ARTIFICIAL INTELLIGENCE

Lecture Set 05

Uninformed Search

By Engr. Ali Asghar Manjotho
Lecturer Department of Computer Systems Engineering
MUET Jamshoro
ali.manjotho@faculty.muet.edu.pk
ali.manjotho.ali@gmail.com

Outlines

- Uninformed Search
- Breadth-First Search
- Backward Breadth-First Search
- Bi-Directional Breadth-First Search
- Depth-First Search
- Depth Limited Search
- Depth-First Iterative Deeping Search (DFID)
- Evaluating Search Algorithms
- Uniform Cost Search
- Repeated States
- How to deal with repeated states?

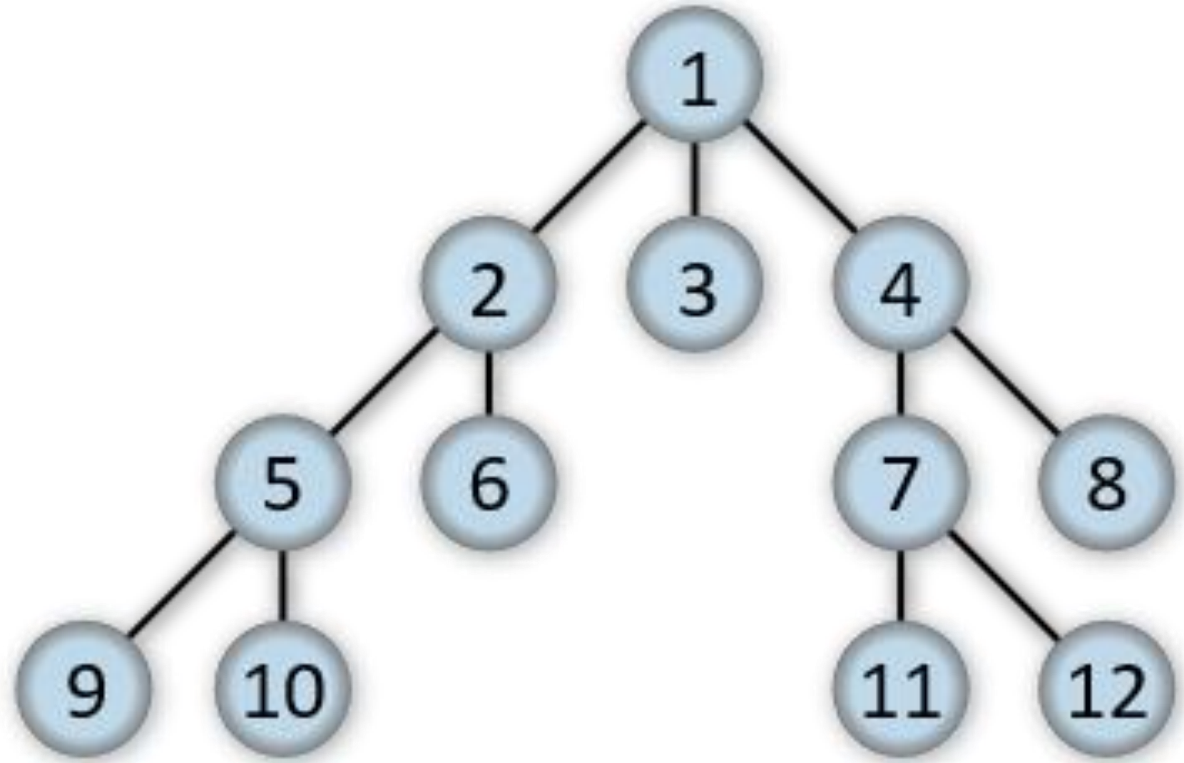
Uninformed Search

- Also called as Blind Search or Brute Force Search.
- A blind search is a search that has no information about its domain. The only thing that a blind search can do is to distinguish a non-goal state from a goal state.
  - *Breadth-First search*
  - *Depth-First search*
  - *Uniform-Cost search*
  - *Depth-First Iterative Deepening search*

Breadth-First Search

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors.
- Breadth First Search explores the state space in a level by level fashion. Only when there are no more states to be explored at a given level then the algorithm move onto the next level.

Breadth-First Search

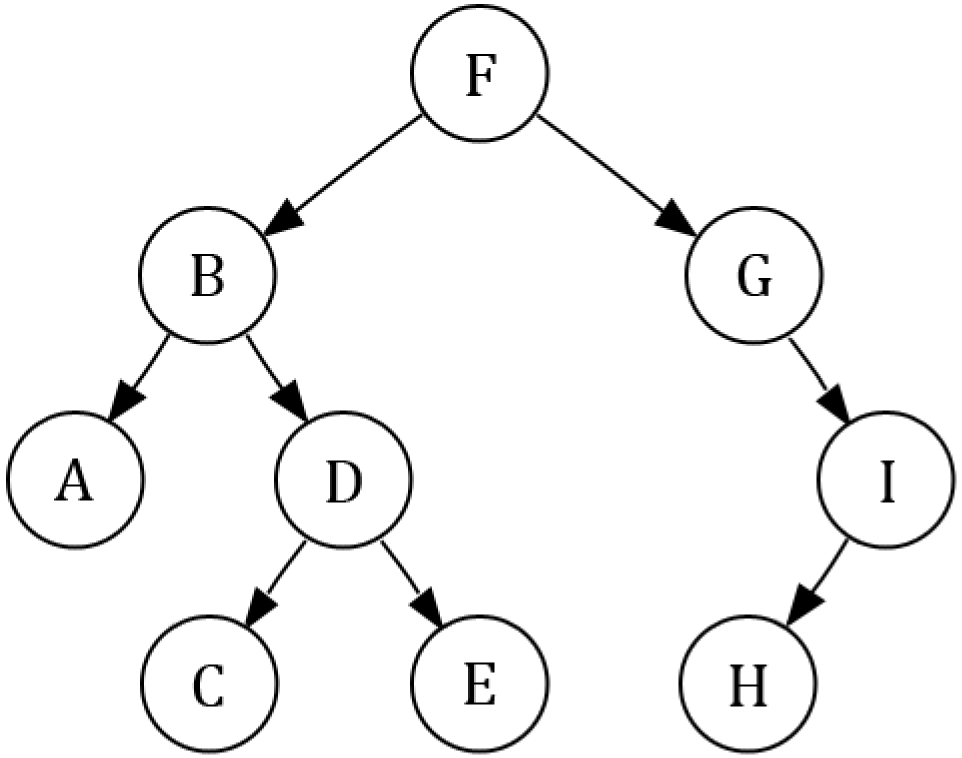


Breadth-First Search

Algorithm:

- Step 01: Choose the starting node, mark it as visited, make it current node.
- Step 02: Find adjacent unvisited node of current node, place it in Queue and mark it visited.
- Step 03: If no unvisited adjacent node found, dequeue node from Queue and make it current node.
- Step 04: Repeat Step 02 to Step 03 until Queue is empty.

Breadth-First Search (Problem 01)



BFS Traversal: F B G A D I C E H

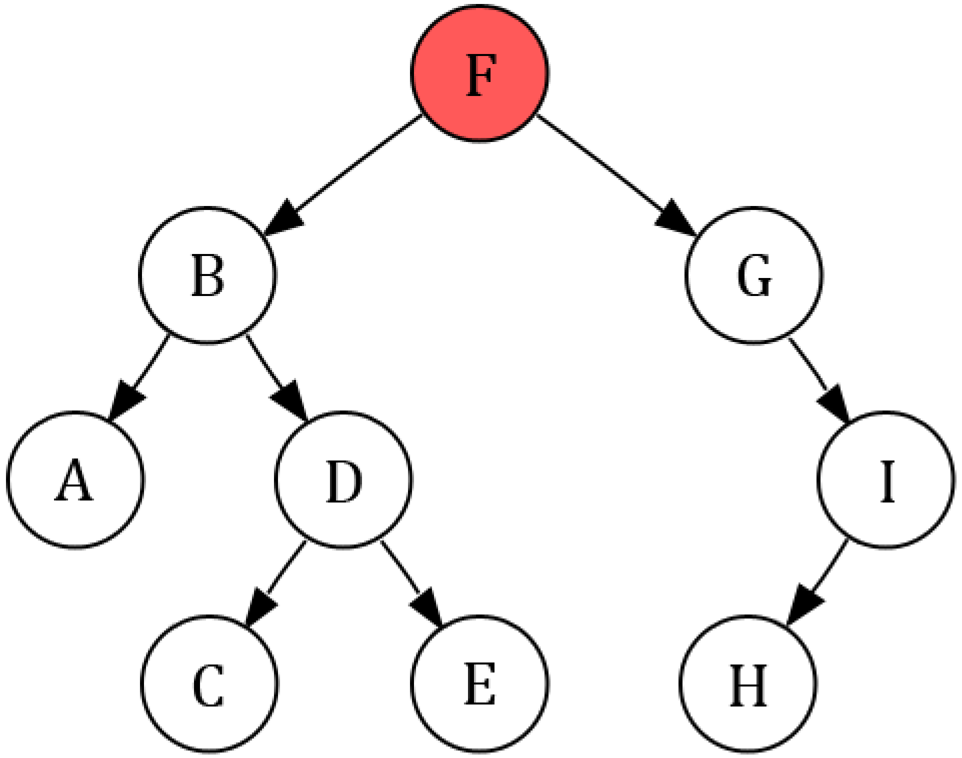
Breadth-First Search (Problem 01)

Current Node = F

Queue:

Visited:

F									



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

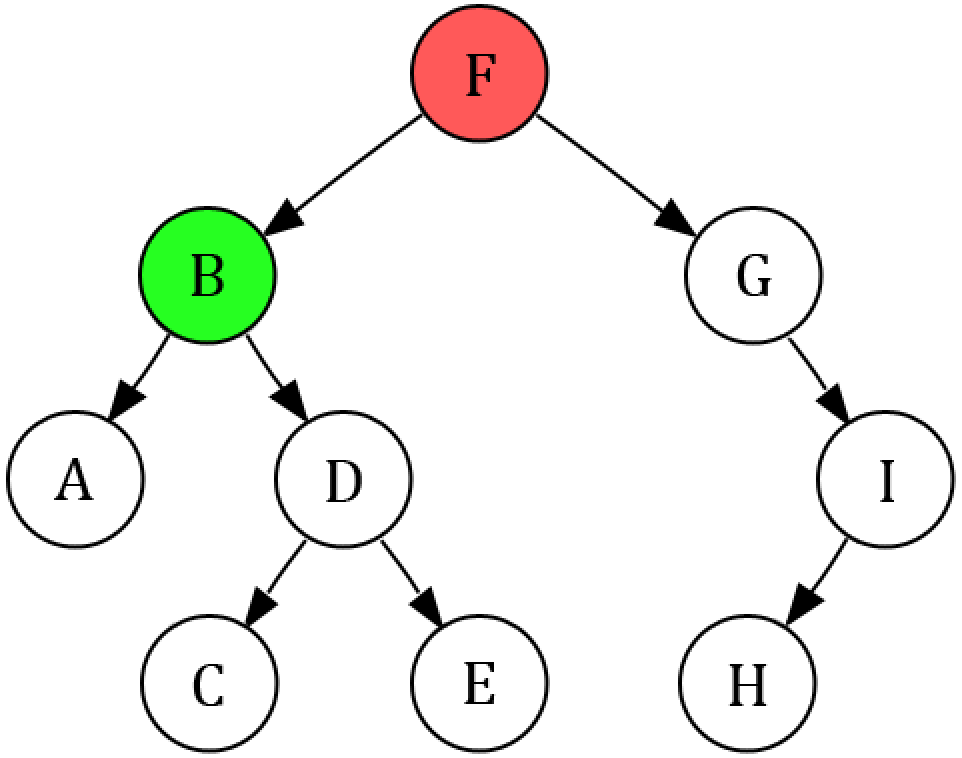
Current Node = F

Queue:

B									

Visited:

F	B								



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

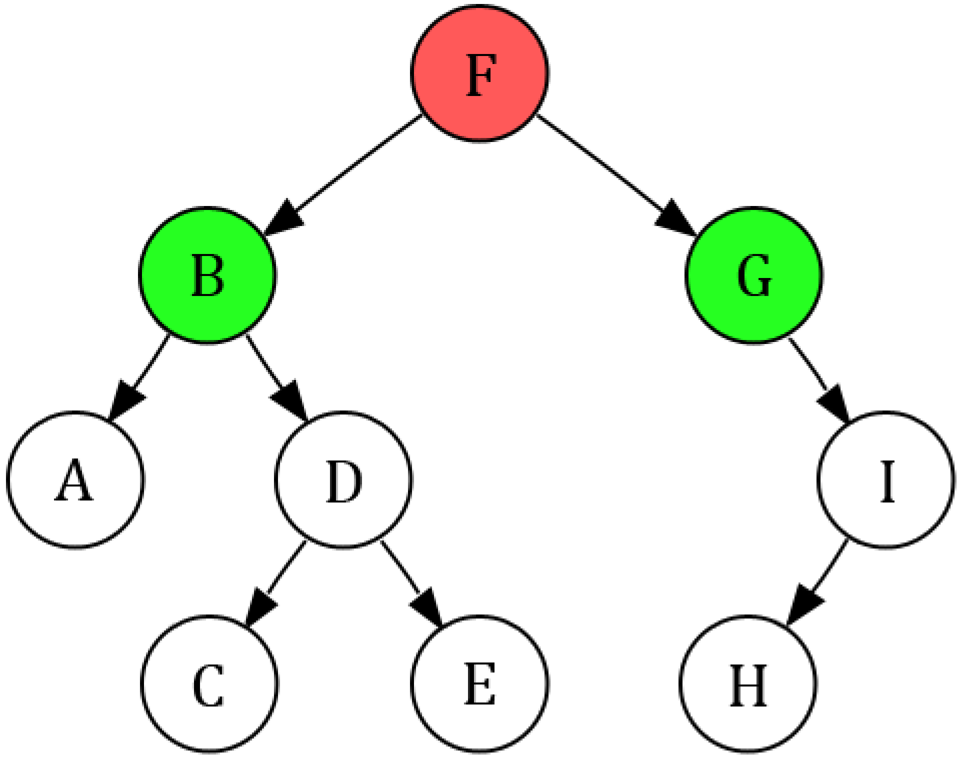
Current Node = F

Queue:

G	B								

Visited:

F	B	G							



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 01)

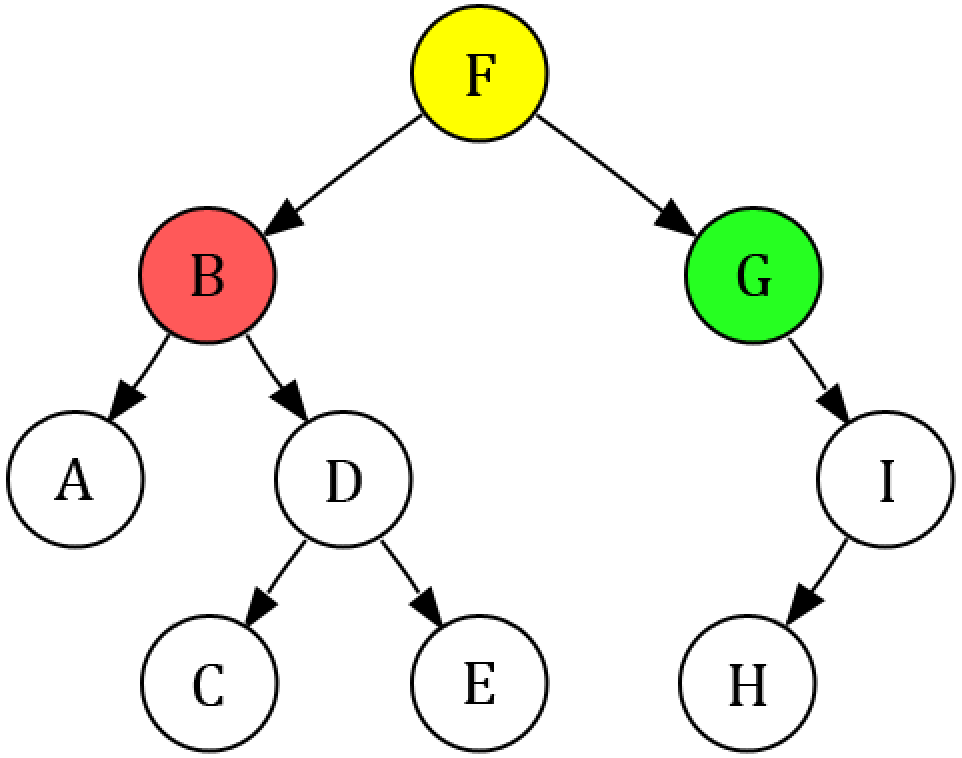
Current Node = B

Queue:

G									

Visited:

F	B	G							



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

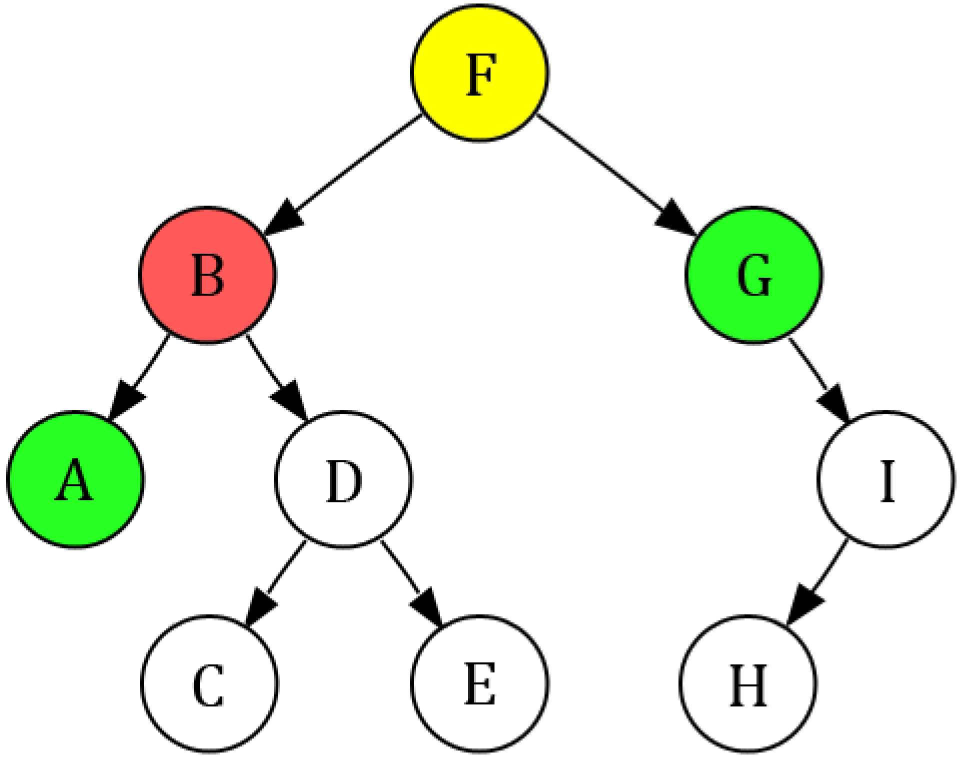
Current Node = B

Queue:

A	G								

Visited:

F	B	G	A						



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

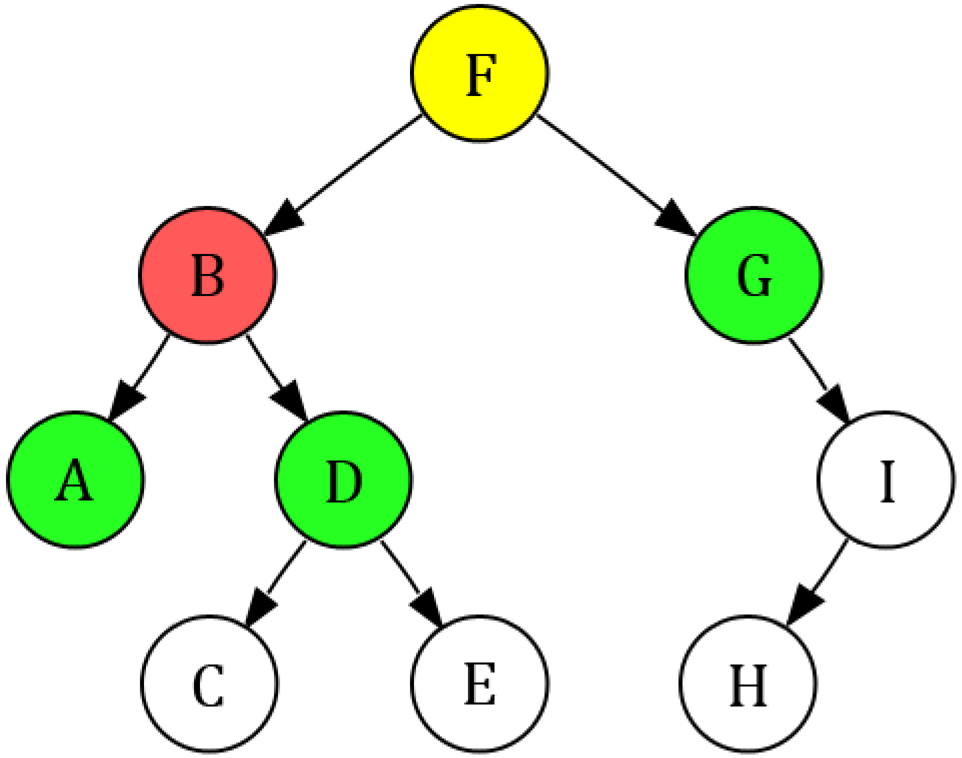
Current Node = B

Queue:

D	**A**	**G**							

Visited:

F	**B**	**G**	**A**	**D**					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

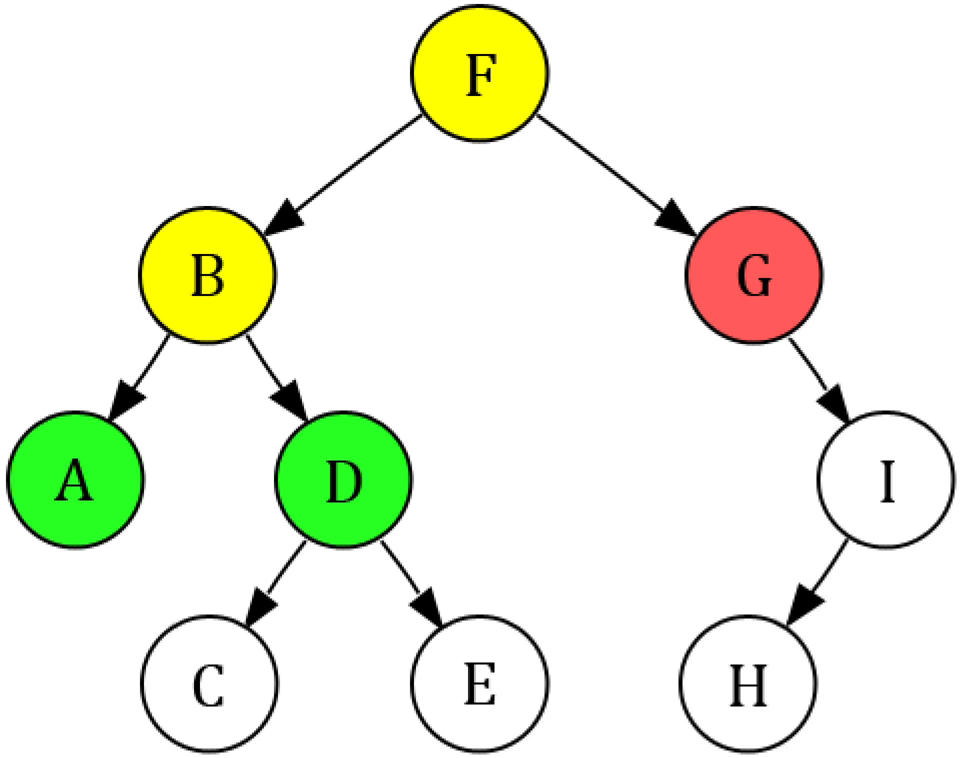
Current Node = G

Queue:

D	**A**								

Visited:

F	**B**	**G**	**A**	**D**					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

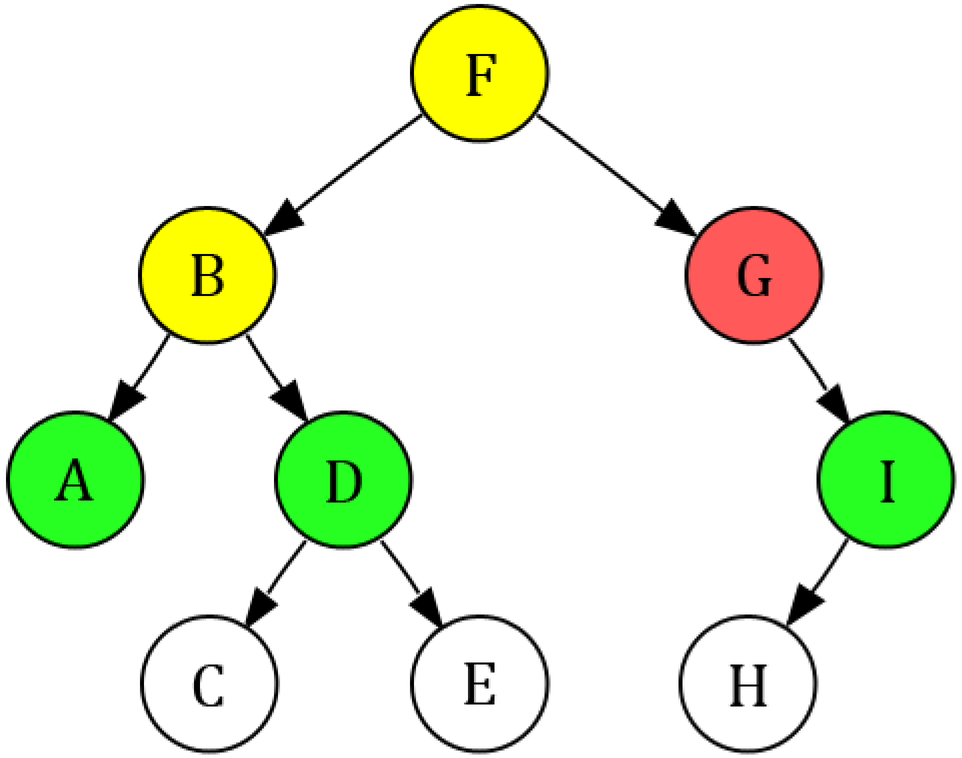
Current Node = G

Queue:

I	D	A							

Visited:

F	B	G	A	D	I				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

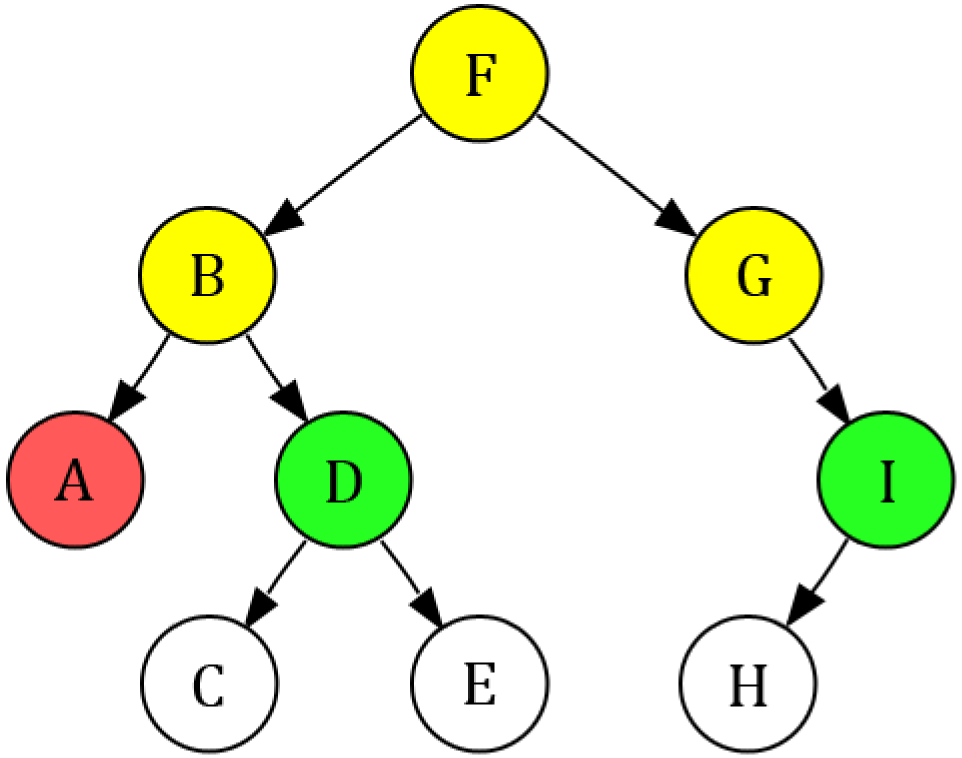
Current Node = A

Queue:

I	D								

Visited:

F	B	G	A	D	I				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

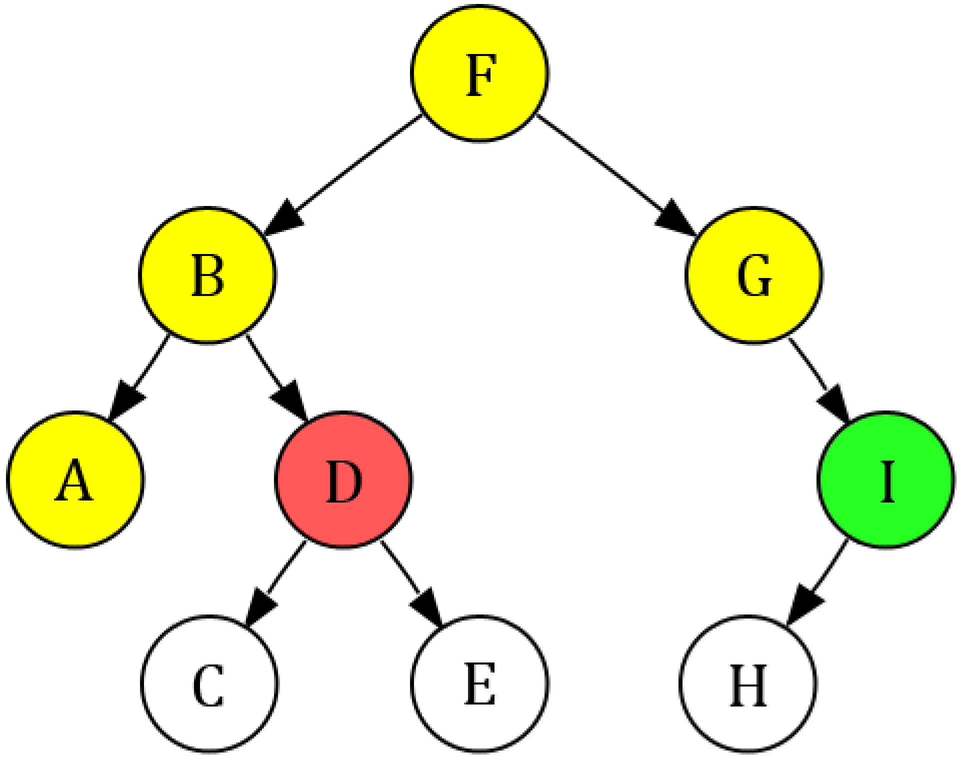
Current Node = D

Queue:

I									

Visited:

F	B	G	A	D	I				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

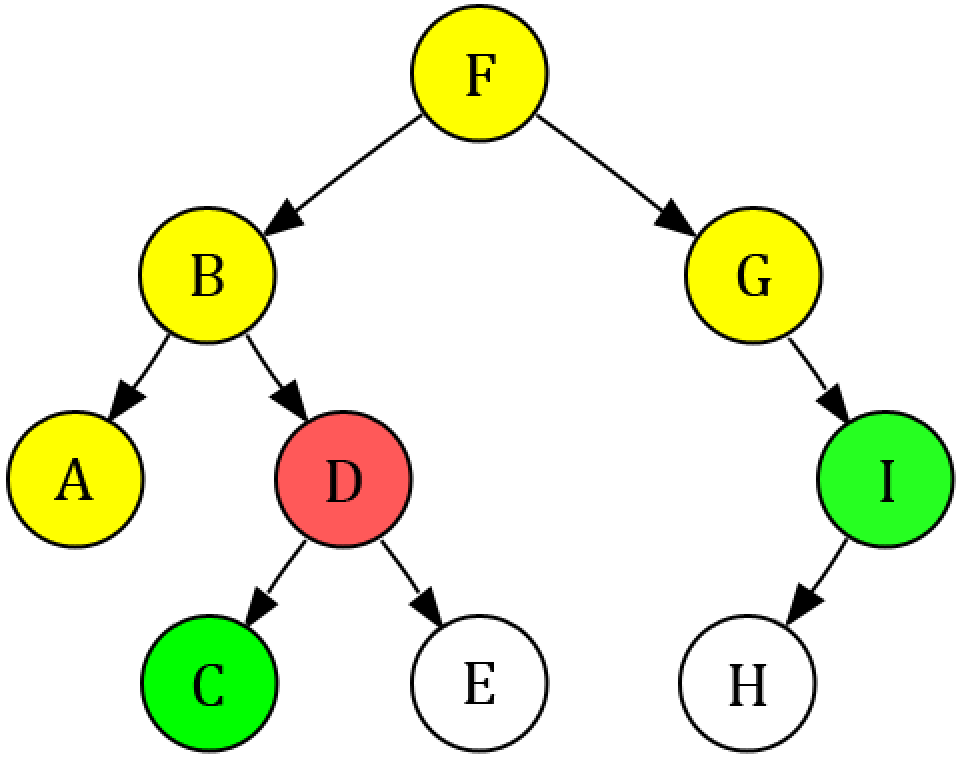
Current Node = D

Queue:

C	I								

Visited:

F	B	G	A	D	I	C			



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

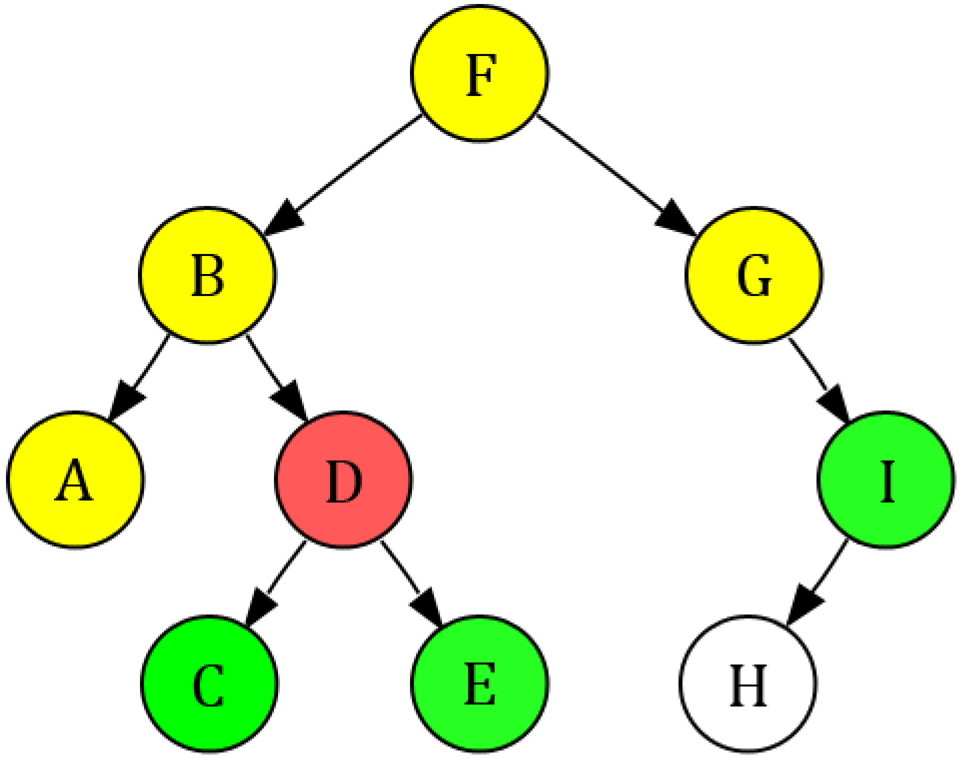
Current Node = D

Queue:

E	C	I							

Visited:

F	B	G	A	D	I	C	E		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

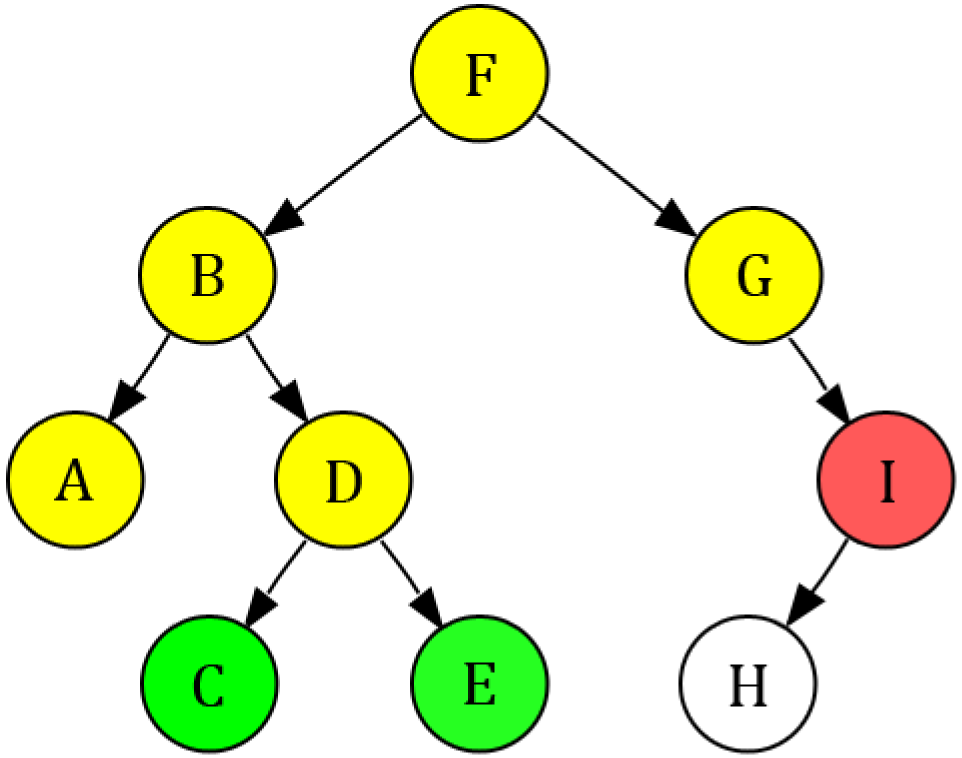
Current Node = I

Queue:

E	C								

Visited:

F	B	G	A	D	I	C	E		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 01)

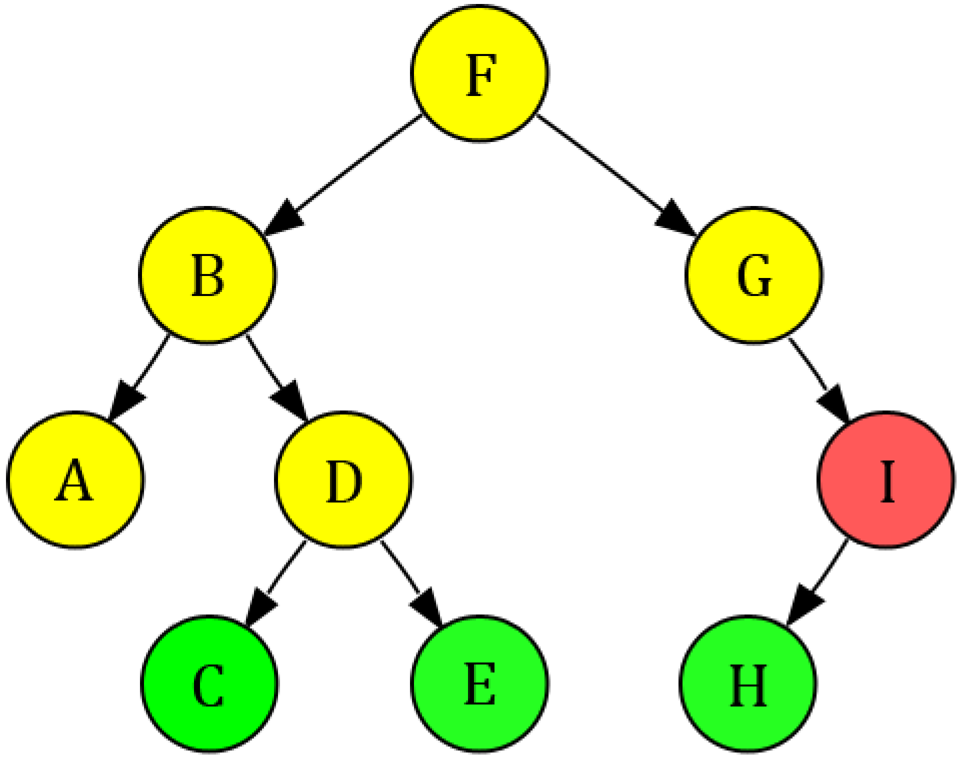
Current Node = I

Queue:

H	E	C							

Visited:

F	B	G	A	D	I	C	E	H	



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 01)

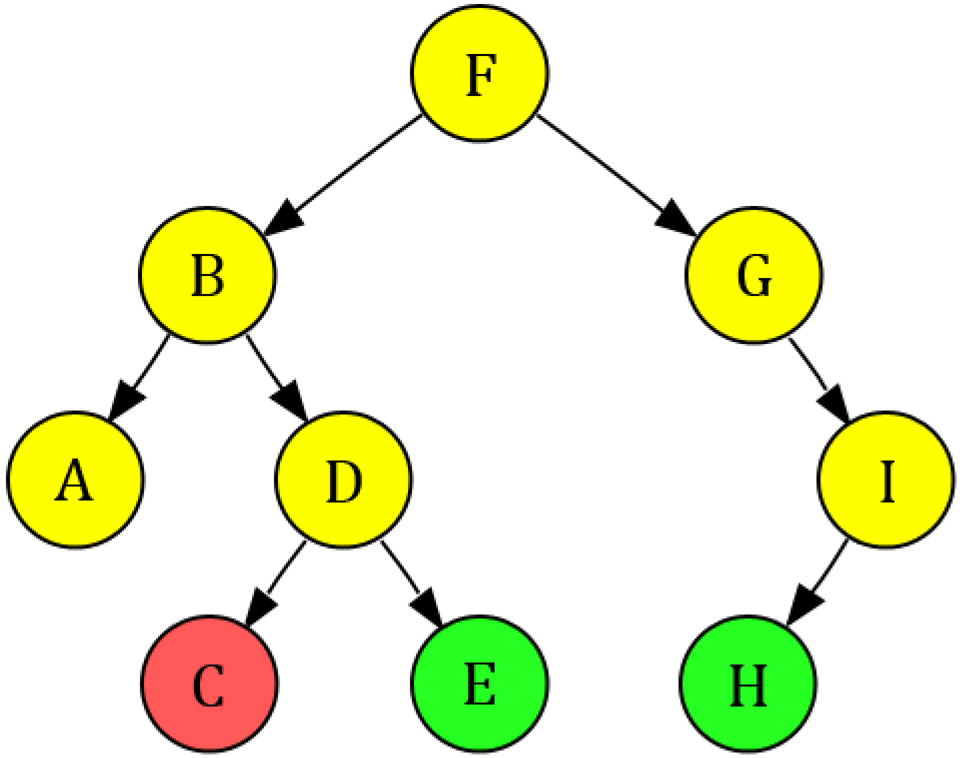
Current Node = C

Queue:

H	E								

Visited:

F	B	G	A	D	I	C	E	H	



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 01)

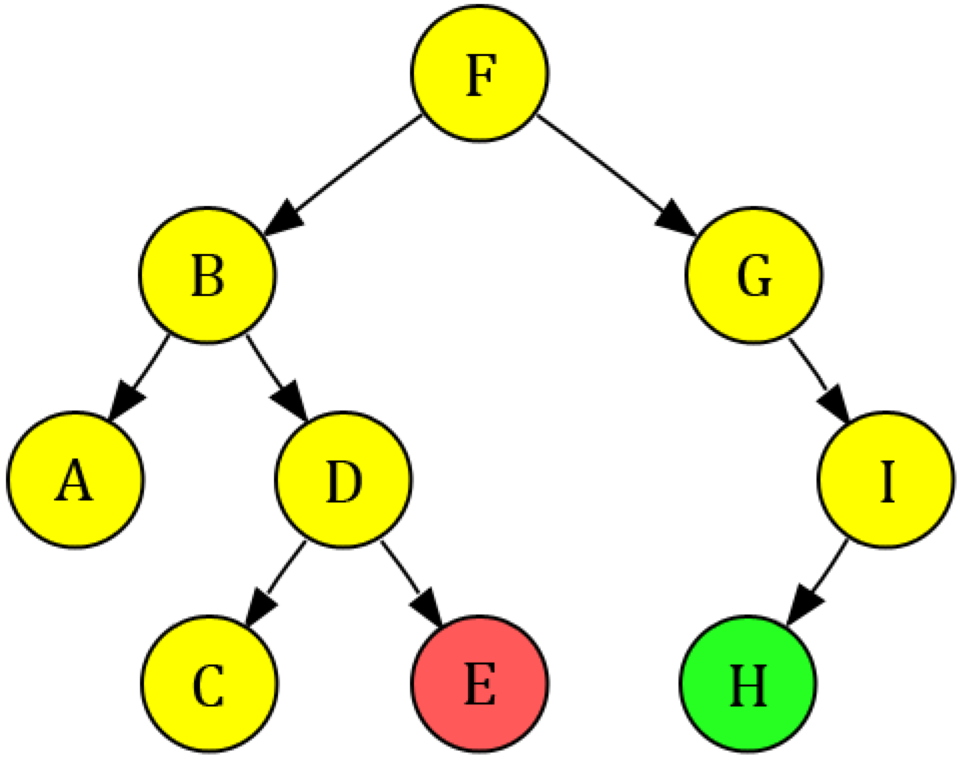
Current Node = E

Queue:

H									

Visited:

F	B	G	A	D	I	C	E	H	



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

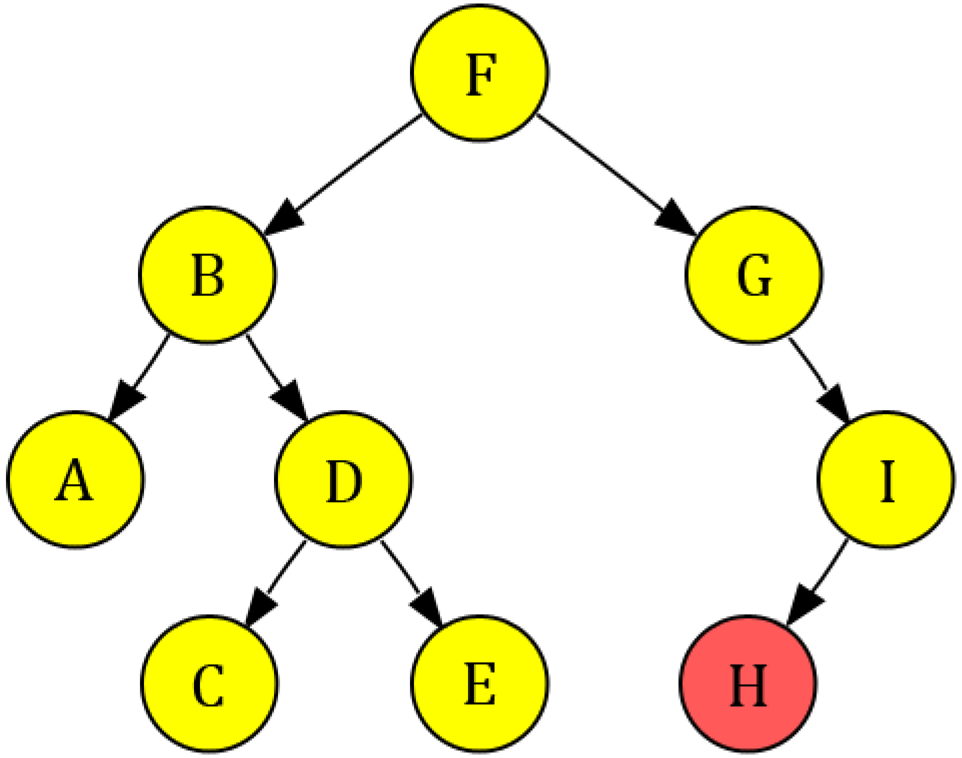
Breadth-First Search (Problem 01)

Current Node = H

Queue:

Visited:

F	**B**	**G**	**A**	**D**	**I**	**C**	**E**	**H**	



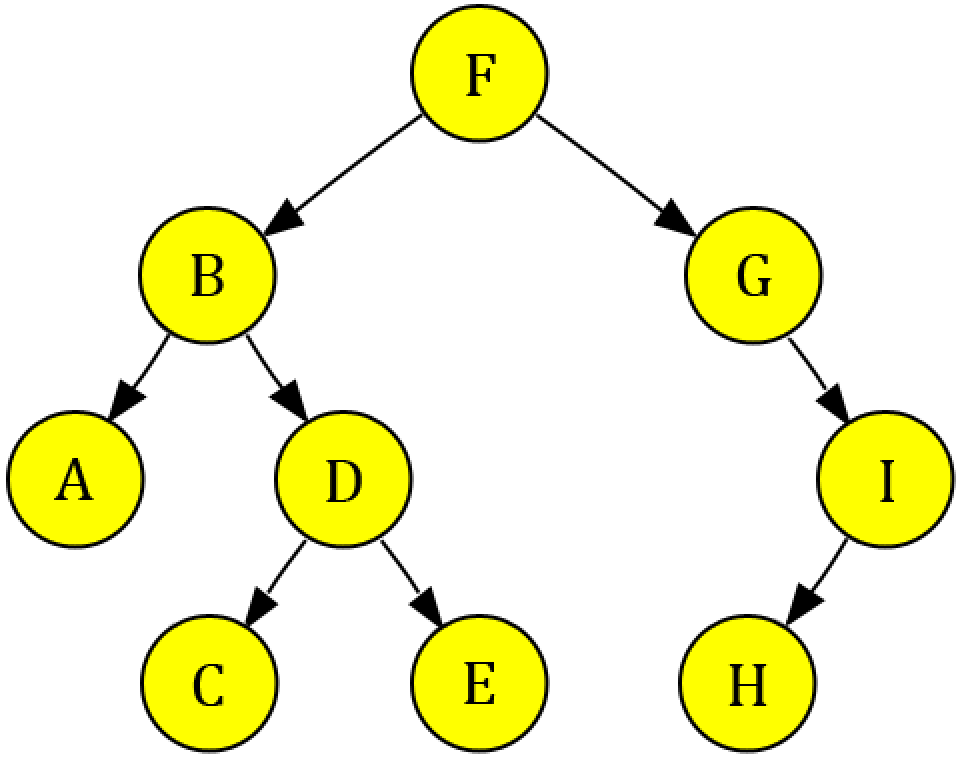
🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 01)

Queue:

Visited:

F	**B**	**G**	**A**	**D**	**I**	**C**	**E**	**H**	

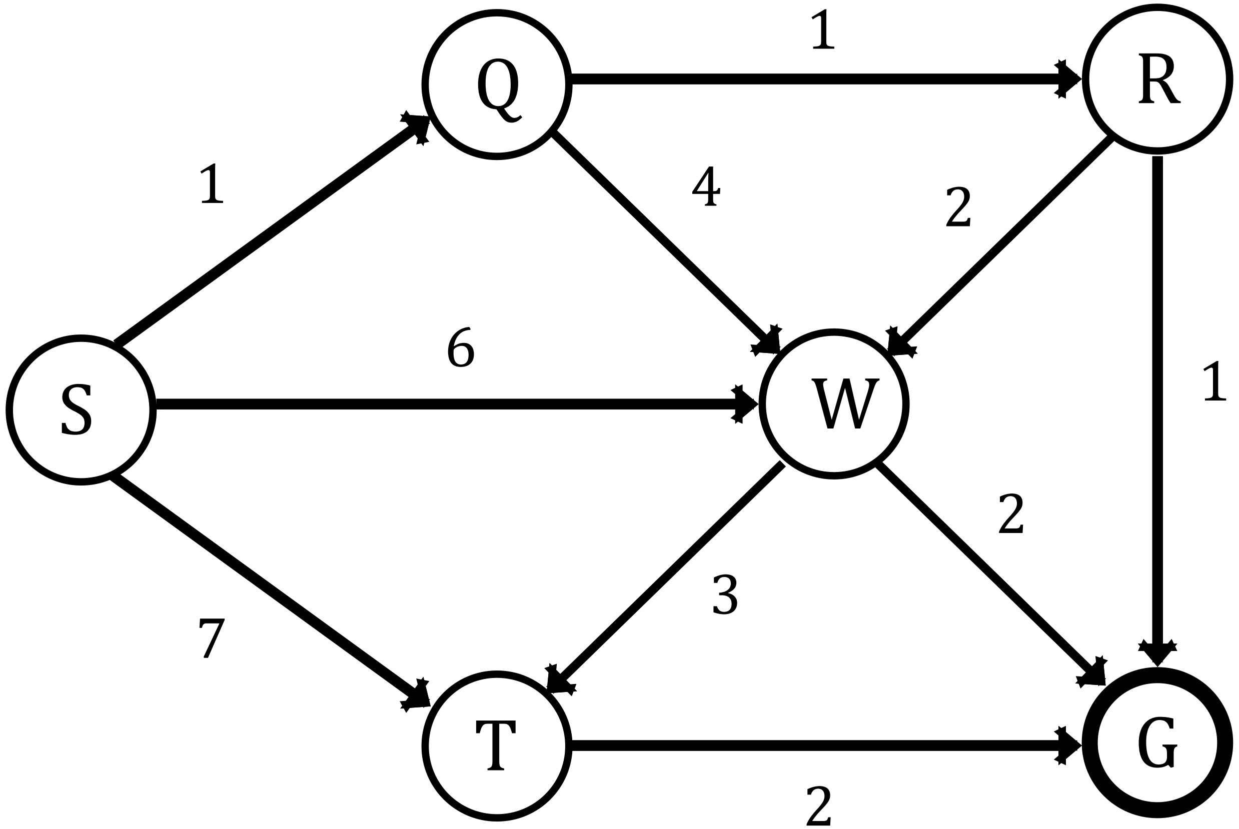


BFS Traversal: F B G A D I C E H

Breadth-First Search (Problem 02)

Queue:

Visited:



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

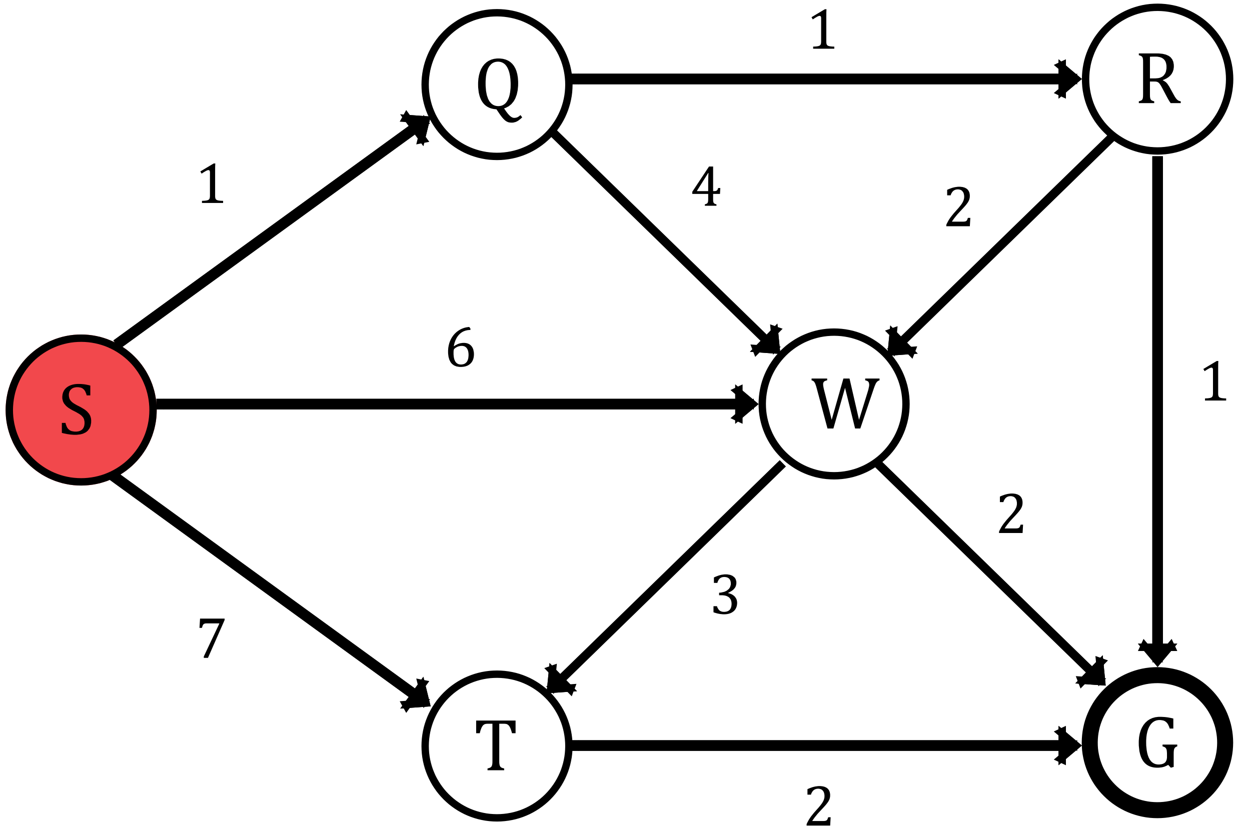
Breadth-First Search (Problem 02)

Current Node = S

Queue:

Visited:

S					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

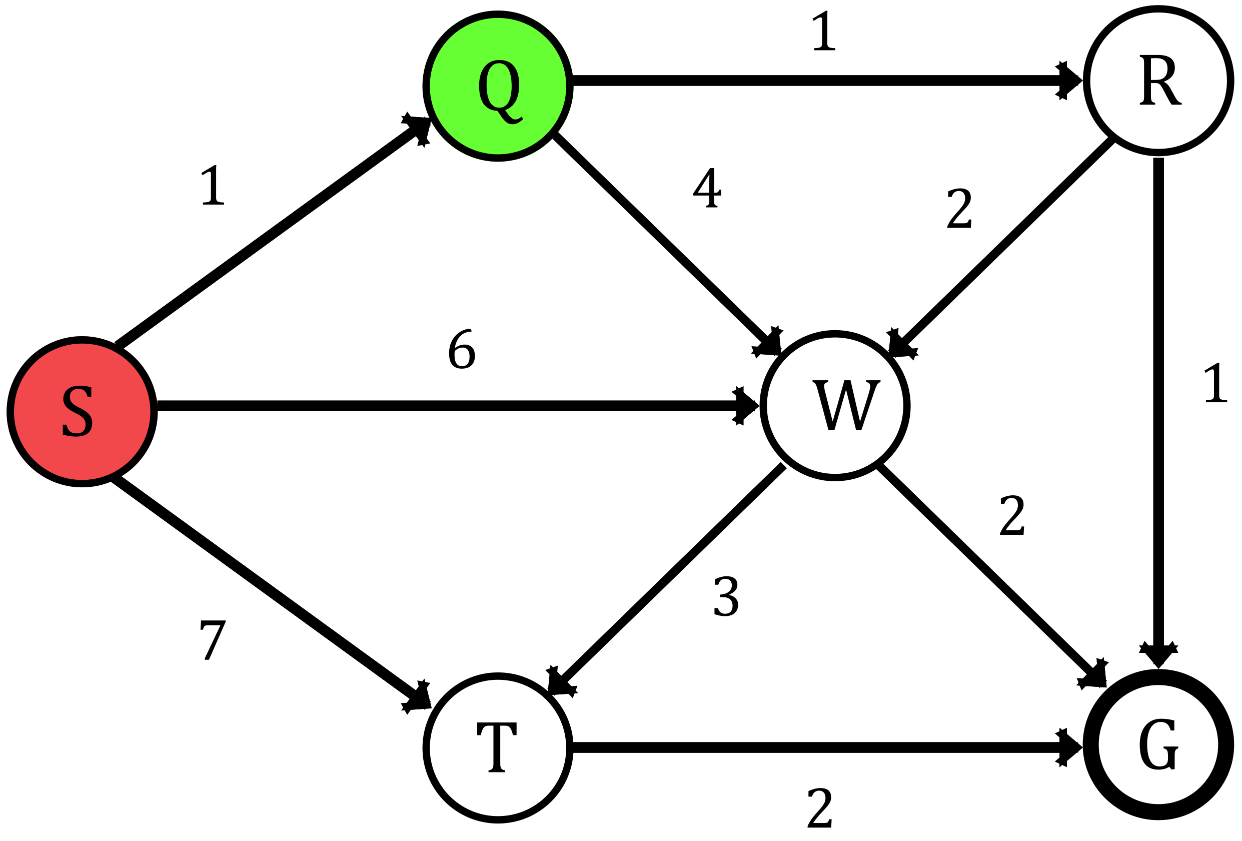
Current Node = S

Queue:

Q					

Visited:

S	**Q**				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

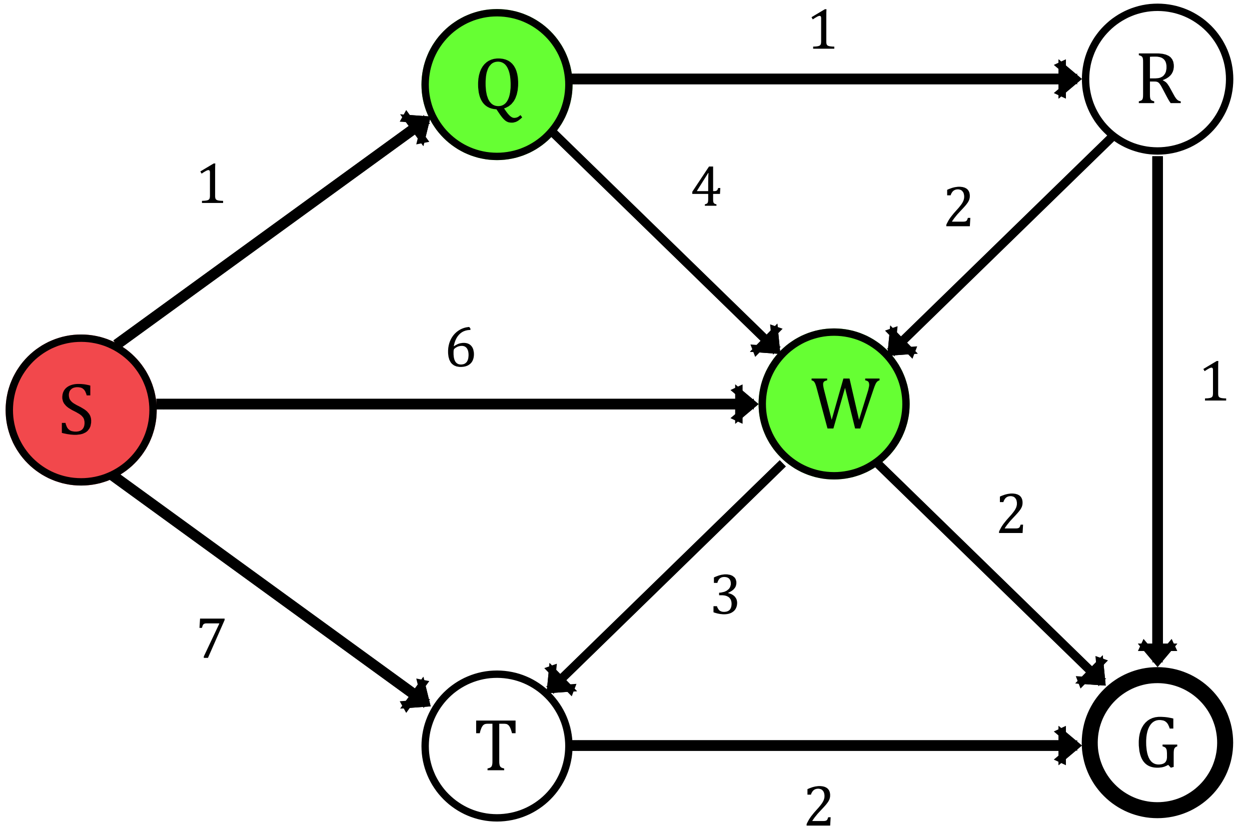
Current Node = S

Queue:

W	**Q**				

Visited:

S	**Q**	**W**			



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

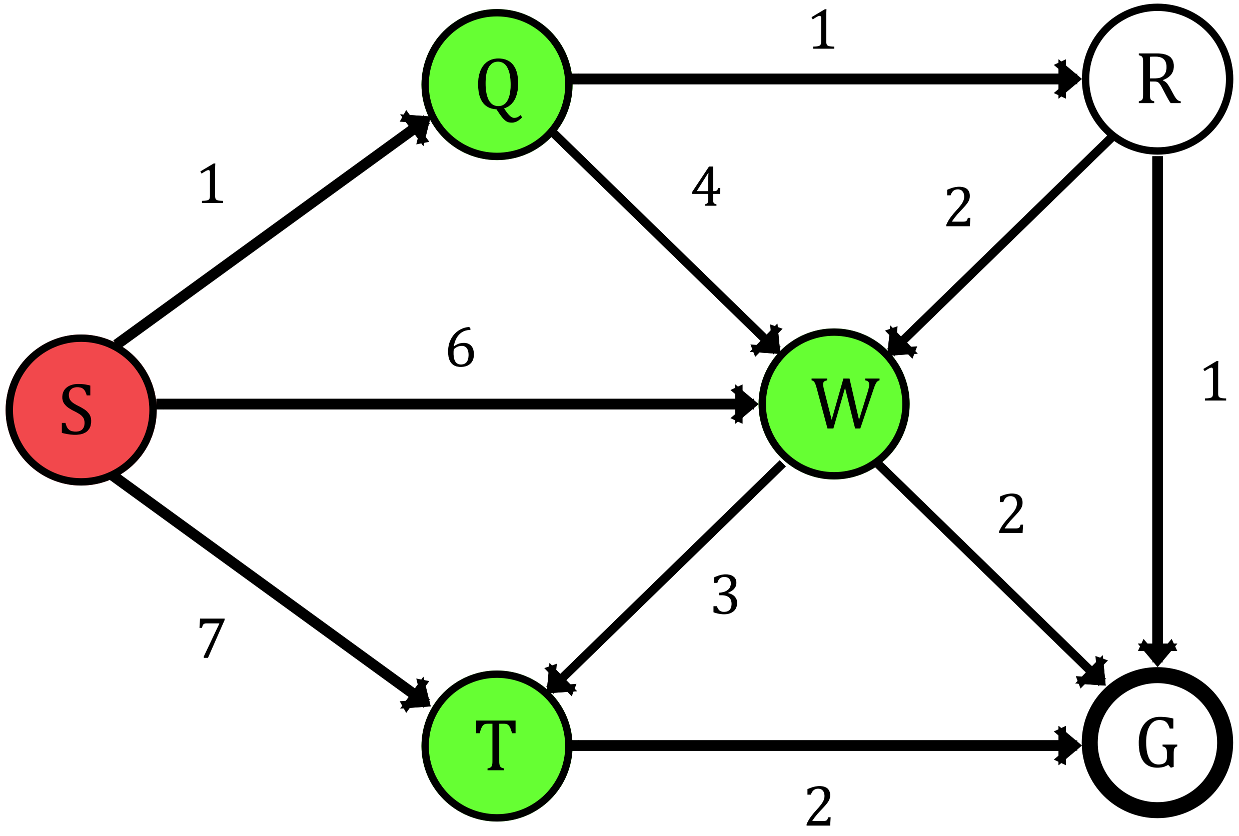
Current Node = S

Queue:

T	W	Q			

Visited:

S	Q	W	T		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

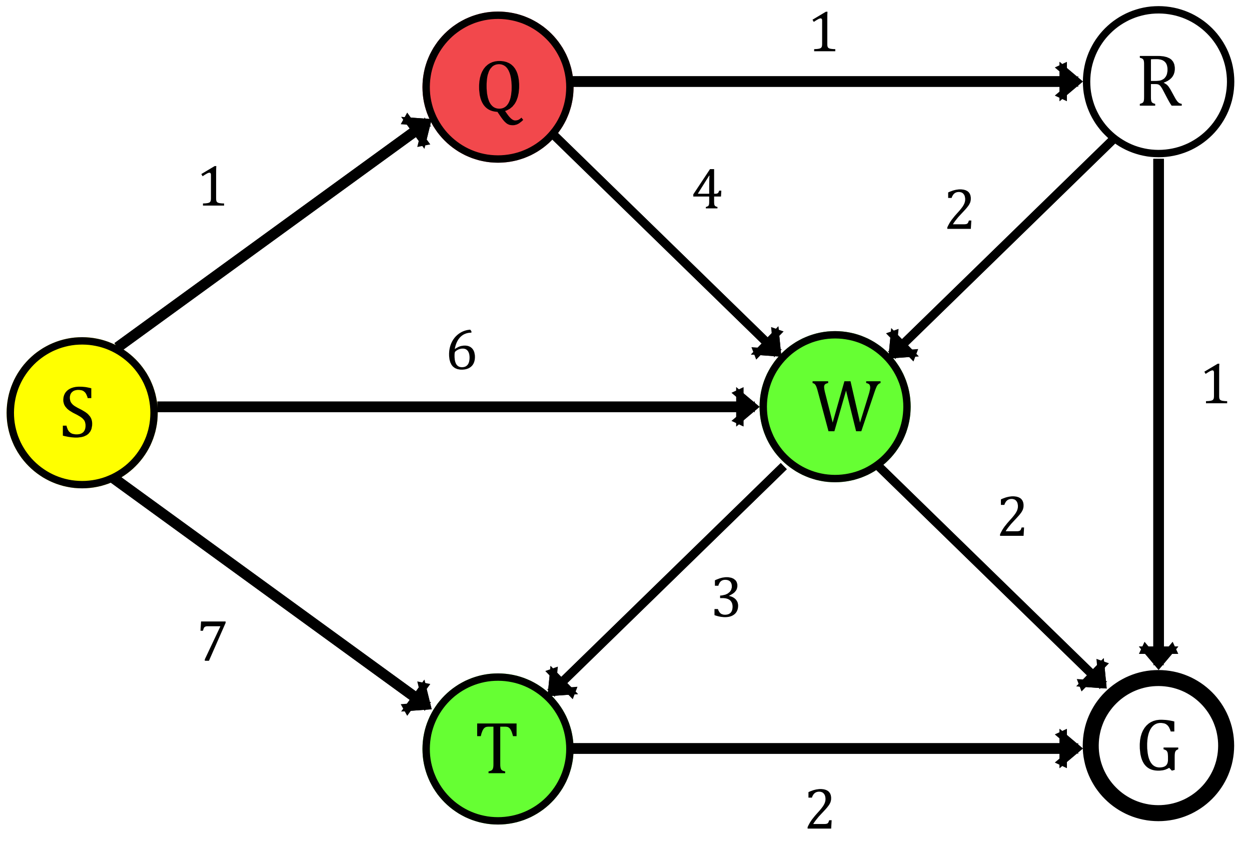
Current Node = Q

Queue:

T	W				

Visited:

S	Q	W	T		



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 02)

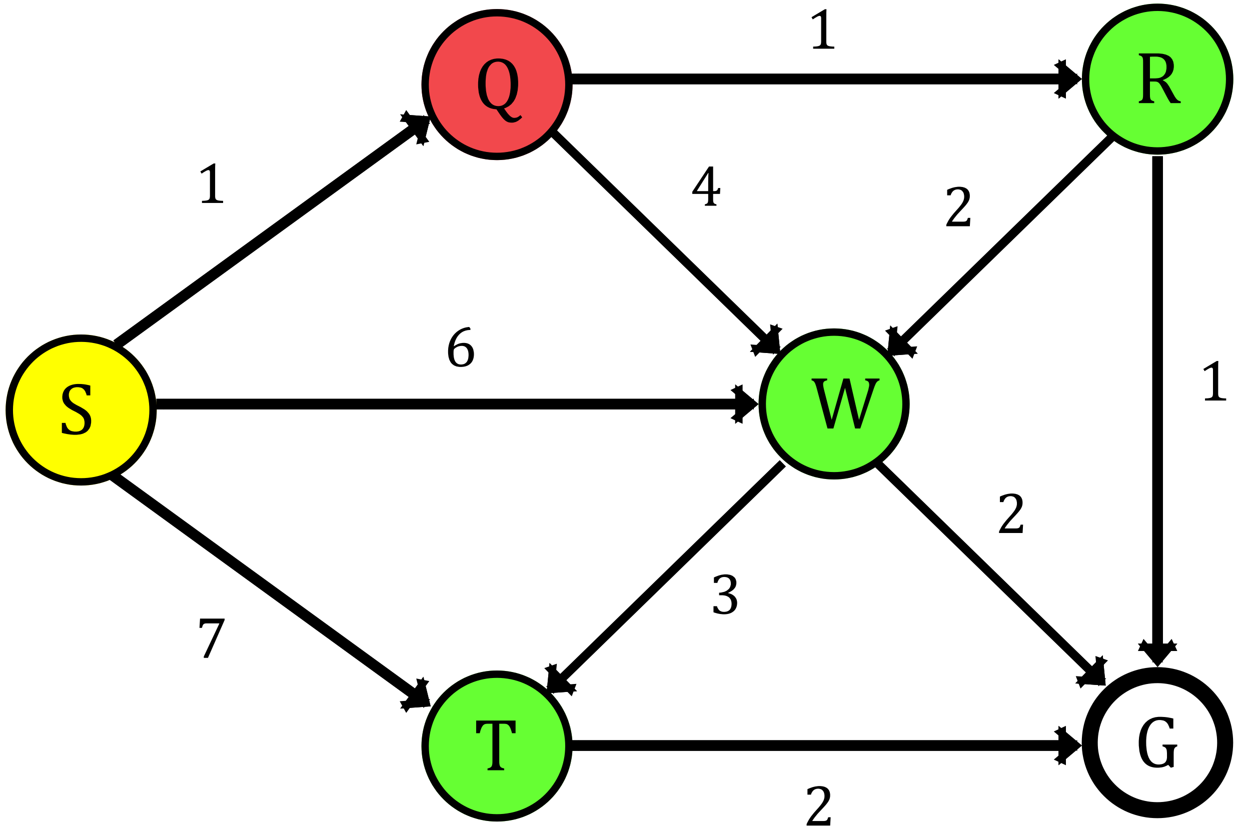
Current Node = Q

Queue:

R	T	W			

Visited:

S	Q	W	T	R	



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Breadth-First Search (Problem 02)

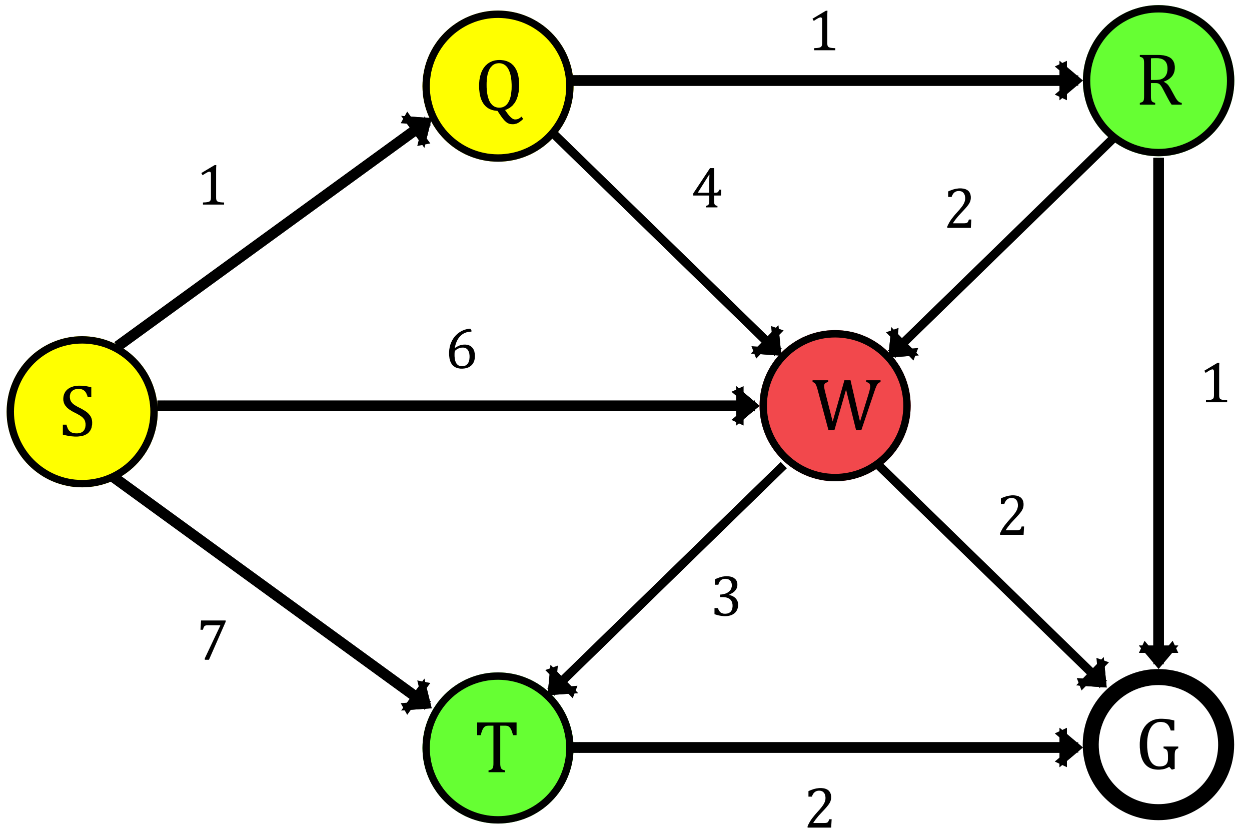
Current Node = W

Queue:

R	T				

Visited:

S	Q	W	T	R	



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

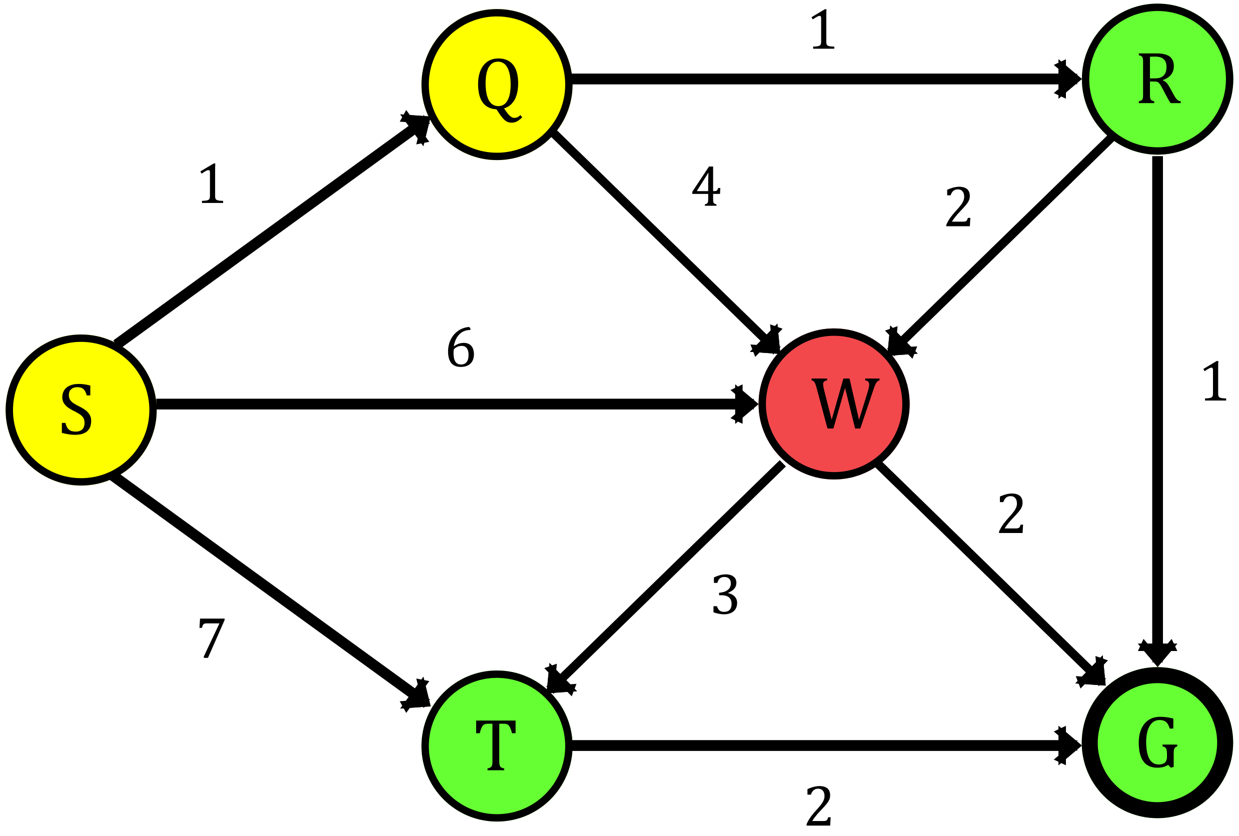
Current Node = W

Queue:

G	R	T			

Visited:

S	Q	W	T	R	G



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

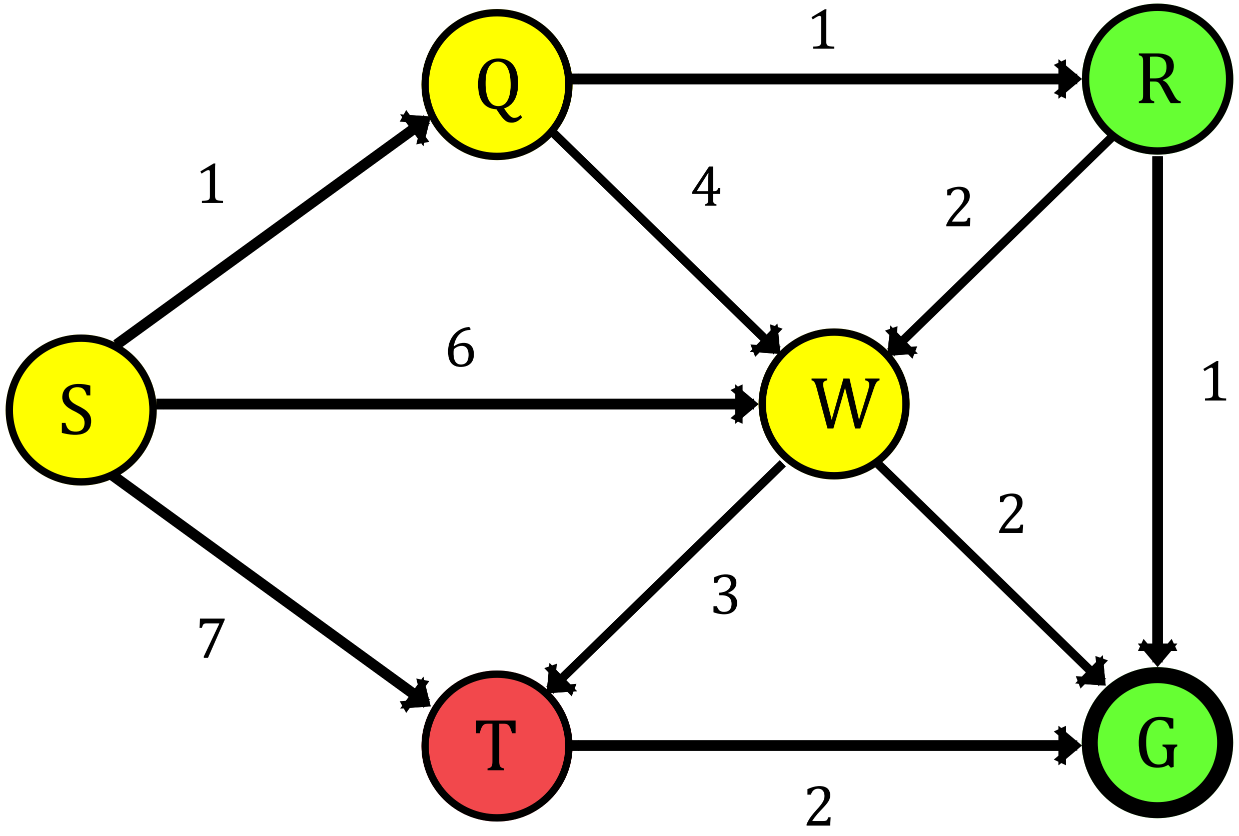
Current Node = T

Queue:

G	R				

Visited:

S	Q	W	T	R	G



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

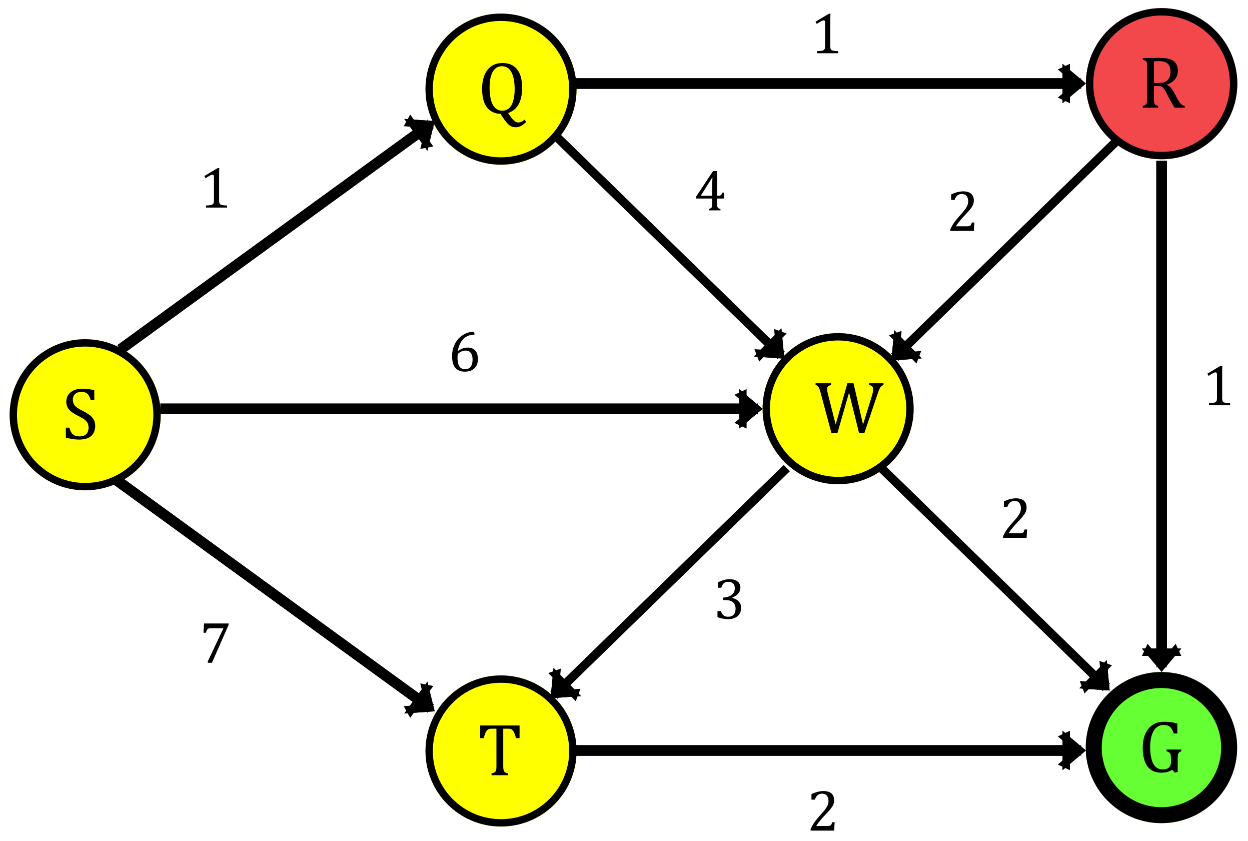
Current Node = R

Queue:

G					

Visited:

S	**Q**	**W**	**T**	**R**	**G**



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

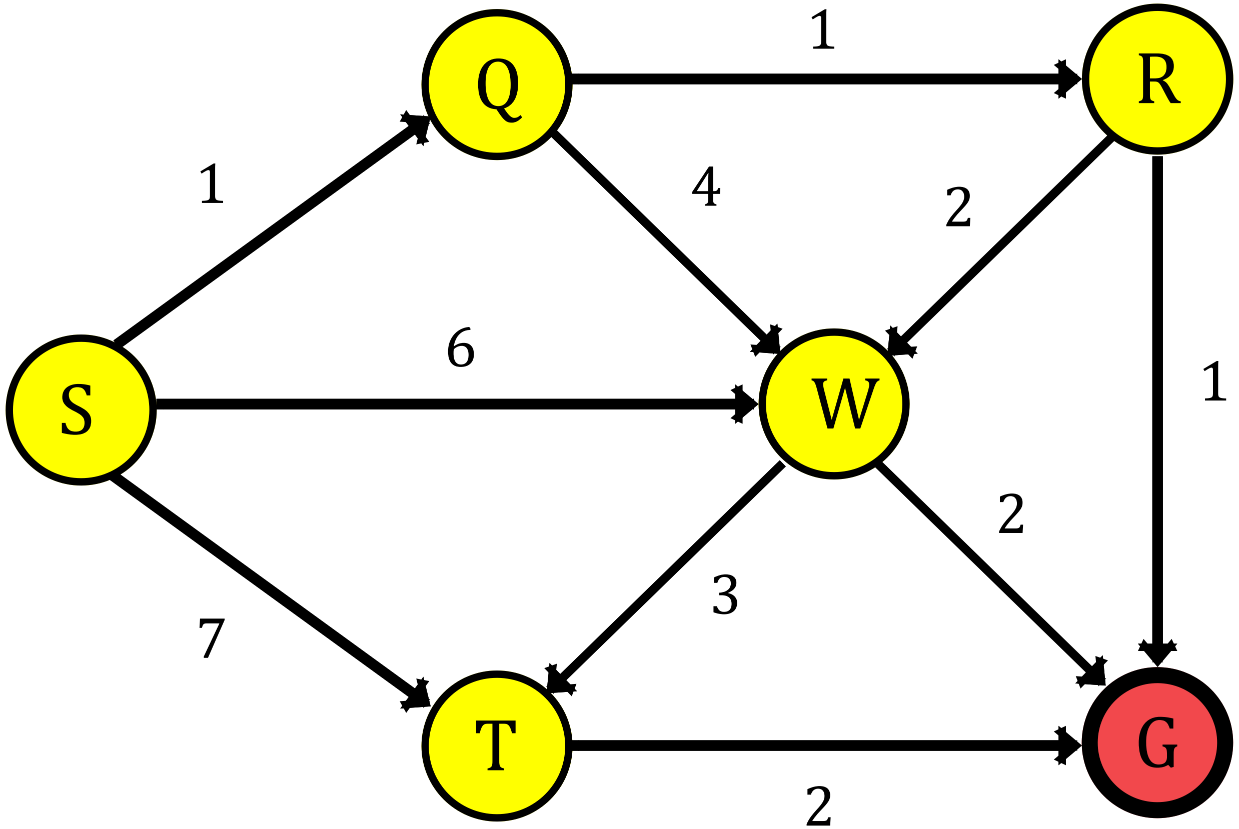
Breadth-First Search (Problem 02)

Current Node = G

Queue:

Visited:

S	**Q**	**W**	**T**	**R**	**G**



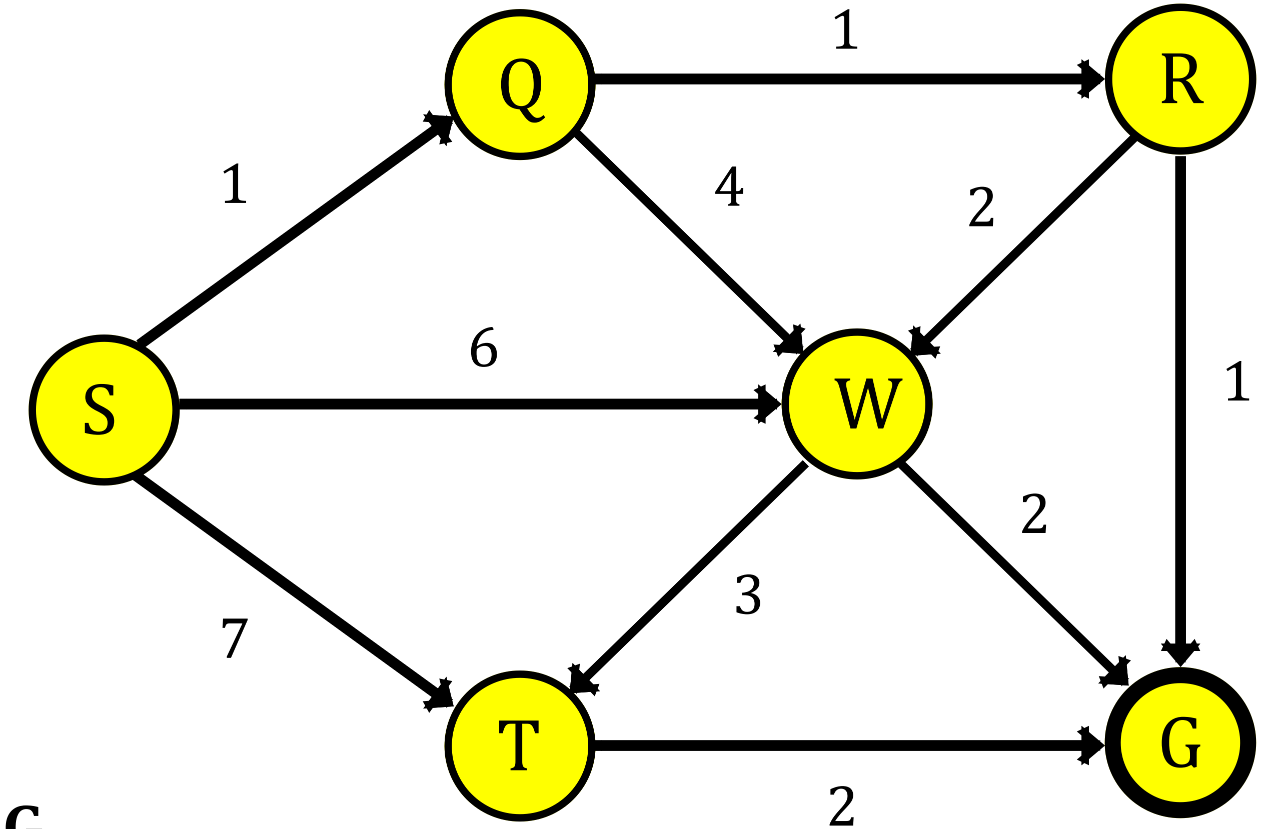
🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Breadth-First Search (Problem 02)

Queue:

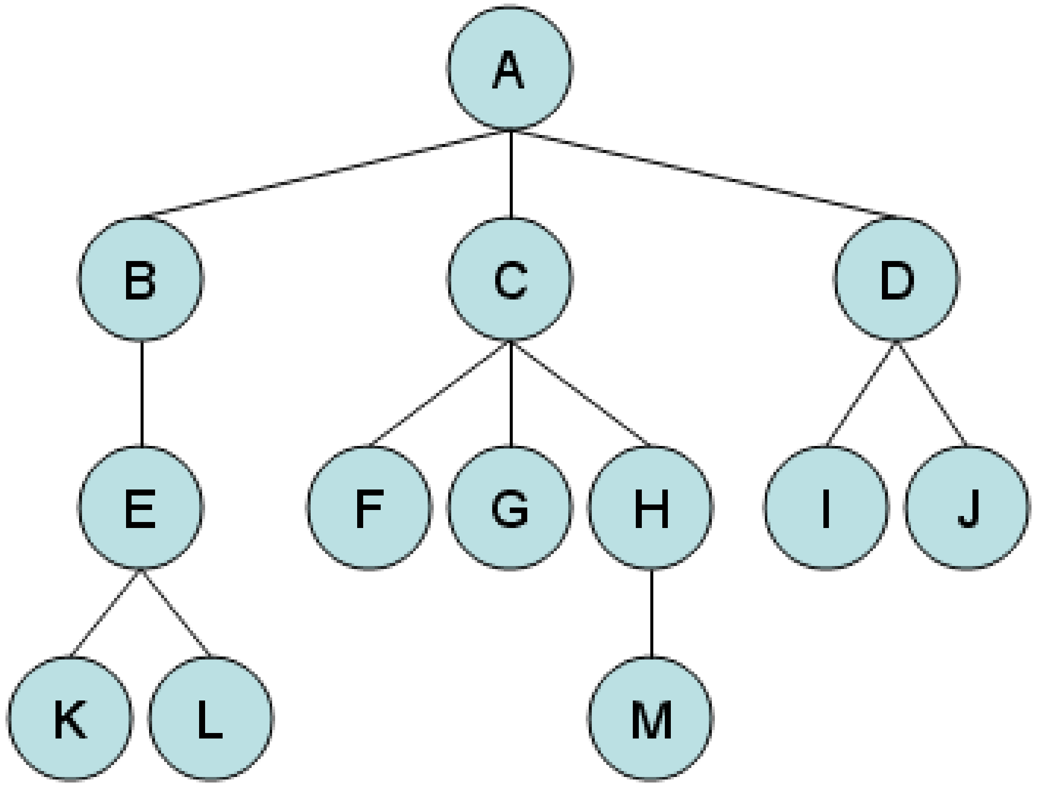
Visited:

S	**Q**	**W**	**T**	**R**	**G**



BFS Traversal: S Q W T R G

Breadth-First Search (Problem 03)



BFS Traversal: A B C D E F G H I J K L M

Breadth-First Search

Time Complexity and Space Complexity:

- If we look at how BFS expands from the root we see that it first expands on a set number of nodes, say b.
  - *On the second level it becomes b^2.*
  - *On the third level it becomes b^3.*
  - *And so on until it reaches b^d for some depth d.*
- $1 + b + b^2 + b^3 + \ldots + b^d$ which is $O(b^d)$
- Since all leaf nodes need to be stored in memory, space complexity is the same as time complexity.

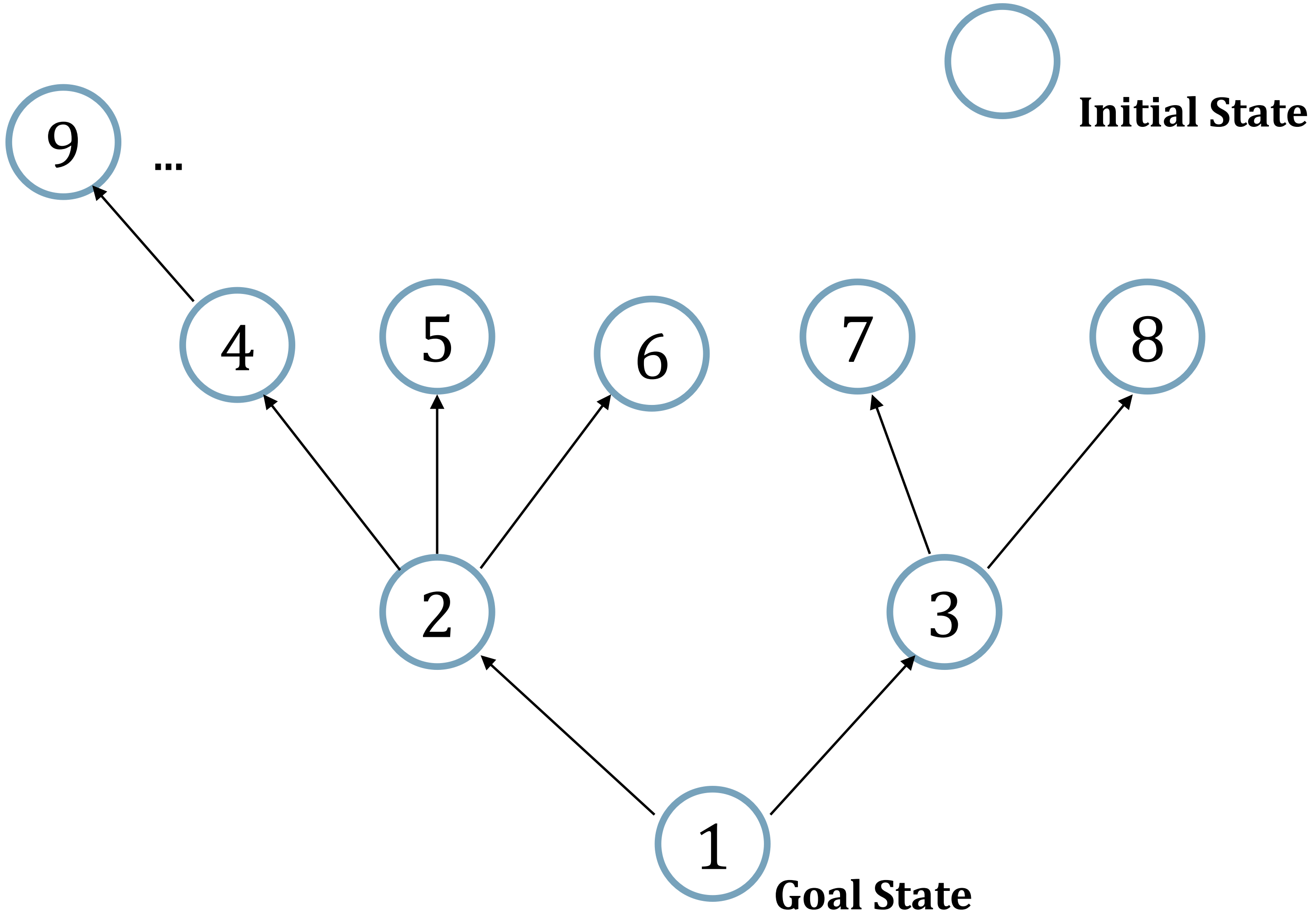
Breadth-First Search

- As you can see BFS is:
  - *Very systematic*
  - *Guaranteed to find a solution*
- What does this mean? From the four criteria, it means
  - *BFS is complete. If there exists an answer, it will be found (b should be finite).*
  - *BFS is optimal (if cost = 1 per step). The path from initial state to goal state is shallow.*

Backward Breadth-First Search

- Searches backwards from a goal state to a start state.
- Obviously this works best when the actions are reversible, and the set of goal states is small.

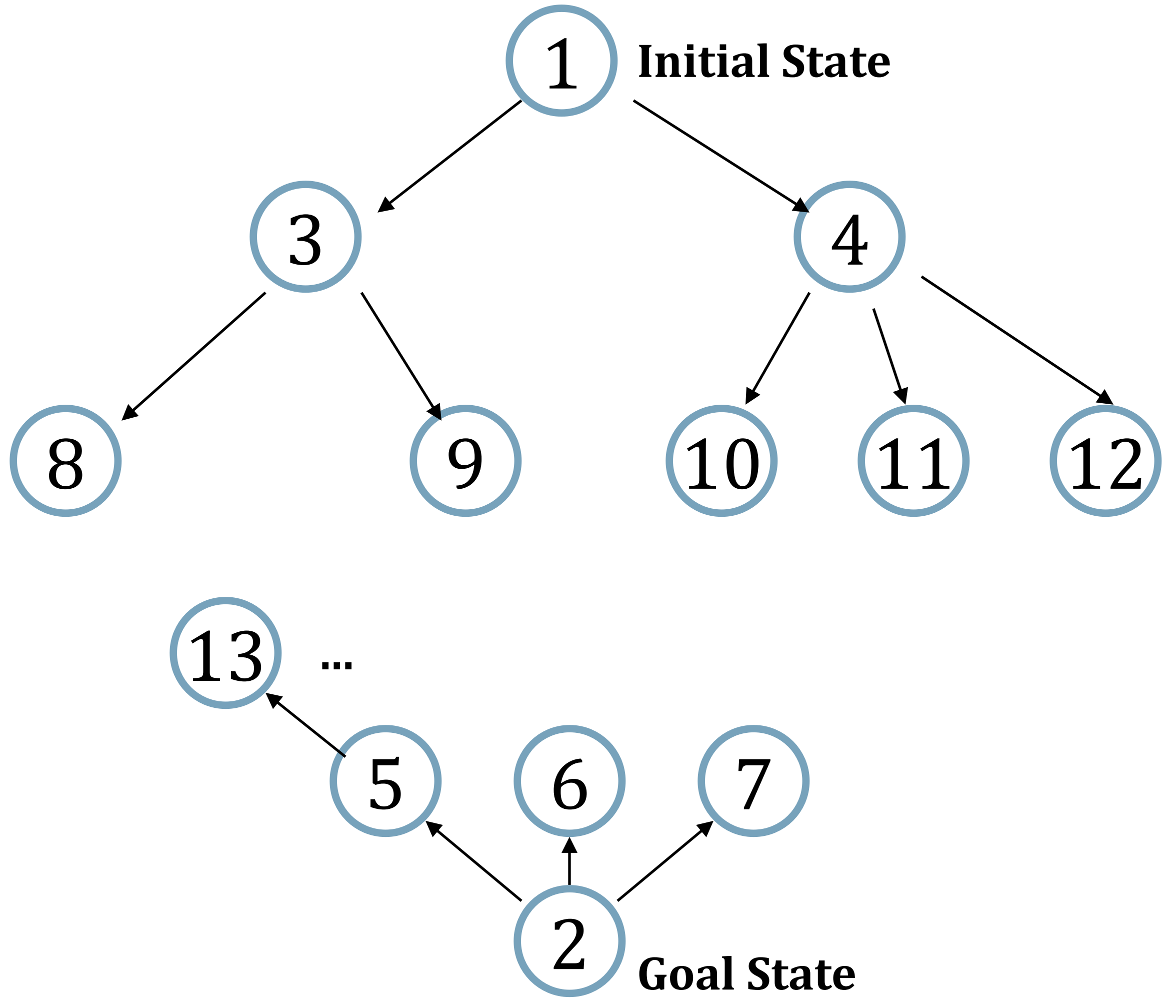
Backward Breadth-First Search



Bi-Directional Breadth-First Search

- The idea behind bidirectional search is to simultaneously search both forward from the initial state j and backward from the goal, and stop when the two searches meet in the middle.

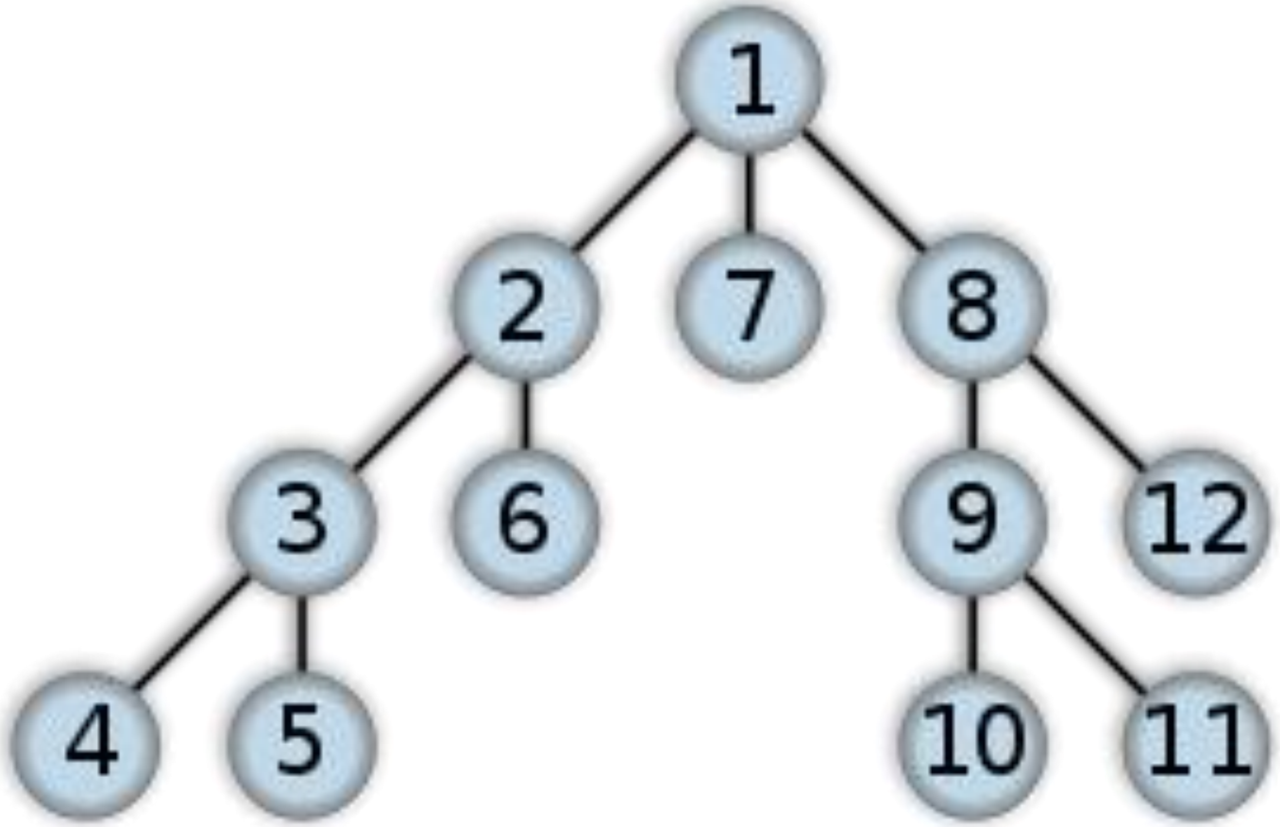
Bi-Directional Breadth-First Search



Depth-First Search

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch before backtracking.
- A depth first search begins at the root node (i.e. Initial node) and works downward to successively deeper levels.
- An operator is applied to the node to generate the next deeper node in sequence. This process continues until a solution is found or backtracking is forced by reaching a dead end.

Depth-First Search

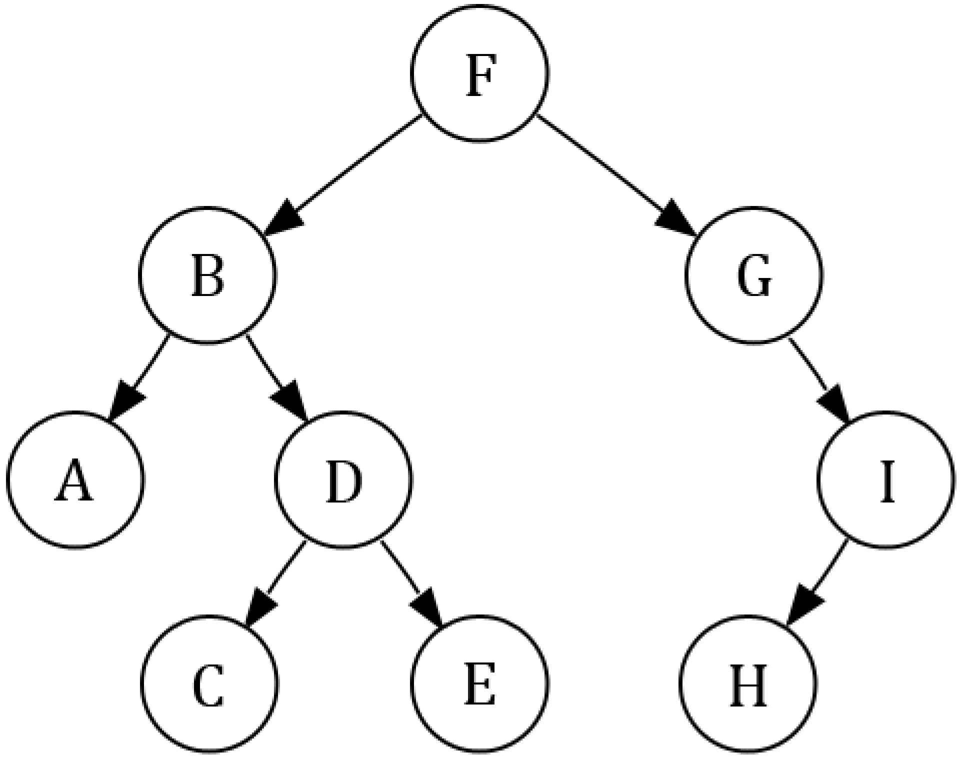


Depth-First Search

Algorithm:

- Step 01: Choose the starting node, mark it as visited, push it on Stack and make it current node.
- Step 02: Find adjacent unvisited node of current node, push it on Stack, mark it visited and make it current node.
- Step 03: If no unvisited adjacent node found, pop out node from Stack and make top of the stack as current node.
- Step 04: Repeat Step 02 to Step 03 until Stack is empty.

Depth-First Search (Problem 01)



DFS Traversal: F B A D C E G I H

Depth-First Search (Problem 01)

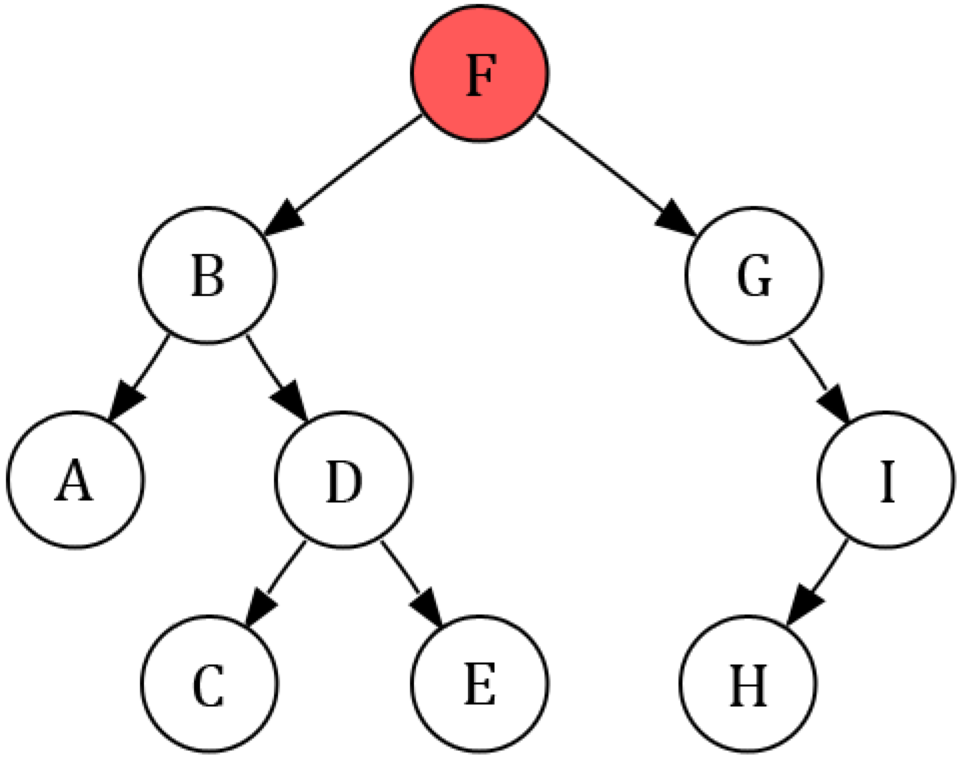
Current Node = F

Stack:

F									

Visited:

F									



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

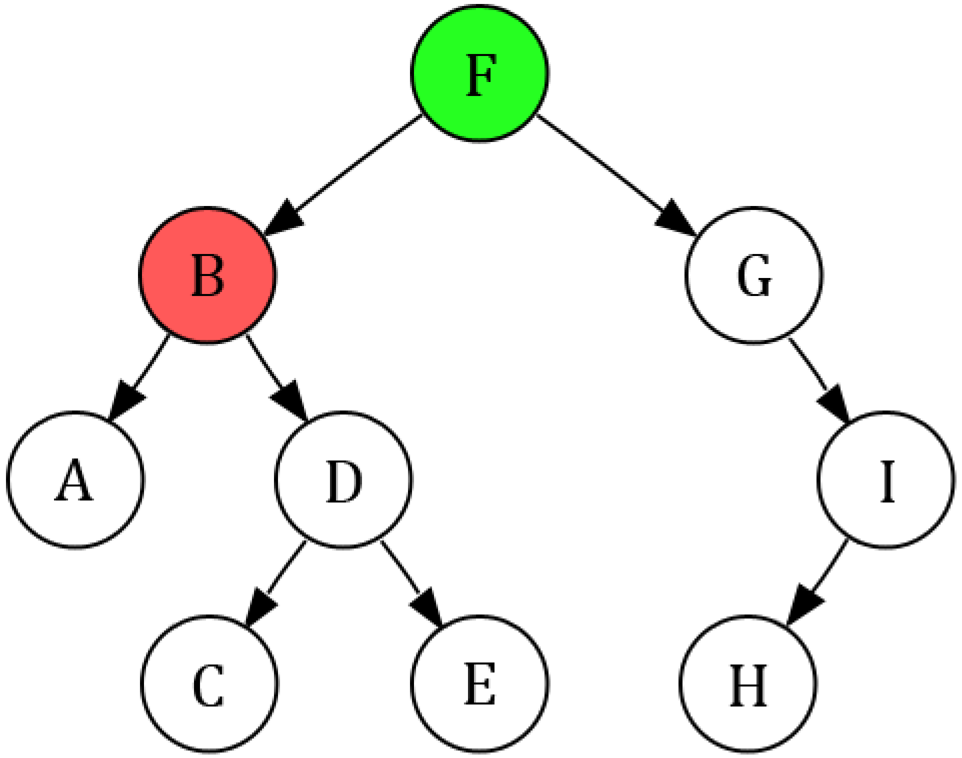
Current Node = B

Stack:

F	**B**								

Visited:

F	**B**								



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

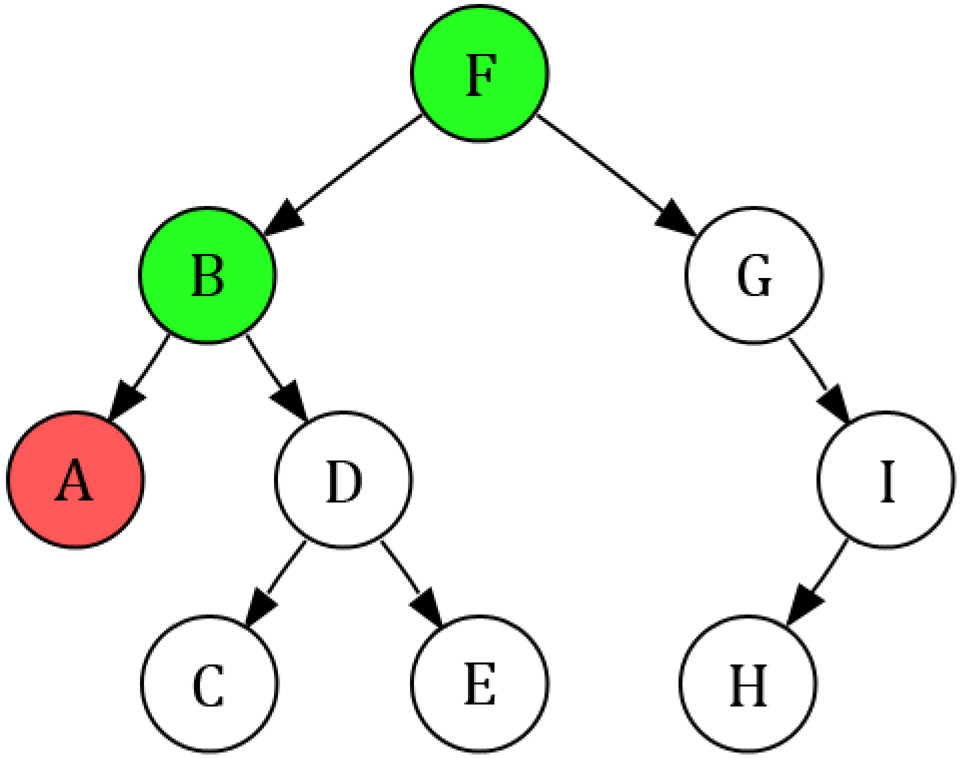
Current Node = A

Stack:

F	B	A							

Visited:

F	B	A							



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 01)

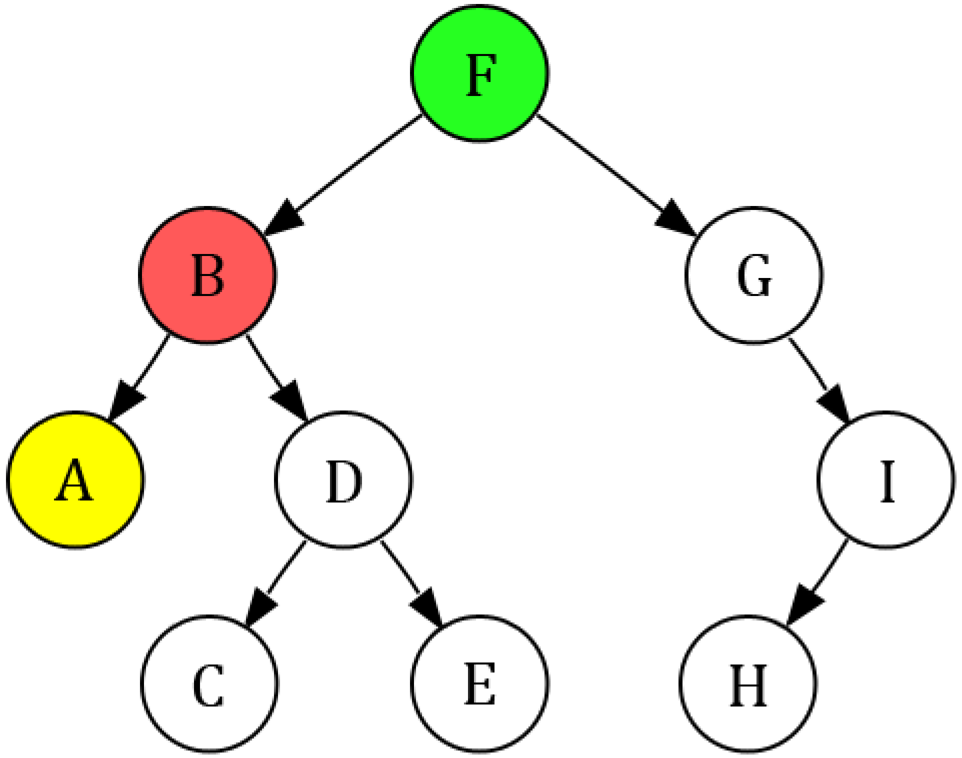
Current Node = B

Stack:

F	**B**								

Visited:

F	**B**	**A**							



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

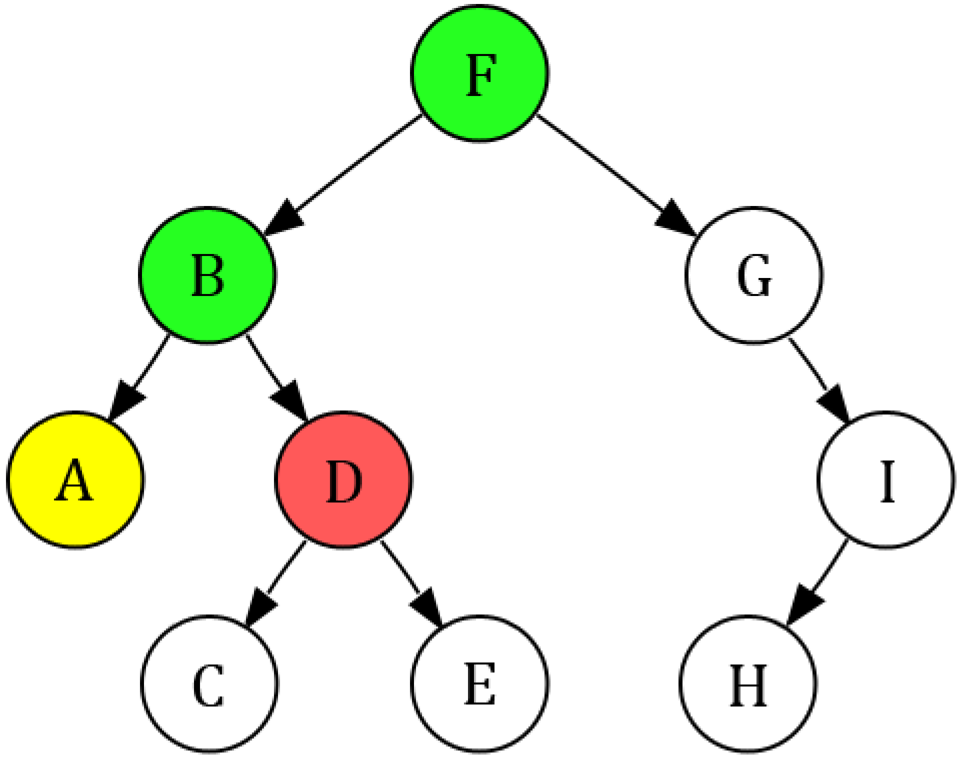
Current Node = D

Stack:

F	**B**	**D**							

Visited:

F	**B**	**A**	**D**						



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

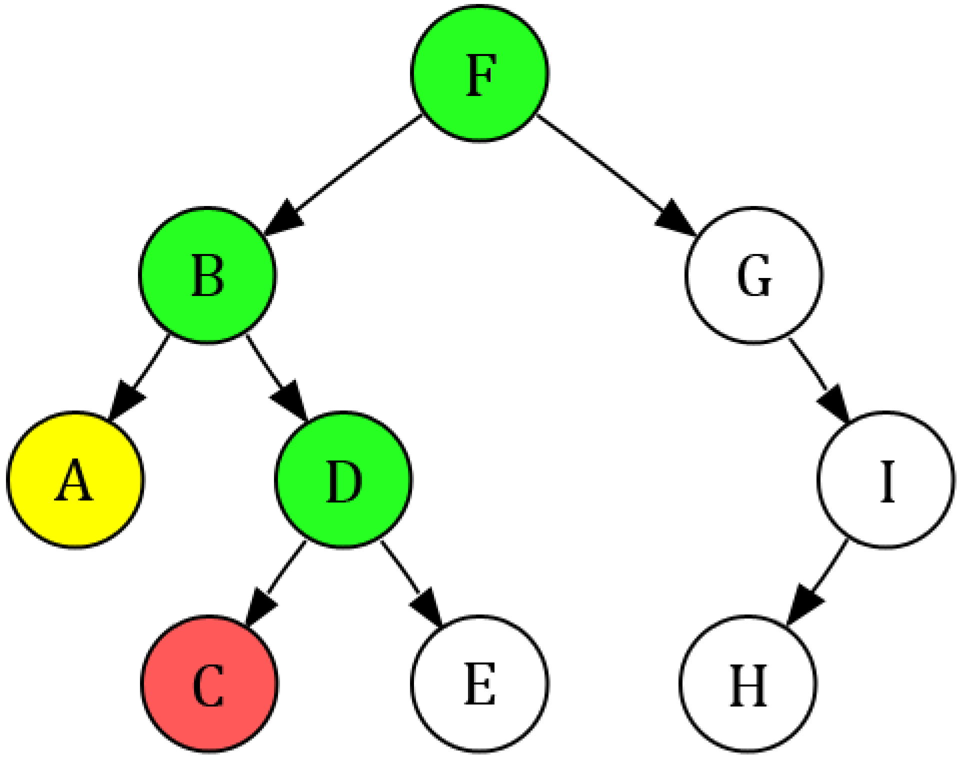
Current Node = C

Stack:

F	B	D	C						

Visited:

F	B	A	D	C					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

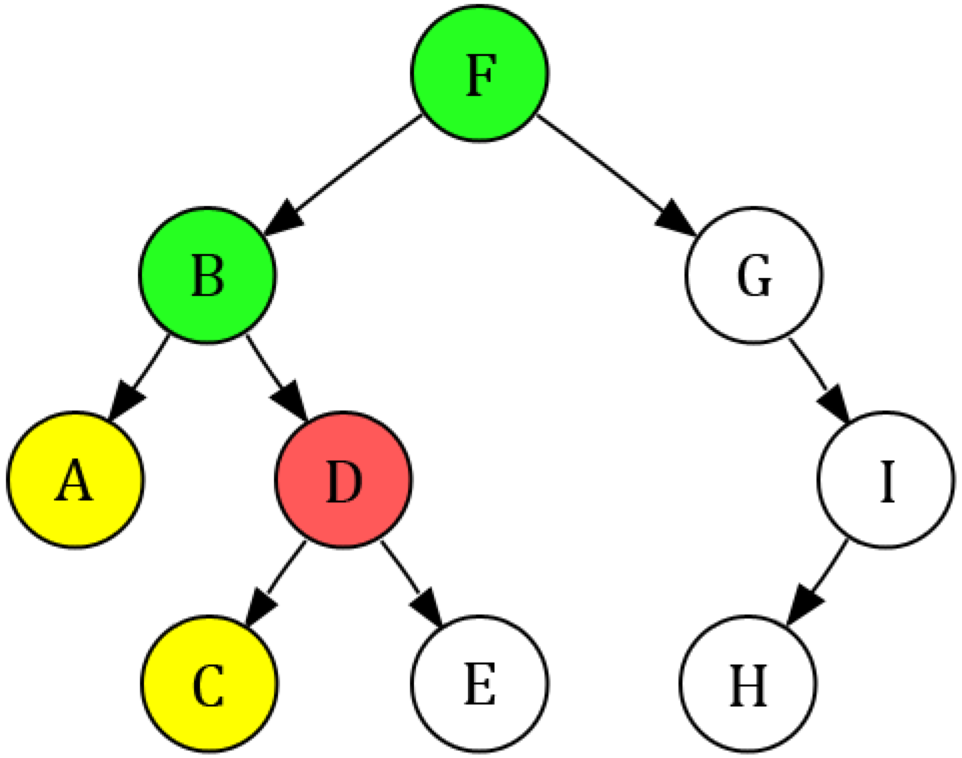
Current Node = D

Stack:

F	**B**	**D**							

Visited:

F	**B**	**A**	**D**	**C**					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

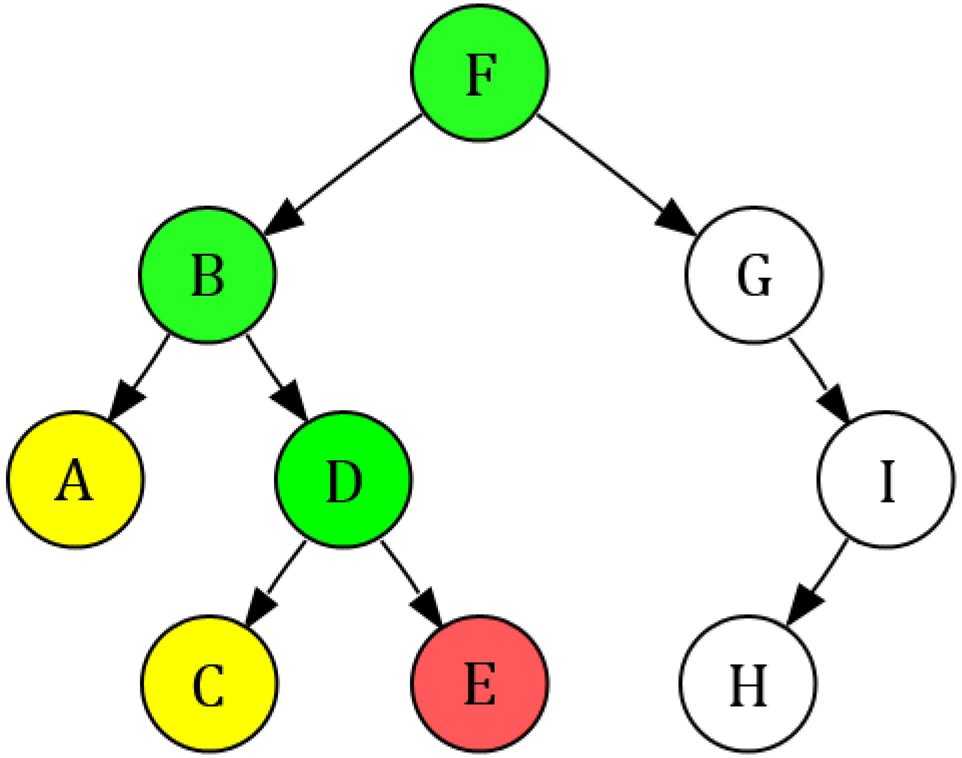
Current Node = E

Stack:

F	**B**	**D**	**E**						

Visited:

F	**B**	**A**	**D**	**C**	**E**				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

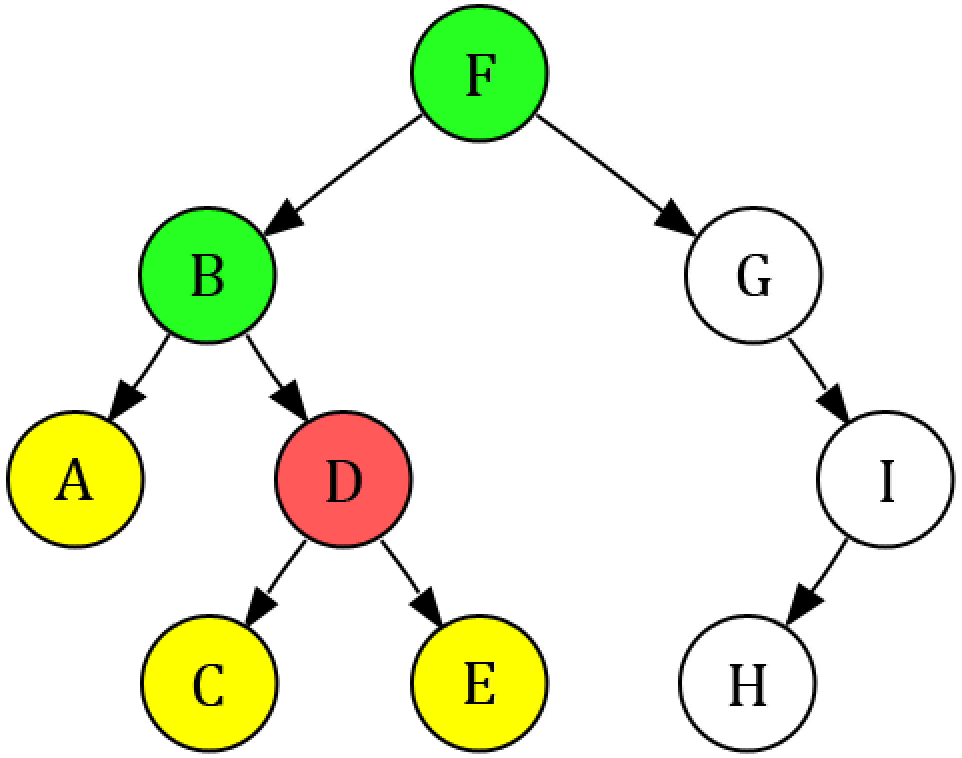
Current Node = D

Stack:

F	B	D							

Visited:

F	B	A	D	C	E				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

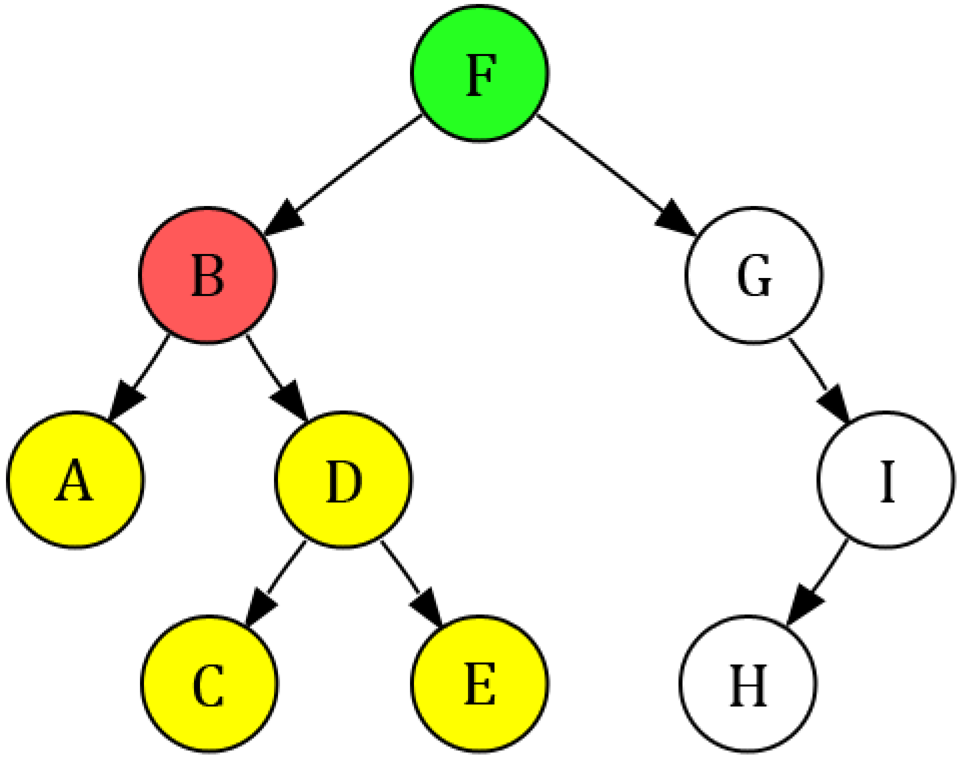
Current Node = B

Stack:

F	B								

Visited:

F	B	A	D	C	E				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

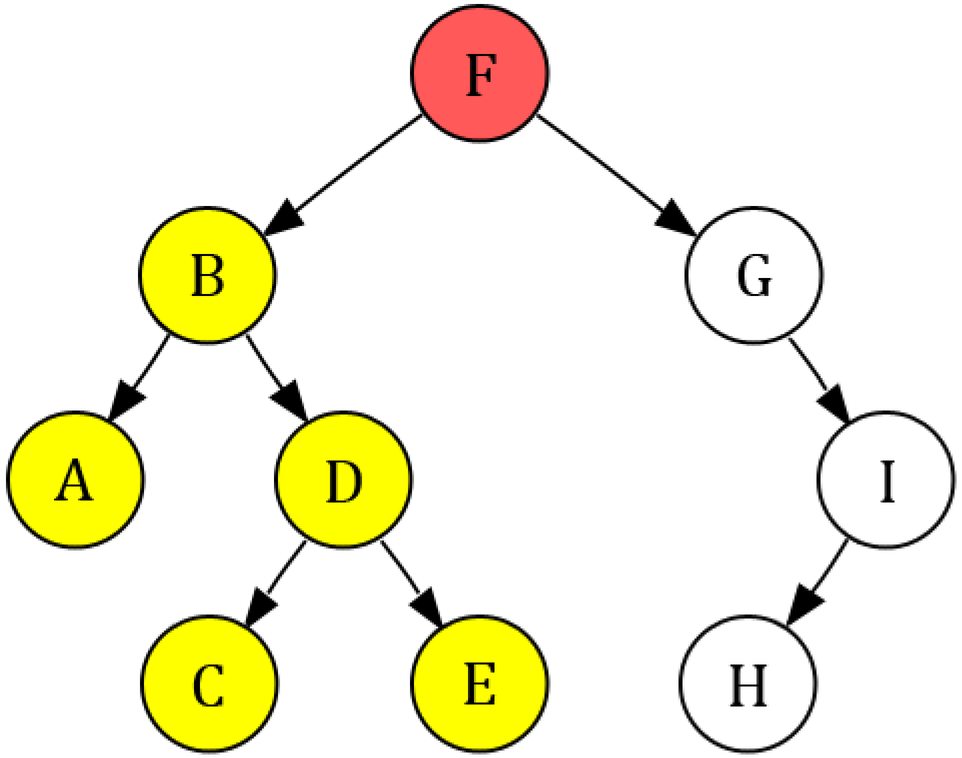
Current Node = F

Stack:

F									

Visited:

F	B	A	D	C	E				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

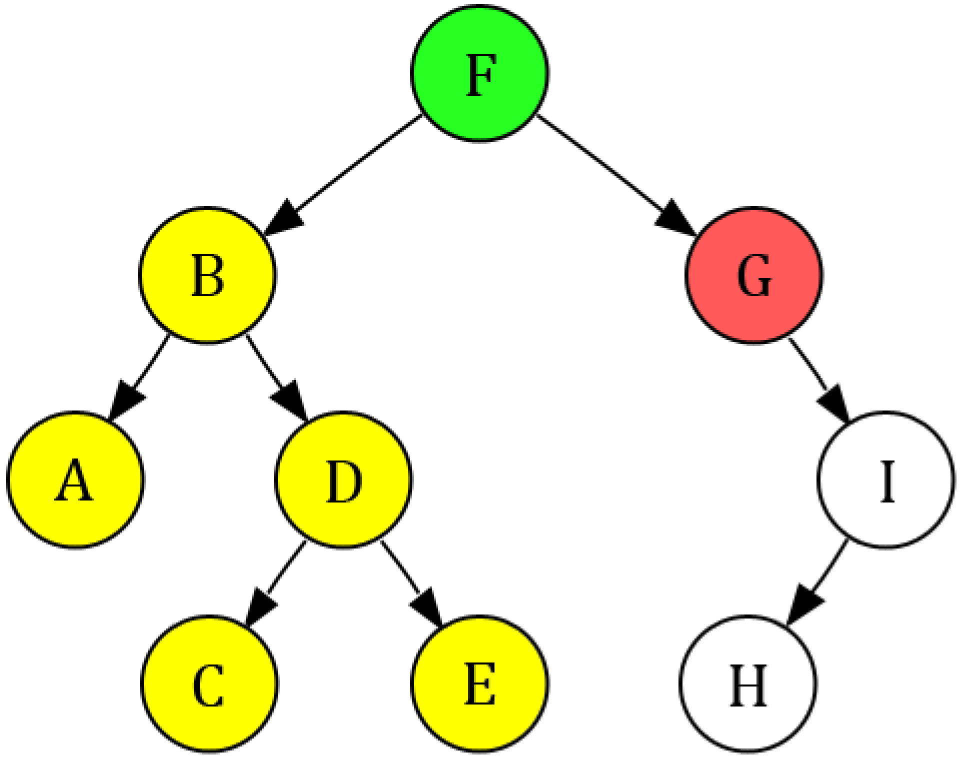
Current Node = G

Stack:

F	**G**								

Visited:

F	**B**	**A**	**D**	**C**	**E**	**G**			



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

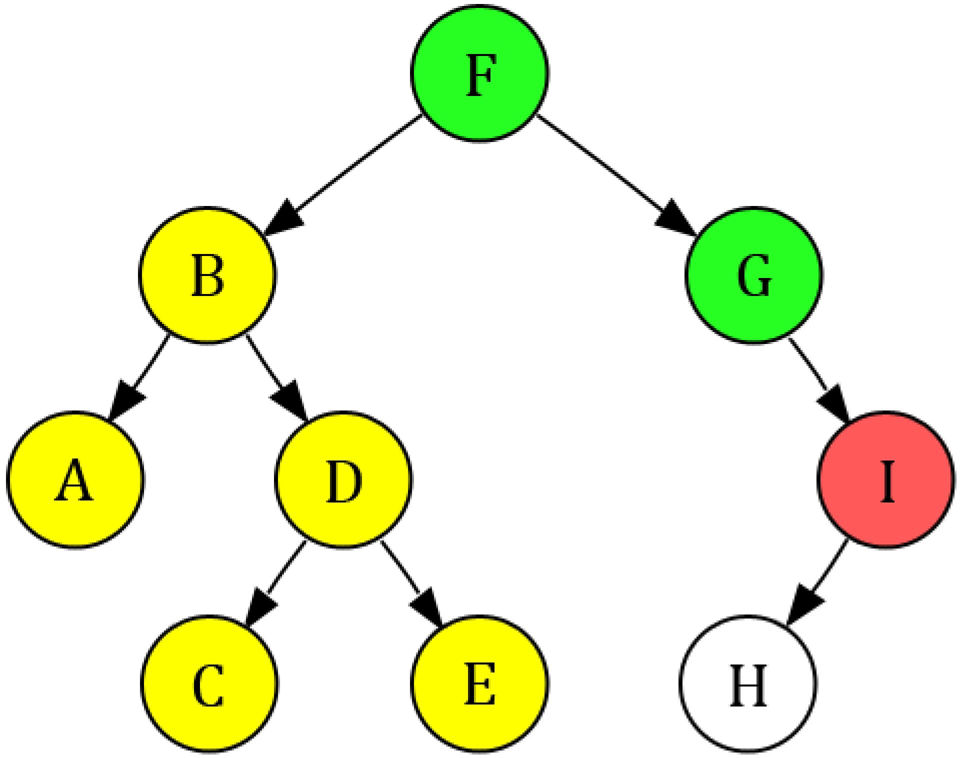
Current Node = I

Stack:

F	G	I							

Visited:

F	B	A	D	C	E	G	I		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

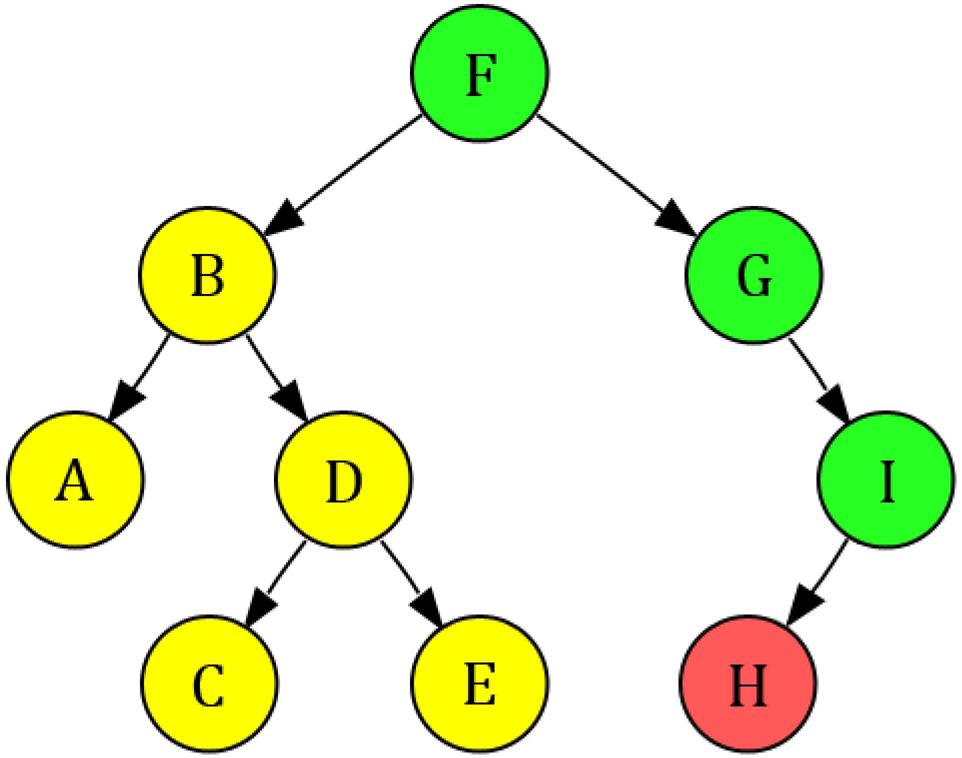
Current Node = H

Stack:

F	G	I	H						

Visited:

F	B	A	D	C	E	G	I	H	



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 01)

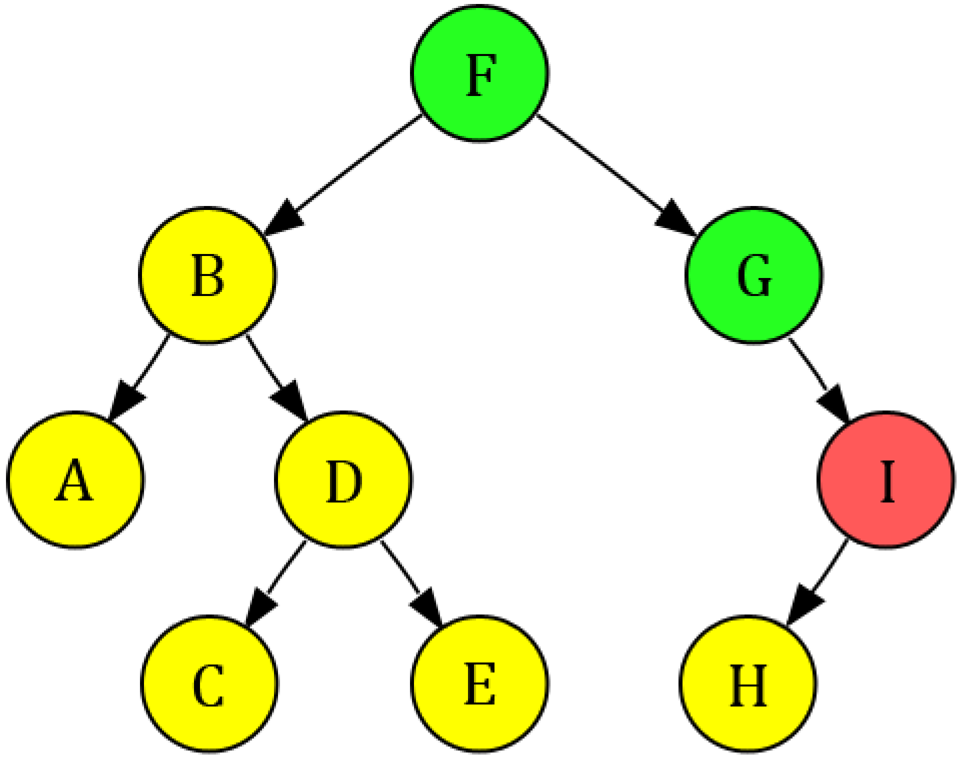
Current Node = I

Stack:

F	G	I							

Visited:

F	B	A	D	C	E	G	I	H	



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

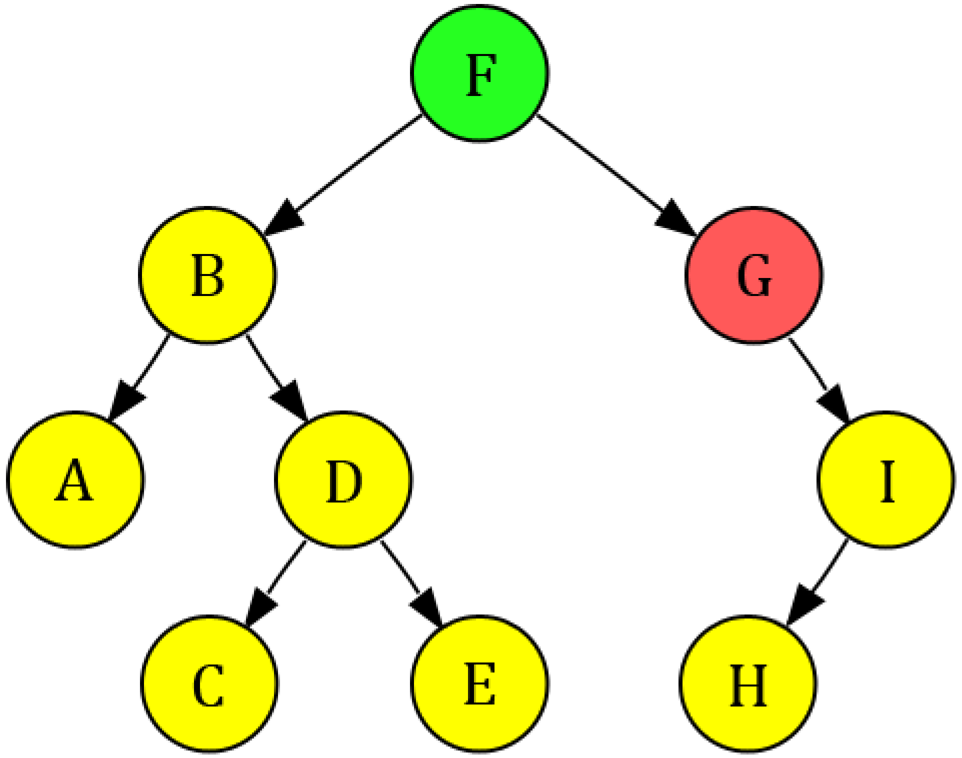
Current Node = G

Stack:

F	G								

Visited:

F	B	A	D	C	E	G	I	H	



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 01)

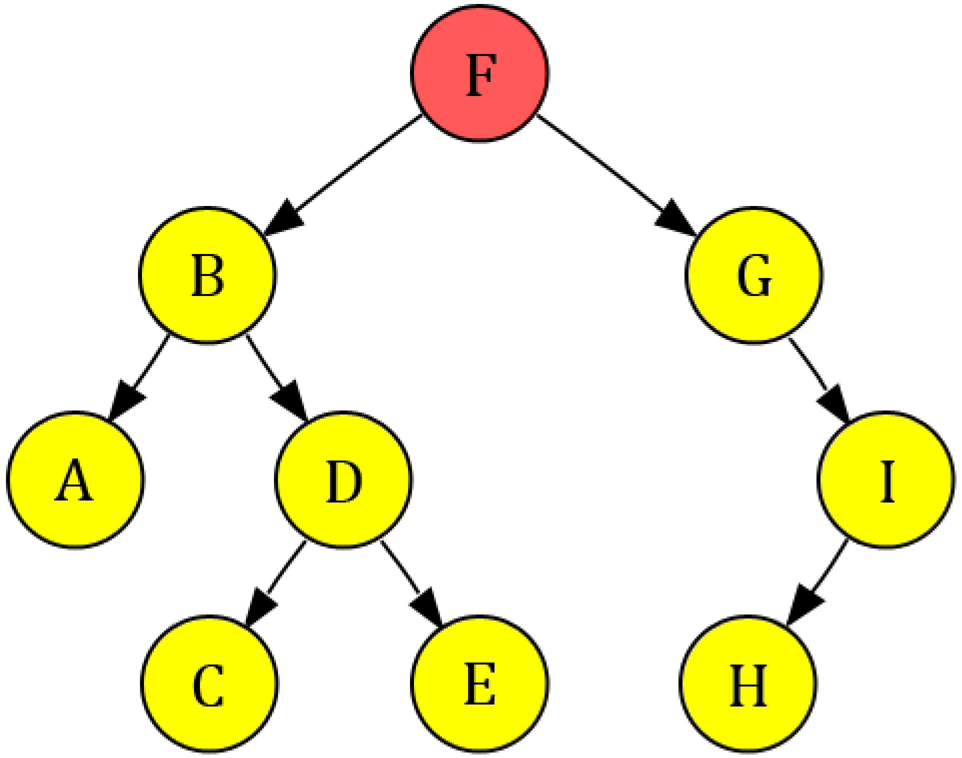
Current Node = F

Stack:

F									

Visited:

F	B	A	D	C	E	G	I	H	



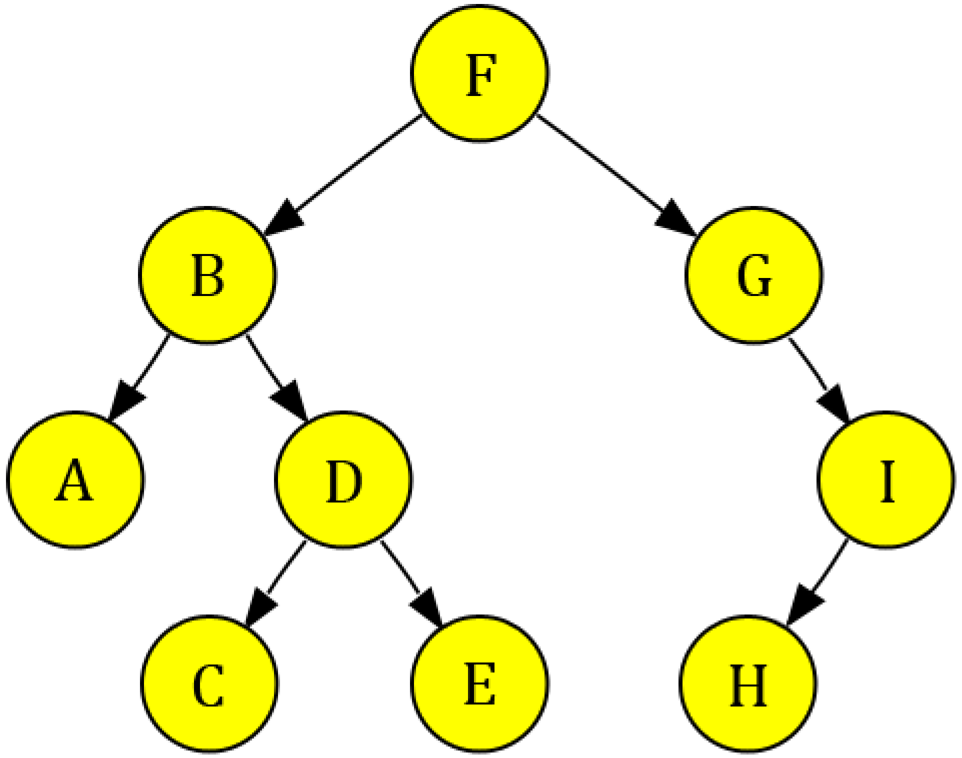
🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 01)

Stack:

Visited:

F	**B**	**A**	**D**	**C**	**E**	**G**	**I**	**H**	



DFS Traversal: F B A D C E G I H

Depth-First Search (Problem 02)

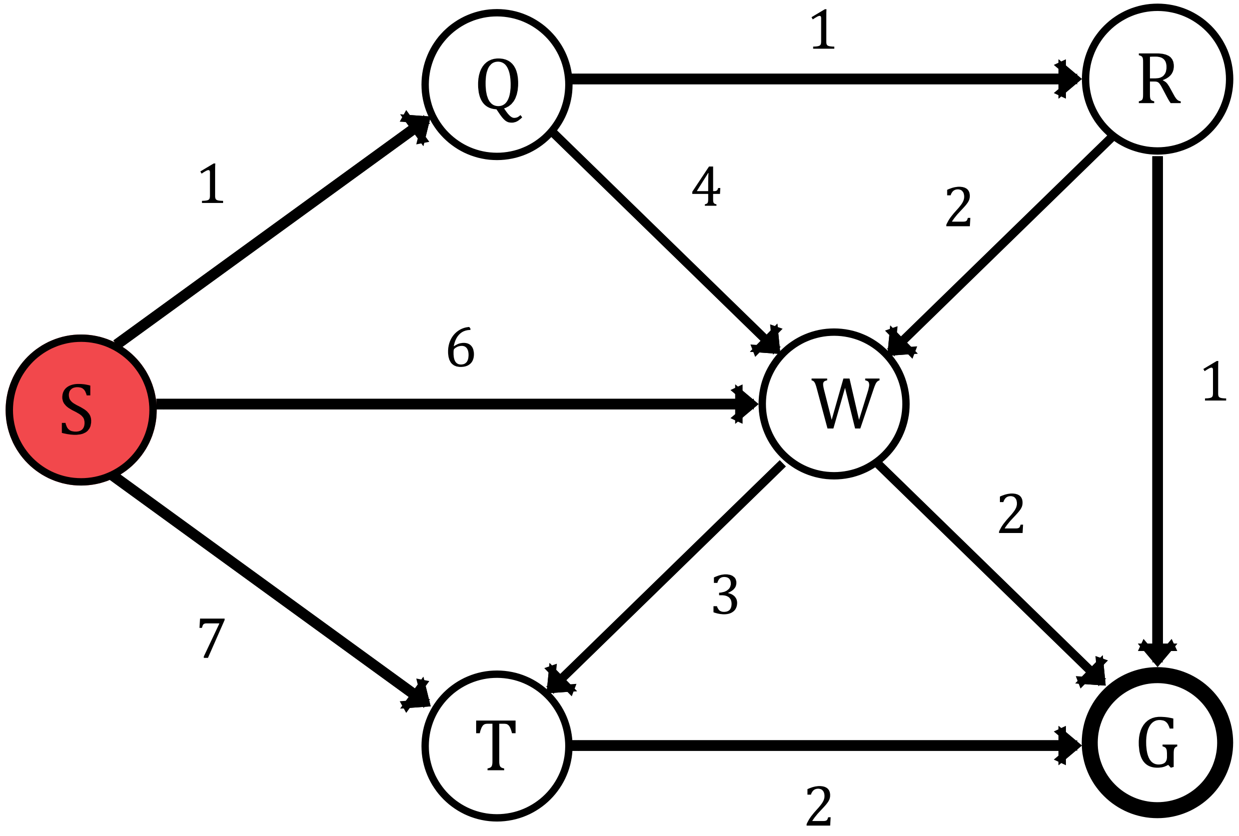
Current Node = S

Stack:

S					

Visited:

S					



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

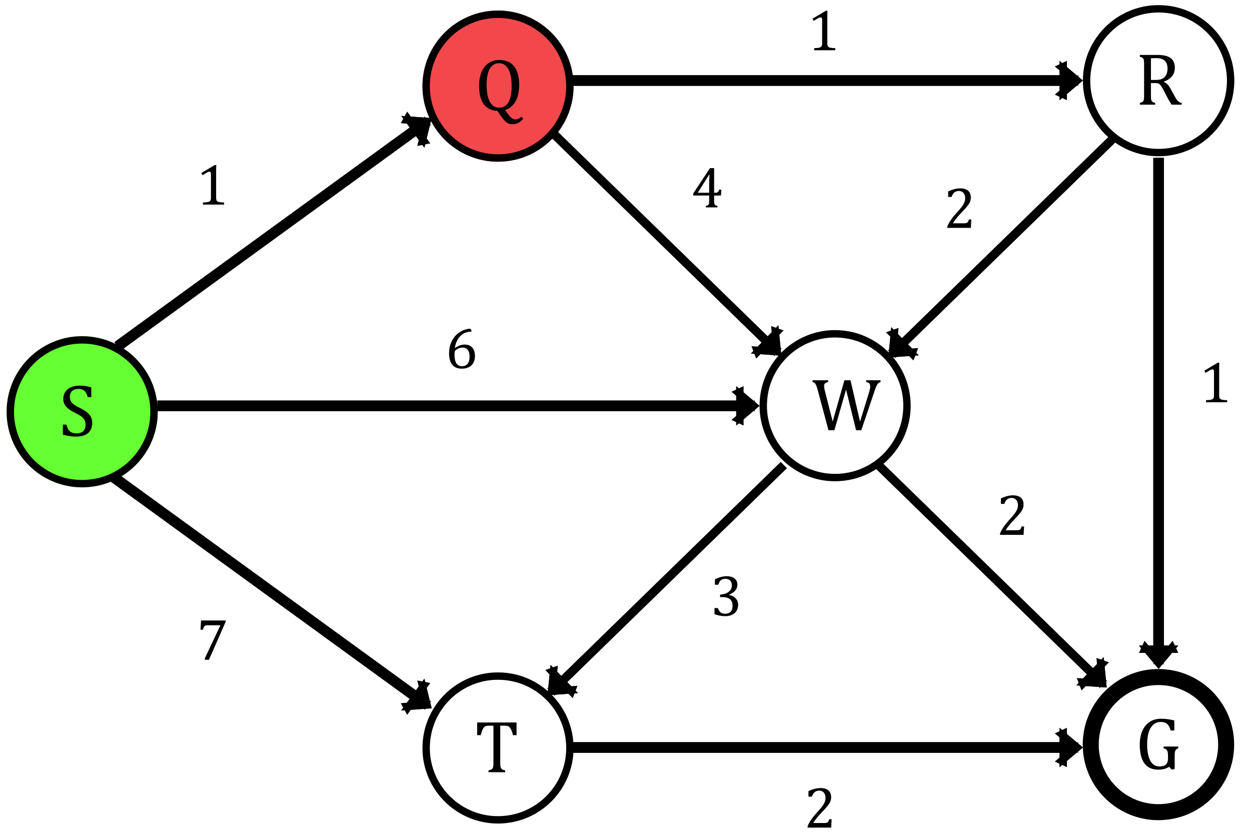
Current Node = Q

Stack:

S	**Q**				

Visited:

S	**Q**				



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

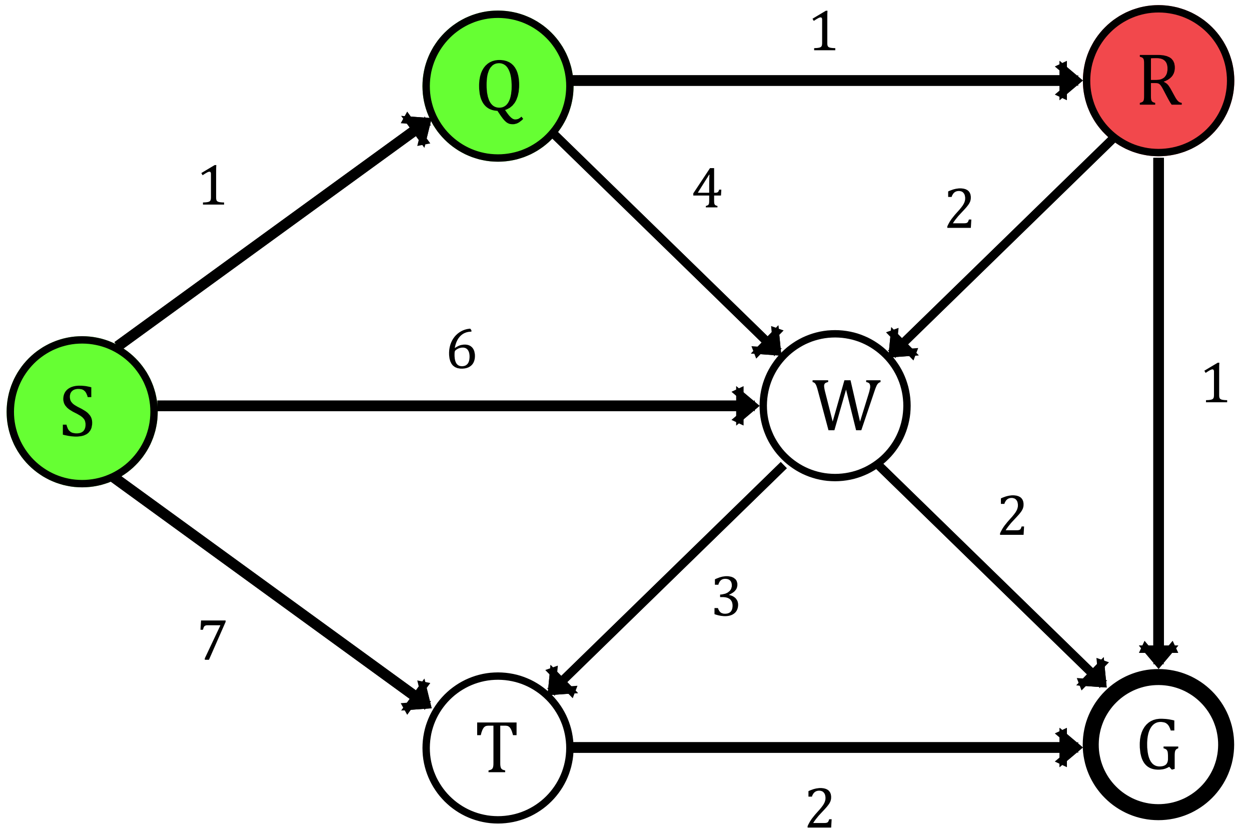
Current Node = R

Stack:

S	Q	R			

Visited:

S	Q	R			



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

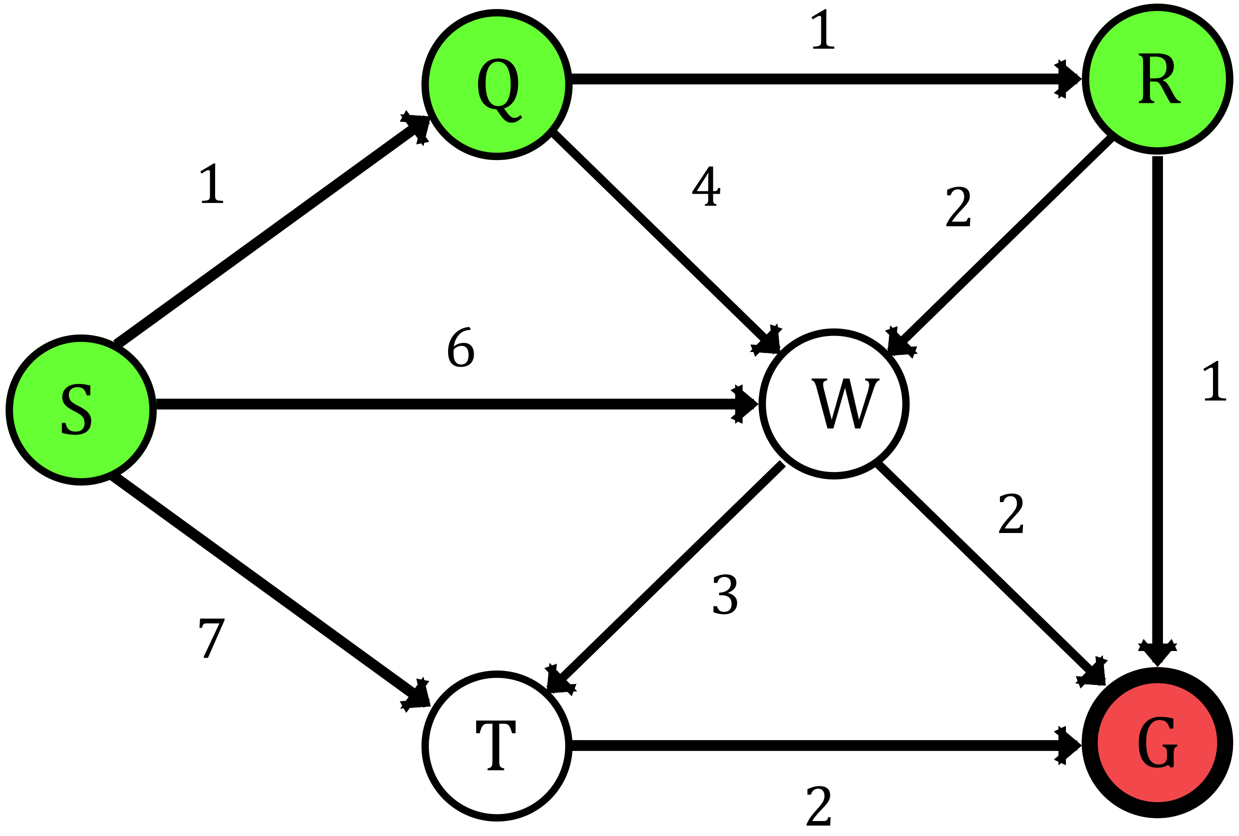
Current Node = G

Stack:

S	**Q**	**R**	**G**		

Visited:

S	**Q**	**R**	**G**		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

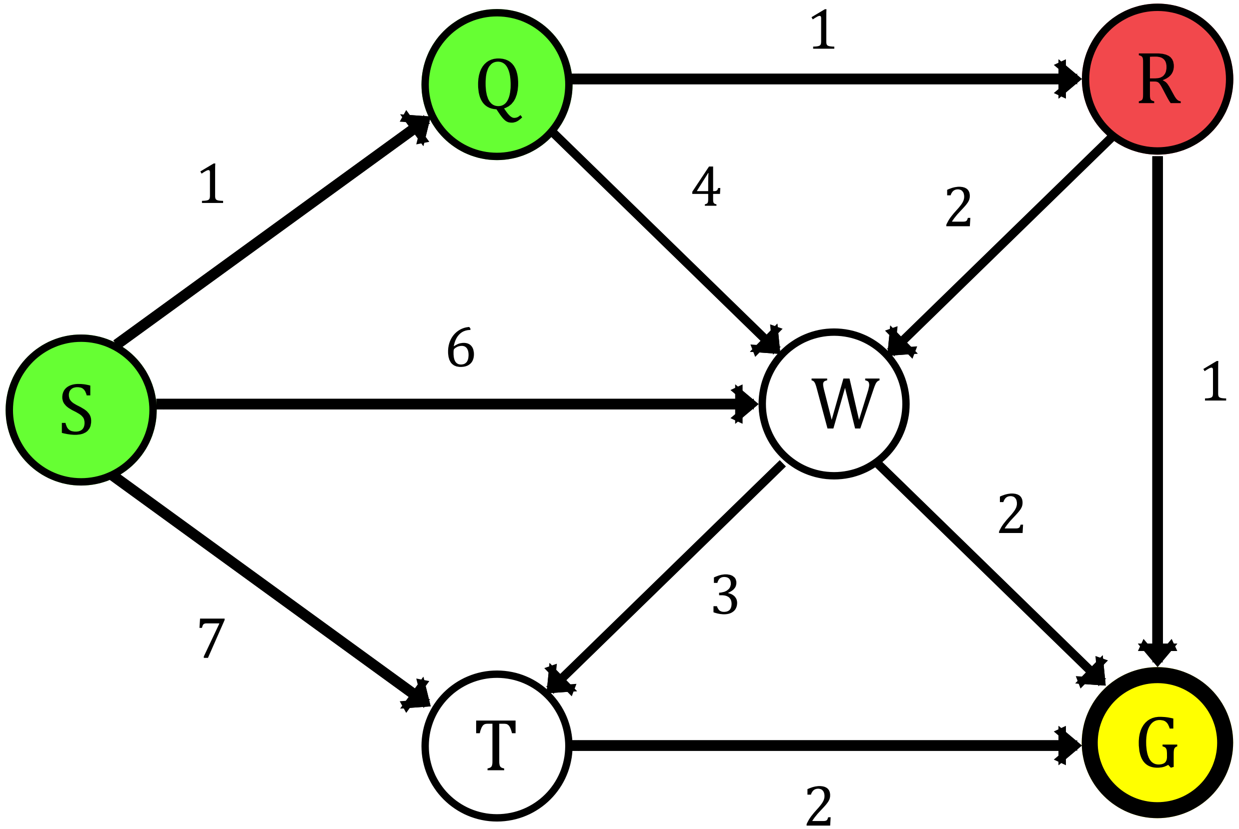
Current Node = R

Stack:

S	Q	R			

Visited:

S	Q	R	G		



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

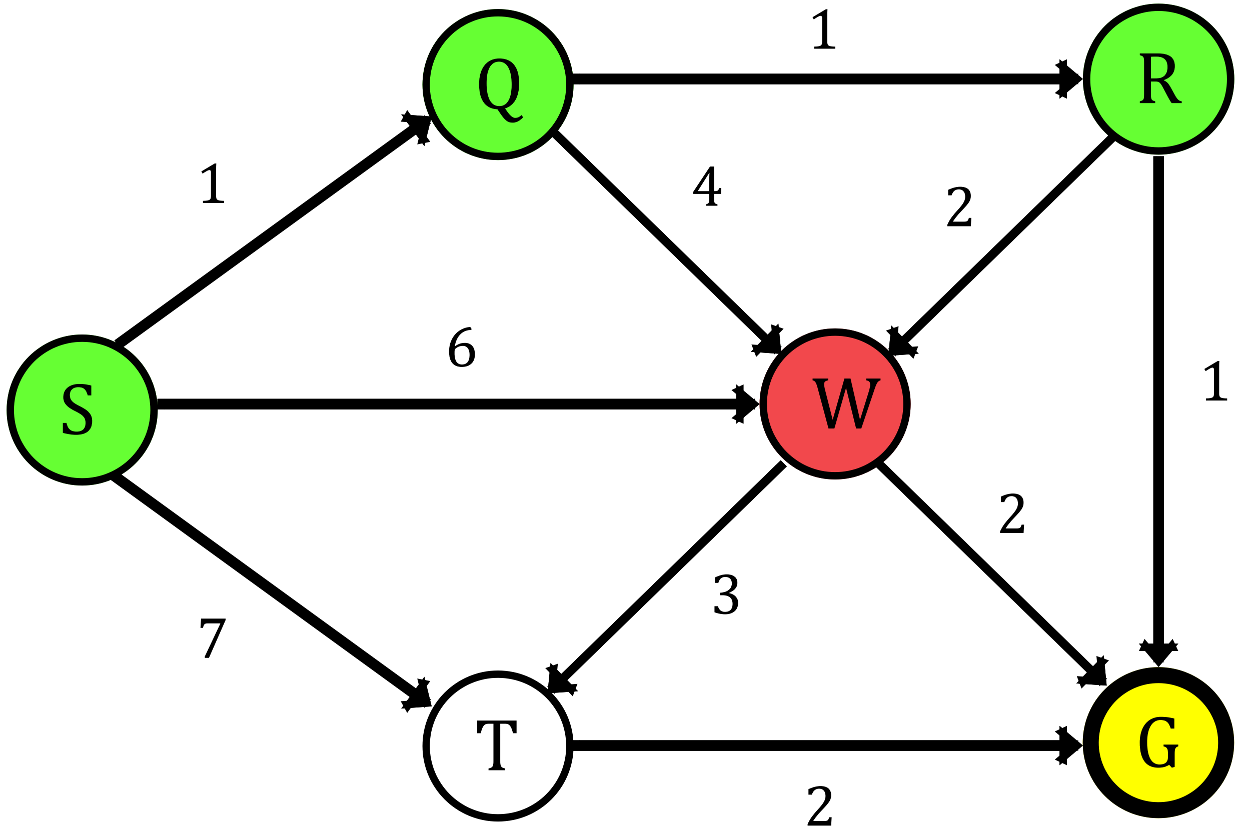
Current Node = W

Stack:

S	Q	R	W		

Visited:

S	Q	R	G	W	



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

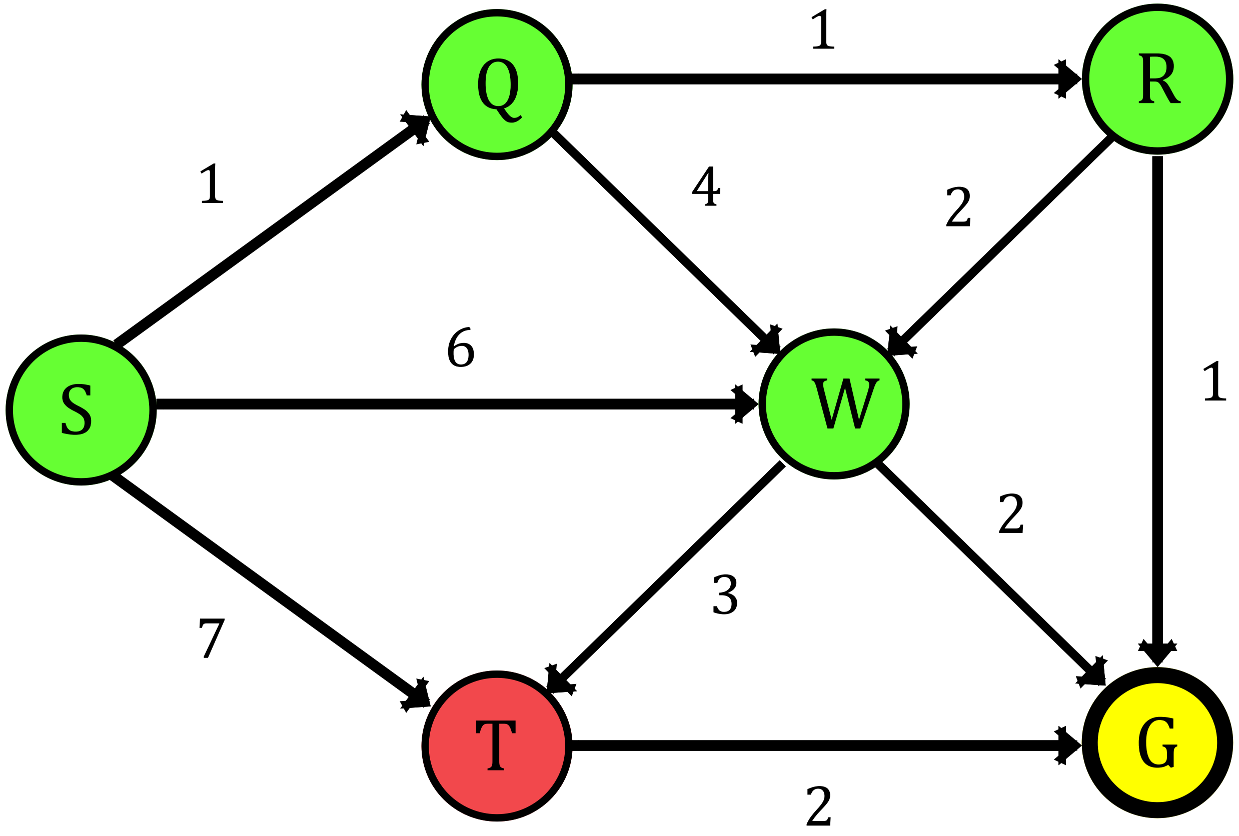
Current Node = T

Stack:

S	**Q**	**R**	**W**	**T**	

Visited:

S	**Q**	**R**	**G**	**W**	**T**



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

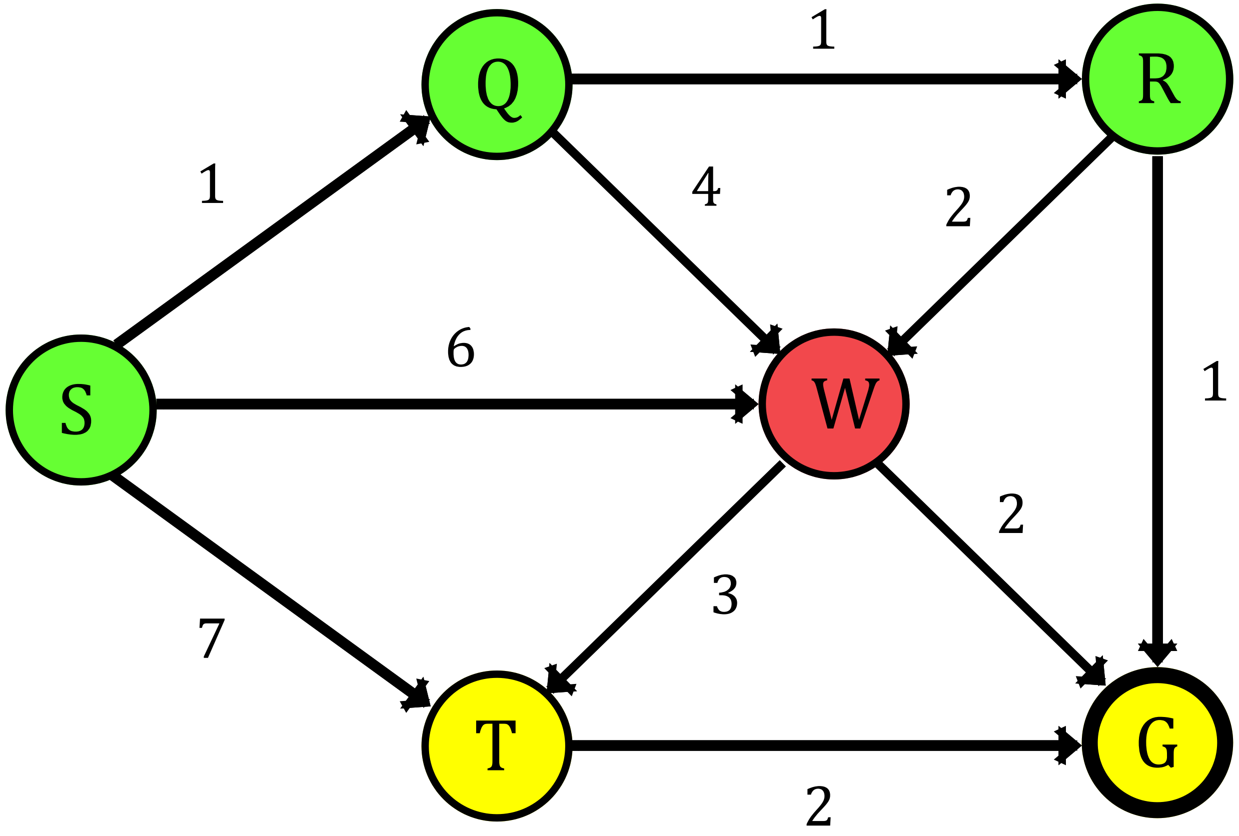
Current Node = W

Stack:

S	Q	R	W		

Visited:

S	Q	R	G	W	T



🟥 **Current Node** 🟩 **Opened Node** 🟨 **Closed Node**

Depth-First Search (Problem 02)

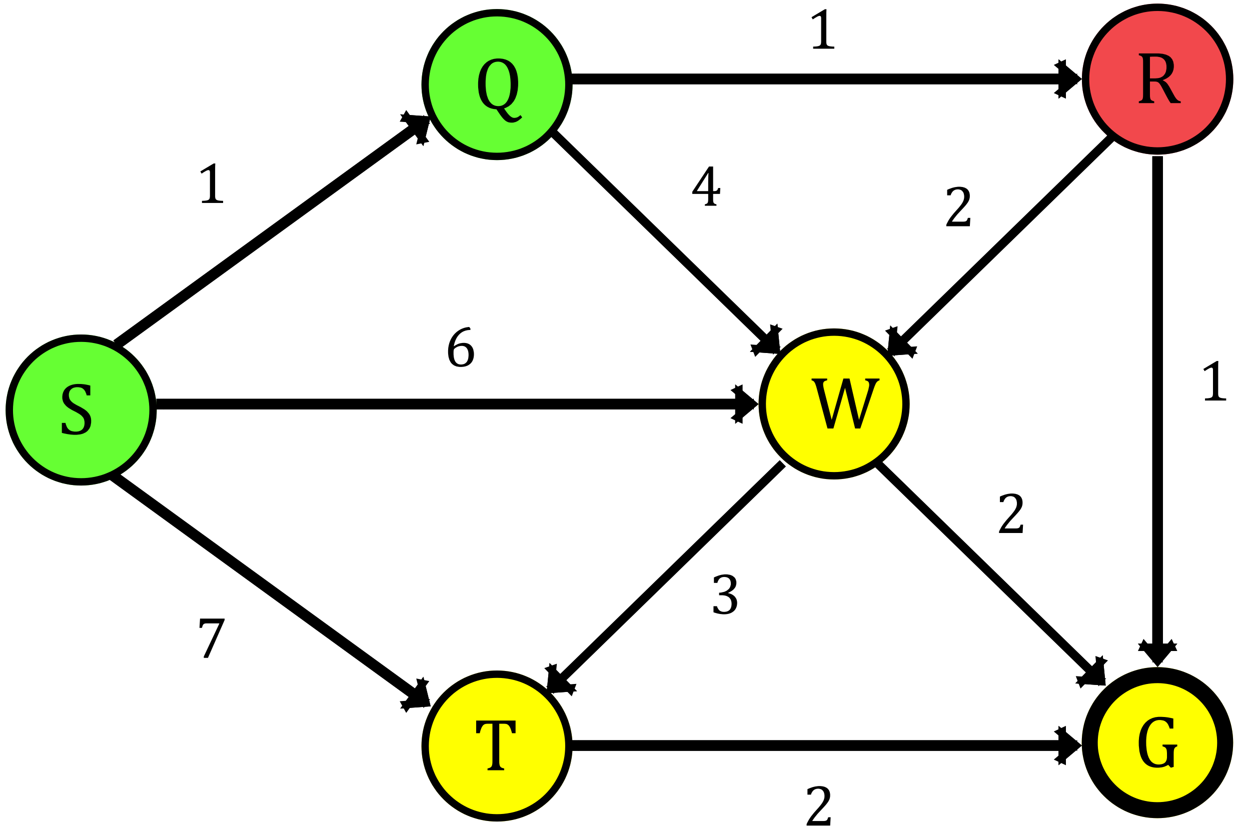
Current Node = R

Stack:

S	Q	R			

Visited:

S	Q	R	G	W	T



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 02)

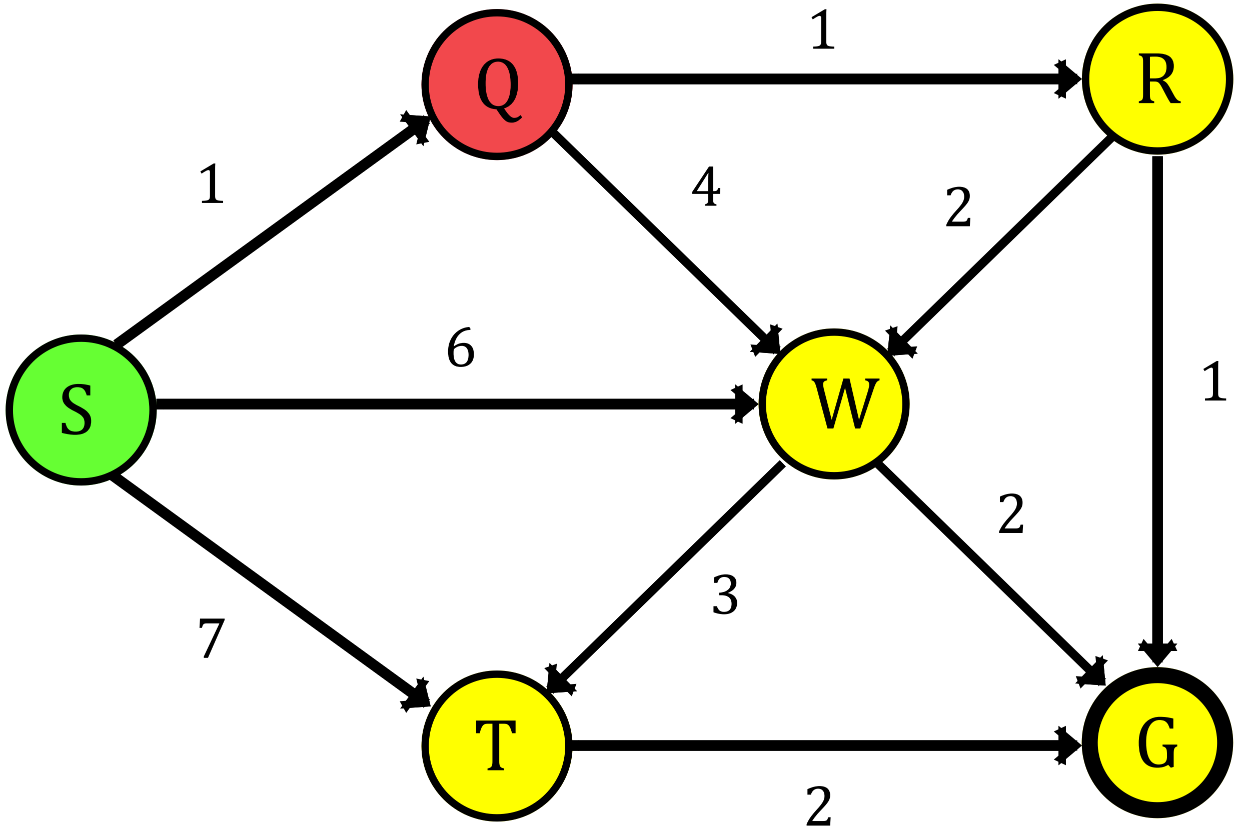
Current Node = Q

Stack:

S	Q				

Visited:

S	Q	R	G	W	T



🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 02)

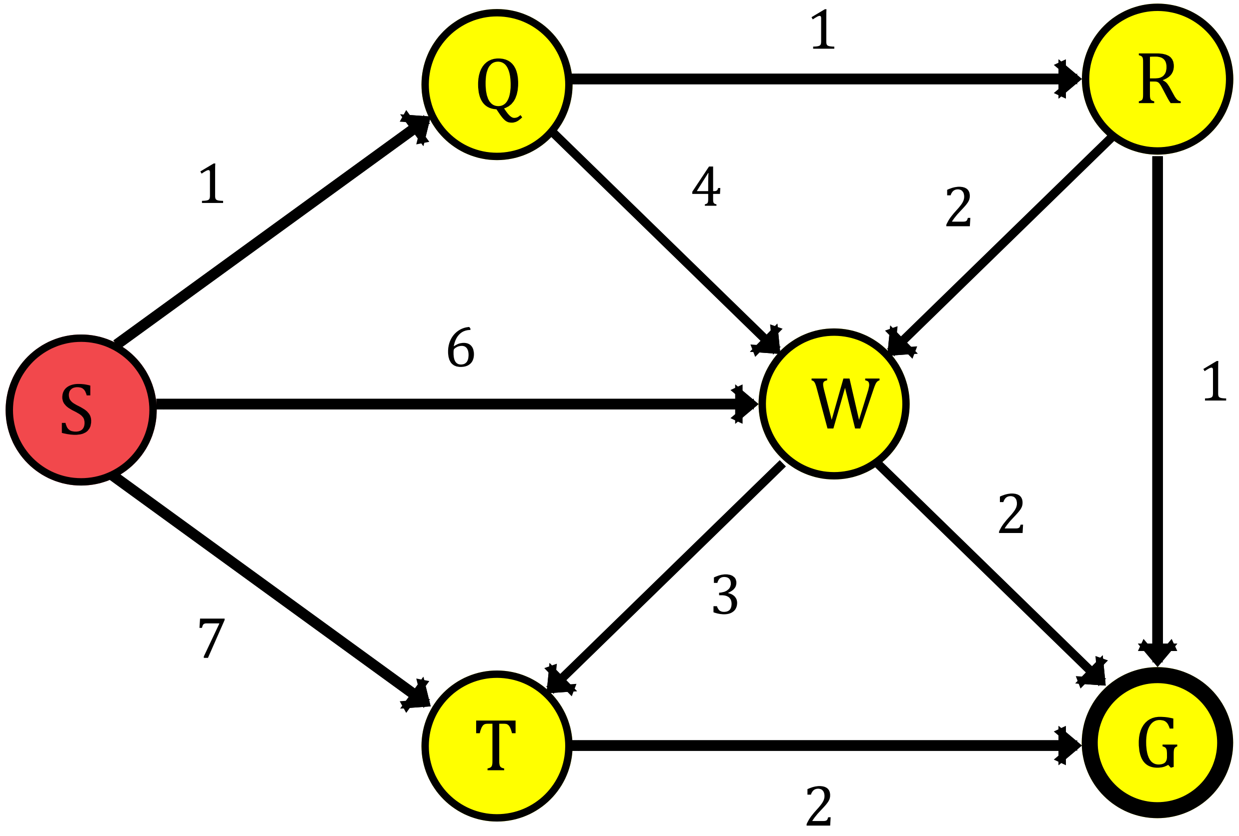
Current Node = S

Stack:

S					

Visited:

S	**Q**	**R**	**G**	**W**	**T**



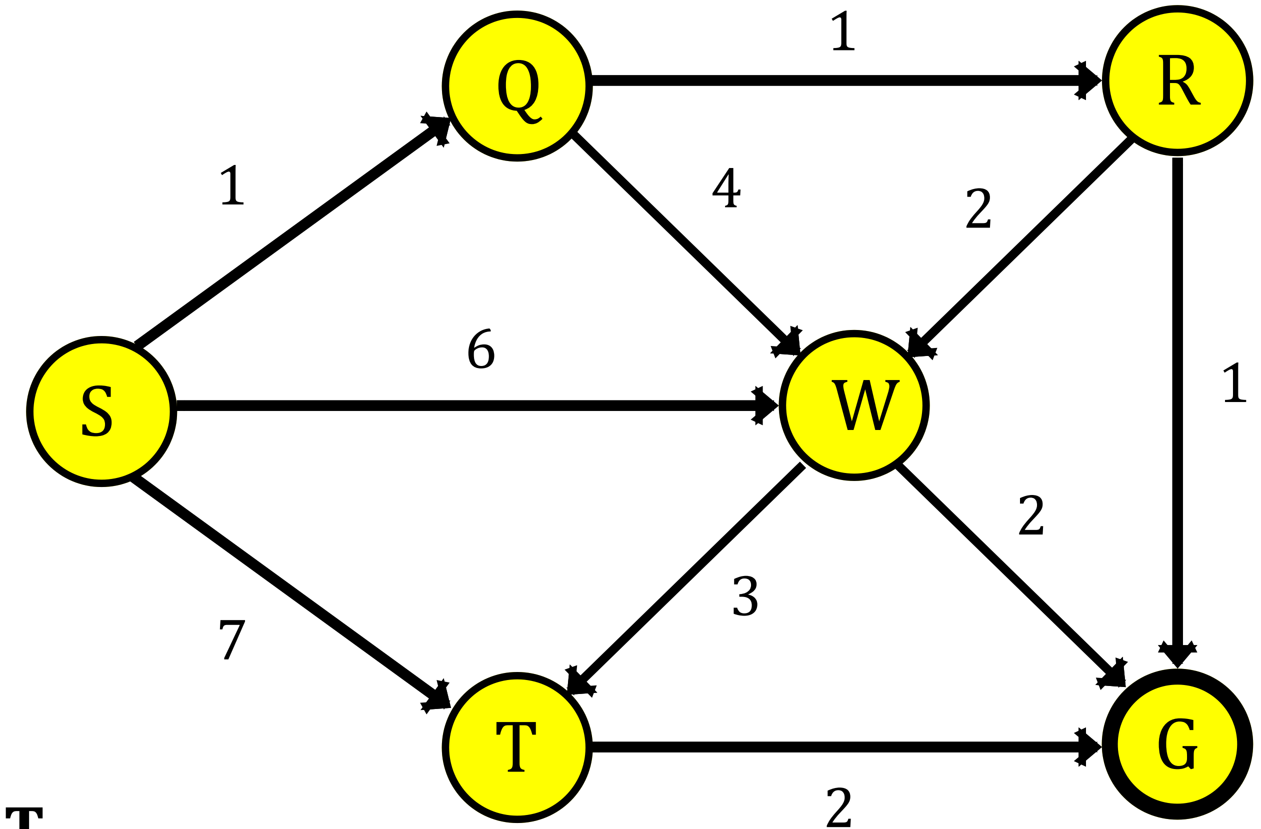
🔴 **Current Node** 🟢 **Opened Node** 🟡 **Closed Node**

Depth-First Search (Problem 02)

Stack:

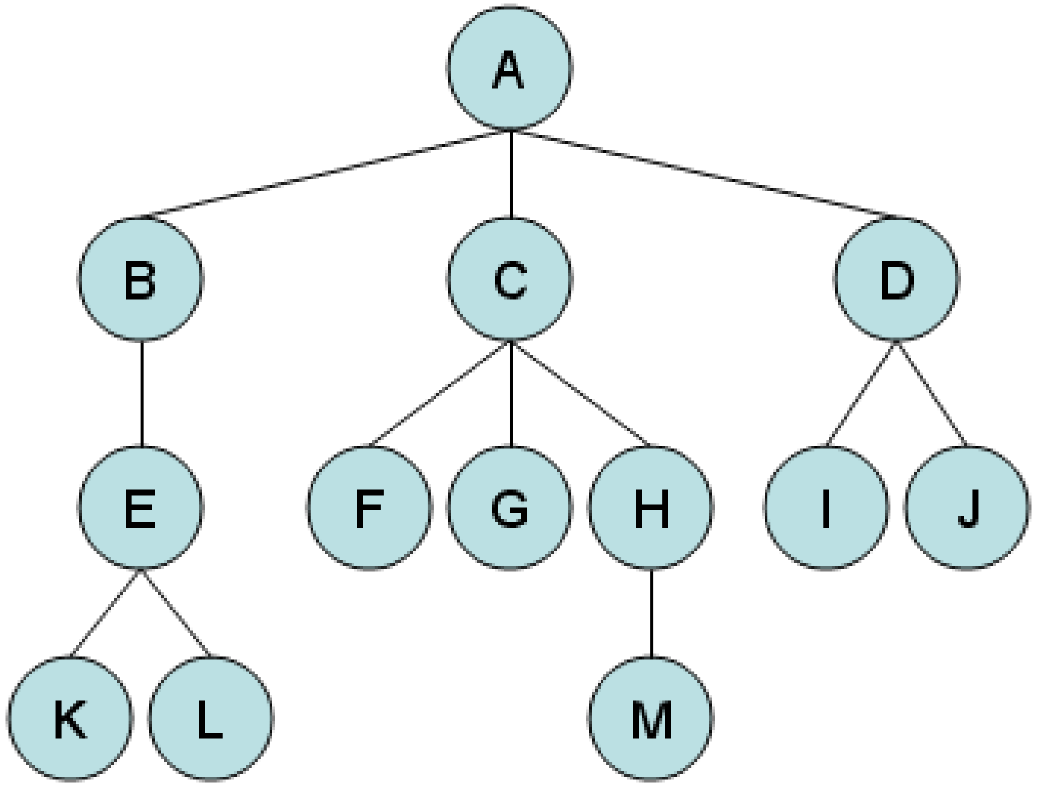
Visited:

S	**Q**	**R**	**G**	**W**	**T**



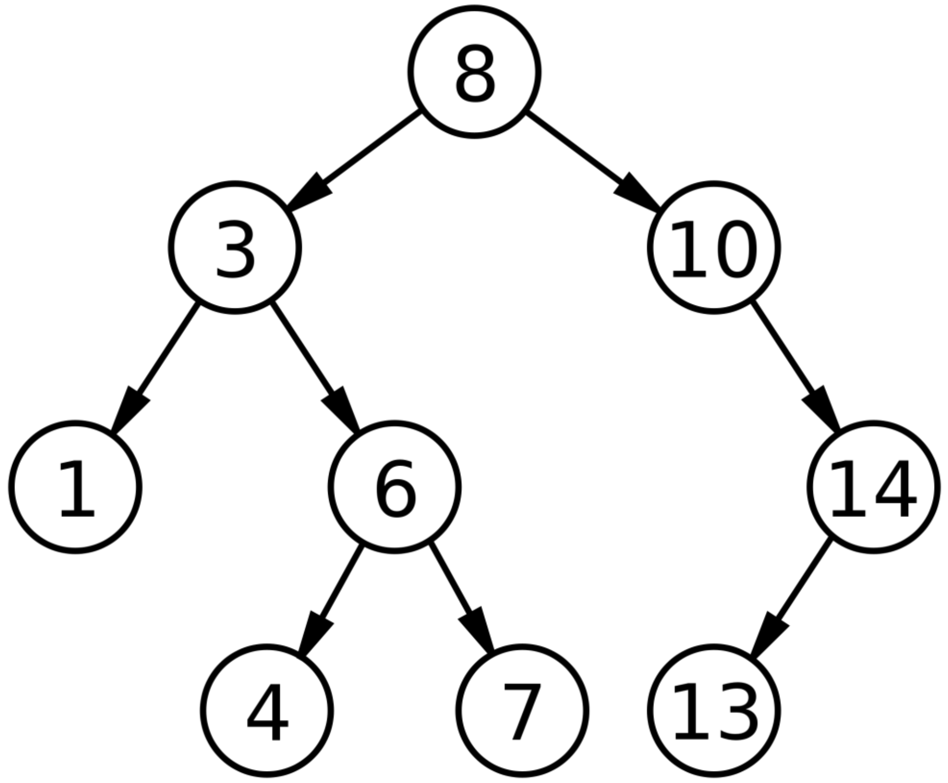
DFS Traversal: S Q R G W T

Depth-First Search (Problem 03)



DFS Traversal: A B E K L C F G H M D I J

Depth-First Search (Problem 04)



DFS Traversal: 8 3 1 6 4 7 10 14 13

Depth-First Search

- Since we don’t expand all nodes at a level, space complexity is modest. For branching factor b and depth d, we require bd number of nodes to be stored in memory. This is much better than b^d.
- In some cases, DFS can be faster than BFS. However, the worse case is still $O(b^d)$.
- If you have deep search trees (or infinite – which is quite possible), DFS may end up running off to infinity and may not be able to recover.
- Thus DFS is neither optimal nor complete.

Depth Limited Search

- Depth-first search will not find a goal if it searches down a path that has infinite length. So, in general, depth-first search is not guaranteed to find a solution, so it is not complete.
- This problem is eliminated by limiting the depth of the search to some value l. However, this introduces another way of preventing depth-first search from finding the goal: if the goal is deeper than l it will not be found.
- Regular depth-first search is a special case, for which $l=\infty$.

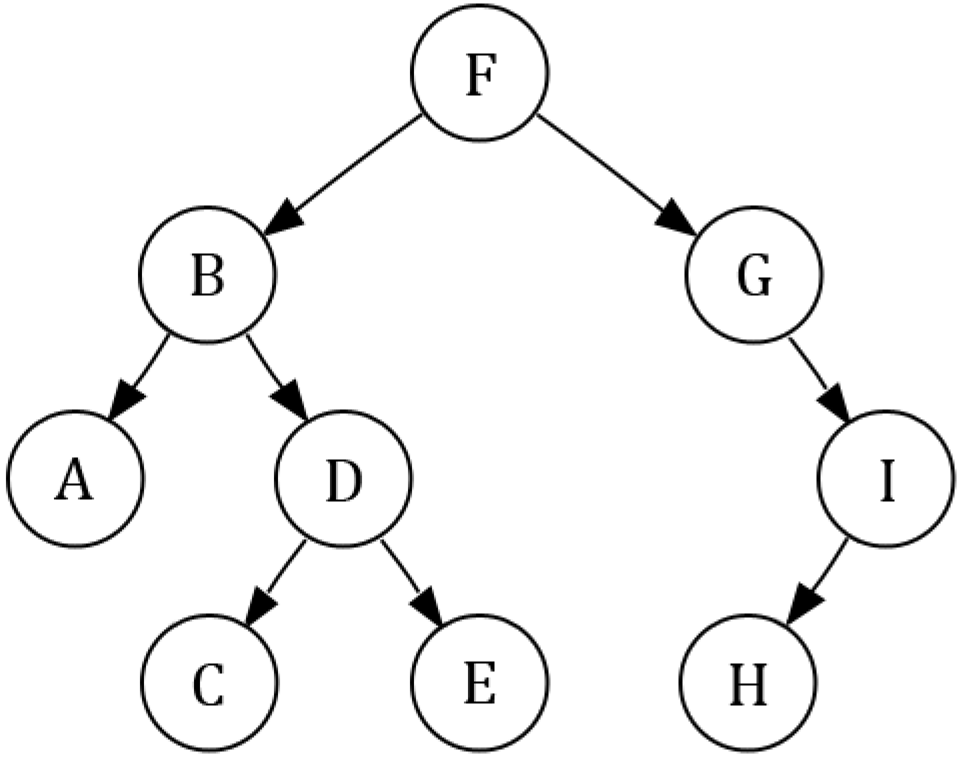
Depth Limited Search

- Depth limited search is complete but not optimal.
- If we choose a depth limit that is too small, then depth limited search is not even complete.
- The time and space complexity of depth limited search is similar to depth first search.

Depth Limited Search (Problem 01)

Visited	Level		
	0	**1**	**2**
F	F		
F B	F	B	
F B A	F	B	A
F B A	F	B	
F B A D	F	B	D
F B A D	F	B	
F B A D	F		
F B A D G	F	G	
F B A D G I	F	G	I
F B A D G I	F	G	
F B A D G I	F		
F B A D G I			

Depth Level Limit:
L = 2



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

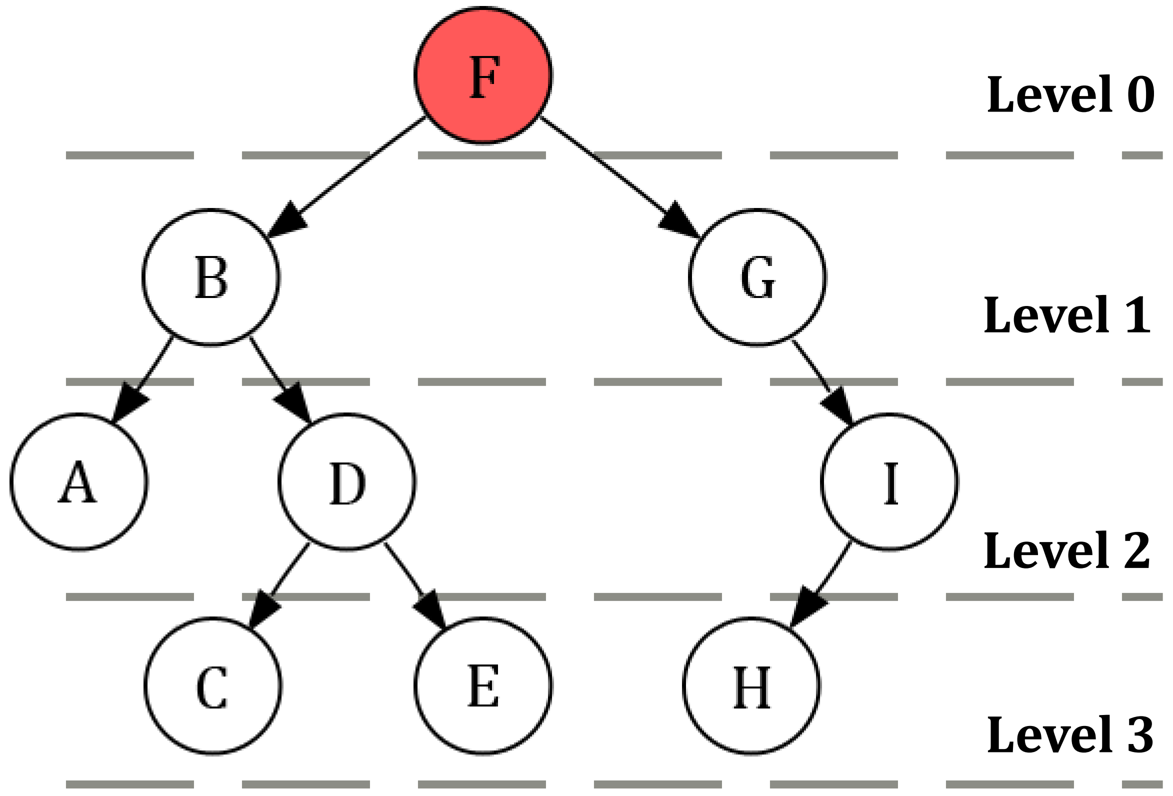
Stack:

F									

Visited:

F									

Current Node = F



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

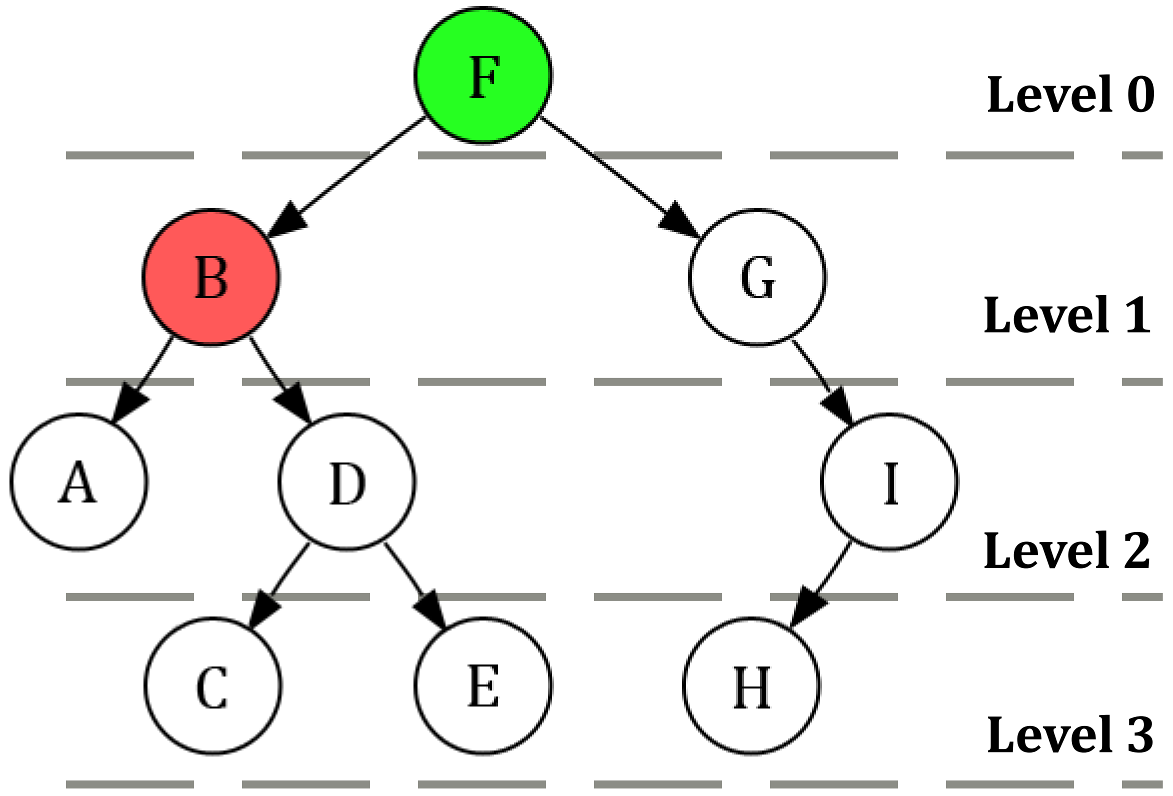
Stack:

F	**B**								

Visited:

F	**B**								

Current Node = B



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

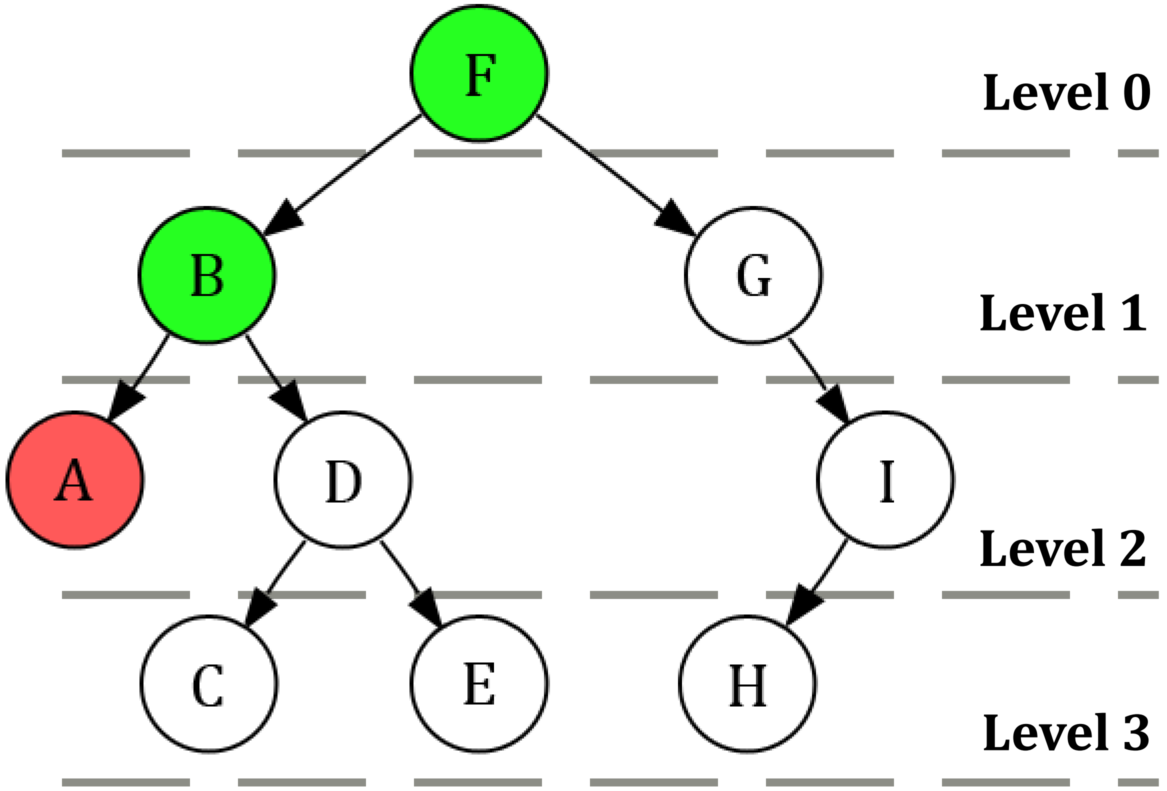
Stack:

F	B	A							

Visited:

F	B	A							

Current Node = A



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

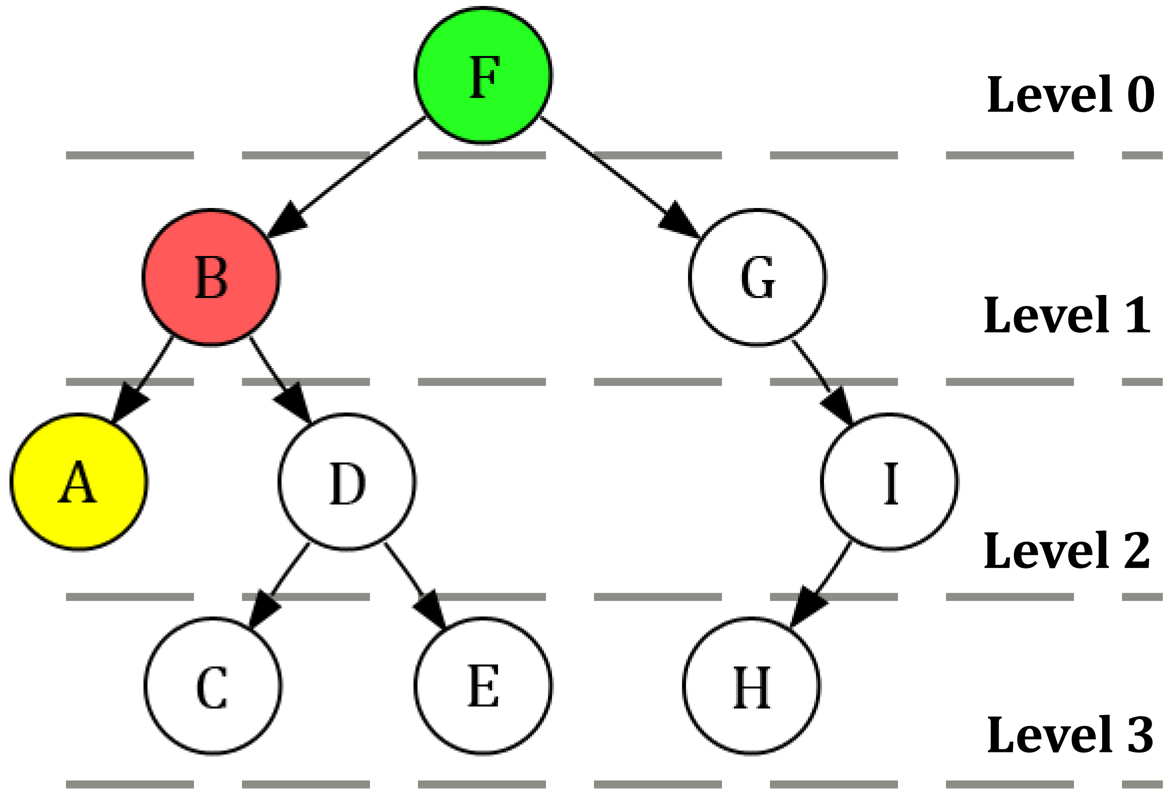
Stack:

F	**B**								

Visited:

F	**B**	**A**							

Current Node = B



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

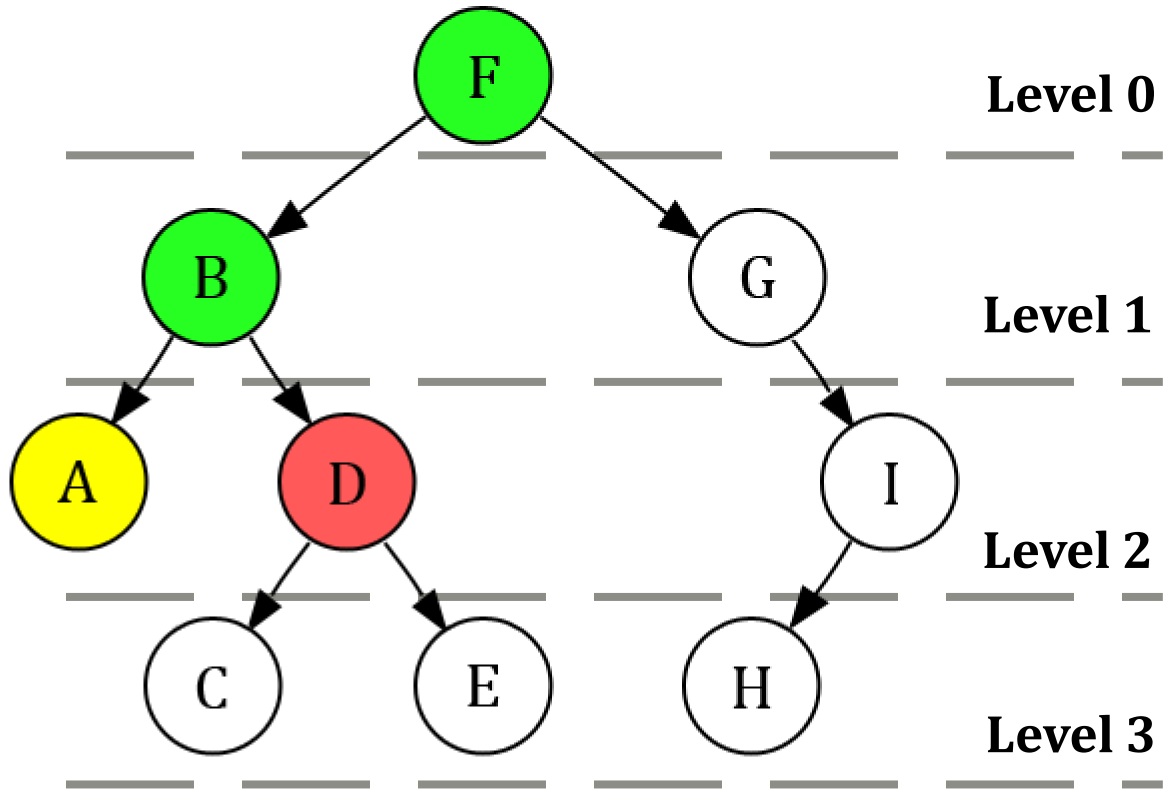
Stack:

F	B	D							

Visited:

F	B	A	D						

Current Node = D



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

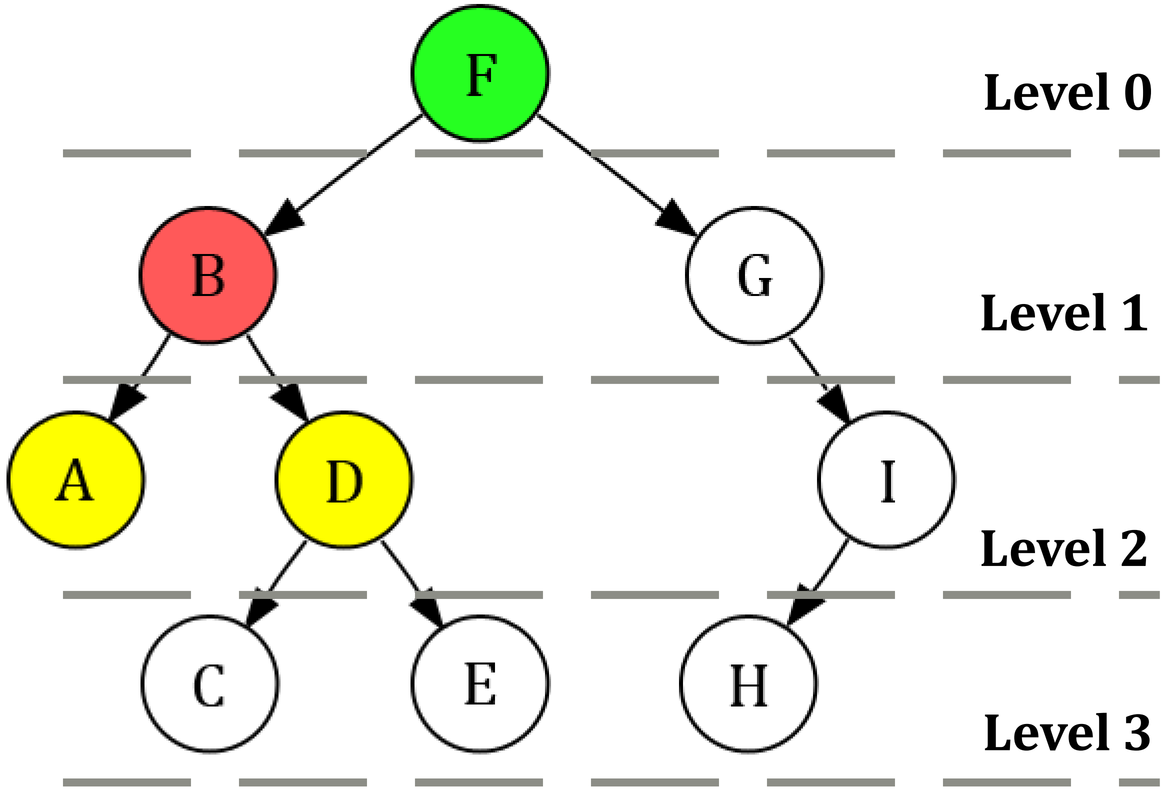
Stack:

F	B								

Visited:

F	B	A	D						

Current Node = B



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

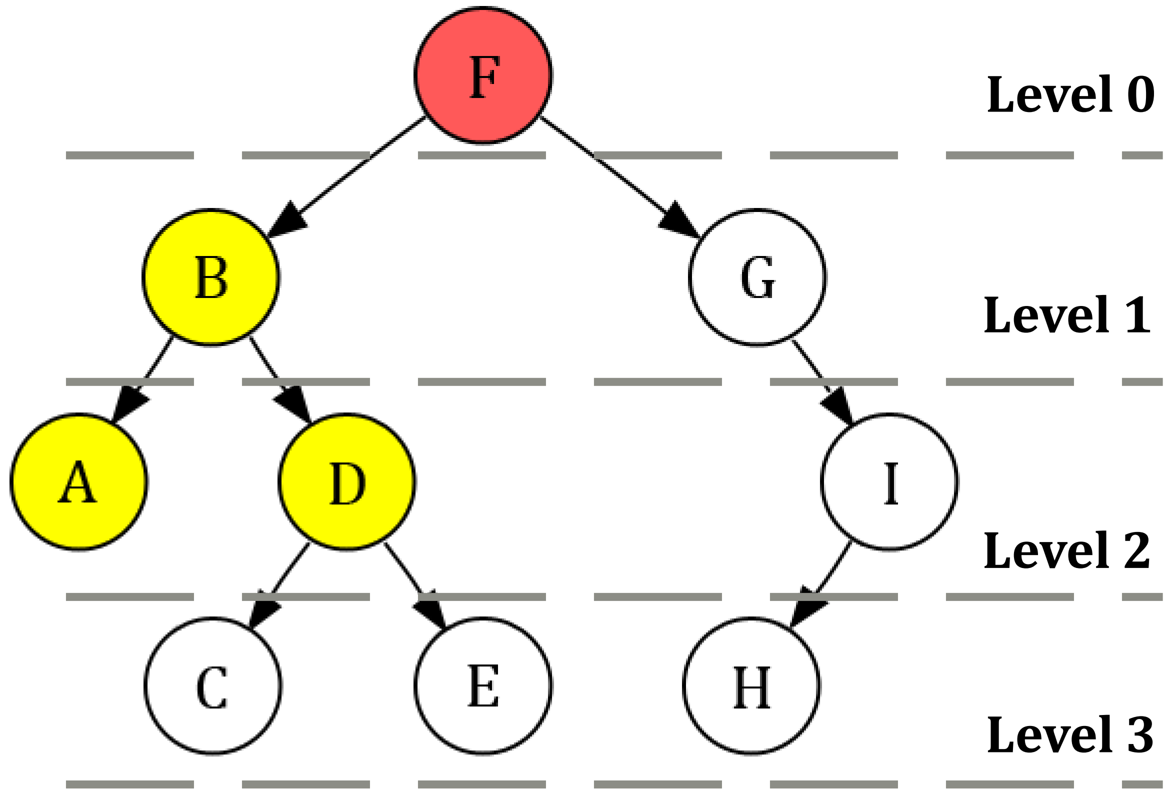
Stack:

F									

Visited:

F	**B**	**A**	**D**						

Current Node = F



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

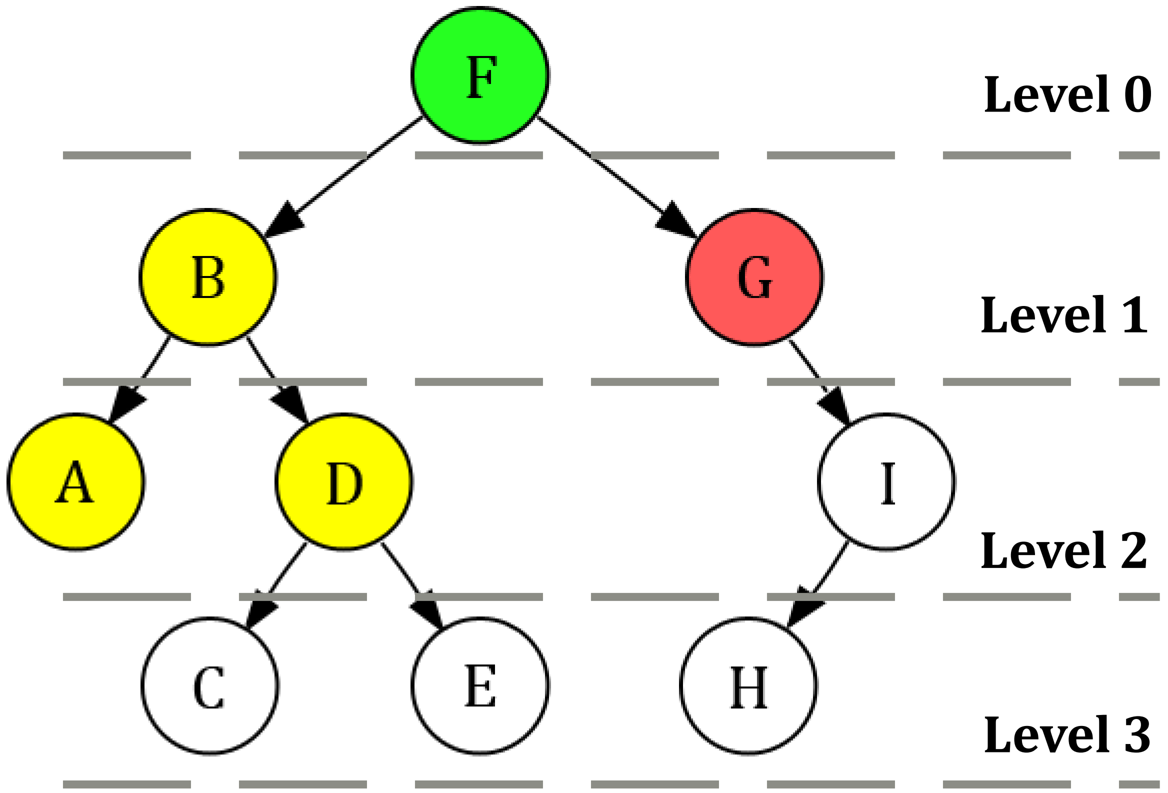
Stack:

F	G								

Visited:

F	B	A	D	G					

Current Node = G



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

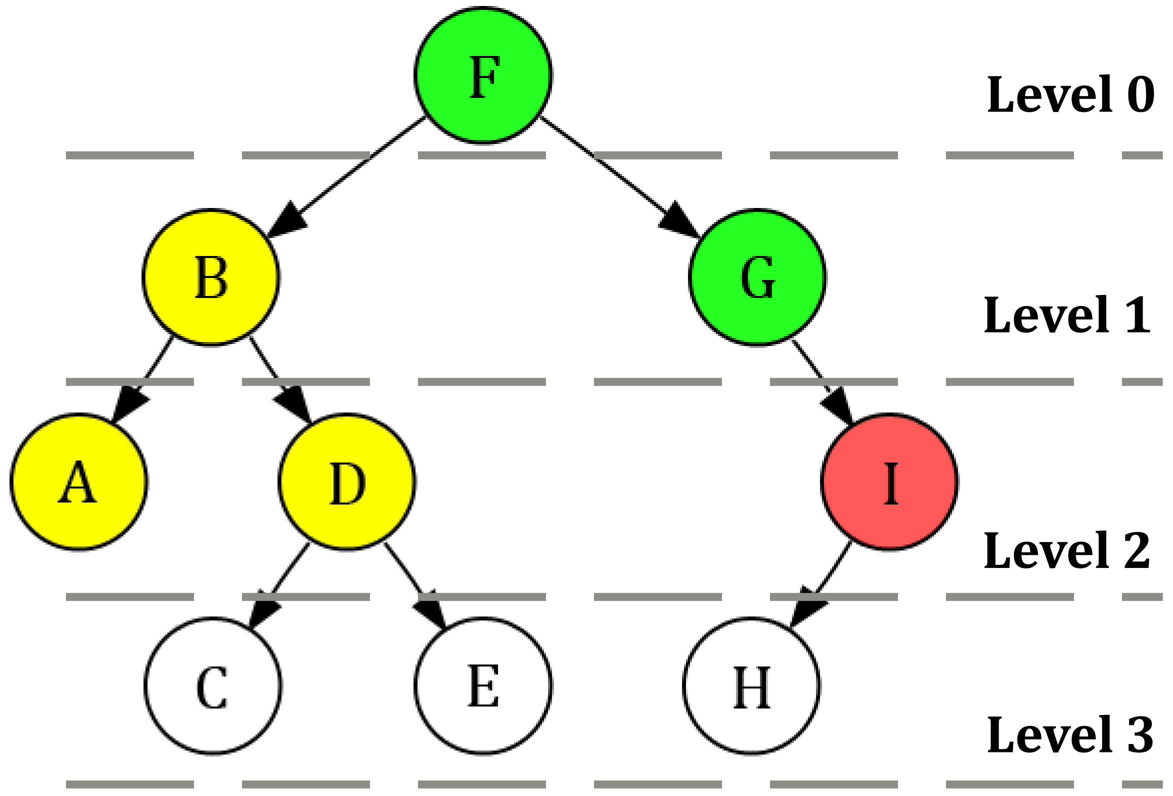
Stack:

F	**G**	**I**							

Visited:

F	**B**	**A**	**D**	**G**	**I**				

Current Node = I



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

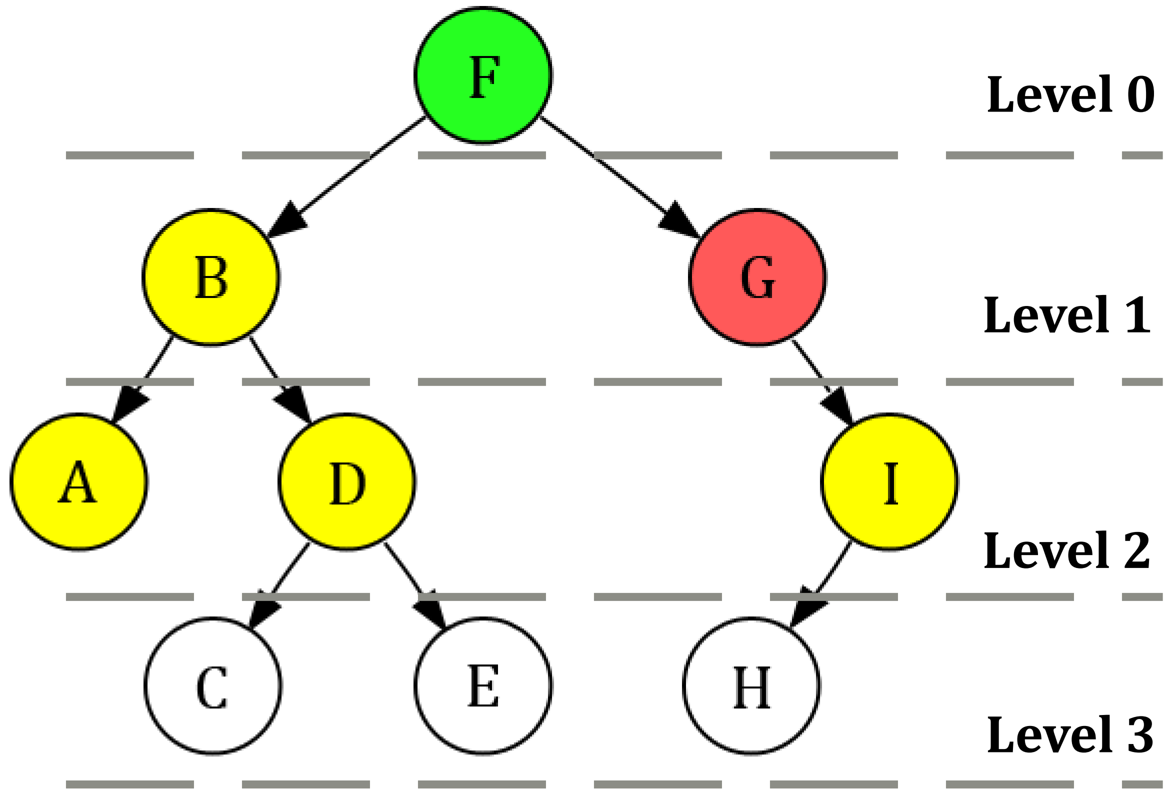
Stack:

F	G								

Visited:

F	B	A	D	G	I				

Current Node = G



Depth Limited Search (Problem 01)

Depth Level Limit:

L = 2

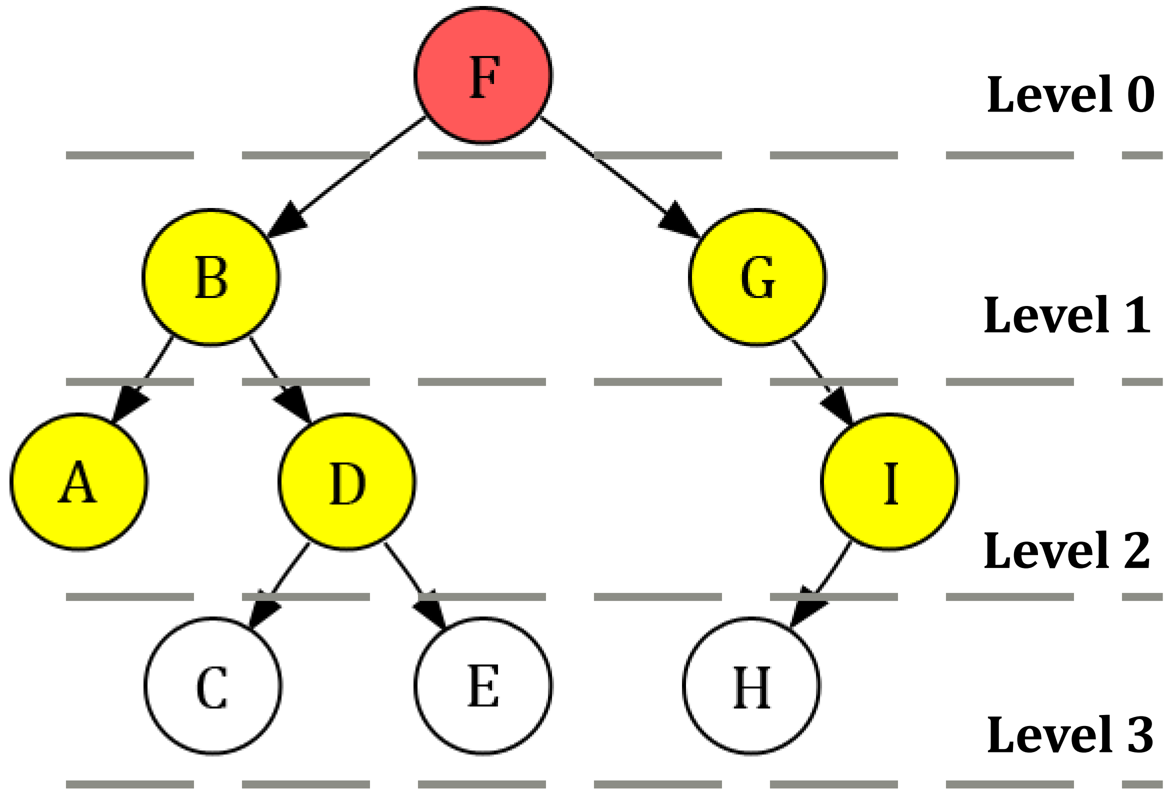
Stack:

F									

Visited:

F	**B**	**A**	**D**	**G**	**I**				

Current Node = F



Depth Limited Search (Problem 01)

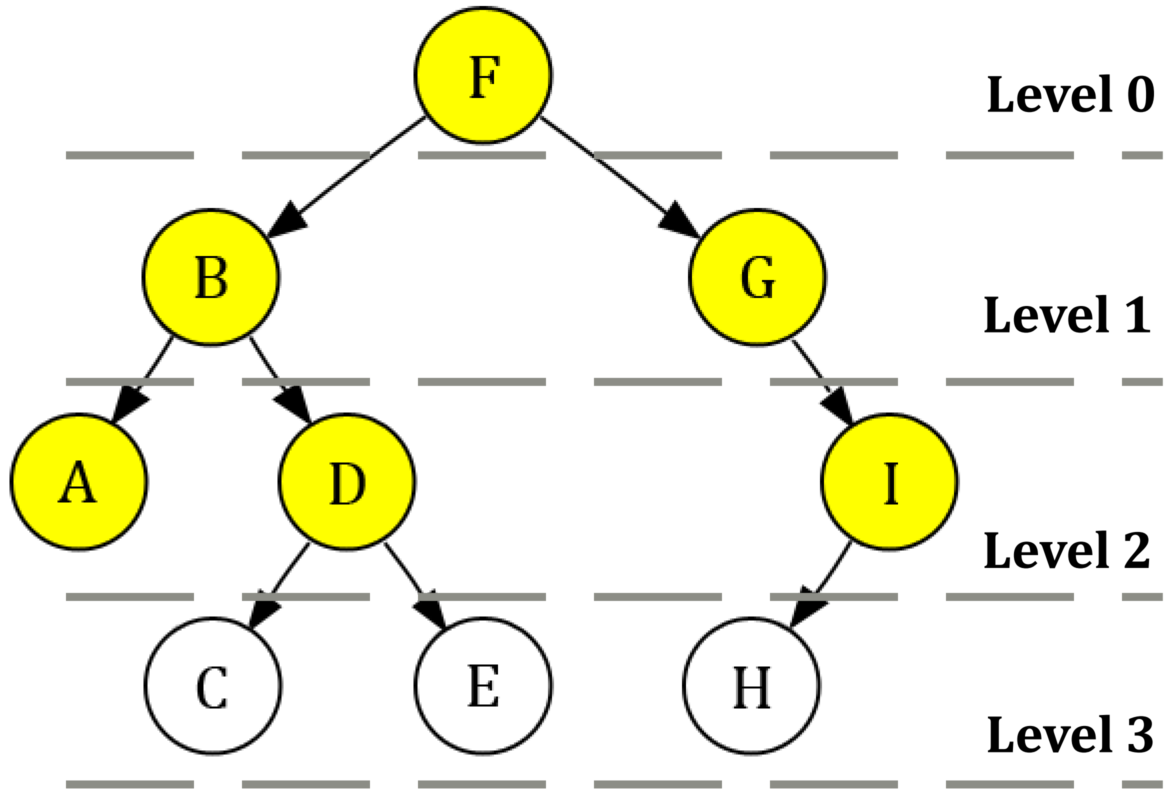
Depth Level Limit:

L = 2

Stack:

Visited:

F	**B**	**A**	**D**	**G**	**I**				



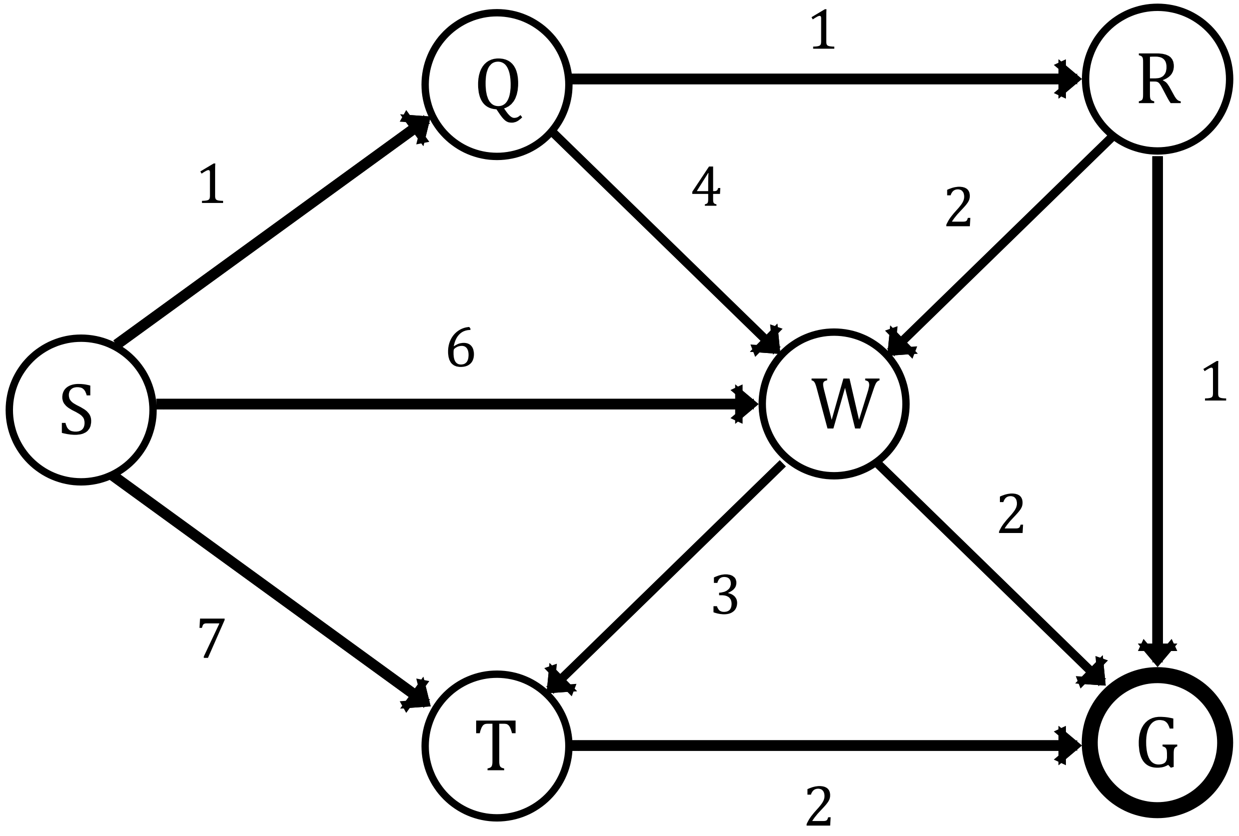
DFS Limited Traversal: F B A D G I

Depth Limited Search (Problem 02)

Visited	Level		
	0	**1**	**2**
S	S		
S Q	S	Q	
S Q R	S	Q	R
S Q R	S	Q	
S Q R W	S	Q	W
S Q R W	S	Q	
S Q R W	S		
S Q R W T	S	T	
S Q R W T G	S	T	G
S Q R W T G	S	T	
S Q R W T G	S		
S Q R W T G			

Depth Level Limit:

L = 2



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

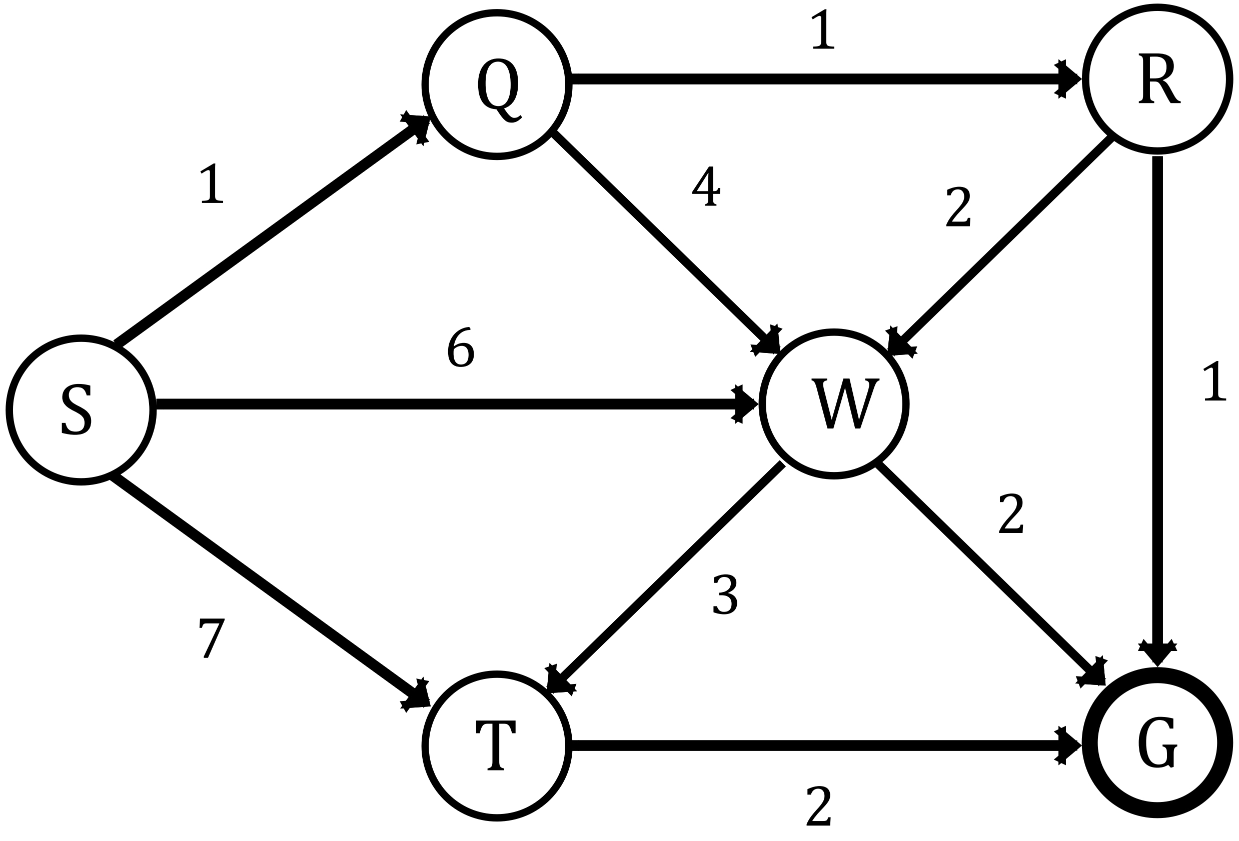
Stack:

S					

Visited:

S					

Current Node = S



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

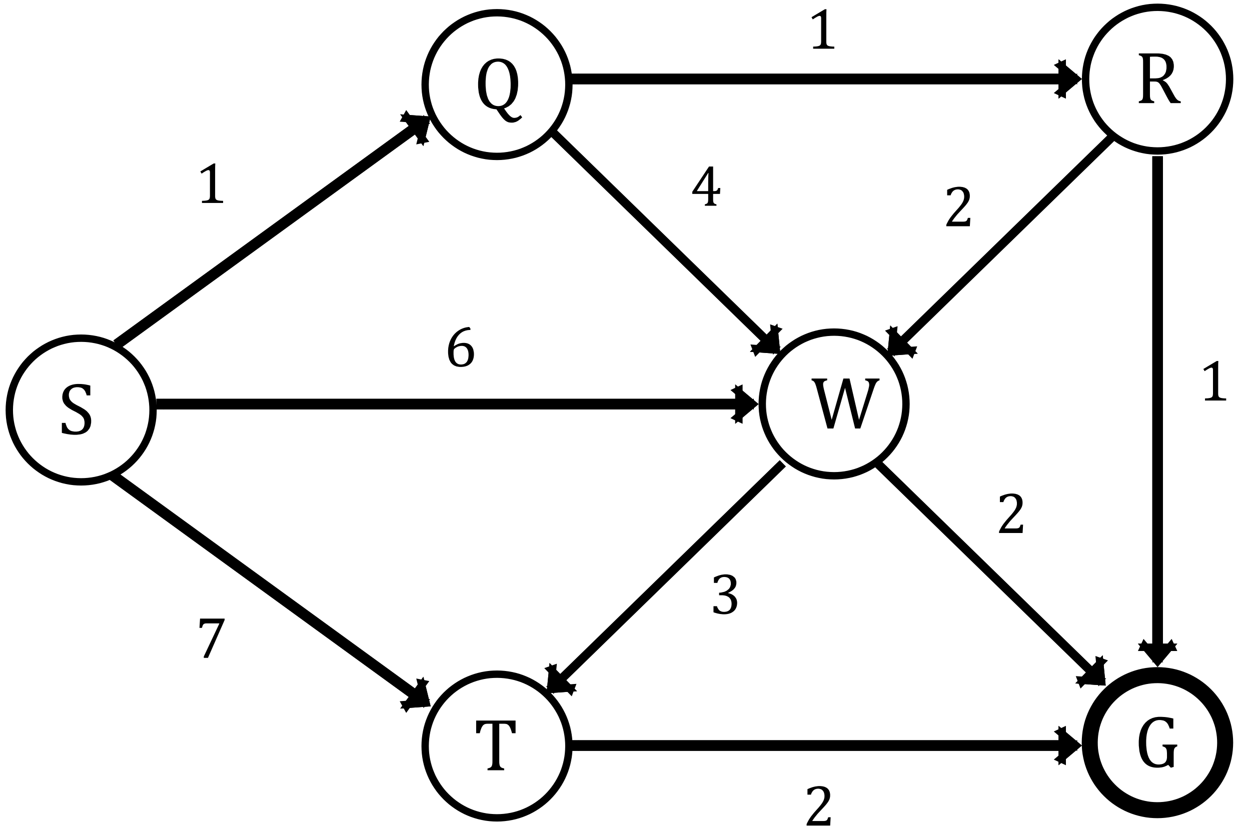
Stack:

S	**Q**				

Visited:

S	**Q**				

Current Node = Q



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

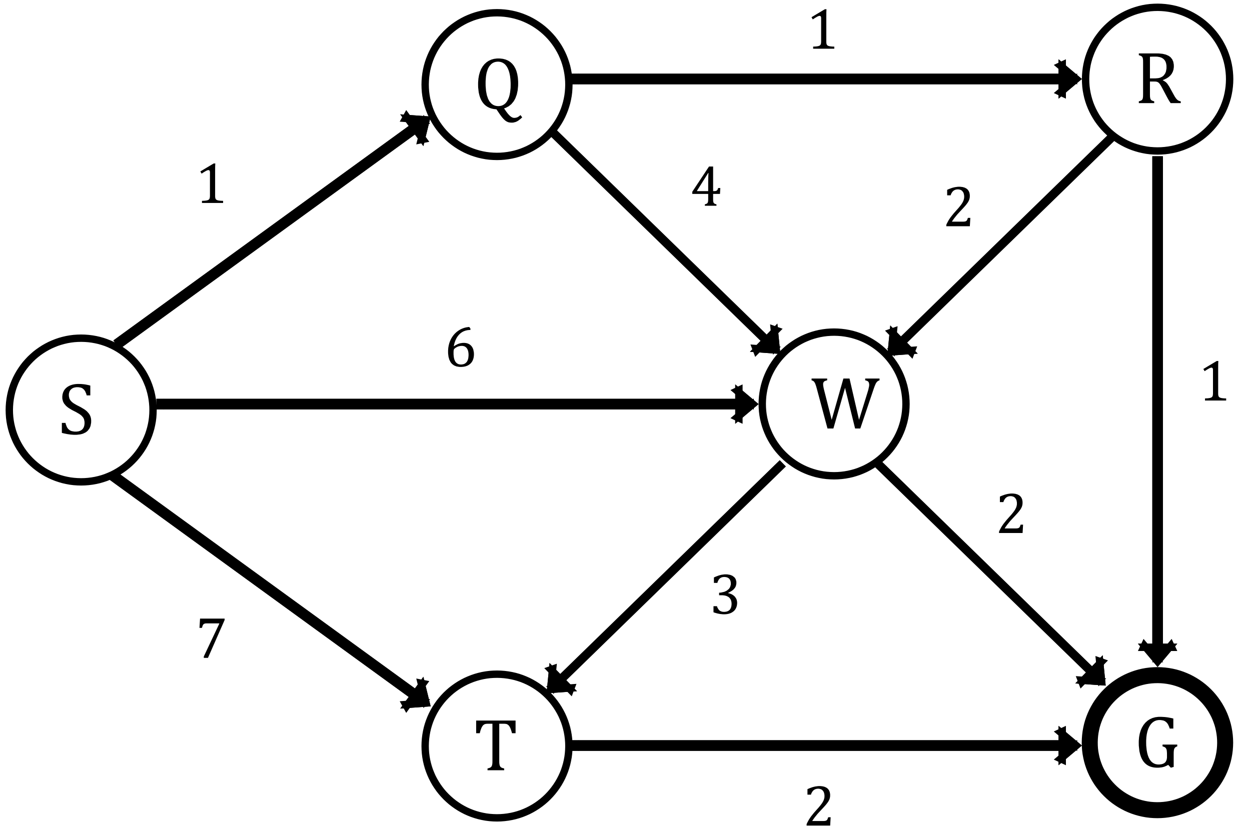
Stack:

S	**Q**	**R**			

Visited:

S	**Q**	**R**			

Current Node = R



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

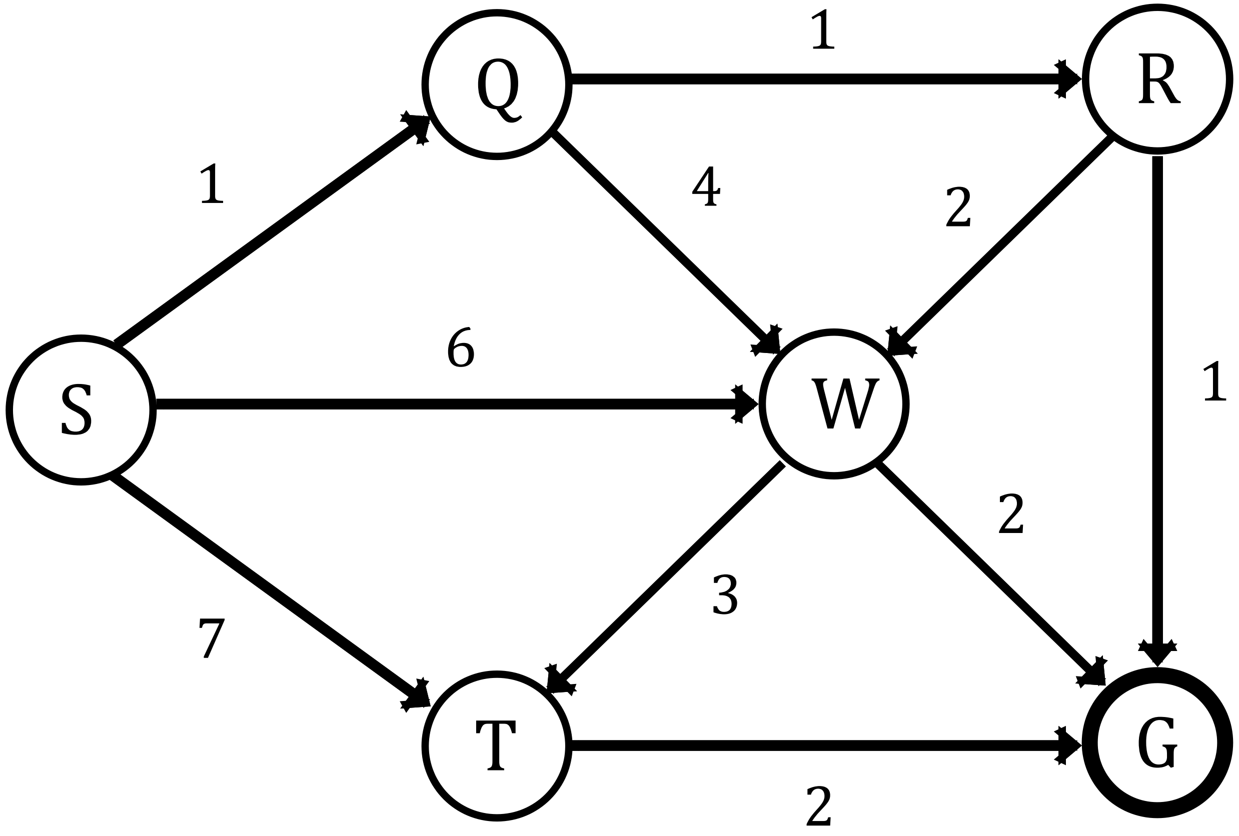
Stack:

S	**Q**				

Visited:

S	**Q**	**R**			

Current Node = Q



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

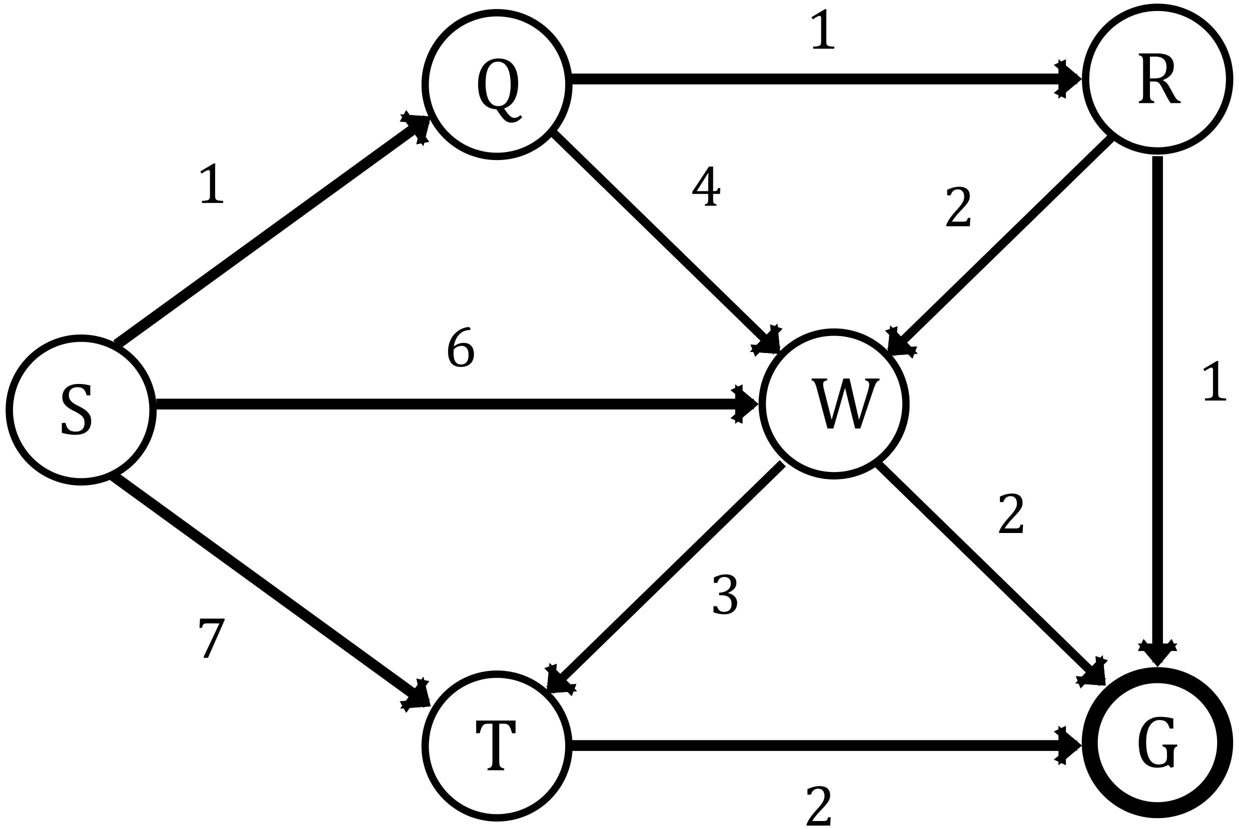
Stack:

S	**Q**	**W**			

Visited:

S	**Q**	**R**	**W**		

Current Node = W


$$\begin{tikzcd}
 & Q \arrow[rr, "1"] \arrow[dr, "4"] & & R \arrow[dl, "2"'] \arrow[dd, "1"] \\
S \arrow[ur, "1"] \arrow[rr, "6"] \arrow[dr, "7"'] & & W \arrow[dl, "3"] \arrow[dr, "2"] & \\
 & T \arrow[rr, "2"'] & & G
\end{tikzcd}$$

Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

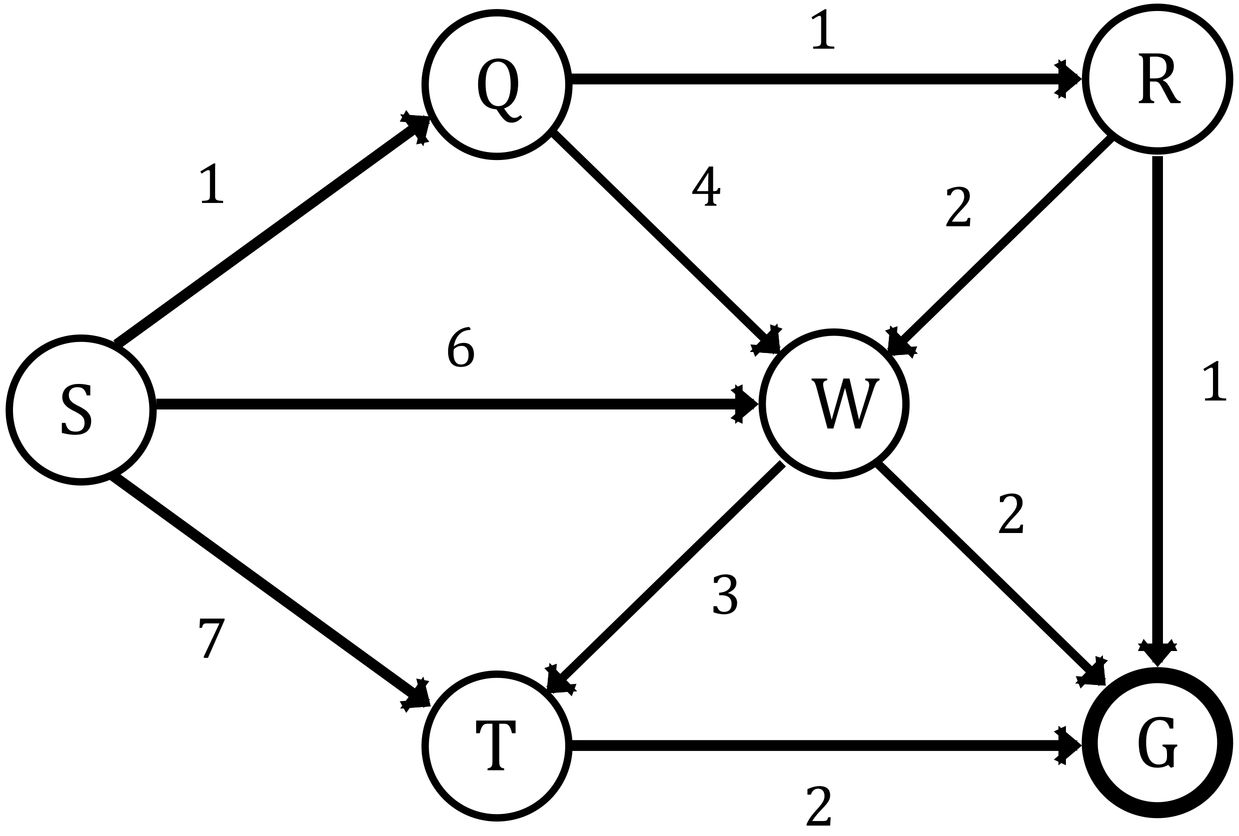
Stack:

S	Q				

Visited:

S	Q	R	W		

Current Node = Q



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

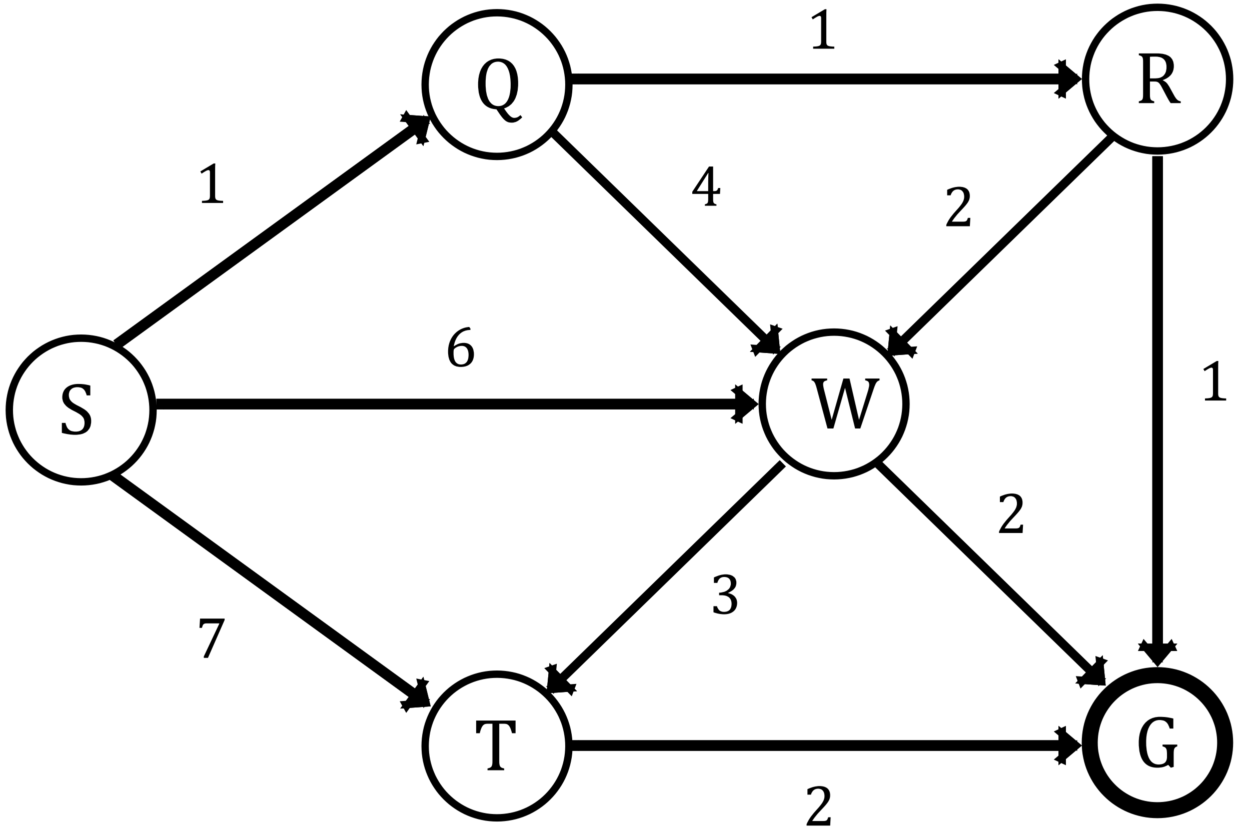
Stack:

S					

Visited:

S	**Q**	**R**	**W**		

Current Node = S



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

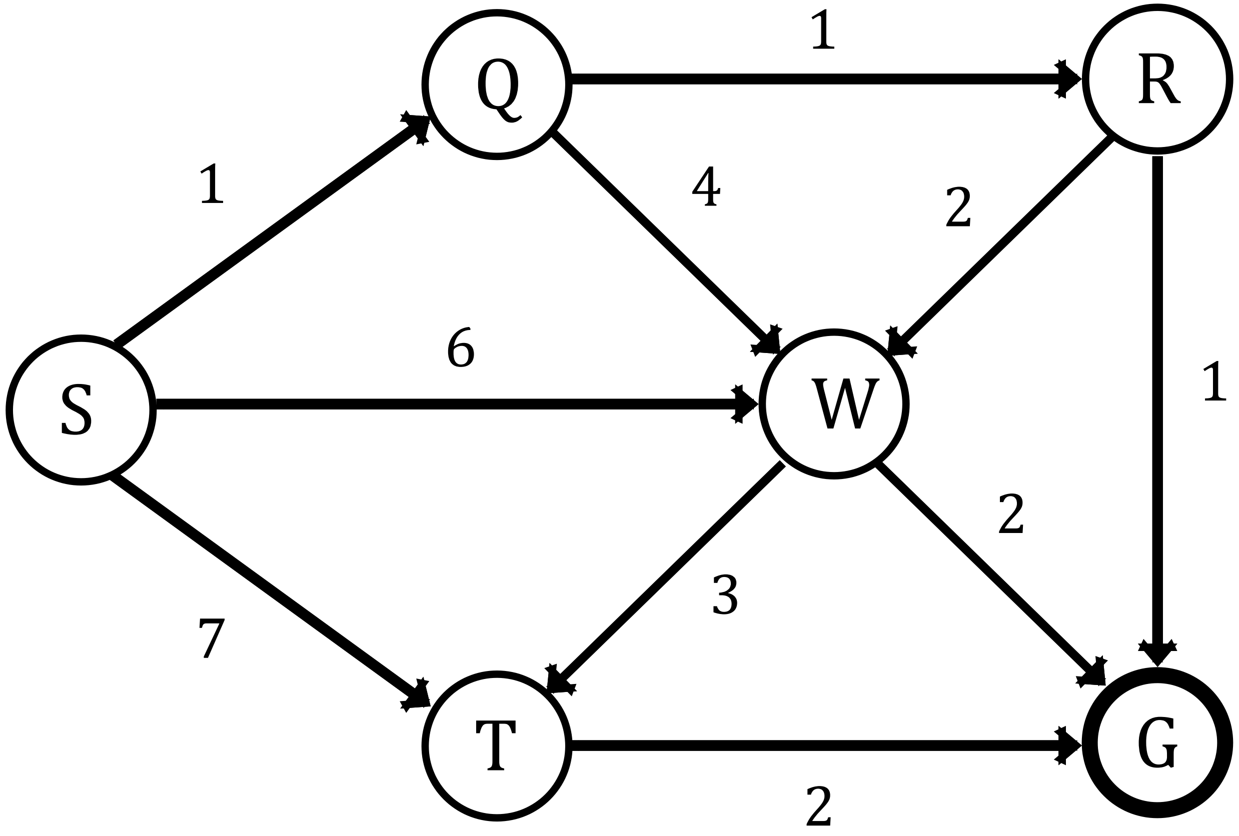
Stack:

S	**T**				

Visited:

S	**Q**	**R**	**W**	**T**	

Current Node = T



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

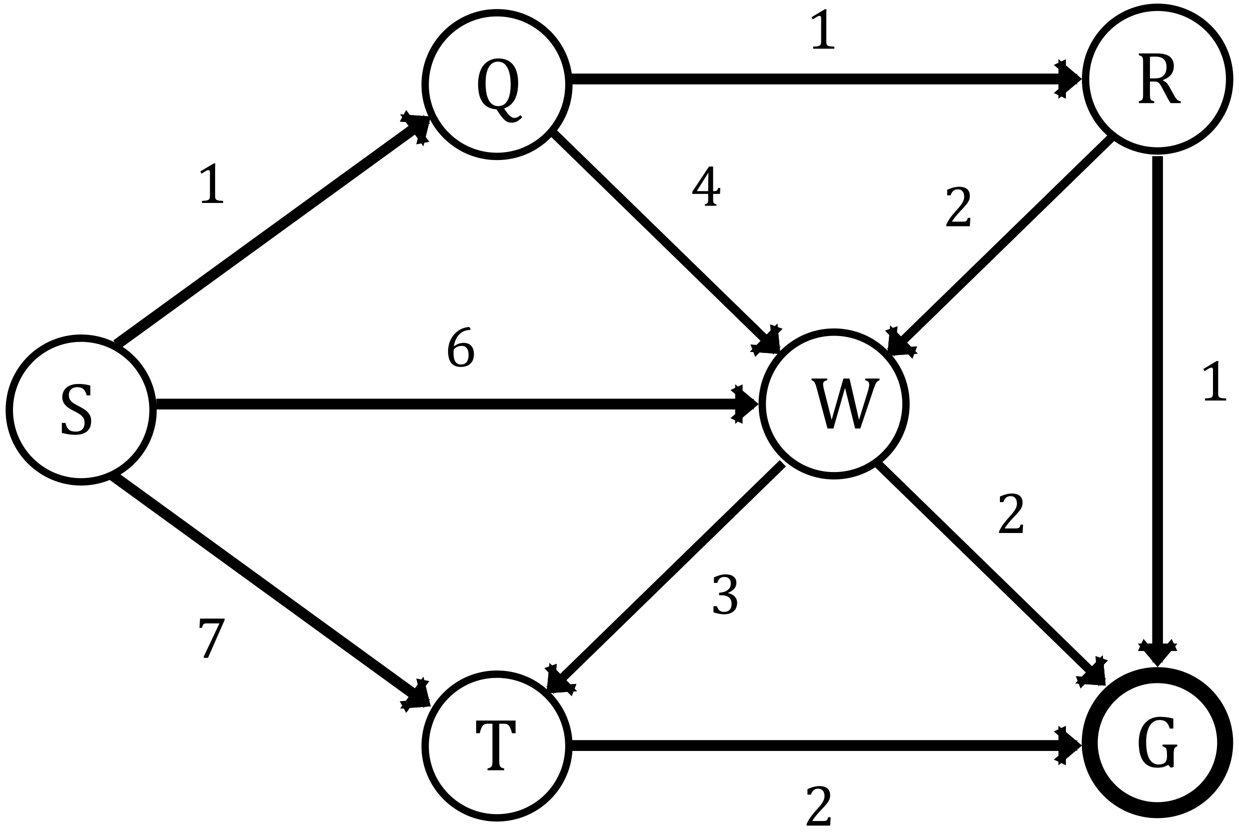
Stack:

S	**T**	**G**			

Visited:

S	**Q**	**R**	**W**	**T**	**G**

Current Node = G



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

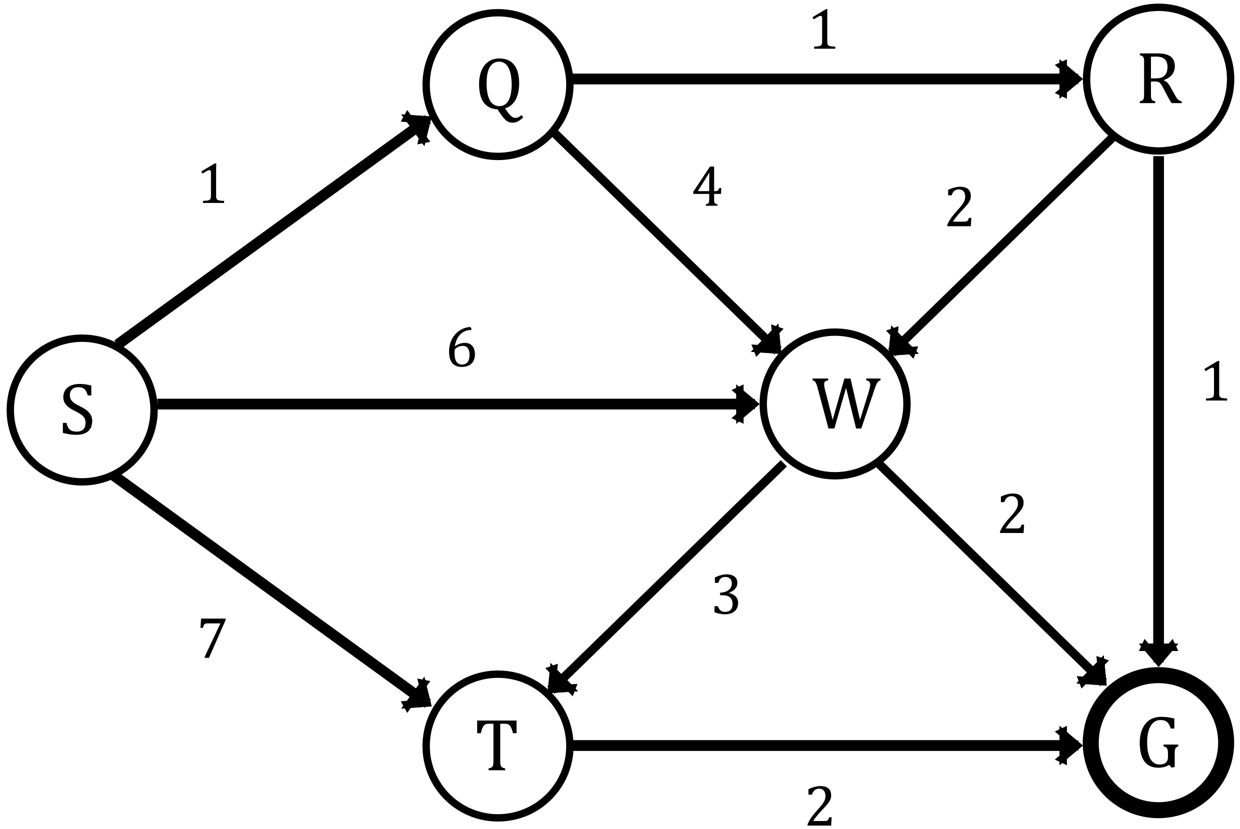
Stack:

S	T				

Visited:

S	Q	R	W	T	G

Current Node = T



Depth Limited Search (Problem 02)

Depth Level Limit:

L = 2

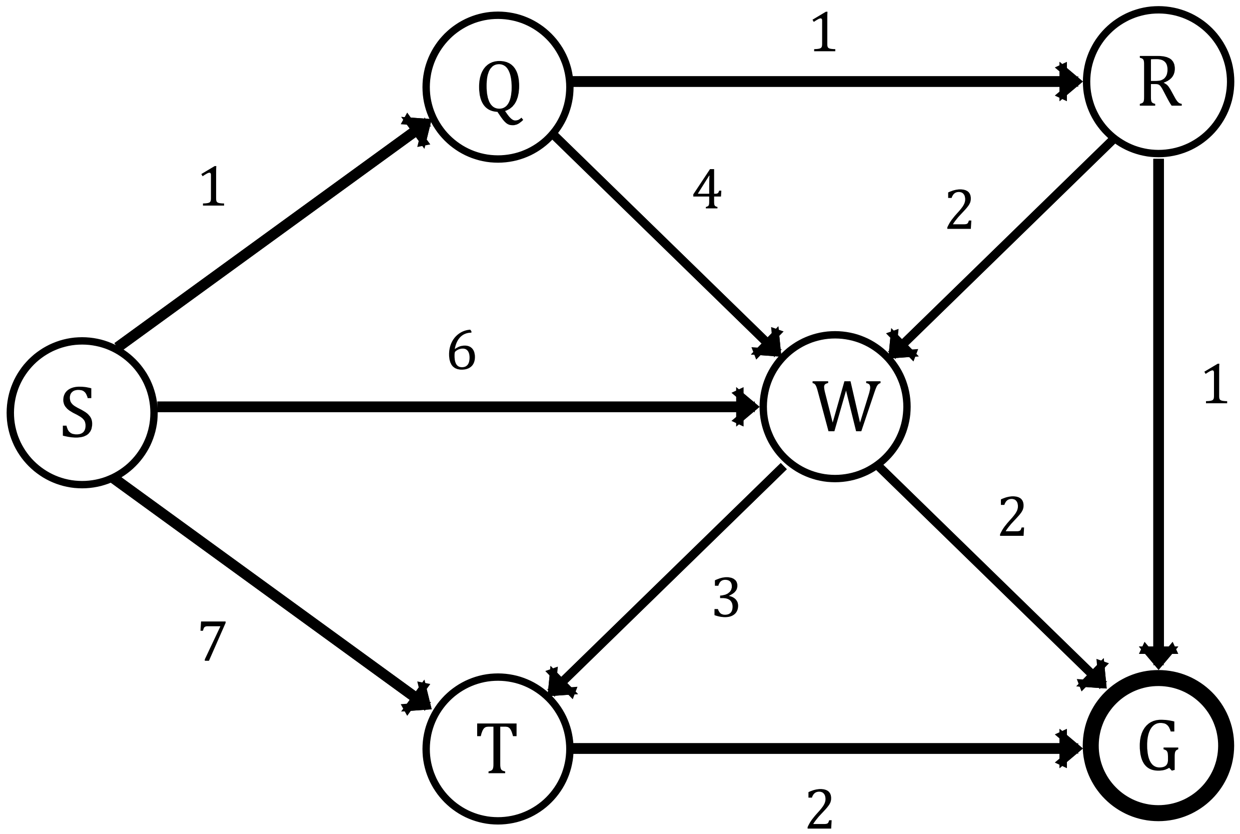
Stack:

S					

Visited:

S	**Q**	**R**	**W**	**T**	**G**

Current Node = S



Depth Limited Search (Problem 02)

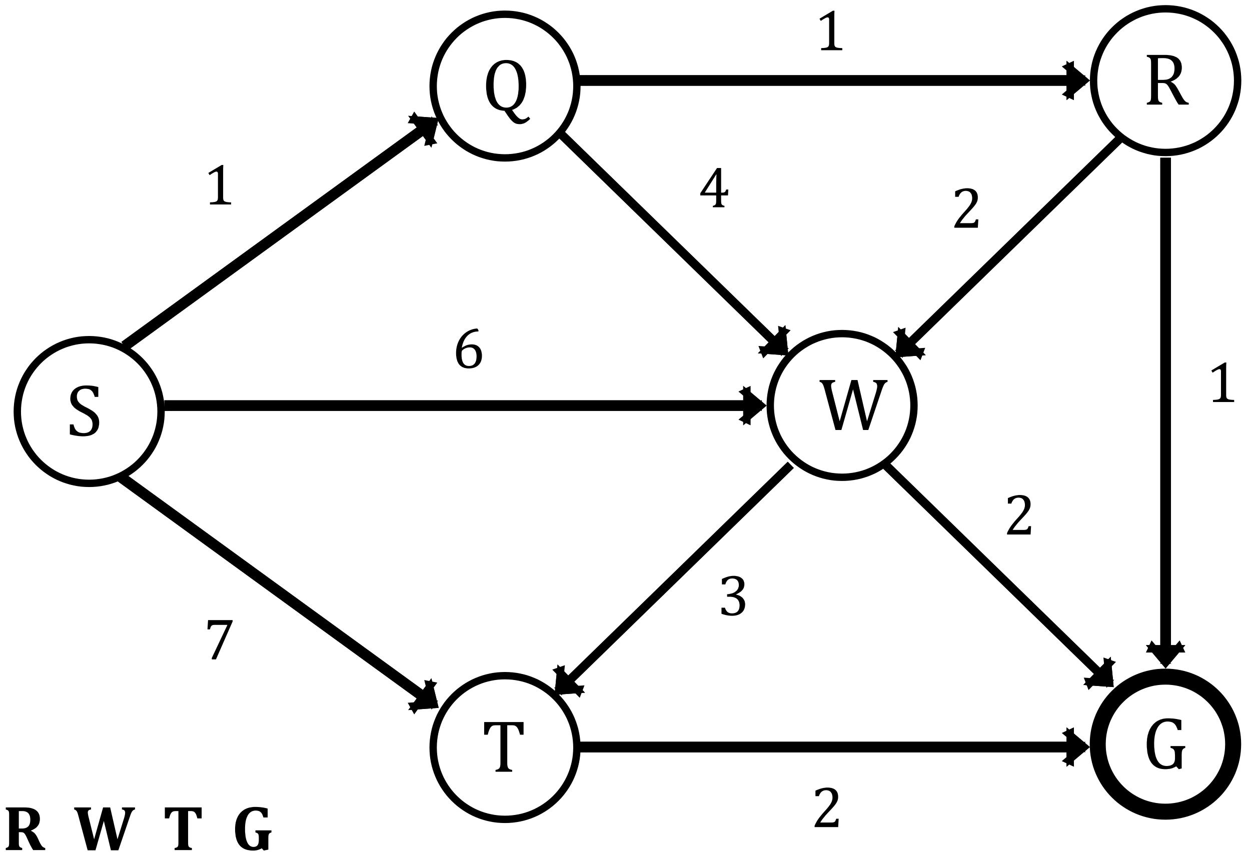
Depth Level Limit:

L = 2

Stack:

Visited:

S	**Q**	**R**	**W**	**T**	**G**



DFS Limited Traversal: S Q R W T G

Depth-First Iterative Deeping Search (DFID)

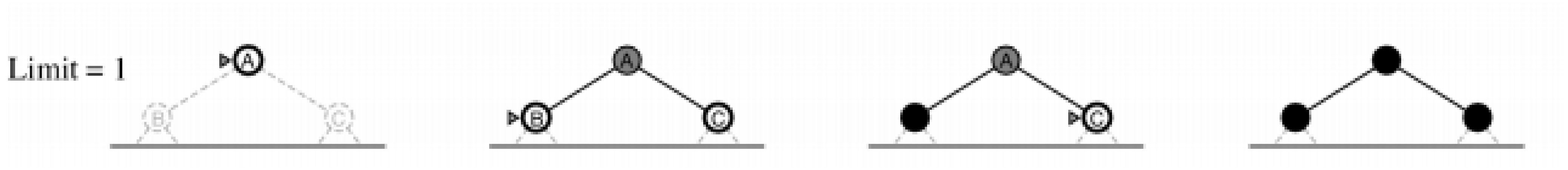
- Depth-first search will not find a goal if it searches down a path that has infinite length. So, in general, depth-first search is not guaranteed to find a solution, so it is not complete.
- This problem is eliminated by limiting the depth of the search to some value l. However, this introduces another way of preventing depth-first search from finding the goal: if the goal is deeper than l it will not be found.
- Regular depth-first search is a special case, for which l=∞.

Depth-First Iterative Deeping Search (DFID)

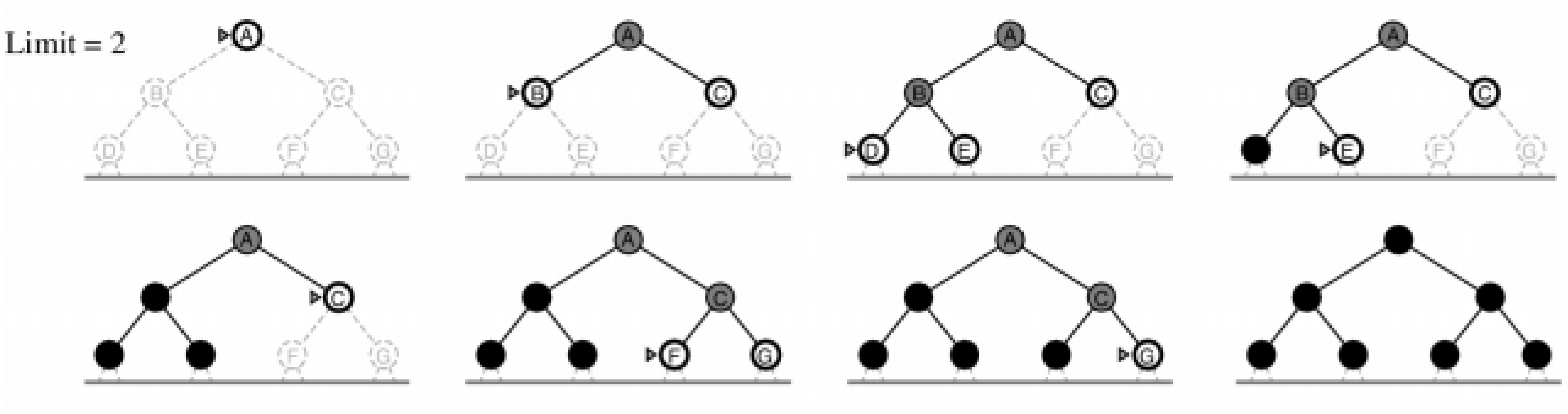
- If a depth-limited depth-first search limited to depth l does not find the goal, try it again with limit at l+1. Continue this until goal is found.
- Make depth-limited depth-first search complete by repeatedly applying it with greater values for the depth limit l.
- Feels like breadth-first search, in that a level is fully explored before extending the search to the next level. But, unlike breadth-first search, after one level is fully explored, all nodes already expanded are thrown away and the search starts with a clear memory.

Depth-First Iterative Deeping Search (DFID)
Limit = 0

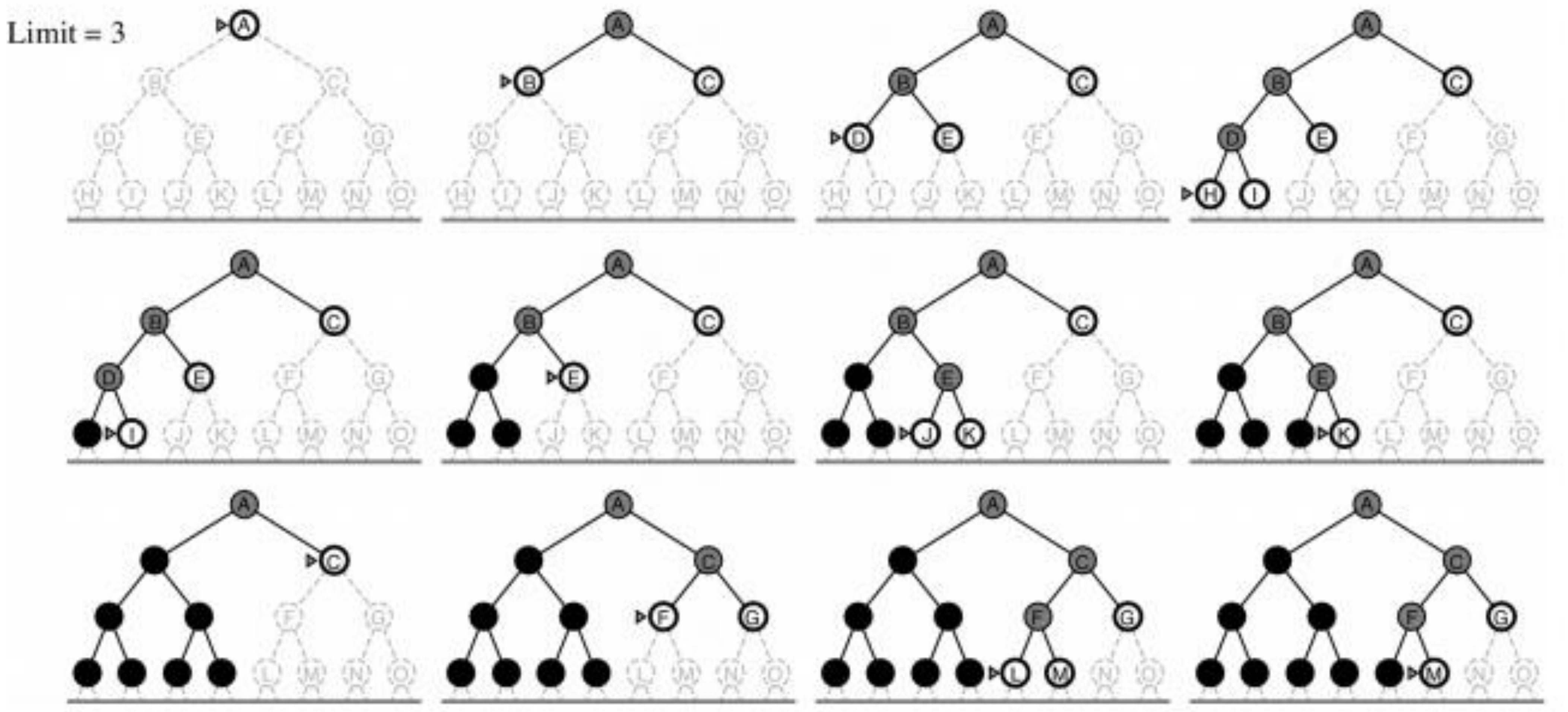
Depth-First Iterative Deeping Search (DFID)
Limit = 1



Depth-First Iterative Deeping Search (DFID) Limit = 2



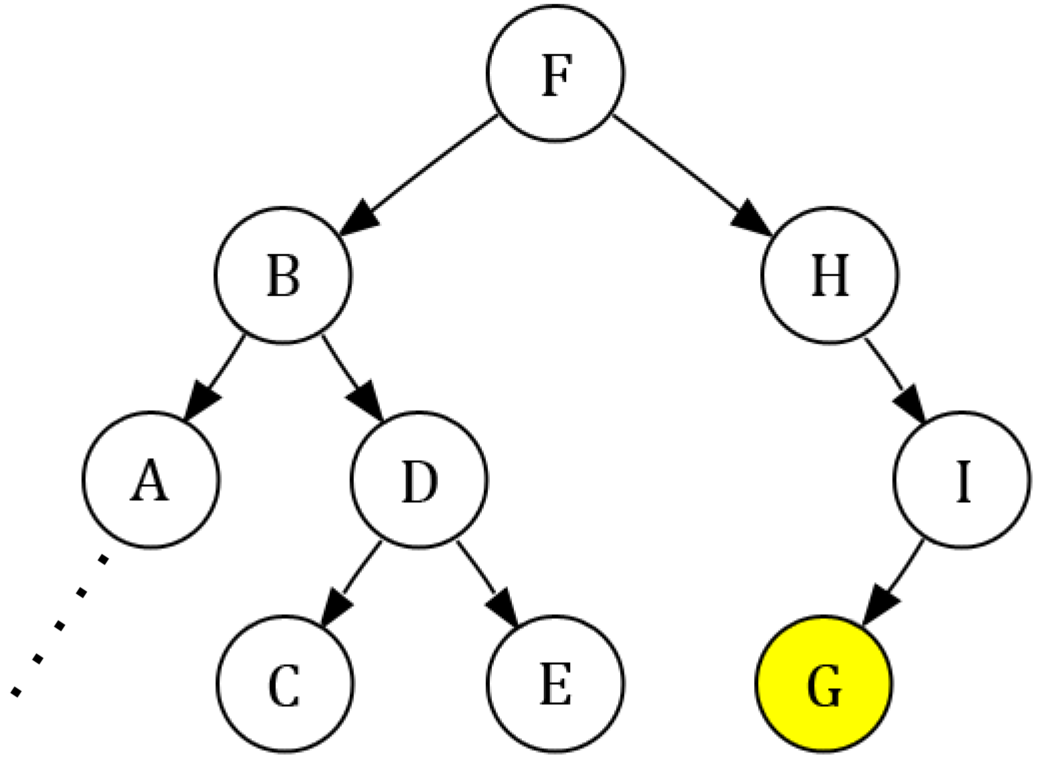
Depth-First Iterative Deeping Search (DFID) Limit = 3



Evaluating Search Algorithms

Depth First Search

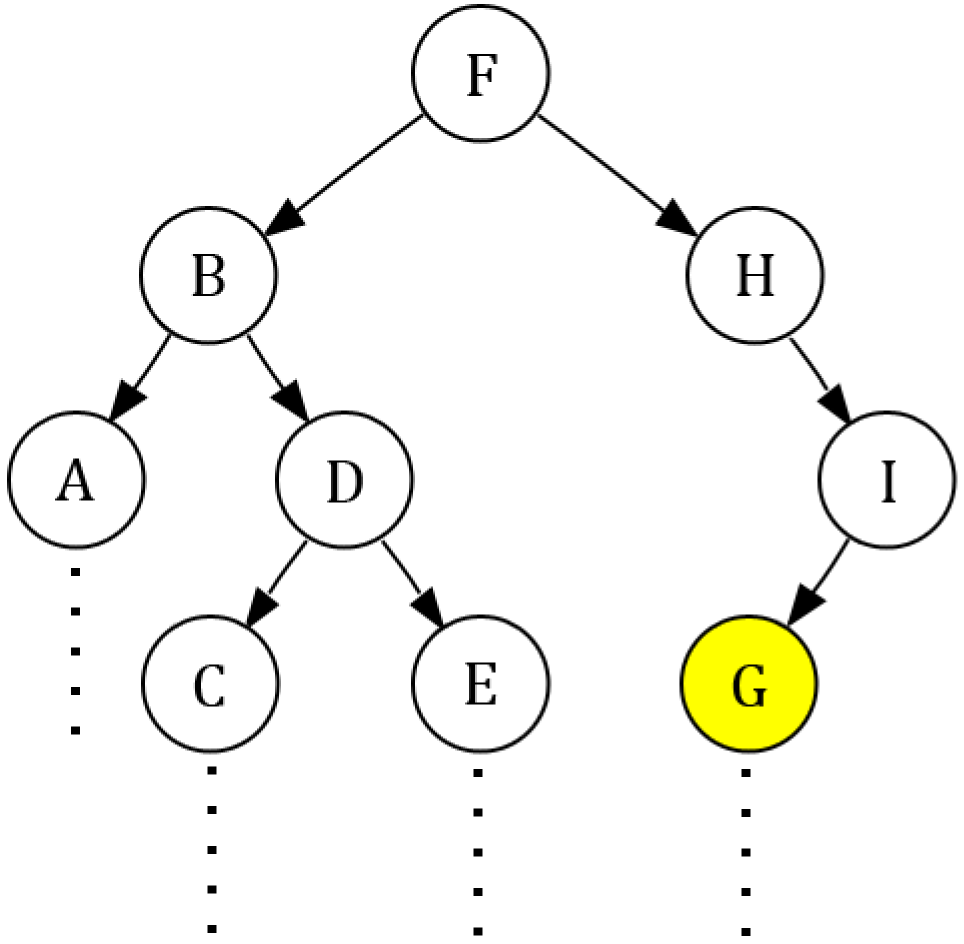
- If the first branch contains infinite nodes and the goal is G (at second branch).
- The DFS will not reach the goal because it will not be able to backtrack from first branch.
- **DFS is not complete** (It is not guaranteed to find the goal)



Evaluating Search Algorithms

Breadth First Search

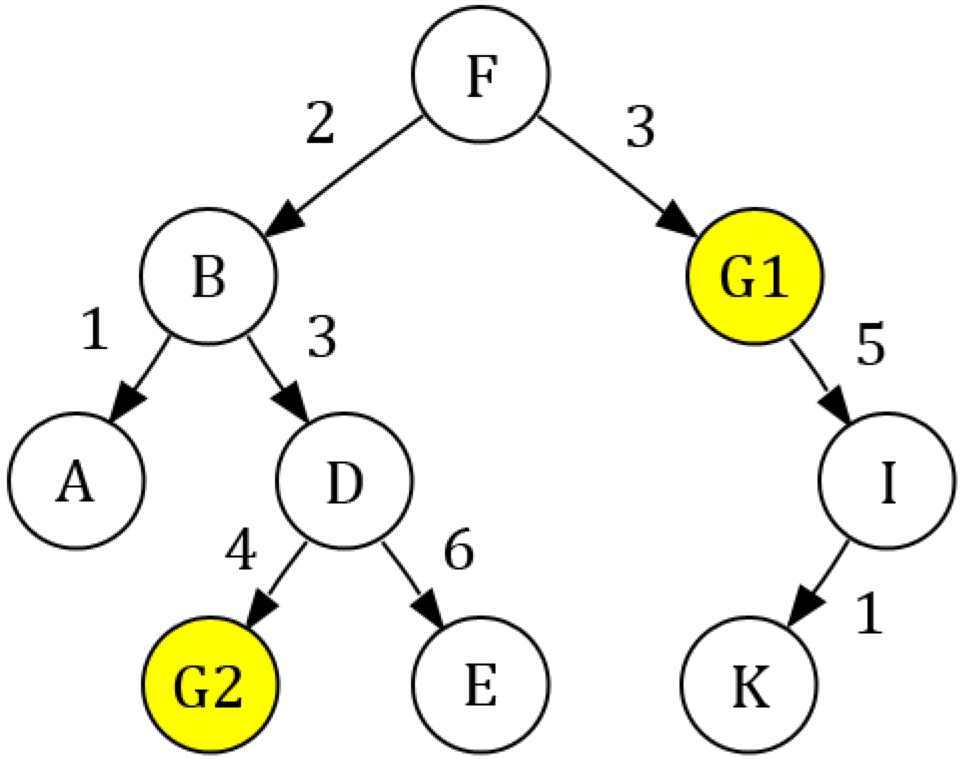
- If number of levels in the graph are infinite, but the goal is present in the solution.
- The BFS will always reach the goal.
- **BFS is complete** (It is guaranteed to find the goal, if it is present).



Evaluating Search Algorithms

Depth First Search

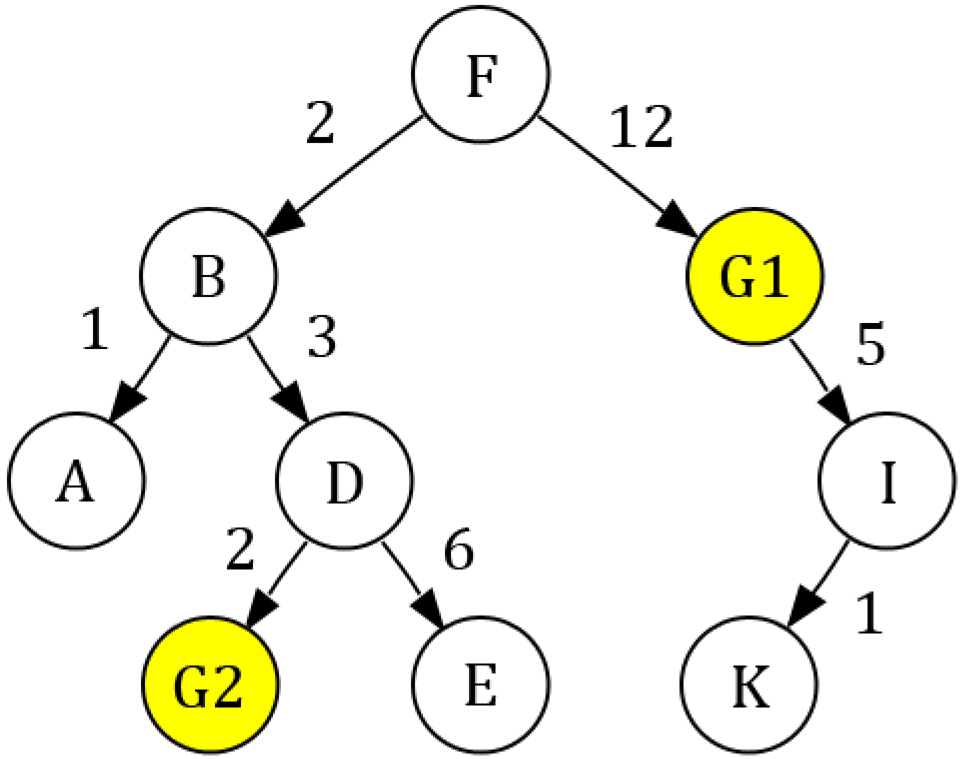
- If DFS is applied on this graph, it will find the goal G2.
- The path cost of G2 is 2+3+4=9.
- But there is another optimal goal present in the solution G1 having path cost 3.
- **DFS is not optimal** (It will not always return the optimal solution).



Evaluating Search Algorithms

Breadth First Search

- If BFS is applied on this graph, it will find the goal G1.
- The path cost of G1 is 12.
- But there is another optimal goal present in the solution G2 having path cost 2+3+2=7.
- **BFS is not optimal** (It will not always return the optimal solution).



Uniform Cost Search

- Breadth-first search finds the shallowest goal state, but this may not always be the least-cost solution for a general path cost function.
- Uniform cost search modifies the breadth-first strategy by always expanding the lowest-cost node rather than the lowest-depth node.
- It maintain a priority Queue.

Uniform Cost Search

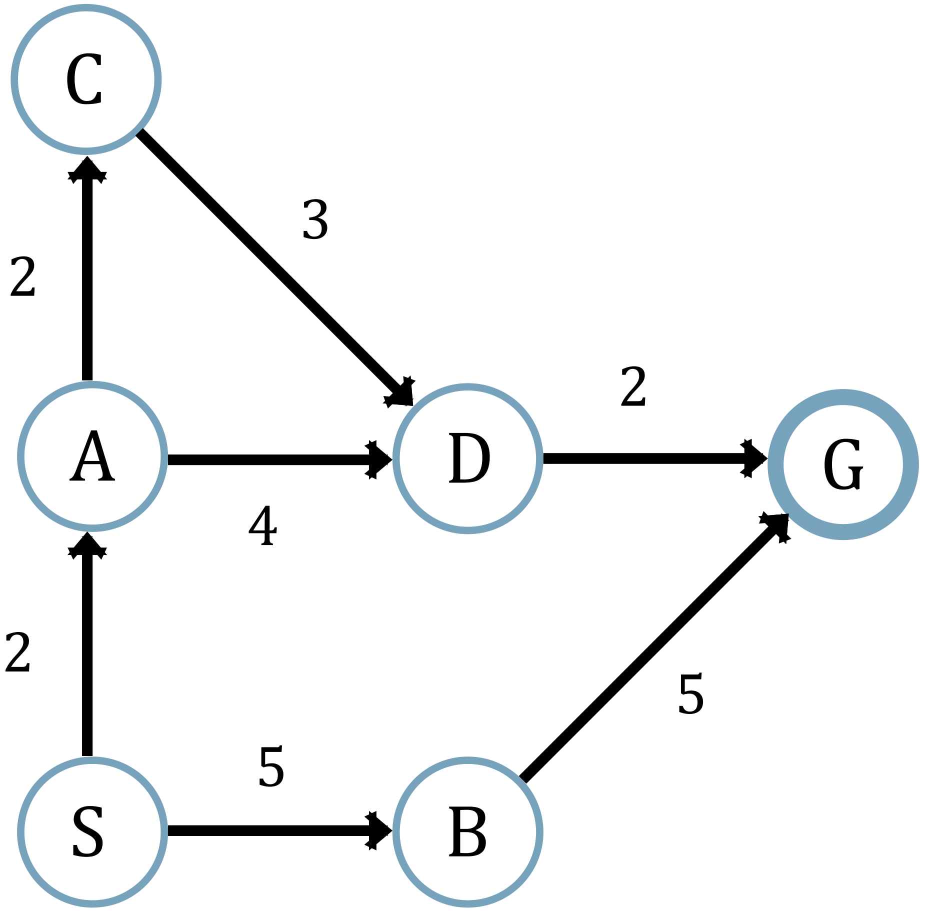
Algorithm:

- Step 01: Initialize Queue with Starting Node.
- Step 02: Pick minimum cost path from Queue.
- Step 03: If the minimum path is goal, stop algorithm you found the solution.
- Step 04: For each neighbor node v add expanded path < v ,P > to Queue.
- Step 05: Repeat Step 02 to Step 04 until Queue is empty

Uniform Cost Search (Problem 01)

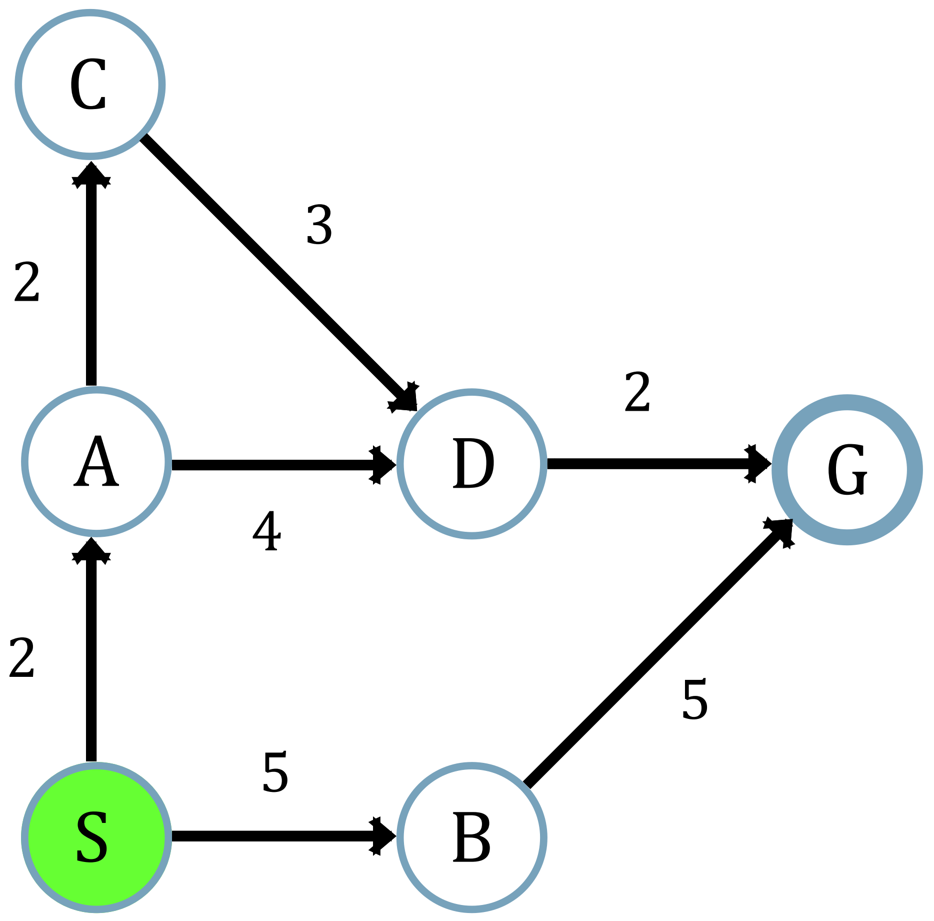
- S = Initial State
- G = Goal State

Queue	
Path	**Cost**



Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<S>	0



From the path we have:
<S> = 0

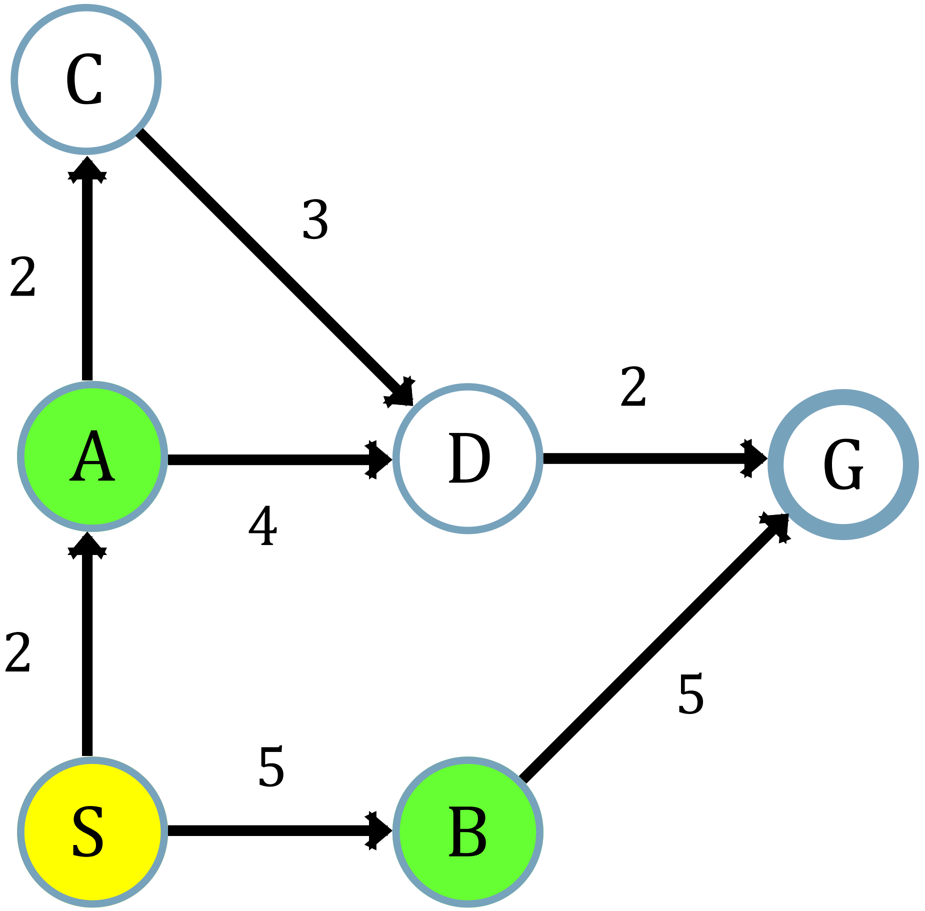
<A,S> = 2
<B,S> = 5

Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<A,S>	2
<B,S>	5

From the path we have:
<A,S> = 2

<C,A,S> = 4
<D,A,S> = 6

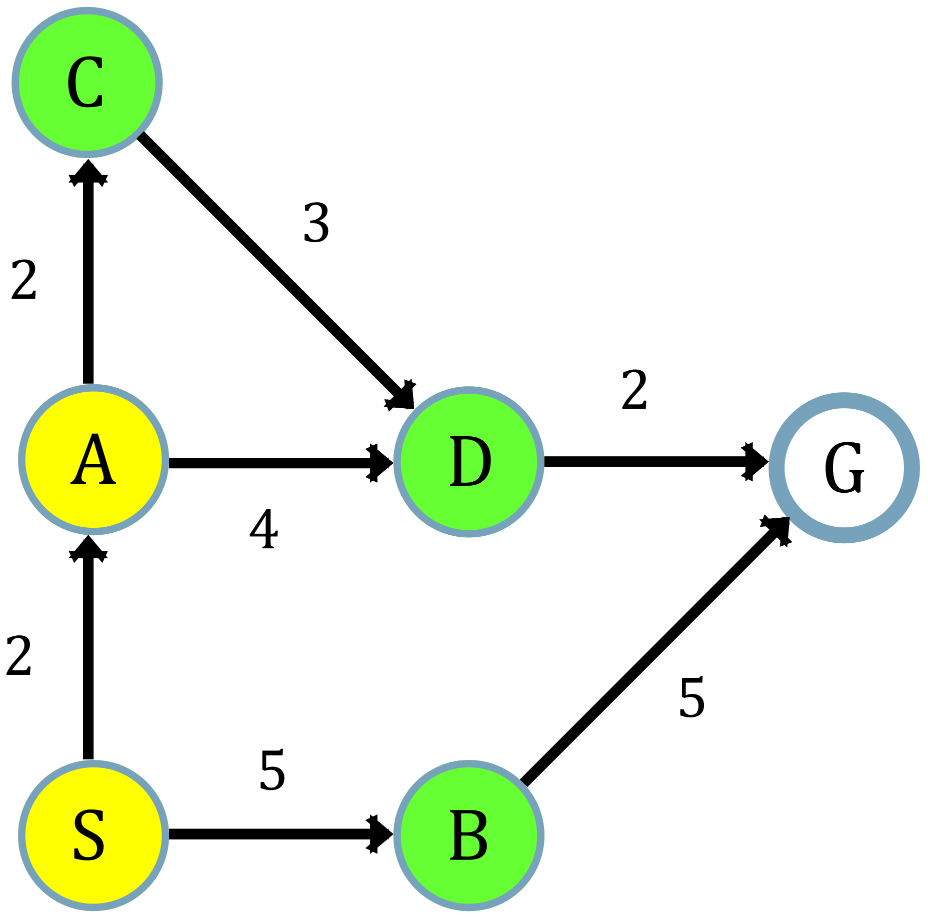


Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<C,A,S>	4
<B,S>	5
<D,A,S>	6

From the path we have:
<C,A,S> = 4

<D,C,A,S> = 7

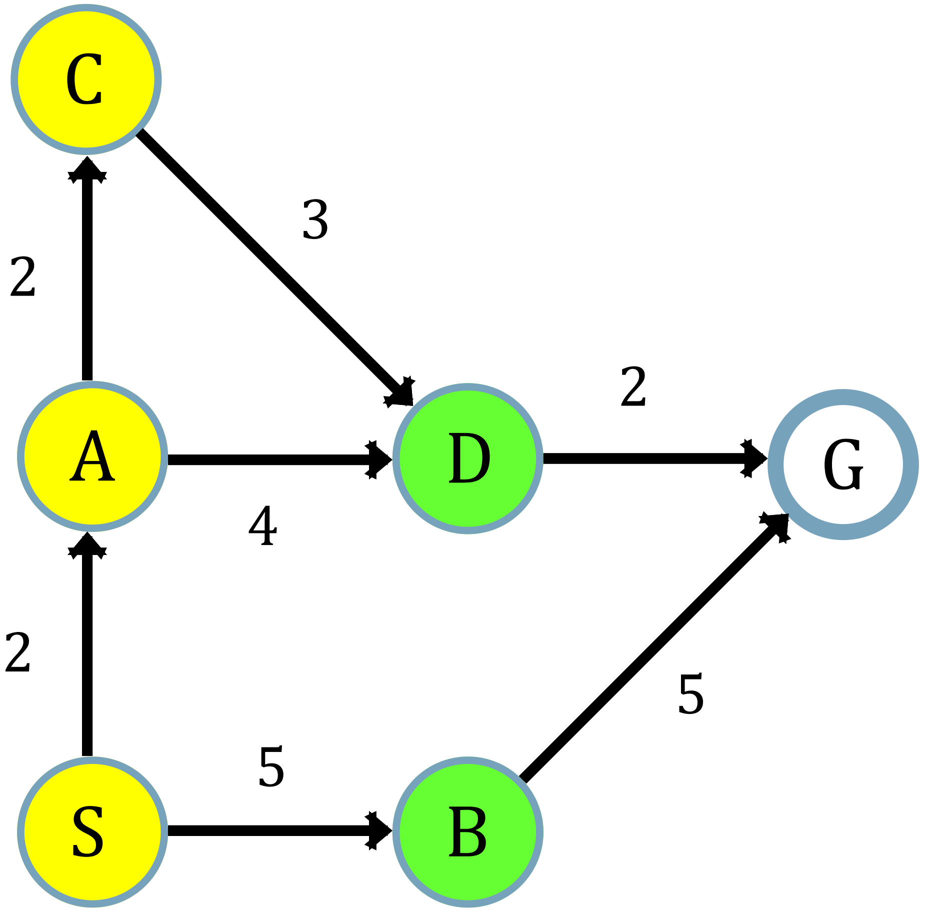


Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<B,S>	5
<D,A,S>	6
<D,C,A,S>	7

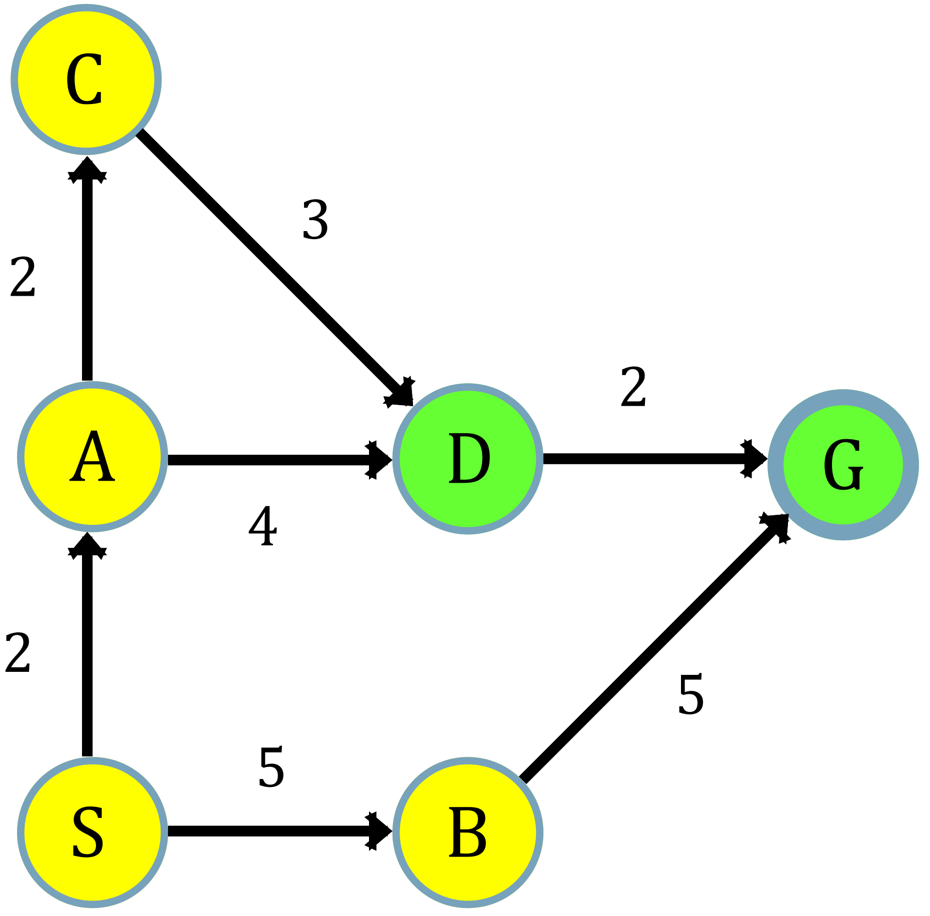
From the path we have:
<B,S> = 5

<G,B,S> = 10



Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<D,A,S>	6
<D,C,A,S>	7
<G,B,S>	10

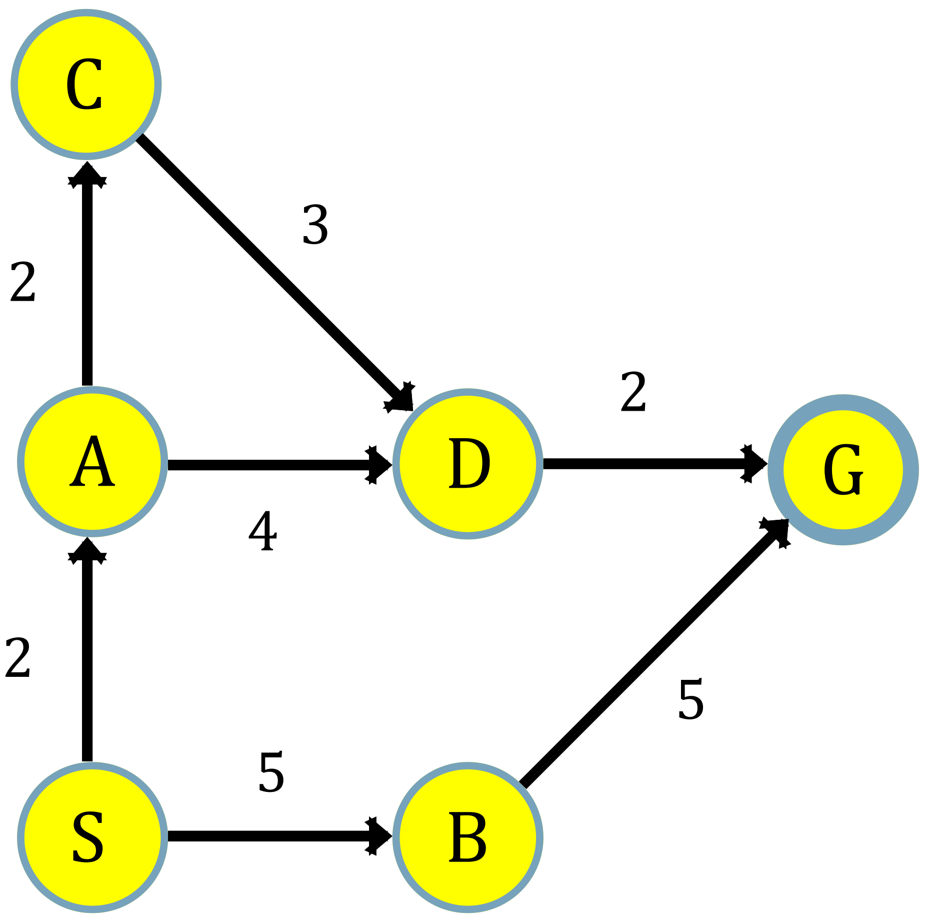

$$\begin{tikzcd}
C \arrow[dr, "3"] & & \\
A \arrow[u, "2"] \arrow[r, "4"'] & D \arrow[r, "2"] & G \\
S \arrow[u, "2"] \arrow[r, "5"] & B \arrow[ur, "5"'] &
\end{tikzcd}$$

From the path we have:
<D,A,S> = 6

<G,D,A,S> = 8

Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<D,C,A,S>	7
<G,D,A,S>	8
<G,B,S>	10

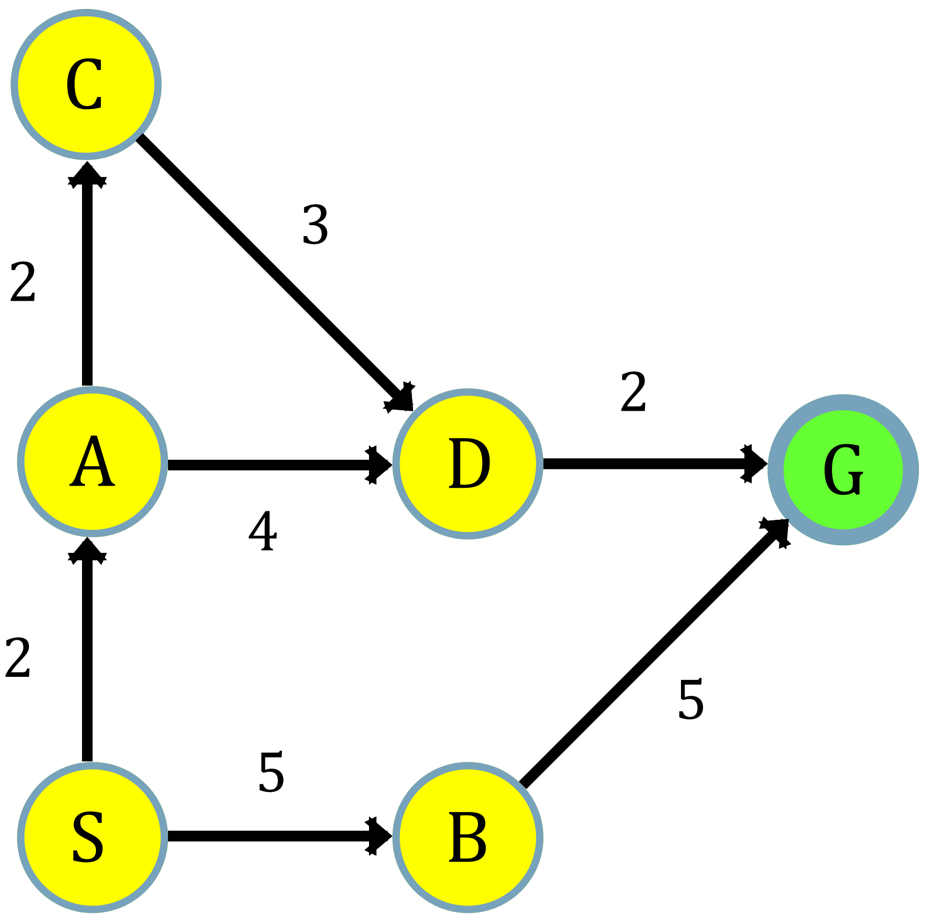


From the path we have:
<D,C,A,S> = 7

<G,D,C,A,S> = 9

Uniform Cost Search (Problem 01)

Queue	
Path	**Cost**
<G,D,A,S>	8
<G,D,C,A,S>	9
<G,B,S>	10



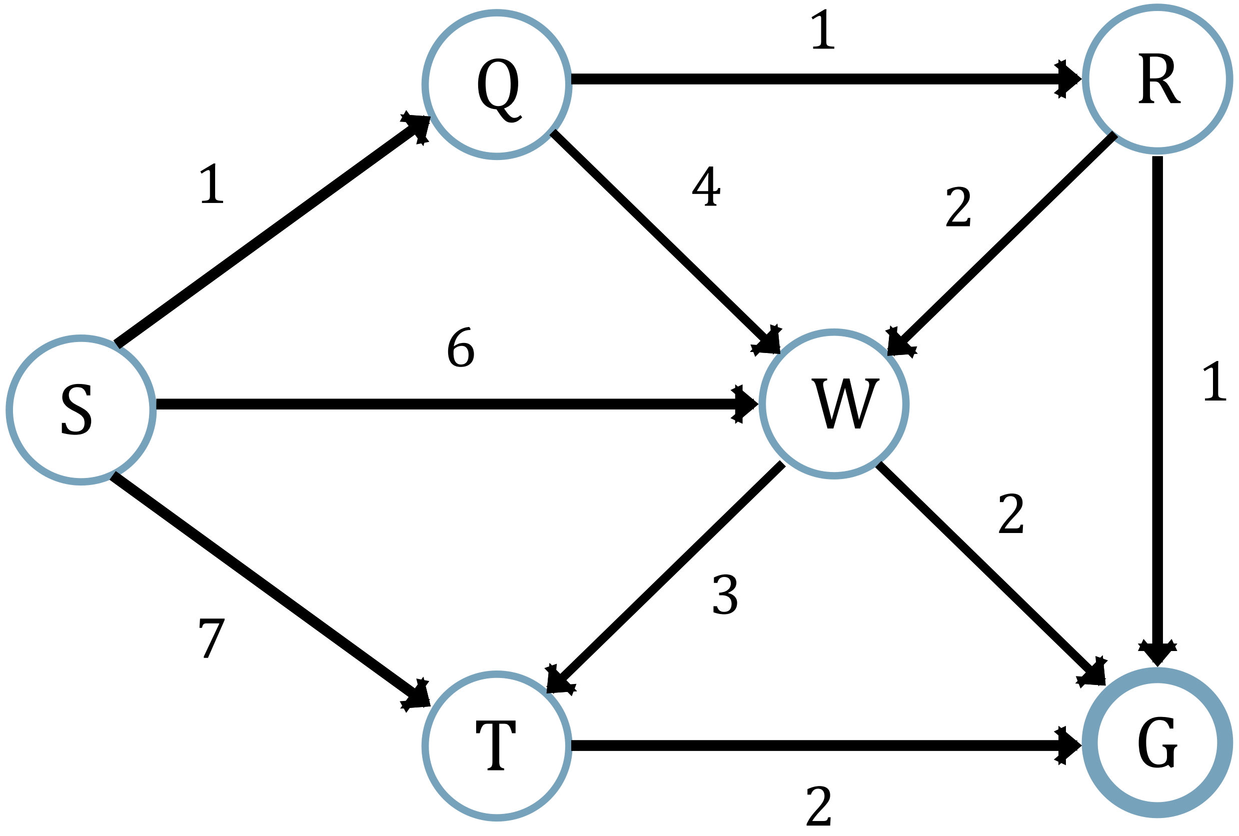
Shortest Path = Solution = S → A → D → G

As minimum path is the goal path, so we have found a solution.

Uniform Cost Search (Problem 02)

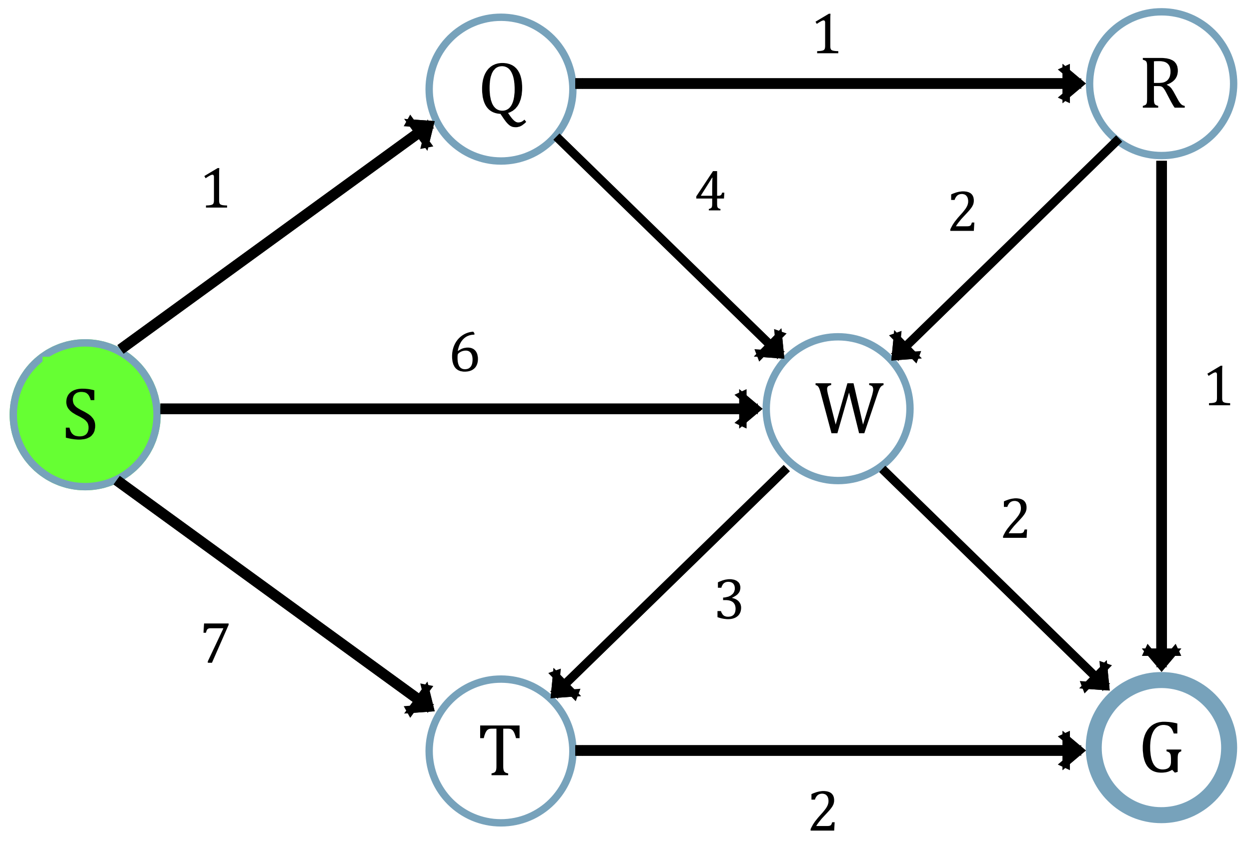
- S = Initial State
- G = Goal State

Queue	
Path	**Cost**



Uniform Cost Search (Problem 02)

Queue	
Path	**Cost**
<S>	0



From the path we have:
<S> = 0

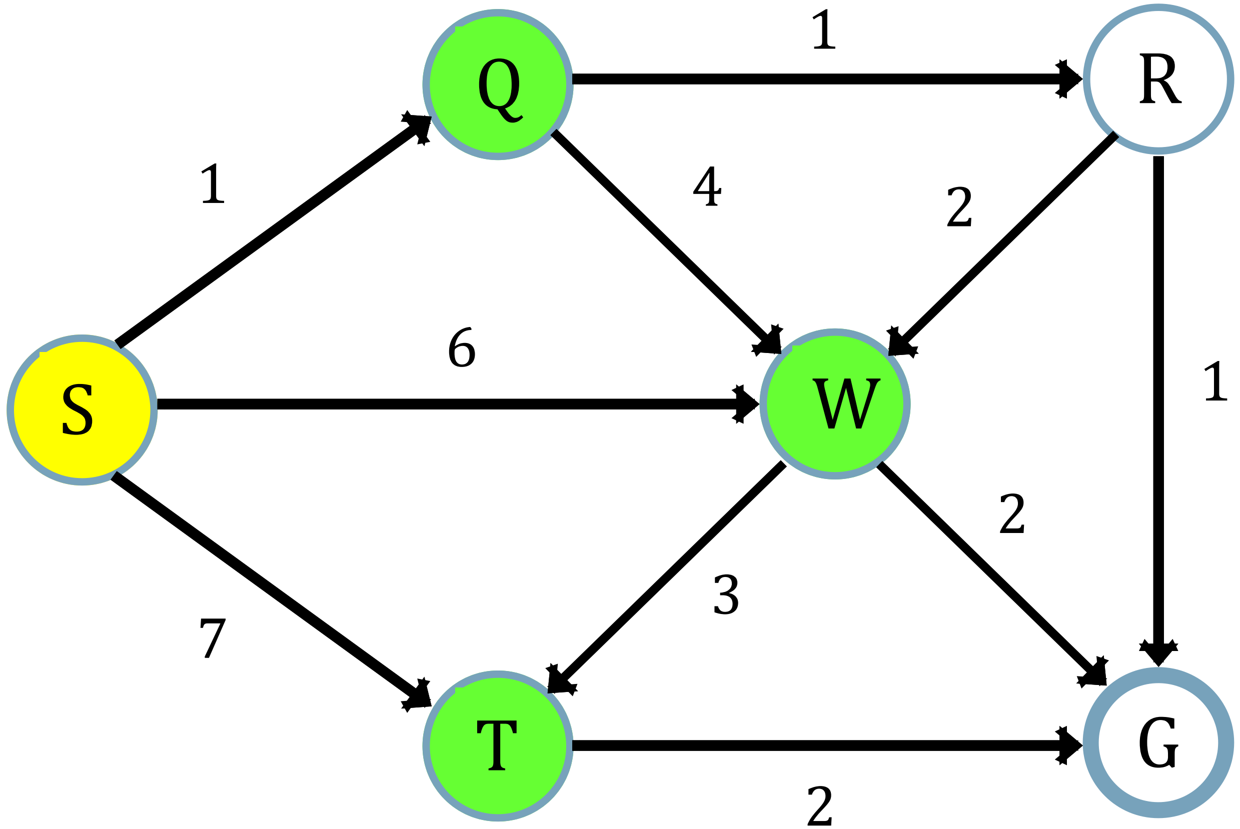
<Q,S> = 1
<W,S> = 6
<T,S> = 7

Uniform Cost Search (Problem 02)

Queue	
Path	**Cost**
<Q,S>	1
<W,S>	6
<T,S>	7

From the path we have:
<Q,S> = 1

<R,Q,S> = 2
<W,Q,S> = 5

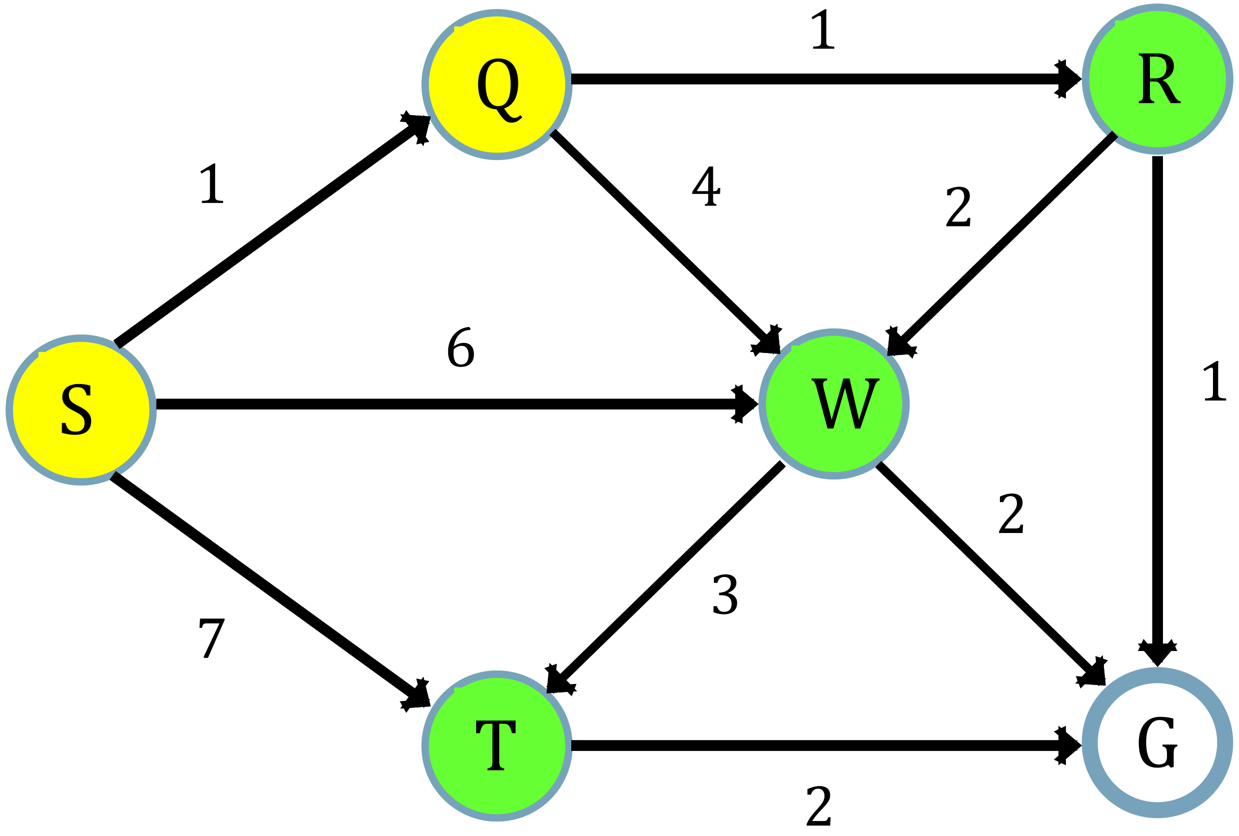


Uniform Cost Search (Problem 02)

Queue	
Path	**Cost**
<R,Q,S>	2
<W,Q,S>	5
<W,S>	6
<T,S>	7

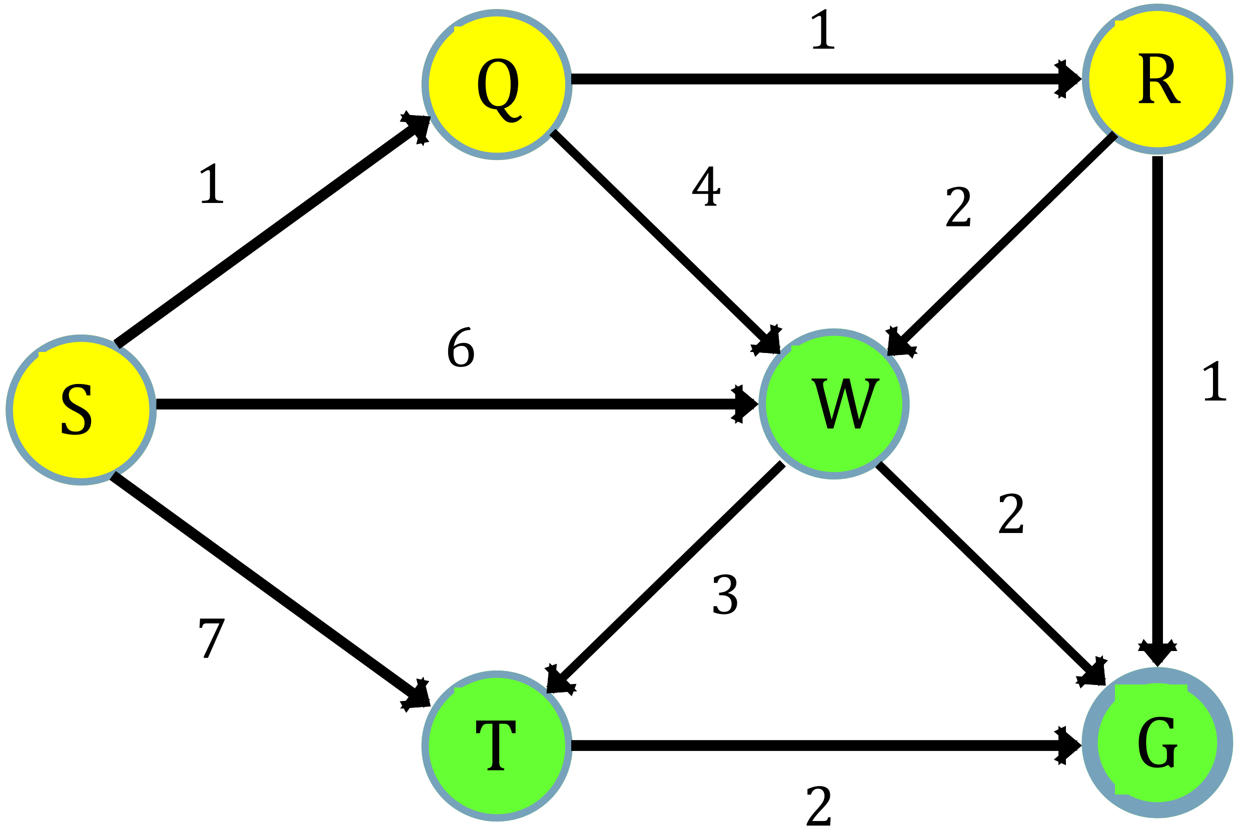
From the path we have:
<R,Q,S> = 2

<W,R,Q,S> = 4
<G,R,Q,S> = 3



Uniform Cost Search (Problem 02)

Queue	
Path	**Cost**
<G,R,Q,S>	3
<W,R,Q,S>	4
<W,Q,S>	5
<W,S>	6
<T,S>	7



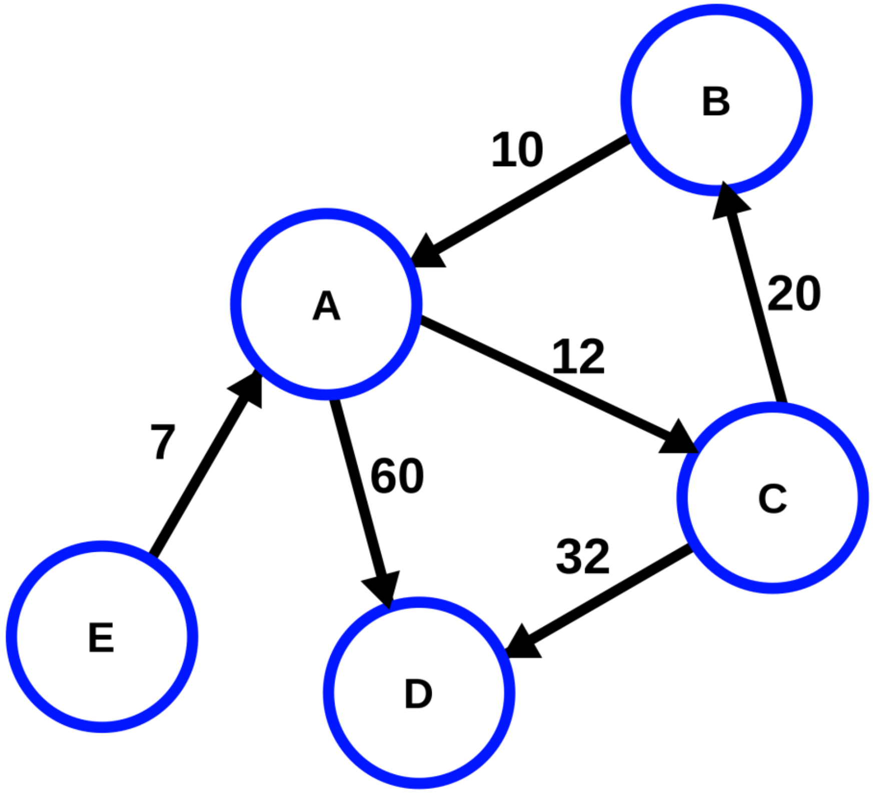
Shortest Path = Solution =
S → Q → R → G

As minimum path is the goal path, so we have found a solution.

Uniform Cost Search (Problem 03)

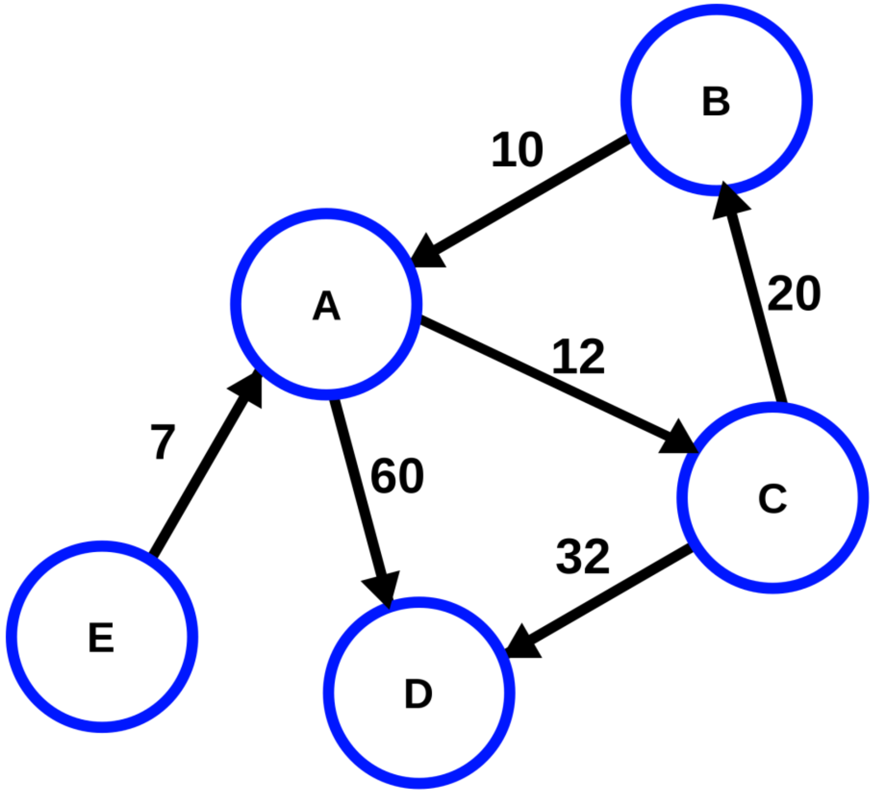
- B = Initial State
- D = Goal State

Queue	
Path	**Cost**



Uniform Cost Search (Problem 03)

Queue	
Path	**Cost**
<B>	0

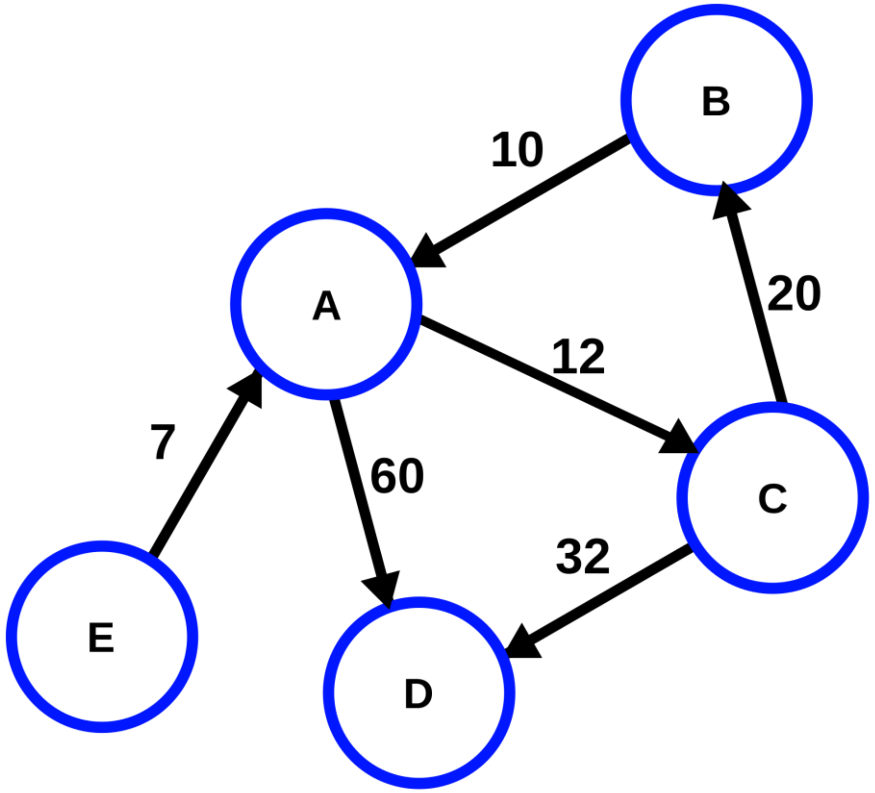


From the path we have:
<B> = 0

<A,B> = 10

Uniform Cost Search (Problem 03)

Queue	
Path	**Cost**
<A,B>	10

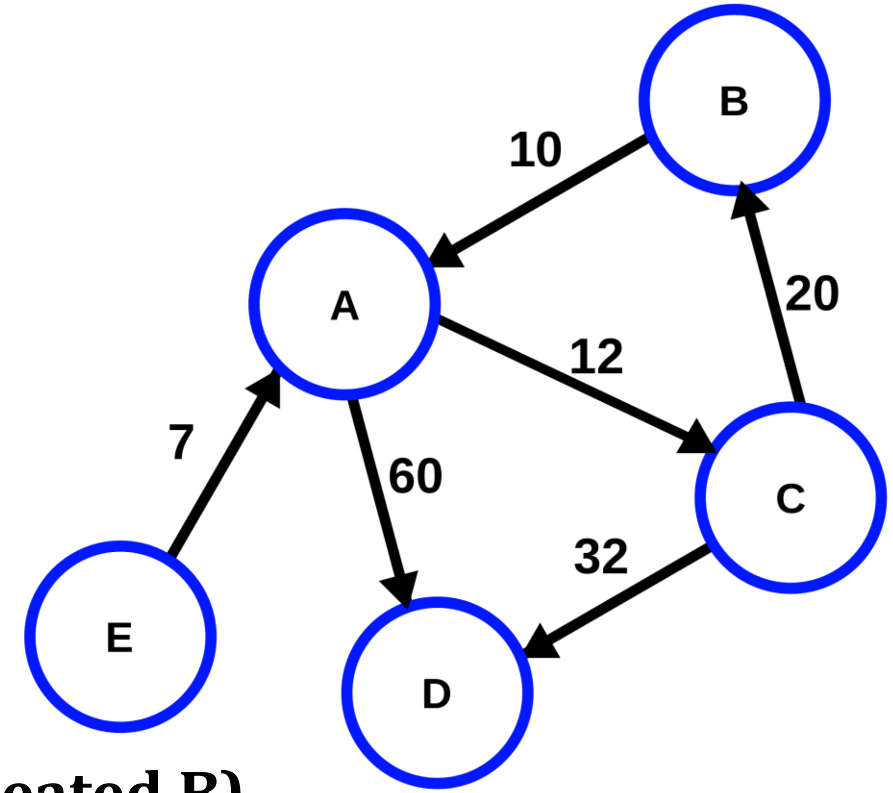


From the path we have:
<A,B> = 10

<C,A,B> = 22
<D,A,B> = 70

Uniform Cost Search (Problem 03)

Queue	
Path	**Cost**
<C,A,B>	22
<D,A,B>	70

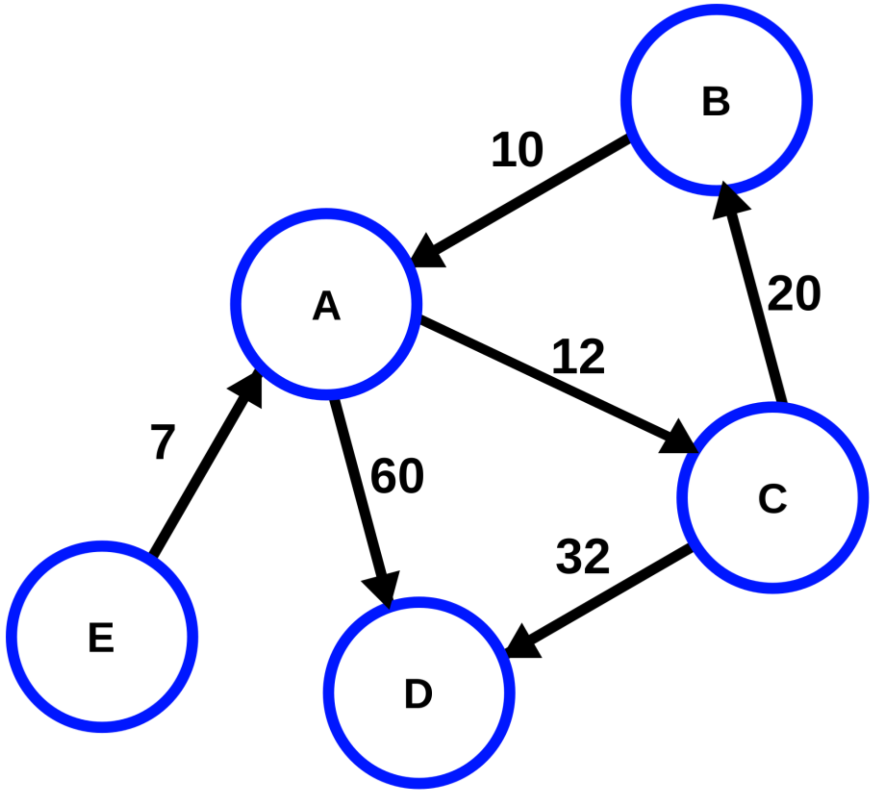


From the path we have:
<C,A,B> = 22

<B,C,A,B> = 42 (It is a cycle as repeated B)
<D,C,A,B> = 54
Avoid Cycles (Discard them)

Uniform Cost Search (Problem 03)

Queue	
Path	**Cost**
<D,C,A,B>	54
<D,A,B>	70



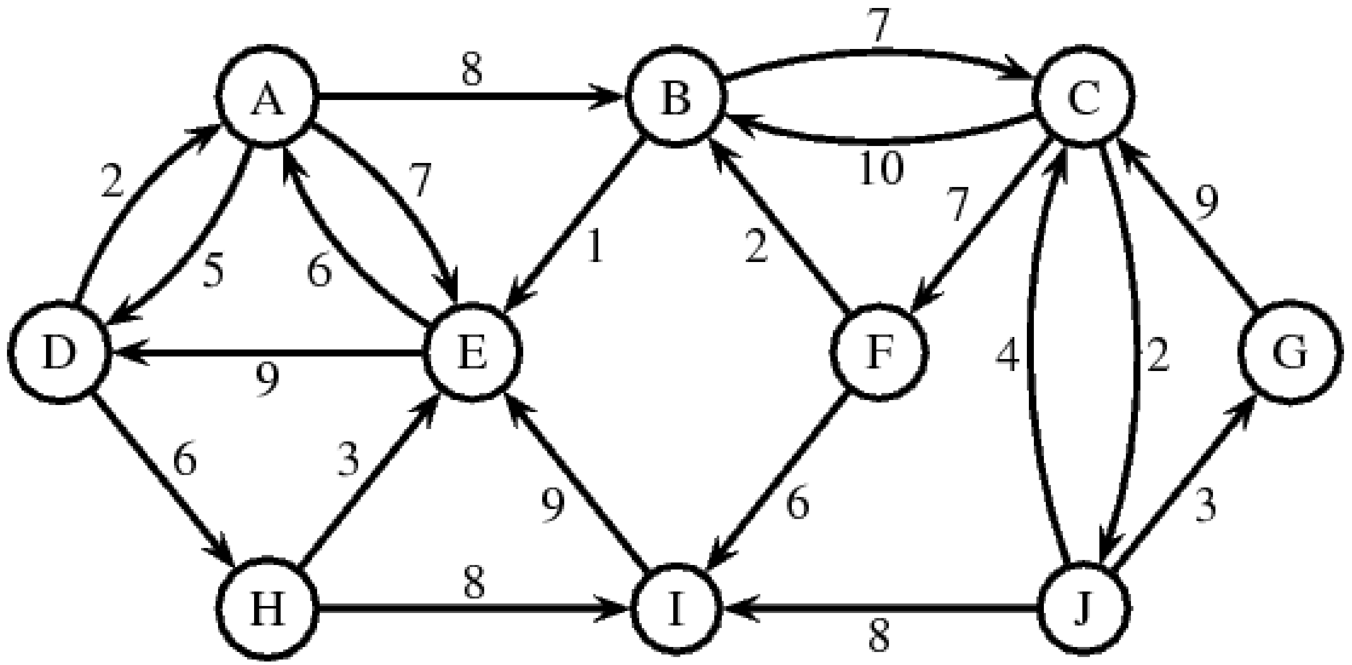
Shortest Path = Solution =
B → A → C → D

As minimum path is the goal path, so we have found a solution.

Uniform Cost Search (Problem 04)

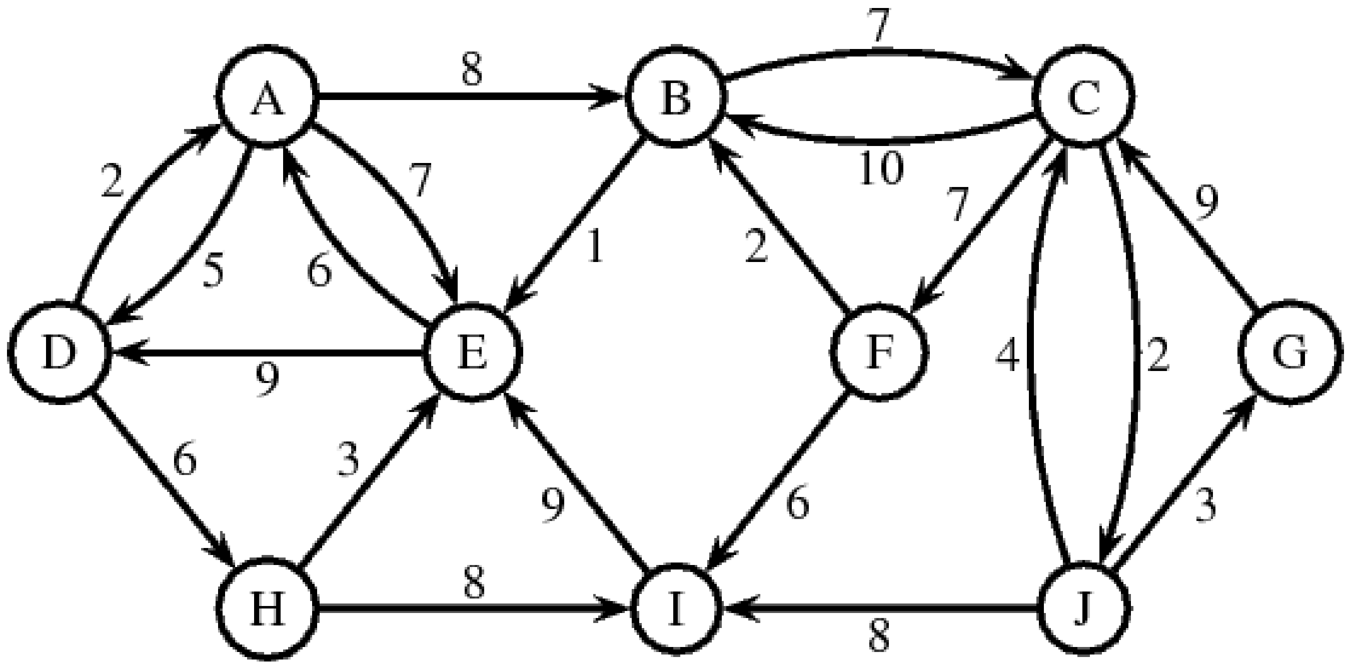
- D = Initial State
- G = Goal State

Queue	
Path	**Cost**



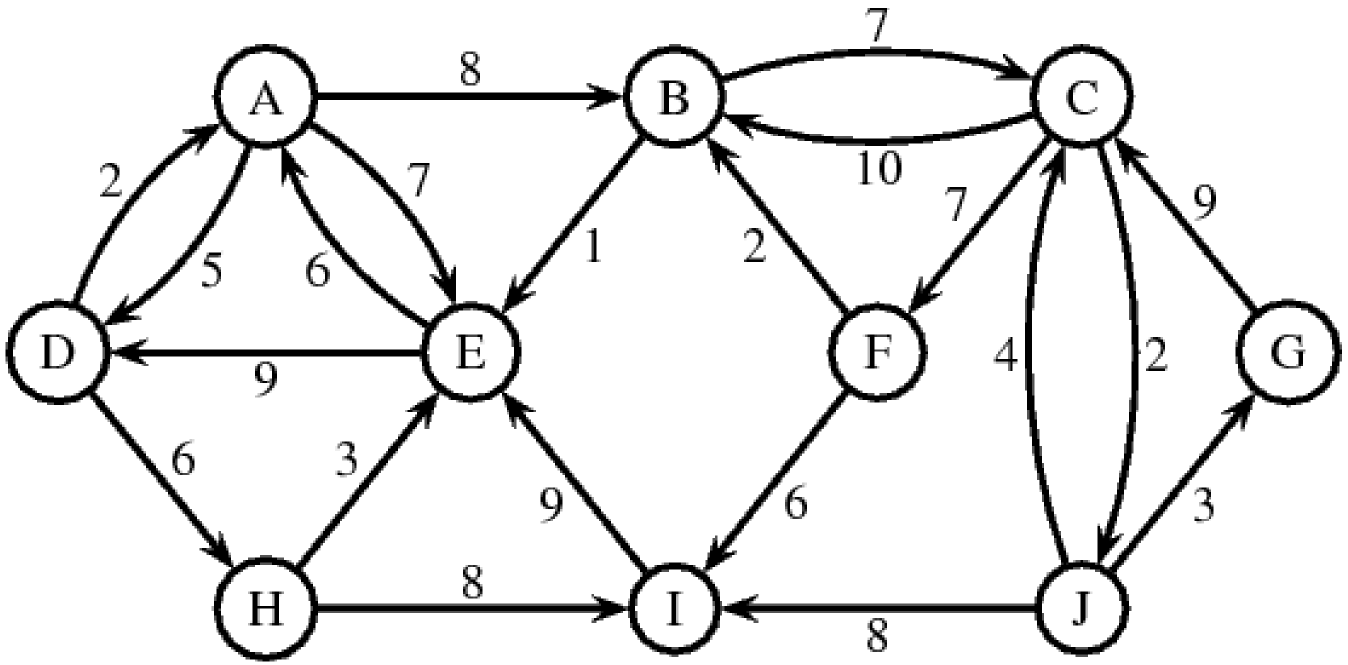
Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<D>	0



Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<A,D>	2
<H,D>	6

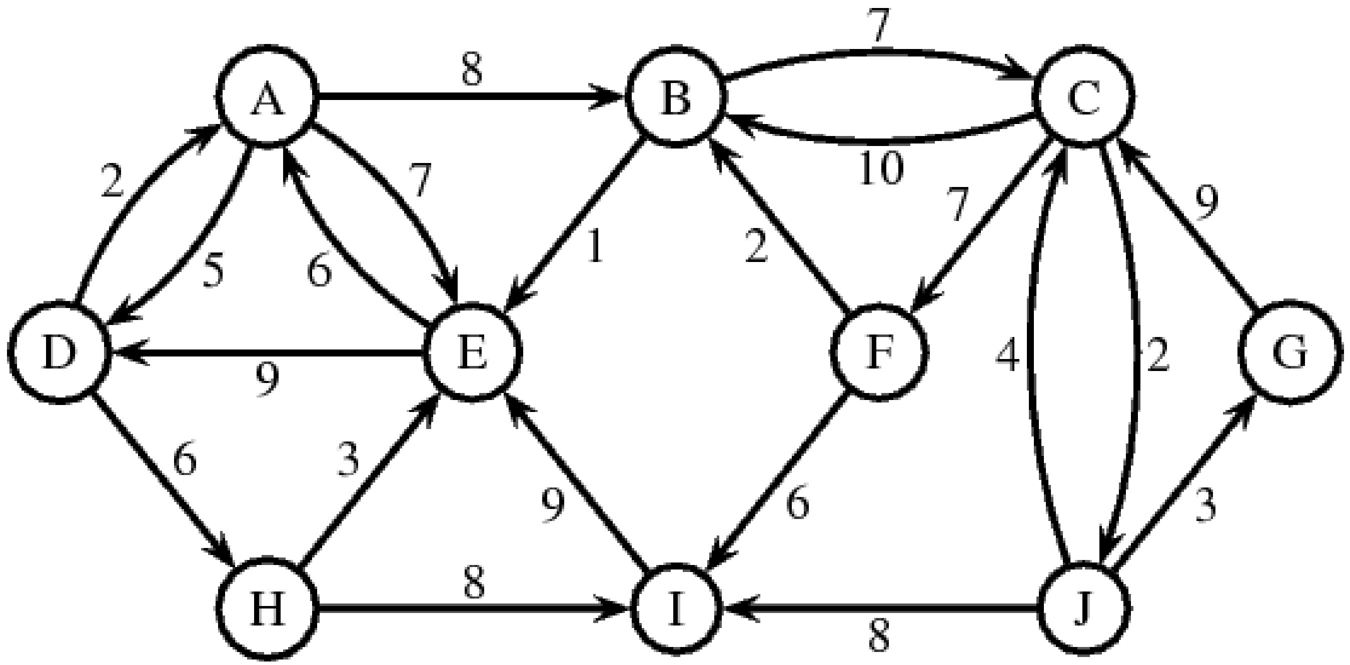


From the path we have:
<A,D> = 2

<B,A,D> = 10
<D,A,D> = 7 (It is a cycle as repeated D)
<E,A,D> = 9
Avoid Cycles (Discard them)

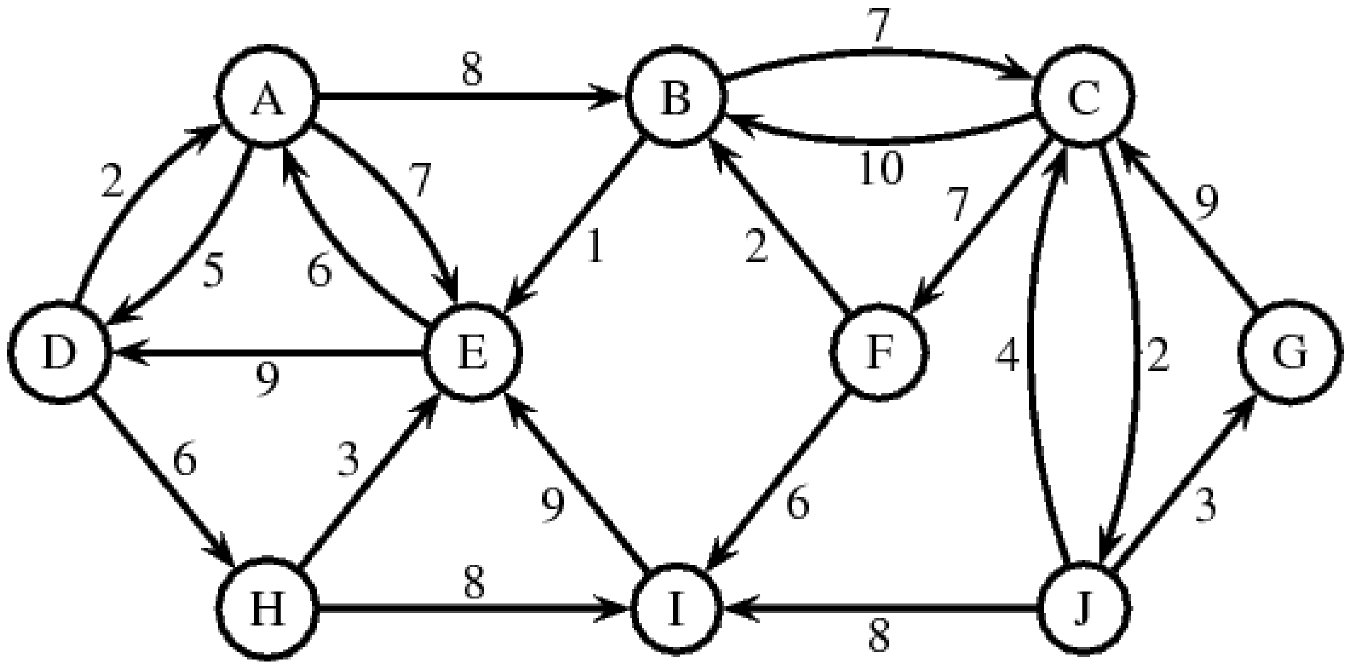
Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<H,D>	6
<E,A,D>	9
<B,A,D>	10



Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<E,H,D>	9
<E,A,D>	9
<B,A,D>	10
<I,H,D>	14

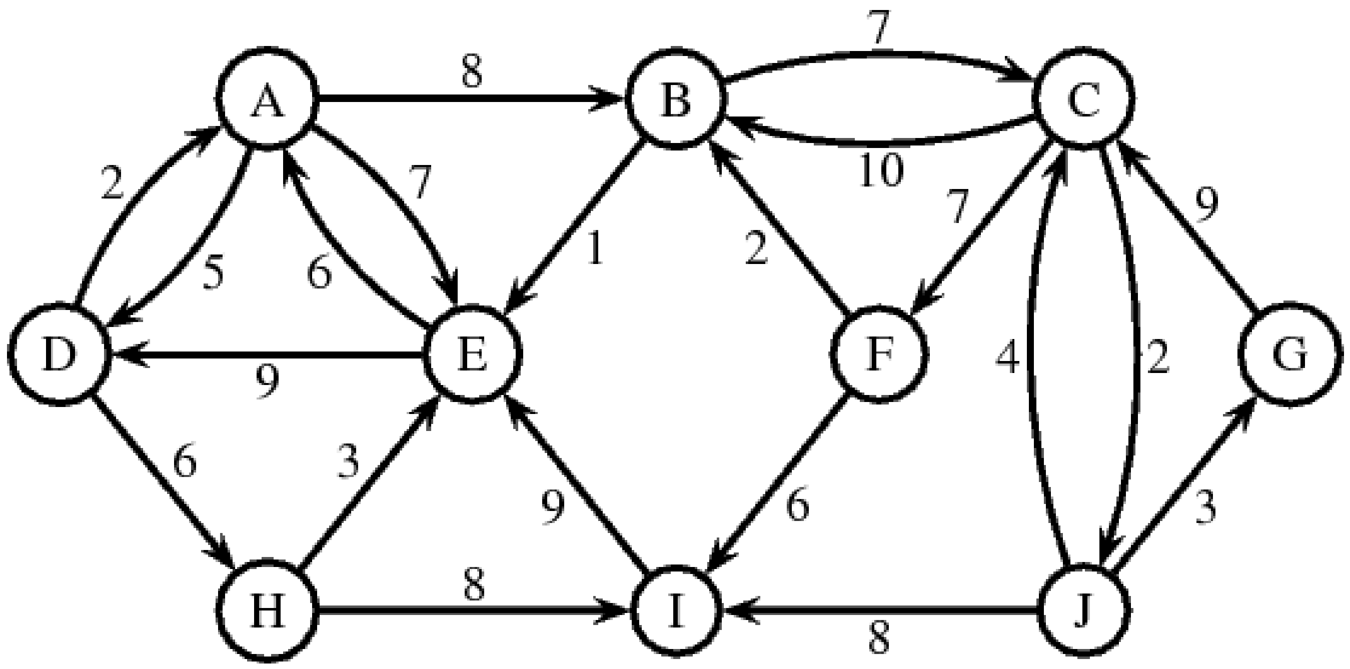


From the path we have:
<E,H,D> = 9

<A,E,H,D> = 15
<D,E,H,D> = 18 (It is a cycle as repeated D)
Avoid Cycles (Discard them)

Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<E,A,D>	9
<B,A,D>	10
<I,H,D>	14
<A,E,H,D>	15

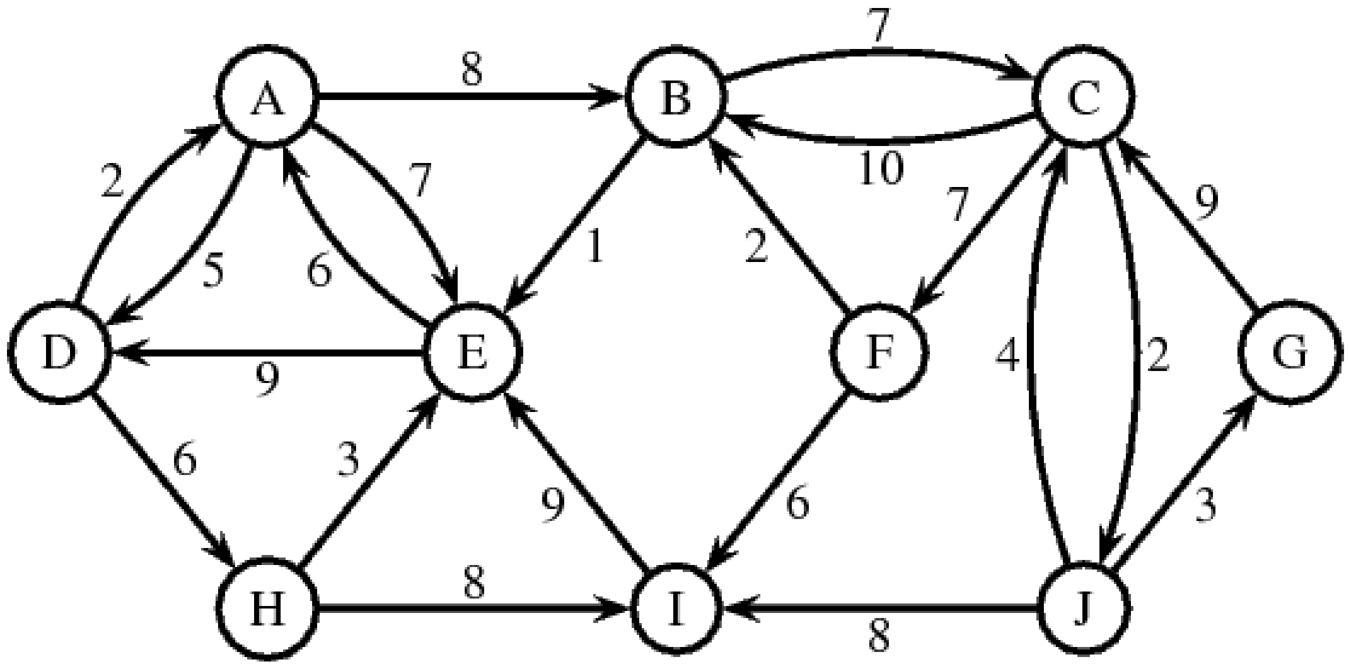


From the path we have:
<E,A,D> = 9

<A,E,A,D> = 15 (Cycle)
<D,E,A,D> = 18 (Cycle)
Avoid Cycles (Discard them)

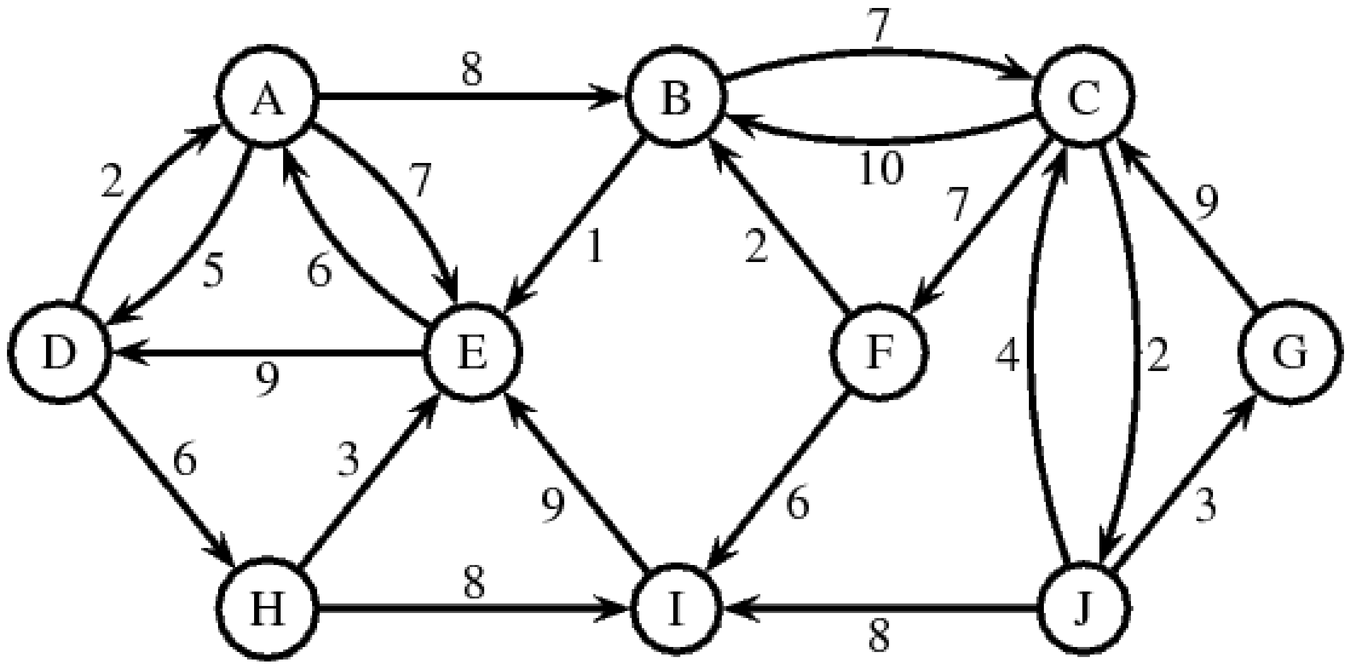
Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<E,B,A,D>	11
<I,H,D>	14
<A,E,H,D>	15
<C,B,A,D>	17



Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<E,B,A,D>	11
<I,H,D>	14
<A,E,H,D>	15
<C,B,A,D>	17

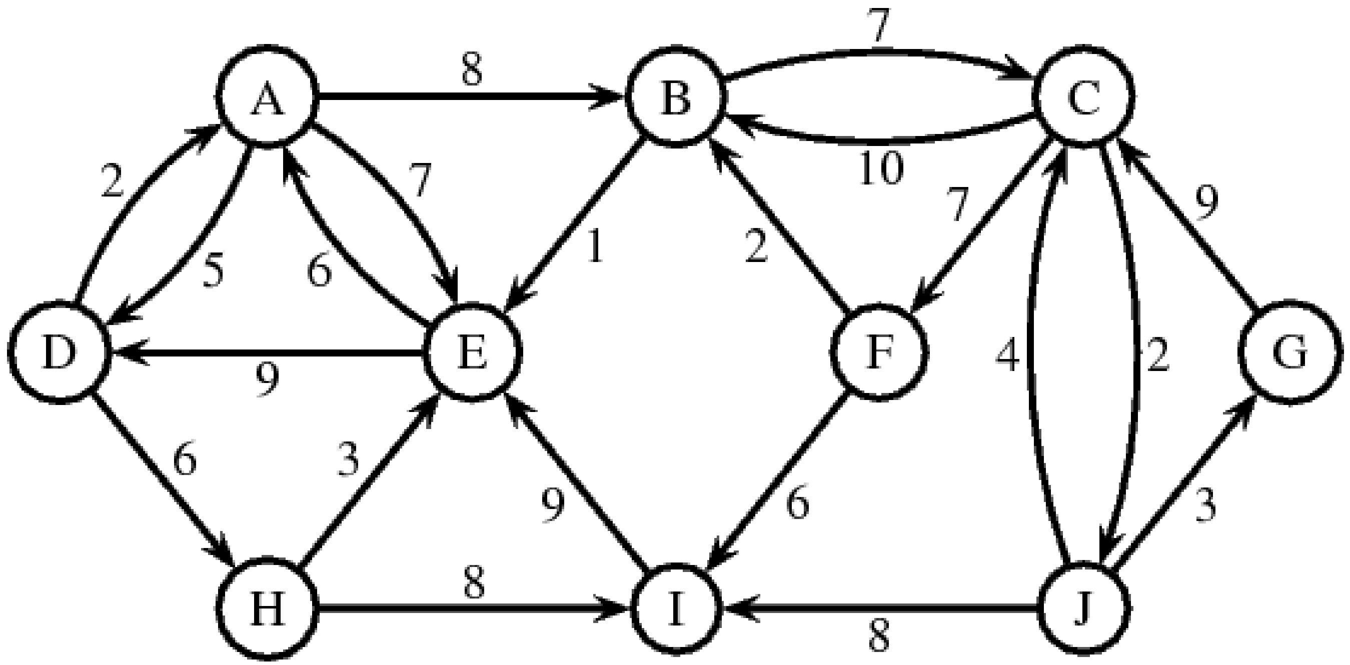


From the path we have:
<E,B,A,D> = 11

<A,E,B,A,D> = 18 (Cycle)
<D,E,B,A,D> = 20 (Cycle)
Avoid Cycles (Discard them)

Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<I,H,D>	14
<A,E,H,D>	15
<C,B,A,D>	17

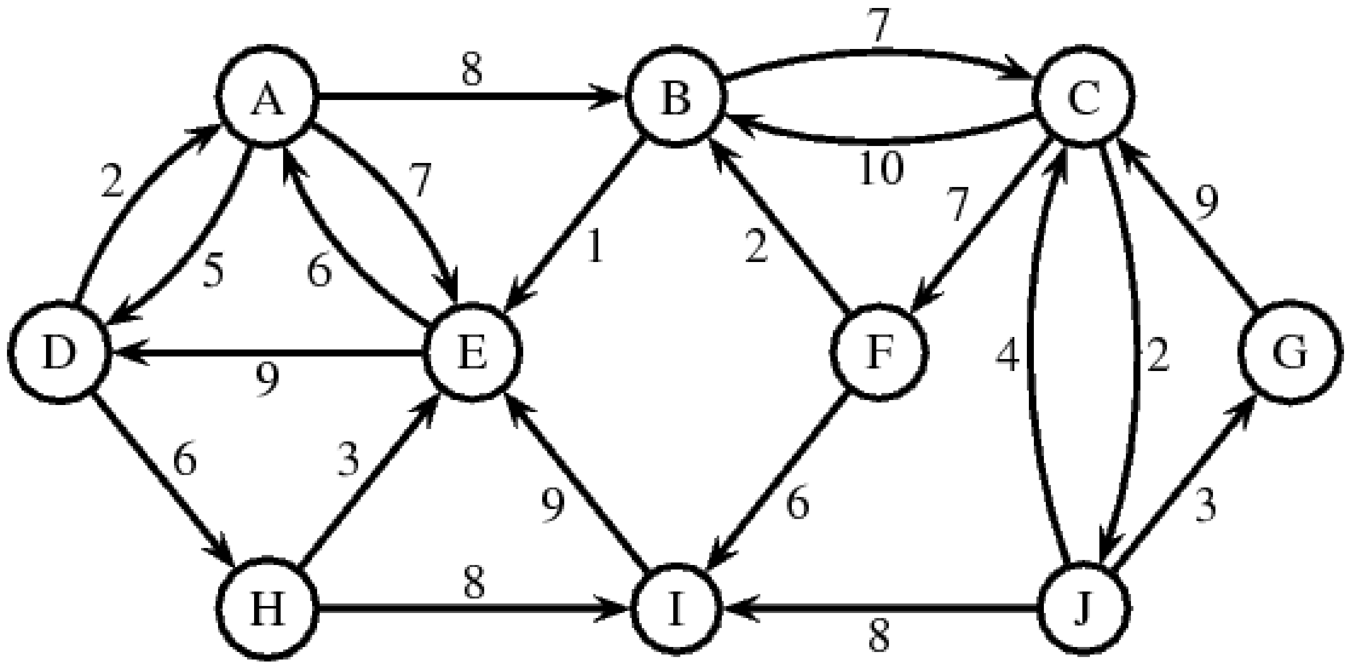


Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<A,E,H,D>	15
<C,B,A,D>	17
<E,I,H,D>	23

From the path we have:
<A,E,H,D> = 15

<D,A,E,H,D> = 20 (Cycle)
<E,A,E,H,D> = 22 (Cycle)
<B,A,E,H,D> = 23
Avoid Cycles (Discard them)

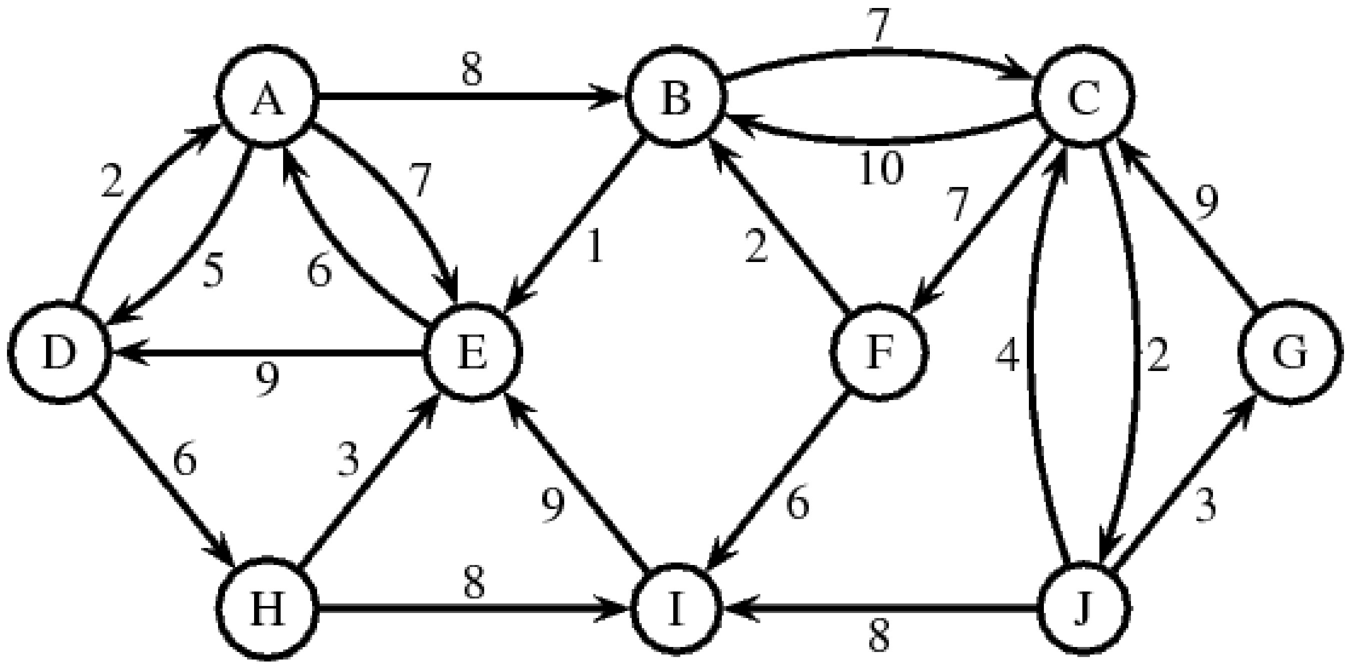


Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<C,B,A,D>	17
<E,I,H,D>	23
<B,A,E,H,D>	23

From the path we have:
<C,B,A,D> = 17

<B,C,B,A,D> = 27 (Cycle)
<F,C,B,A,D> = 24
<J,C,B,A,D> = 19
Avoid Cycles (Discard them)

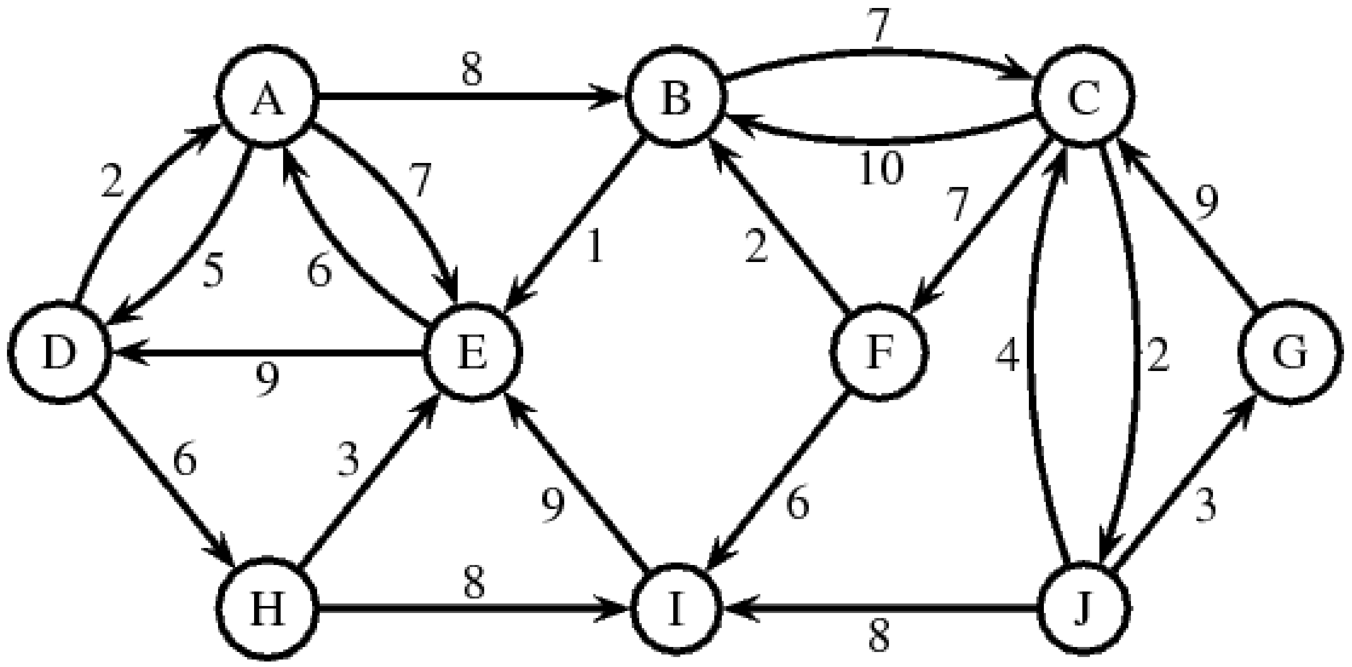


Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<J,C,B,A,D>	19
<E,I,H,D>	23
<B,A,E,H,D>	23
<F,C,B,A,D>	24

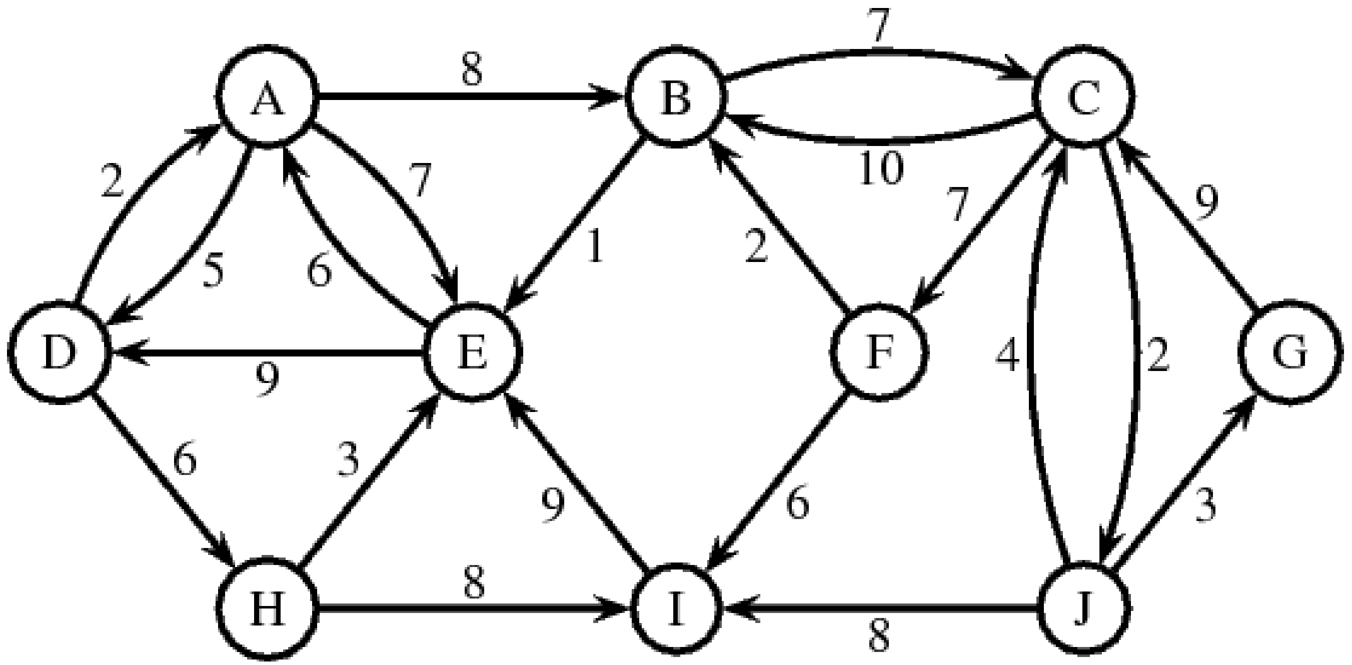
From the path we have:
<J,C,B,A,D> = 19

<I,J,C,B,A,D> = 27
<C,J,C,B,A,D> = 23 (Cycle)
<G,J,C,B,A,D> = 22
Avoid Cycles (Discard them)



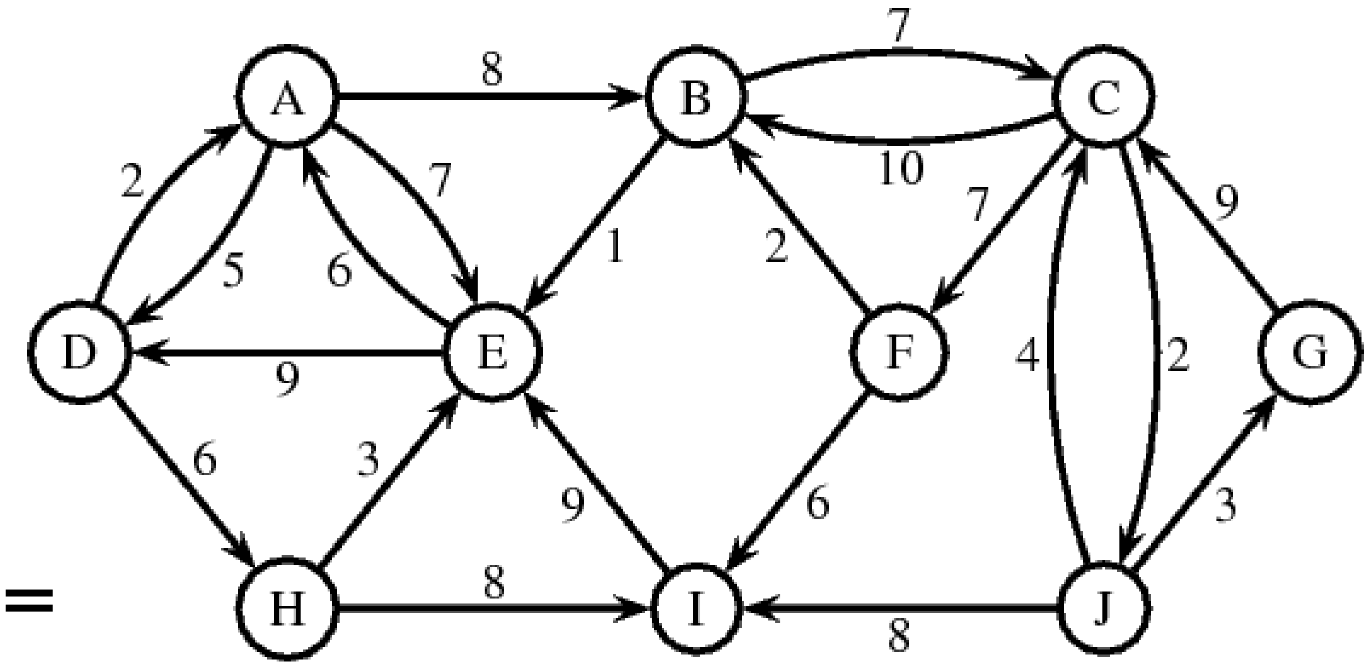
Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<G,J,C,B,A,D>	22
<E,I,H,D>	23
<B,A,E,H,D>	23
<F,C,B,A,D>	24
<I,J,C,B,A,D>	27



Uniform Cost Search (Problem 04)

Queue	
Path	**Cost**
<G,J,C,B,A,D>	22
<E,I,H,D>	23
<B,A,E,H,D>	23
<F,C,B,A,D>	24
<I,J,C,B,A,D>	27

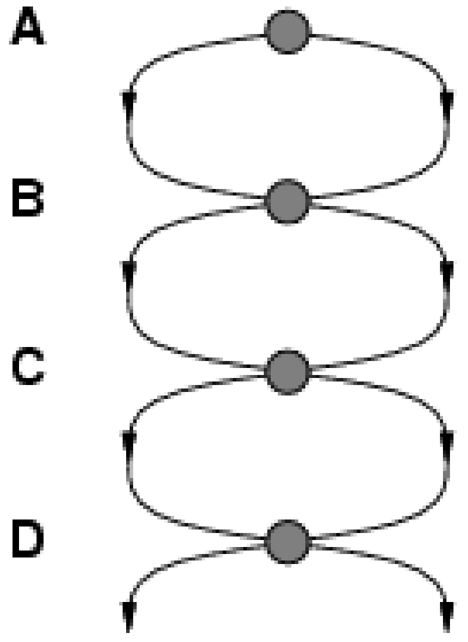


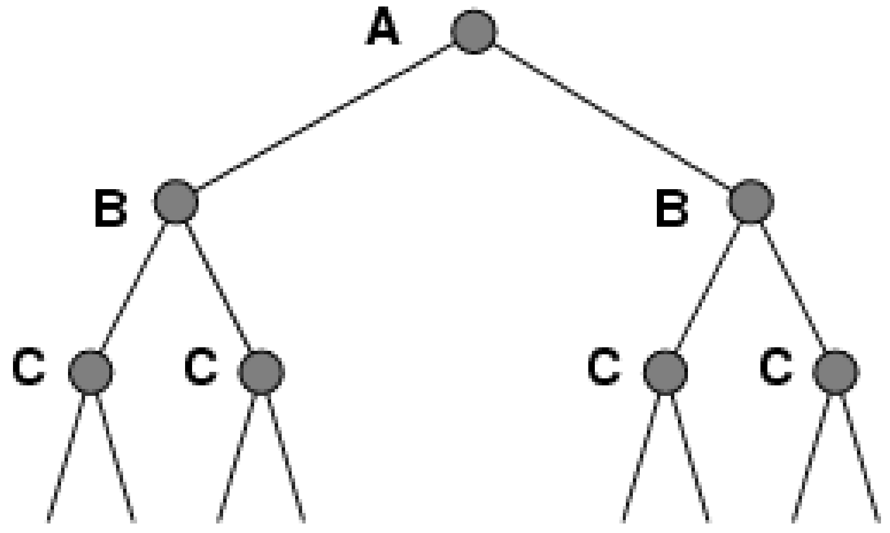
Shortest Path = Solution = D → A → B → C → J → G

As minimum path is the goal path, so we have found a solution.

Repeated States

- Failure to detect repeated states can turn a linear problem into an exponential one!





How to deal with repeated states?

- Do not return to the state you just came from.
  - *Have the expand function refuse to generate any successor that is the same state as the node's parent.*
- Do not repeat paths with cycles in them.
  - *Have the expand function refuse to generate any successor of a node that is the same as any of the node's ancestors.*
- Do not generate any state that was ever generated before.