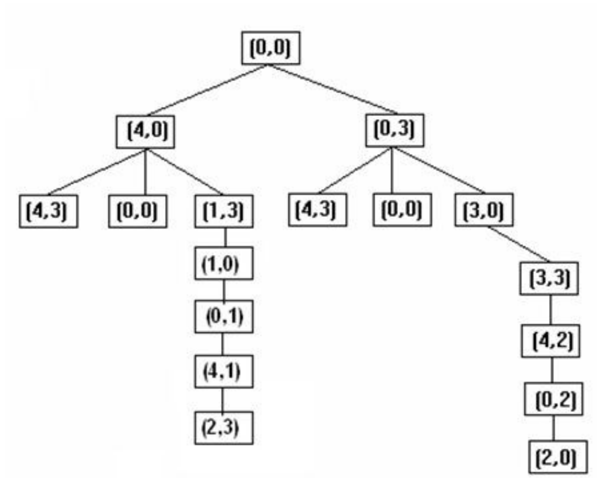
3. Pour water from small jug to big jug

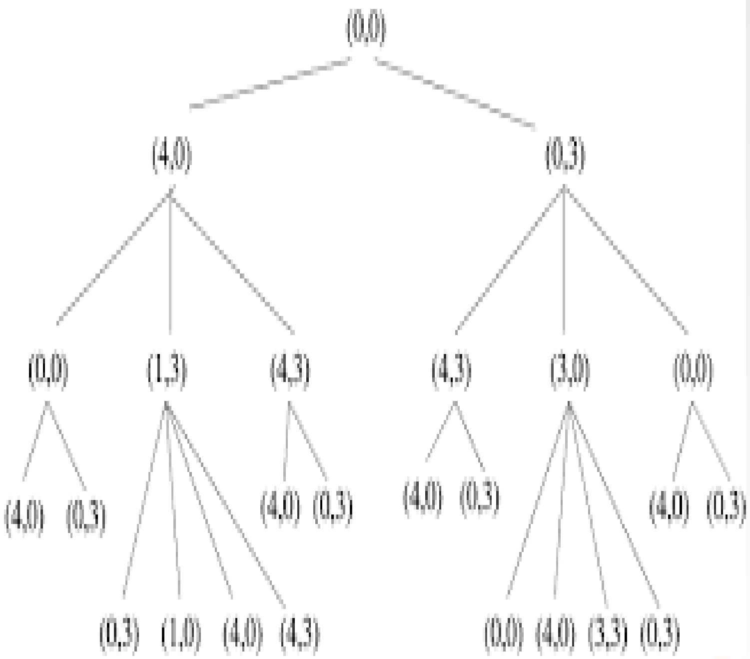
4. Pour water from big jug to small jug

State Space : Water jug

Example: Water jug problem

State space representation





Homework

- *Is there any other solution for a water jug problem?*
- *If there is some other solution, describe in an order pair of how to solve it.*

State Space: Tic- tac- toe problem

Initial state   Goal State

	x	

o		**x**
o	x	x
x	o	

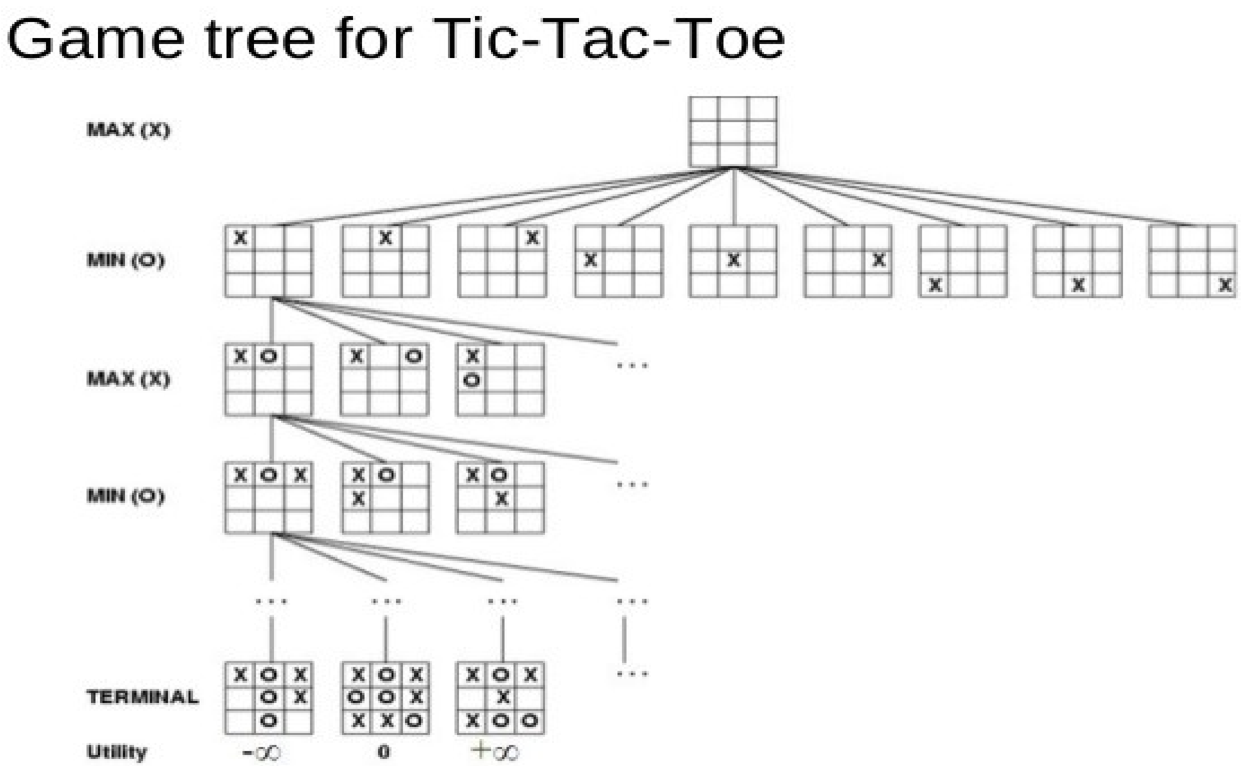
Initial State: *State in figure*

Goal State: *To reach the final winning position*

Operator: *adding 'x' or 'o' in cells one by one*

Path cost: *Each step costs 1 so that the path cost is the length of the path*

State Space: Tic- tac- toe problem



Courtesy : Artificial Intelligence and Soft Computing. Behavioural and Cognitive Modelling of the Human Brain

Question answering question

1. *Marcus was a man.*
2. *Marcus was a Pompeian.*
3. *Marcus was born in 40 A.D.*
4. *All men are mortal.*
5. *All Pompeians died when the volcano erupted in 79 A.D.*
6. *No mortal lives longer than 150 years.*
7. *It is now 1991 A.D.*

Is Marcus alive?

Solution 1

1. Marcus was man.	axiom 1
4. All men are mortal.	axiom 4
8. Marcus is mortal.	1,4
3. Marcus was born in 40 A.D.	axiom 3
7. It is now 1991 A.D.	axiom 7
9. Marcus' age is 1951 years.	3,7
6. No mortal lives longer than 150 years	axiom 6
10. Marcus is dead.	8,6,9

Solution 2

7. It is now 1991 A.D.	*axiom 7*
5. All Pompeians died in 79 A.D.	*axiom 5*
11. All Pompeians are dead now.	*7,5*
2. Marcus was a Pompeian.	*axiom 2*
12. Marcus is dead……	*11,2*

Understanding a sentence

- *The bank president ate a dish of pasta salad with the fork.*

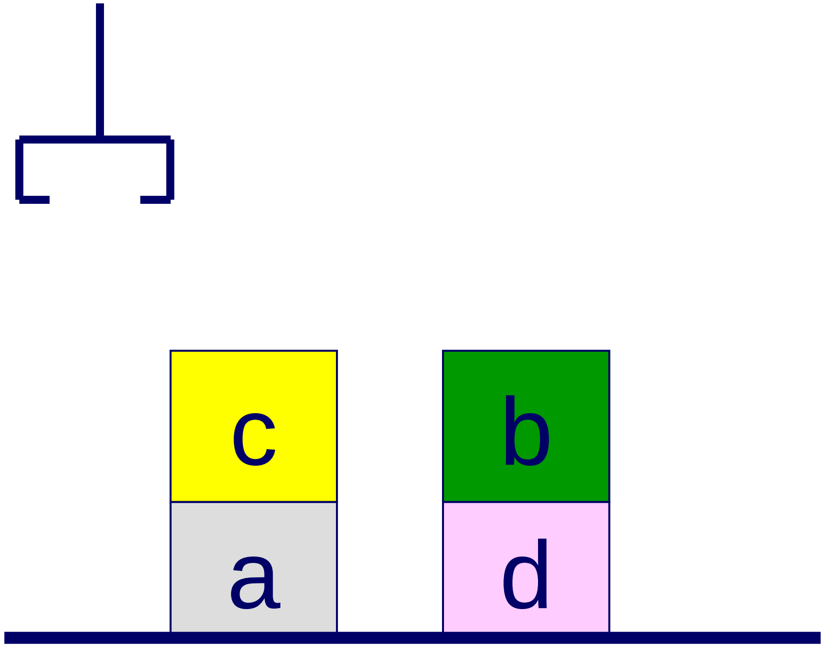
- *bank = financial institution / a side of a river*
- *dish = eat dish / eat pasta*
- *pasta salad : dog food ➔ food with dog meat?*
- *with a fork : with ….her friend. / with vegetable.*
- *solution : state of the world*

Seven problem characteristics

1. *Decomposable Problem*
   - *Block world problem*
2. *Can solution steps be ignored or undone?*
   - *Ignorable : theorem proving*
     - solution steps can be ignored
   - *Recoverable : 8 puzzle*
     - solution steps can be undone (backtracking)
   - *Irrecoverable : chess*
     - solution steps can not be undone

A blocks world

- *on(c,a).*
- *on(b,d).*
- *ontable(a).*
- *ontable(d).*
- *clear(b).*
- *clear(c).*
- *hand_empty.*



Seven problem characteristics

- *3. Is the universe predictable?*
  - *8-puzzel (yes)*
  - *bridge (no) ➔ but we can use probabilities of each possible outcomes*
    - *controlling robot arm ➔ gear of the arm might stuck*
    - *helping the lawyer to decide how to defend his client against a murder charge.*
- *4. Is a good solution absolute or relative?*
- *- formal inference methods*
- *- More than one solution?*
- *- traveling salesman problem*

Seven problem characteristics

5. Is the solution a state or a path ?

- water jug problem ➔ path / plan

6. What is the role of knowledge?

knowledge for perfect program of chess

(need knowledge to constrain the search)

newspaper story understanding

(need knowledge to recognize a solution)

7. Does the task require interaction with a person? solitary/ conversational

Production system

1. *A set of rules.*
2. *Knowledge contains information for a particular task.*
3. *A control strategy.*
   - resolve conflict rule.
   - Breadth-first search
   - Depth-first search

Expert system shells : provide environment for construct knowledge based expert system.

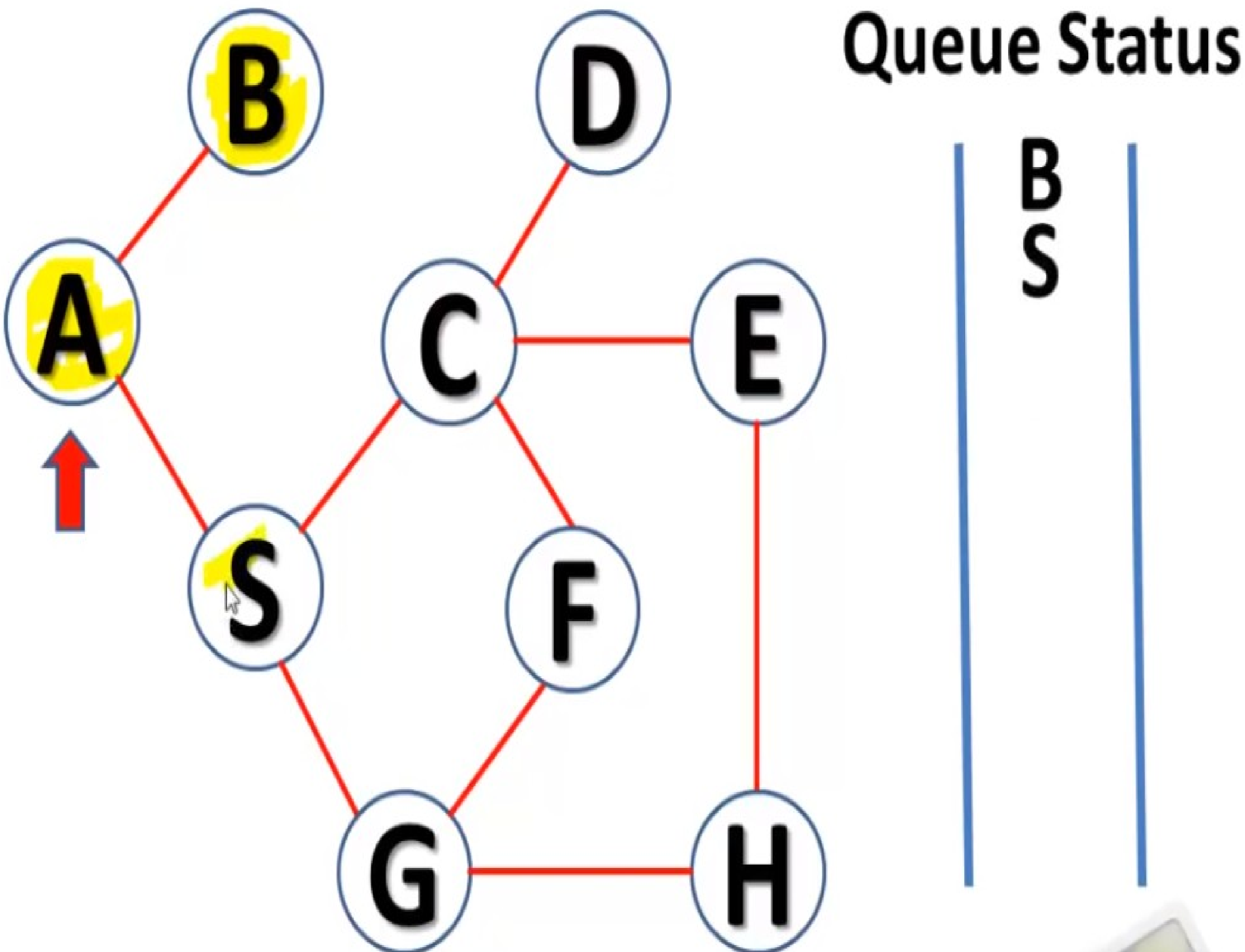
Breadth-first search

- *Algorithm BFS:*

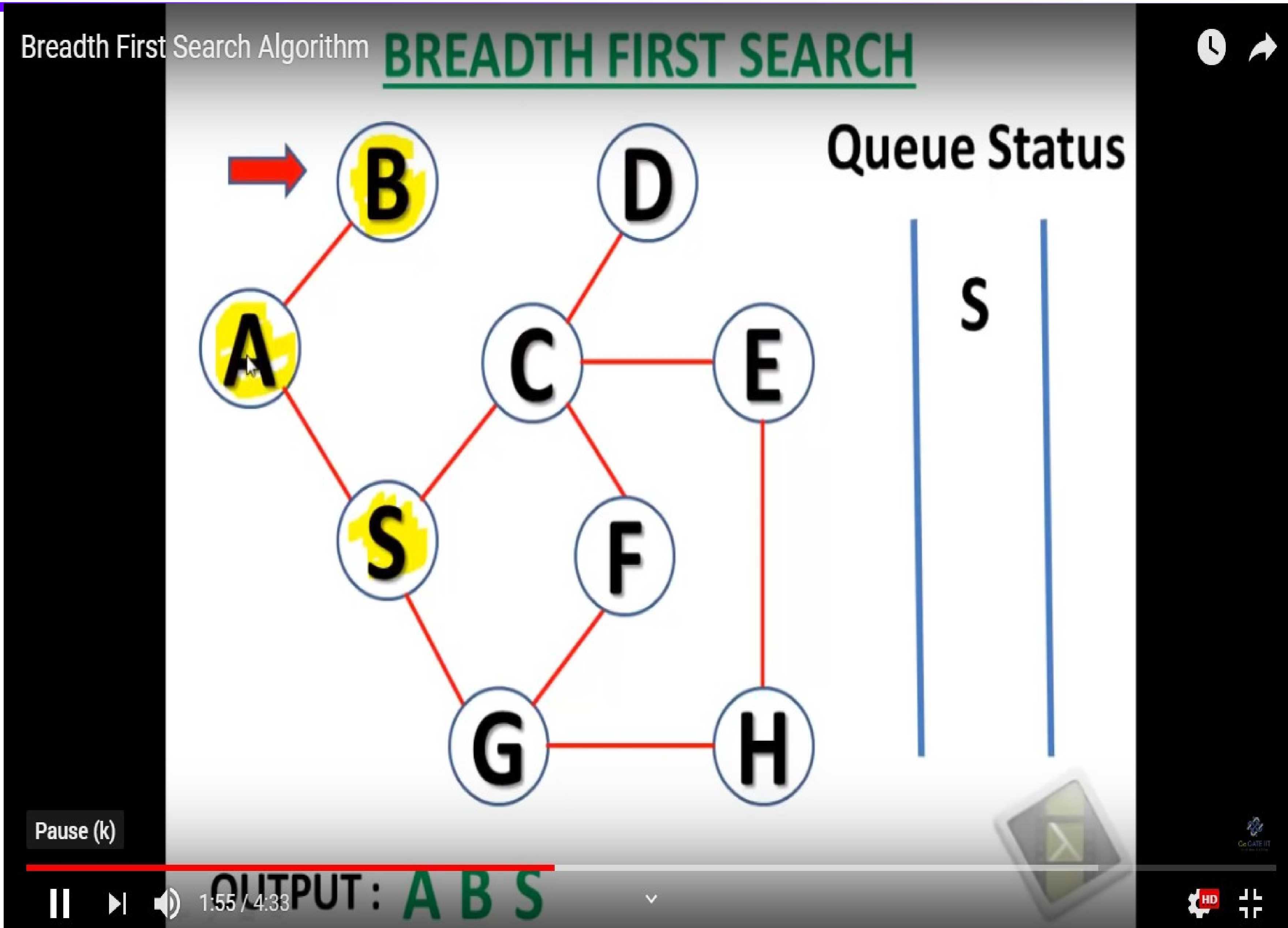
1. *Create a variable called NODE-LIST and set it to the initial state.*
2. *Until a goal state is found or NODE-LIST is empty do:*
   - Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
   - For each way that each rule can match the state described in E do:
     - Apply the rule to generate a new state.
     - If the new state is a goal state, quit and return this state.
     - Otherwise, add the new state to the end of NODE-LIST.

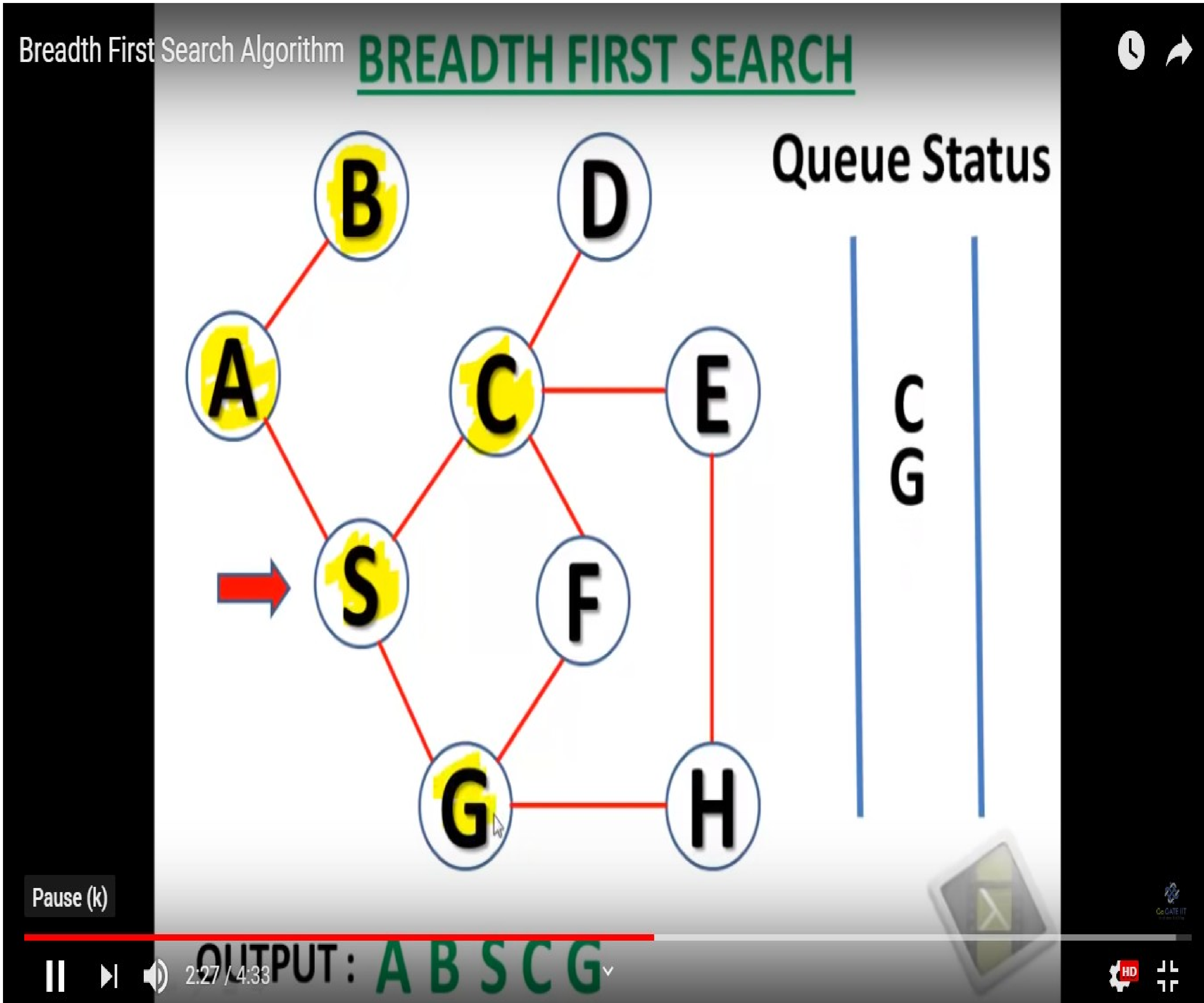
```
BFS-B(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   Q := EmptyQueue
   visited[s] := true
   Enqueue(Q,s)
   while not Empty(Q) do
      u := Dequeue(Q)
      for all w in Adj[u] do
      if not visited[w] then
         visited[w] := true
         Enqueue(Q,w)
      end if
   end while
```

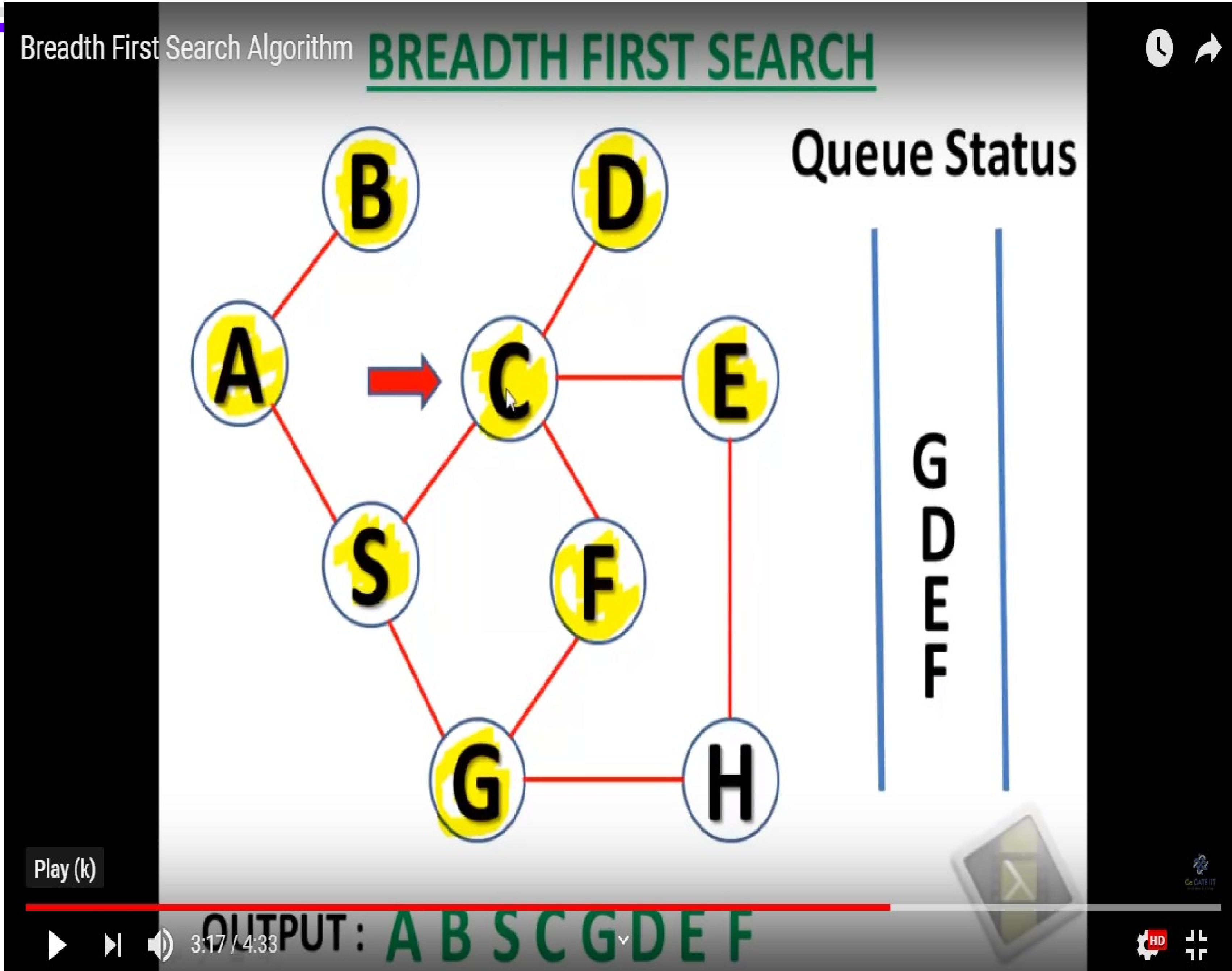
BREADTH FIRST SEARCH

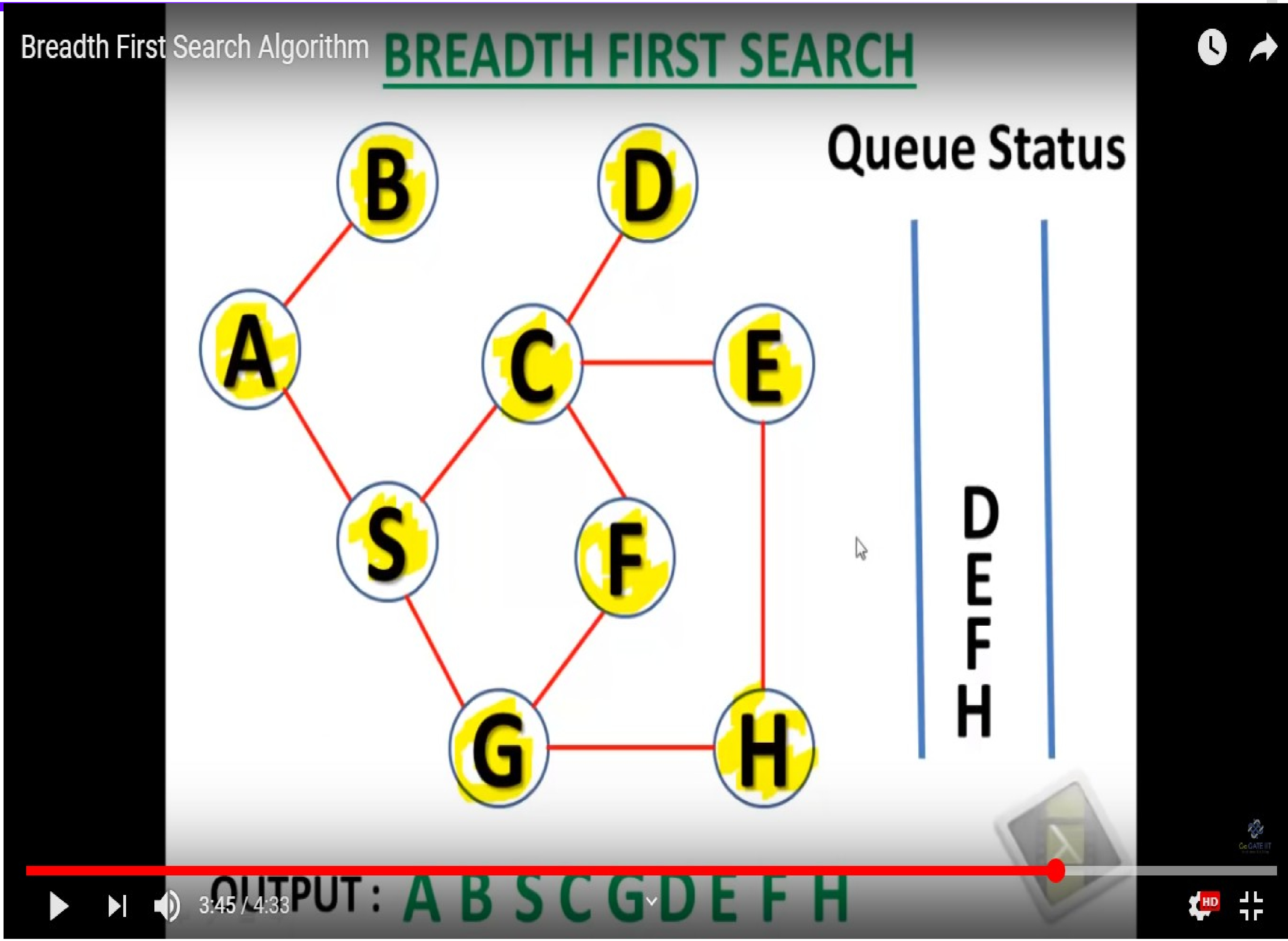


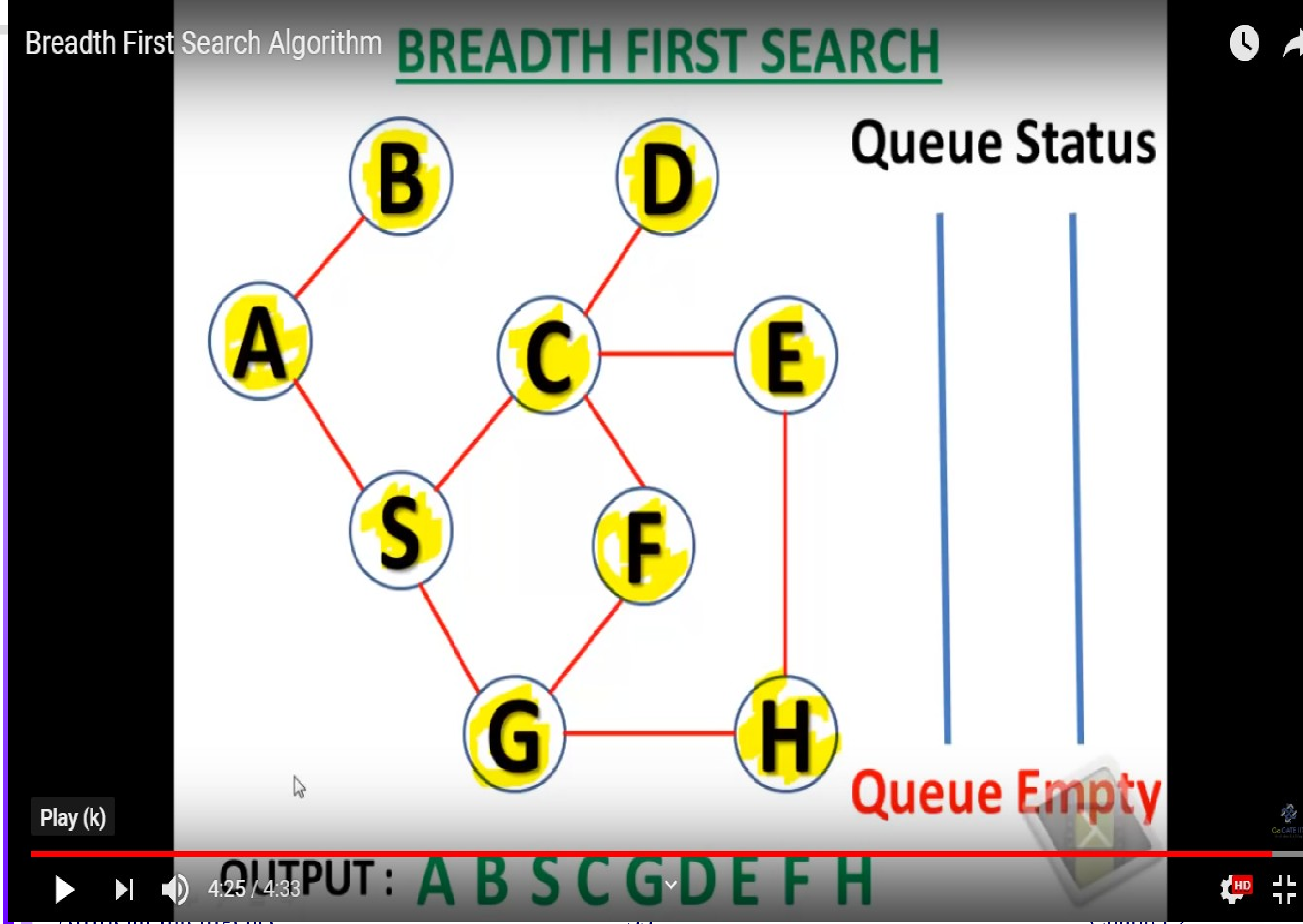
OUTPUT : A B


Breadth First Search Algorithm
BREADTH FIRST SEARCH
Queue Status
B
D
A
C
E
S
F
G
H
S
Pause (k)
1:55 / 4:33
OUTPUT : A B S


Breadth First Search Algorithm
BREADTH FIRST SEARCH
Queue Status
B
D
A
C
E
S
F
G
H
C
G
OUTPUT : A B S C G
Pause (k)
2:27 / 4:33
HD


Breadth First Search Algorithm
BREADTH FIRST SEARCH
Queue Status
B
D
A
C
E
S
F
G
H
G
D
E
F
Play (k)
OUTPUT : A B S C G D E F
3:17 / 4:33


Breadth First Search Algorithm
BREADTH FIRST SEARCH
Queue Status
B
D
A
C
E
S
F
G
H
D
E
F
H
3:45 / 4:33
OUTPUT : A B S C G D E F H


Breadth First Search Algorithm
BREADTH FIRST SEARCH
B
D
A
C
E
S
F
G
H
Queue Status
Queue Empty
Play (k)
4:25 / 4:33
OUTPUT : A B S C G D E F H

Advantage BFS

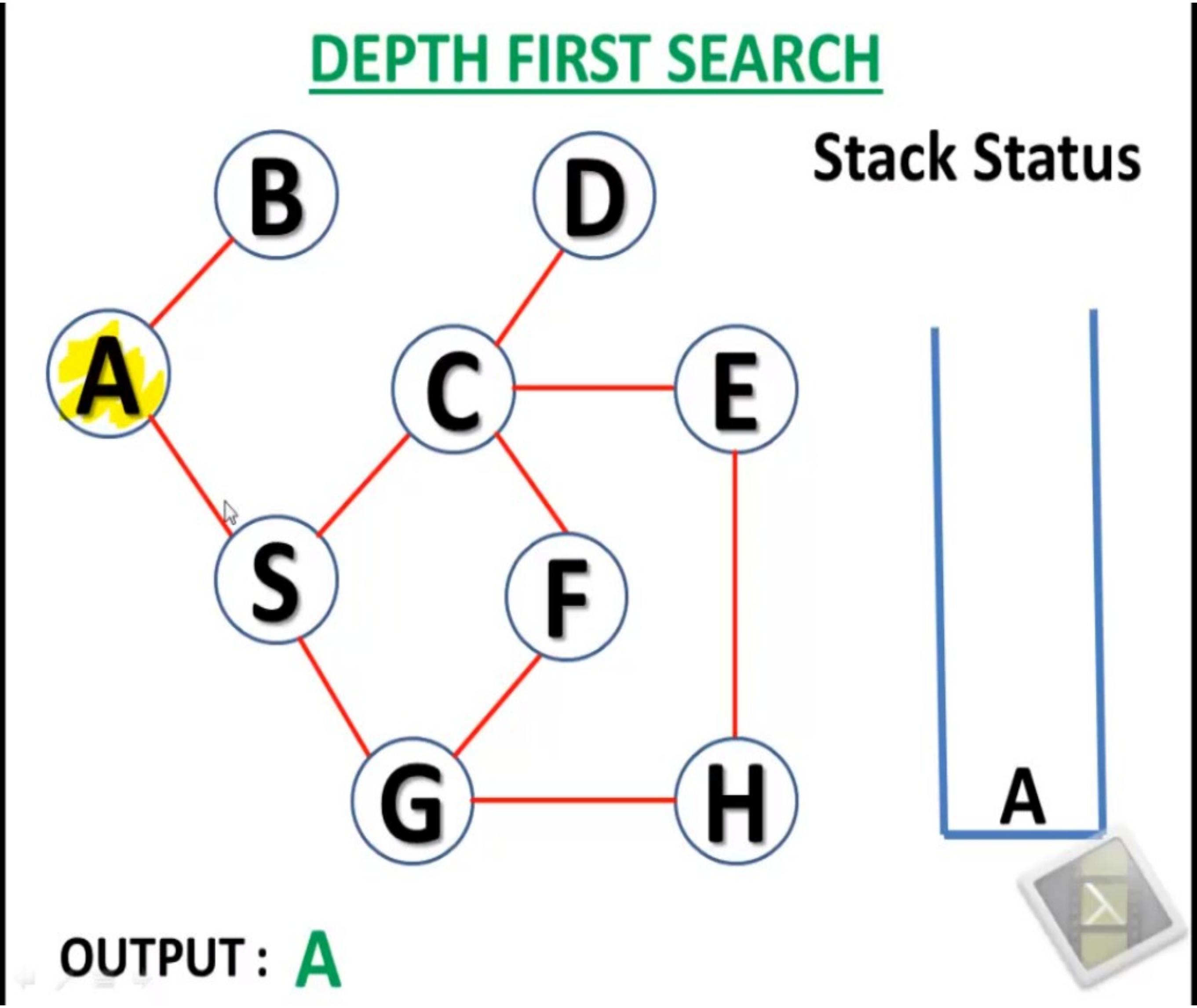
1. *will not trapped exploring a blind alley*
2. *if there is a solution, BFS is guaranteed to find it.*
3. *if there are multiple solutions, a minimum solution will be found.*

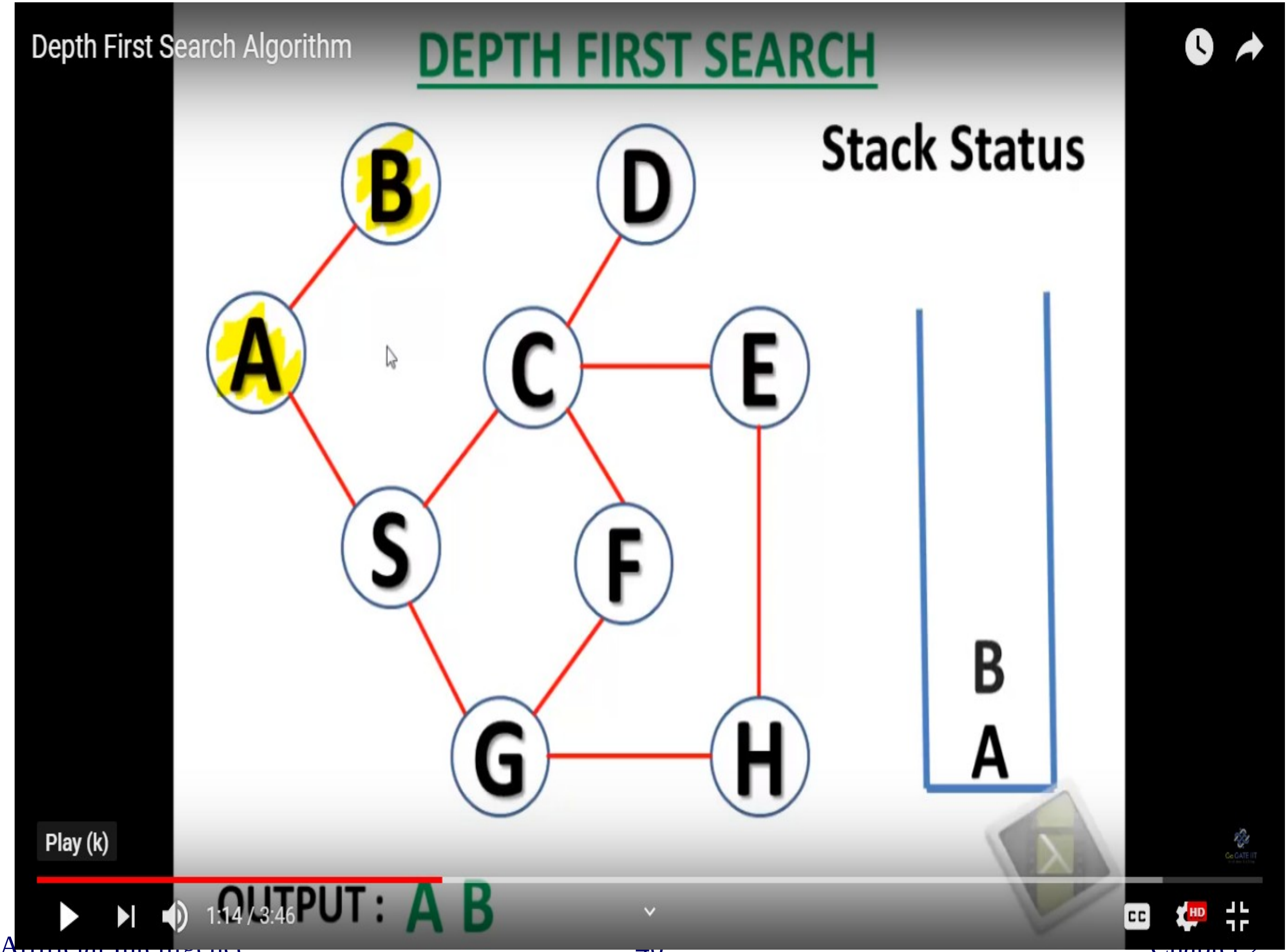
Depth-first search

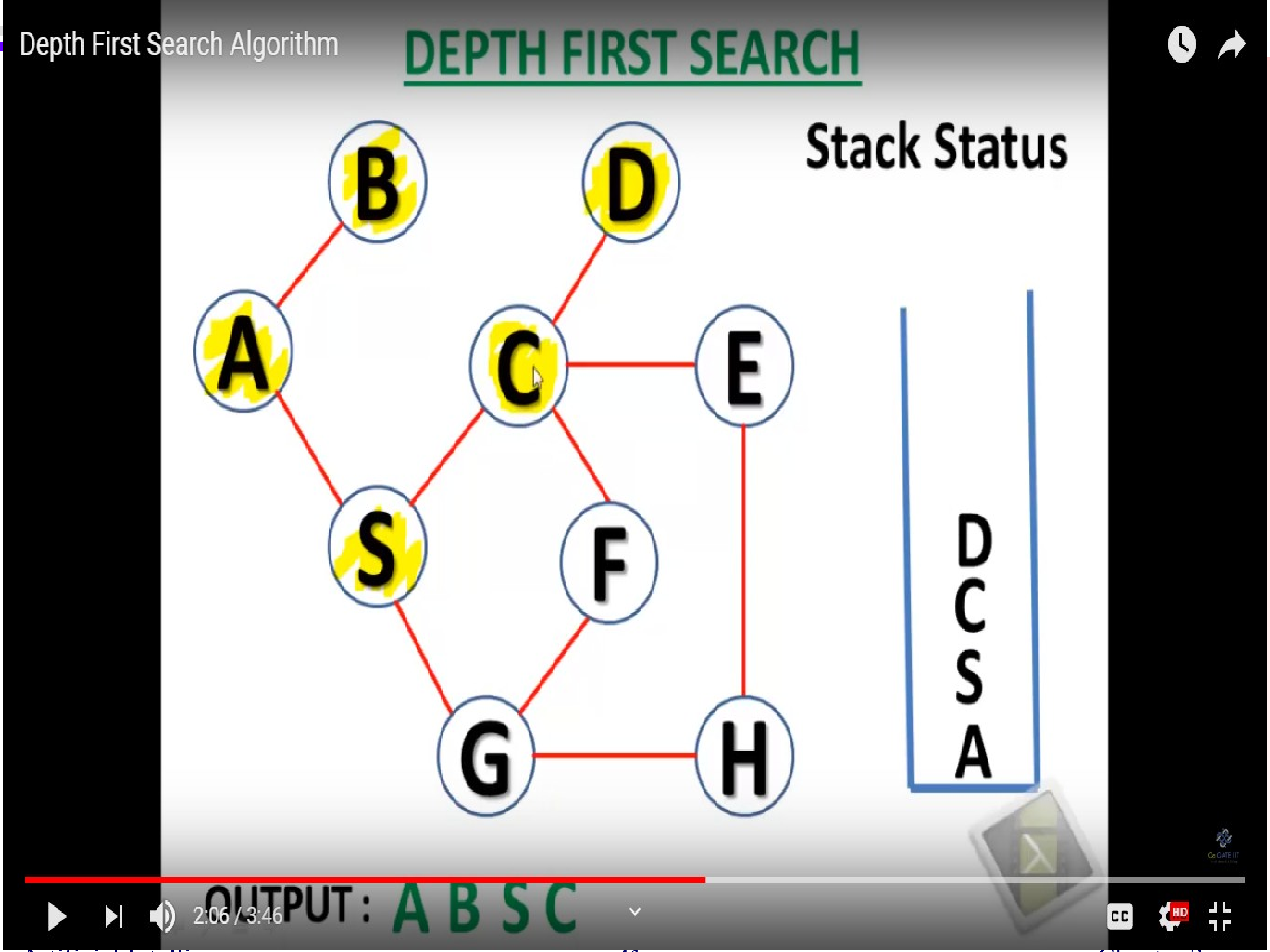
Algorithm DFS:

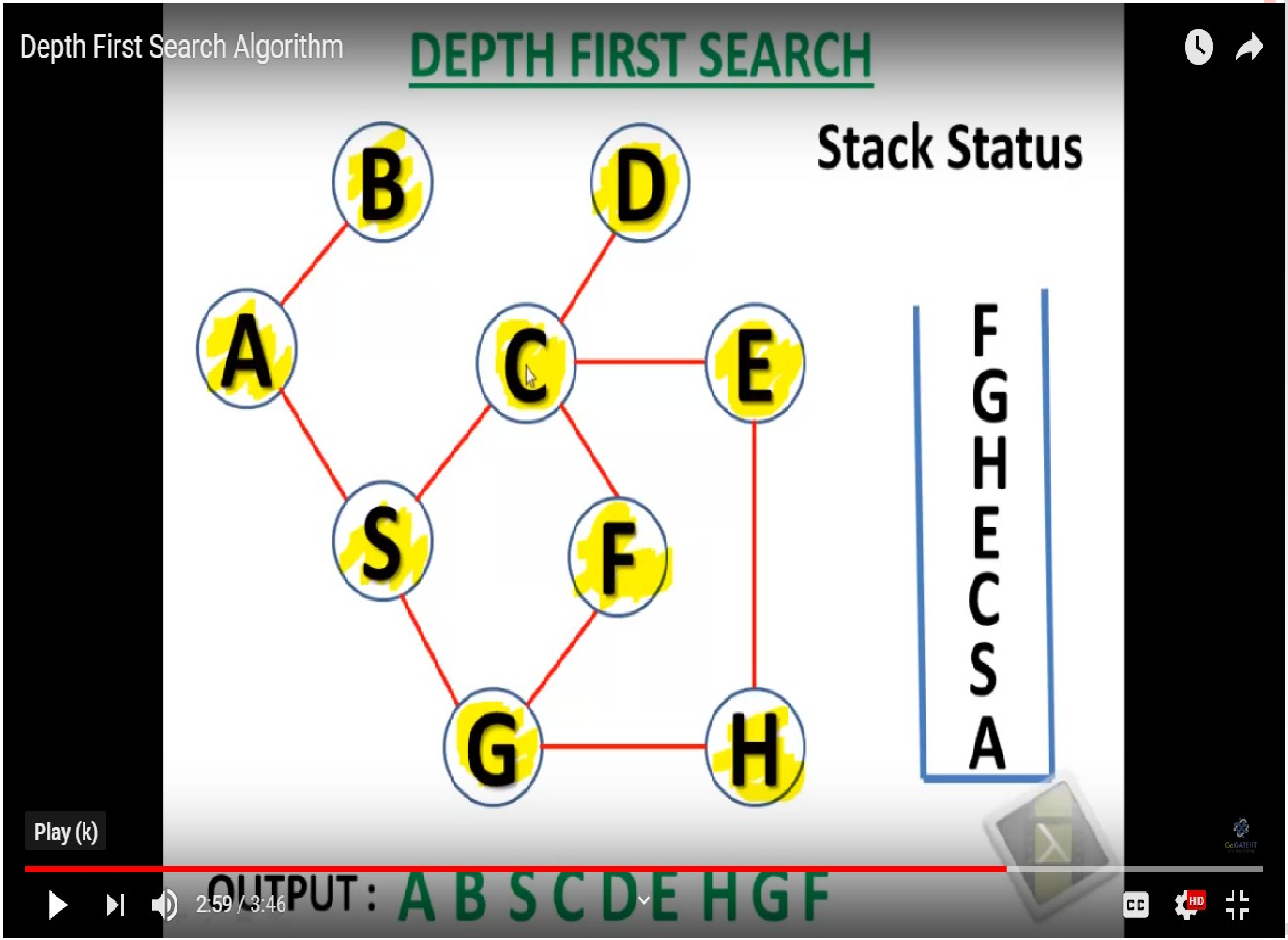
1. *If the initial state is a goal state, quit and return success.*
2. *Otherwise, do the following until success or failure is signaled:*
   - Generate a successor, E, of the initial state. If there are no more successors, signal failure.
   - Call Depth-First Search with E as the initial state.
   - If success is returned, signal success. Otherwise continue in this loop.

```
DFS-A(G,s)
   for all v in V[G] do
      visited[v] := false
   end for
   S := EmptyStack
   Push(S,s)
   while not Empty(S) do
      u := Pop(S)
      if not visited[u] then
         visted[u] := true
         for all w in Adj[u] do
            if not visited[w] then
               Push(S,w)
            end if
         end for
      end if
   end while
```


DEPTH FIRST SEARCH
Stack Status
B
D
A
C
E
S
F
G
H
A
OUTPUT : A


Depth First Search Algorithm
DEPTH FIRST SEARCH
Stack Status
B
D
A
C
E
S
F
G
H
B
A
Play (k)
1:14 / 3:46
OUTPUT : A B


Depth First Search Algorithm
DEPTH FIRST SEARCH
Stack Status
B
D
A
C
E
S
F
G
H
D
C
S
A
OUTPUT : A B S C
2:06 / 3:46


Depth First Search Algorithm
DEPTH FIRST SEARCH
Stack Status
B
D
A
C
E
S
F
G
H
F
G
H
E
C
S
A
Play (k)
OUTPUT : A B S C D E H G F
2:59 / 3:46

Advantage DFS

1. *require less memory*
2. *may find a solution without examining much of the search space.*

Heuristic Search

- *Heuriskein → to discover (Greek word)*
- *Heuristic is a technique that improves the efficiency of the search process.(tour guide)*
- *- It is often useful to introduce heuristics based*
- *on relatively unstructured knowledge.*
- *- can not use Mathematical analysis.*
- *Heuristic function : is the function that maps from problem state descriptions to measures of desirability, usually represent as number. → guide the most profitable direction*

To solve a problem

1. Define the problem precisely. Specify the problem space, and the starting and goal state (s).
2. Analyze the problem to determine where it falls with respect to seven important issues.
3. Identify and represent the knowledge required by the task.
4. Choose one or more techniques for problem solving , and apply those techniques to the problem.

Referencs

Introduction to Artificial Intelligence by Rusell Norving