

# **ME 620: Fundamentals of Artificial Intelligence**

## **Lecture 10: Searching AND/OR Graphs**

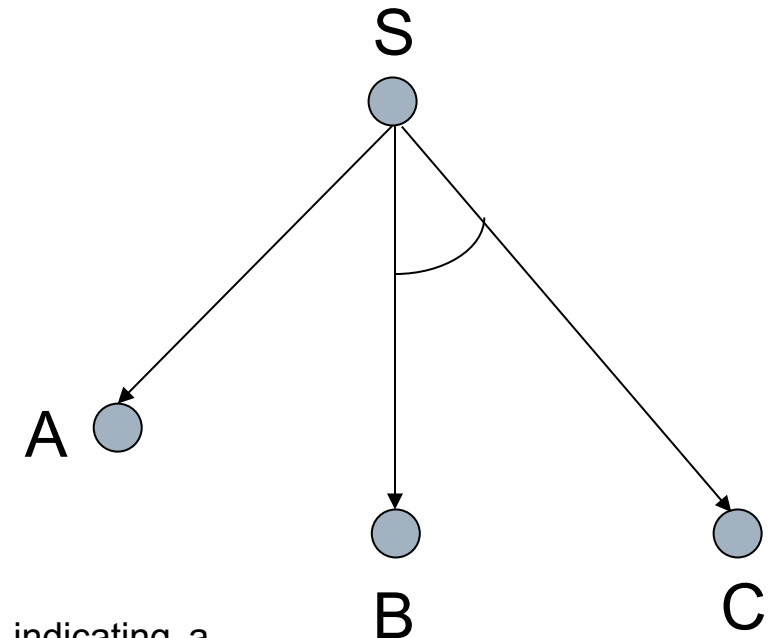


**Shyamanta M Hazarika**

Biomimetic Robotics and Artificial Intelligence Lab  
Mechanical Engineering and M F School of Data Sc. & AI  
IIT Guwahati

# AND-OR Graph

- Represent the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- AND arc may point to **any number of successor node, all of which must be solved** in order for the arc to point to a solution.
- AND arcs are **indicated with a line connecting all the components.**



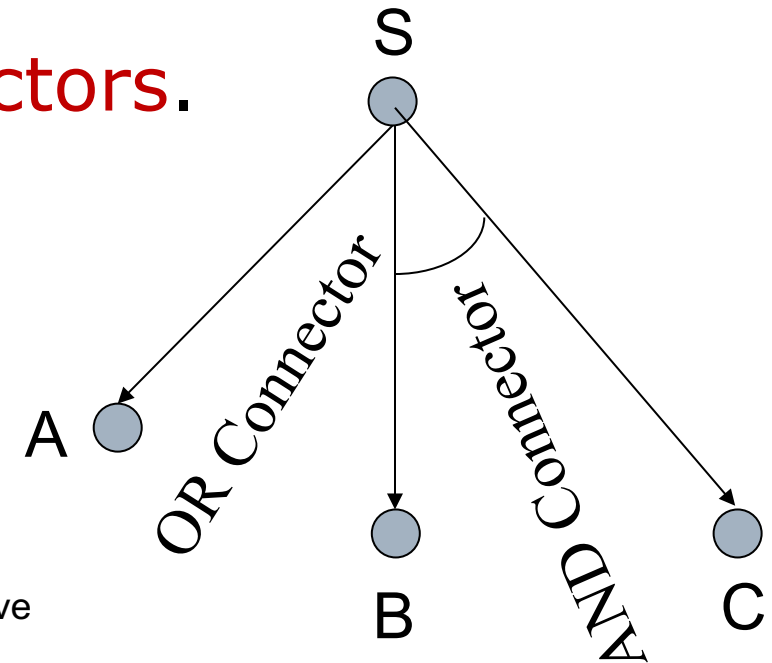
Several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved.

# Hypergraph

□ AND-OR Graphs are defined as hypergraphs.

- Instead of arcs connecting pairs of nodes, there are *hyperarcs* connecting a parent node with a set of successor nodes.
- These hyperarcs are called connectors.

Each *k*-connector is directed from a parent node to a set of *k* successor node.

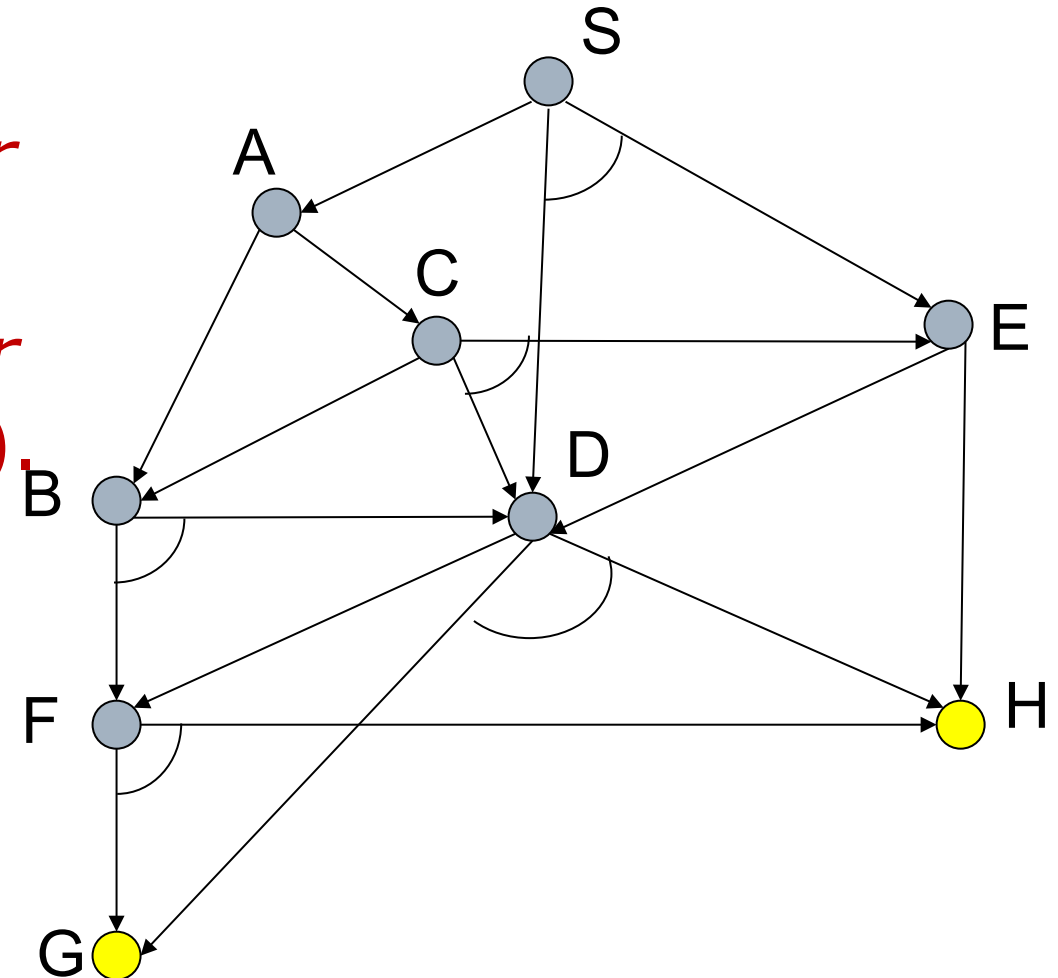


If all of the connectors are 1-connectors, we have the special case of an ordinary graph.

# AND-OR Graph

□ Figure below shows an example AND-OR Graph.

- Node *S* has a *one-connector* directed to *successor A*
- Node *S* has a *two-connector* directed to *successors (D, E)*.

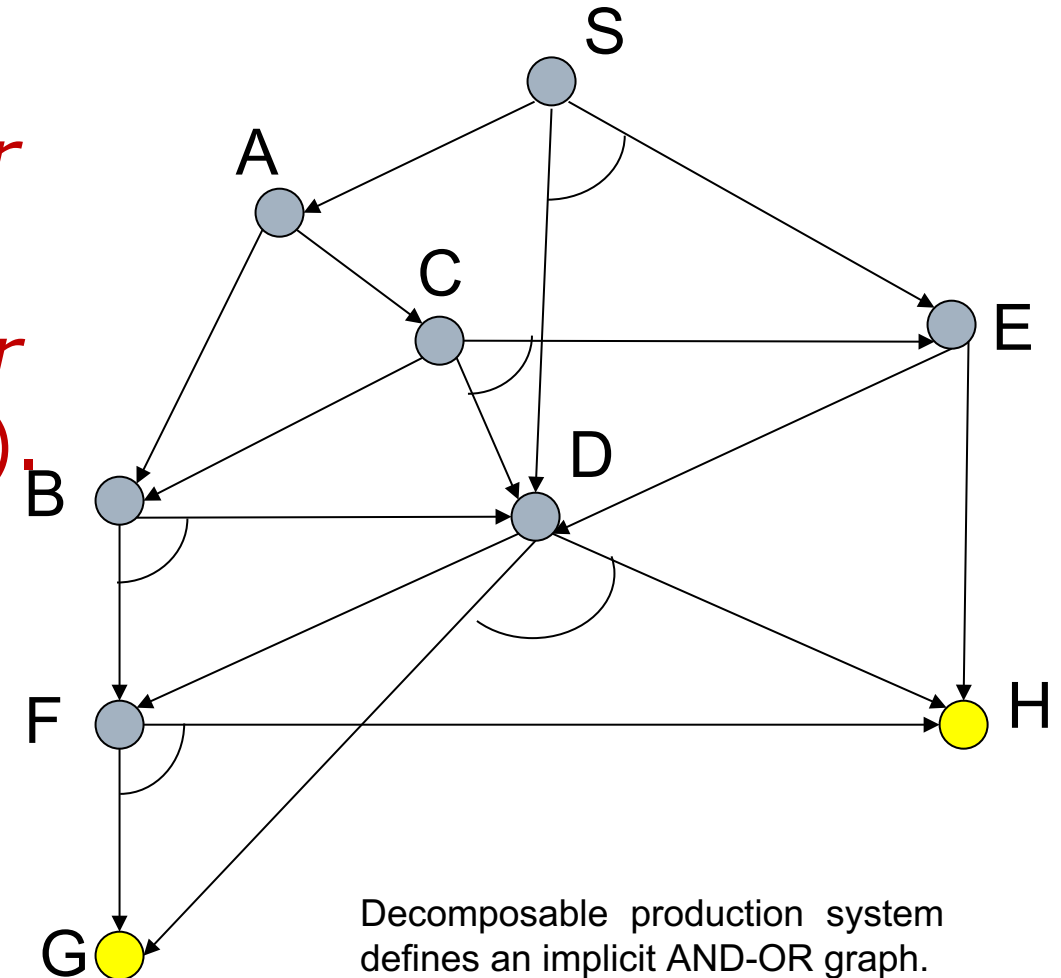
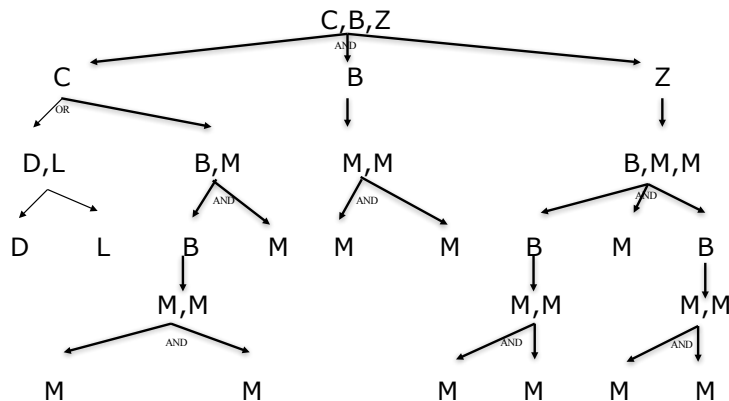


For  $k > 1$ ; each  $k$ -connector is denoted by a curved line joining the arcs from parent to elements of the successor set.

# AND-OR Graph

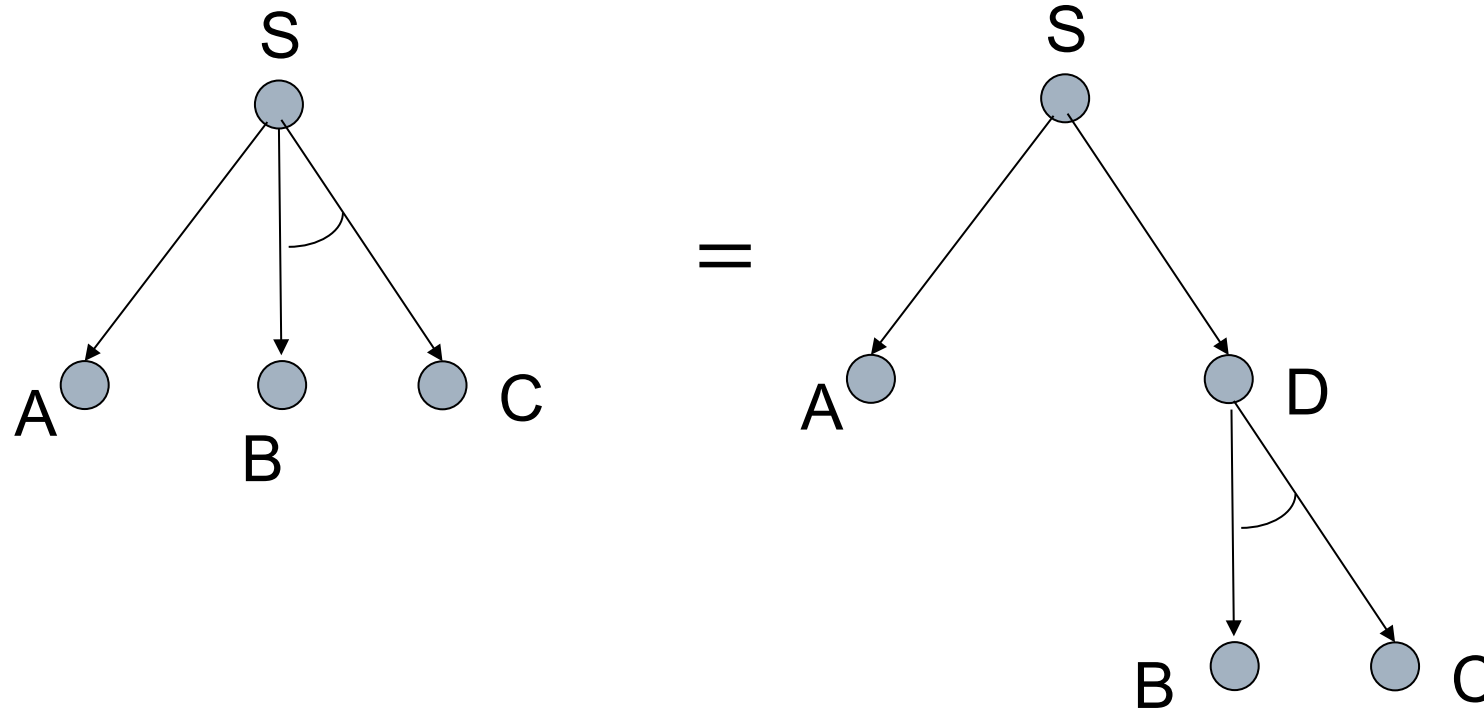
□ Initial database corresponds to start node S.

- Node S has a *one-connector* directed to successor A
- Node S has a *two-connector* directed to successors (D, E).



# AND-OR Graph

If required one can always **transform the graph on the left to one of the right**; of course, keeping track of the cost of the edges.

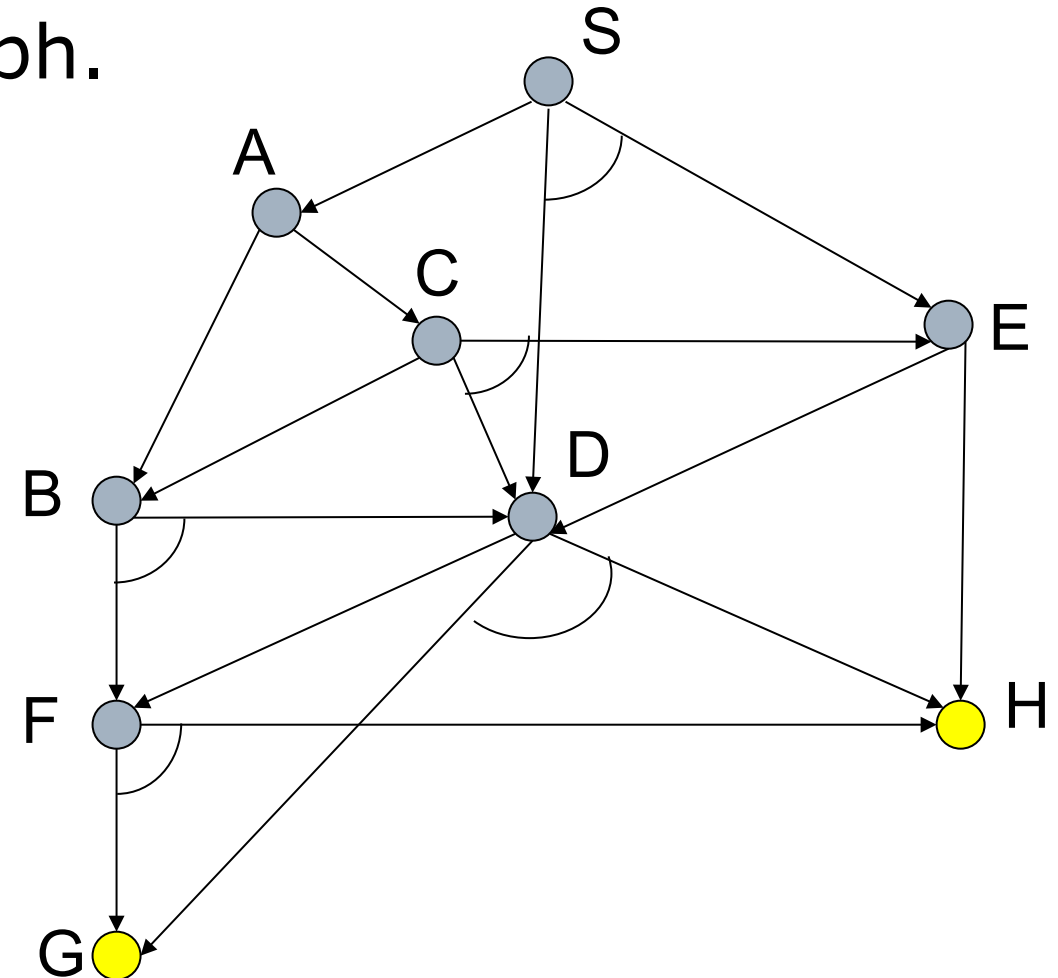


Recall that the AND-OR graphs used in our discussion of decomposable production systems had alternate AND and OR nodes.

# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

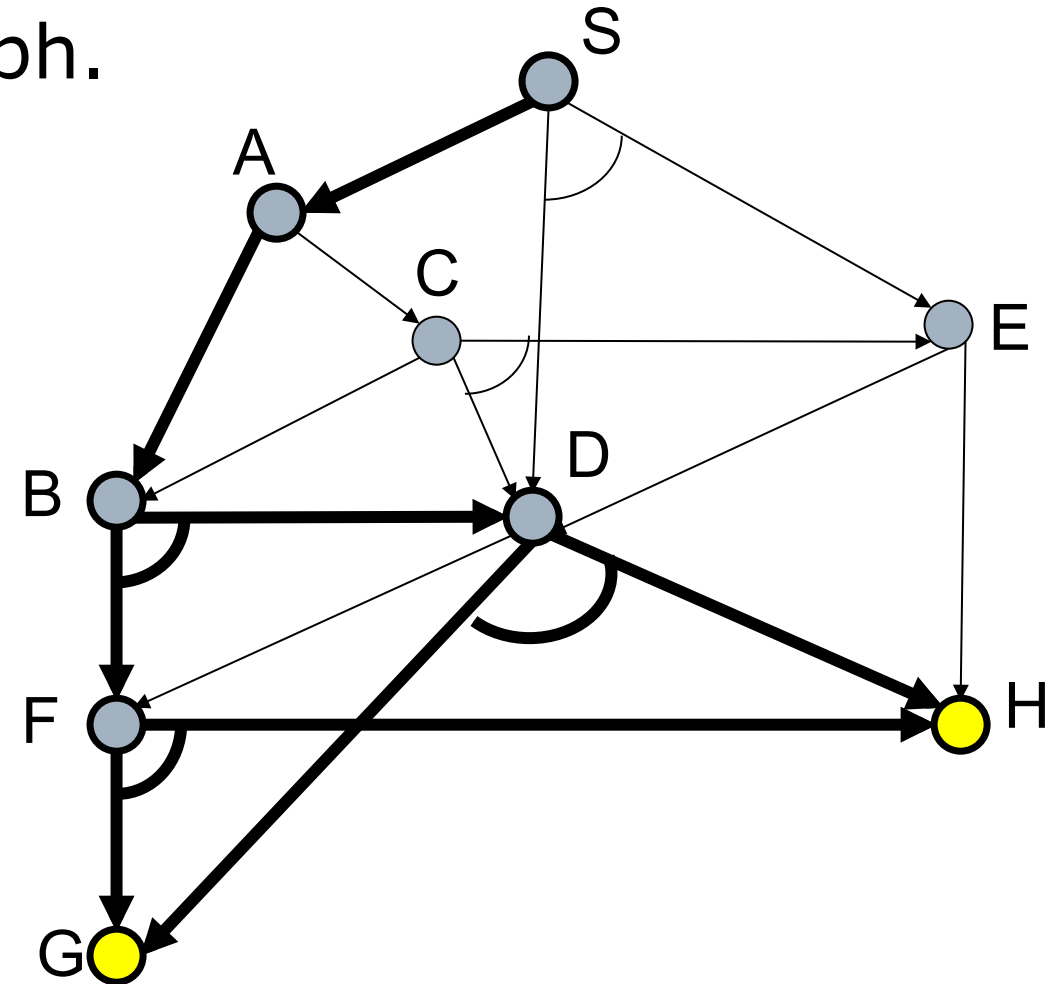
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**

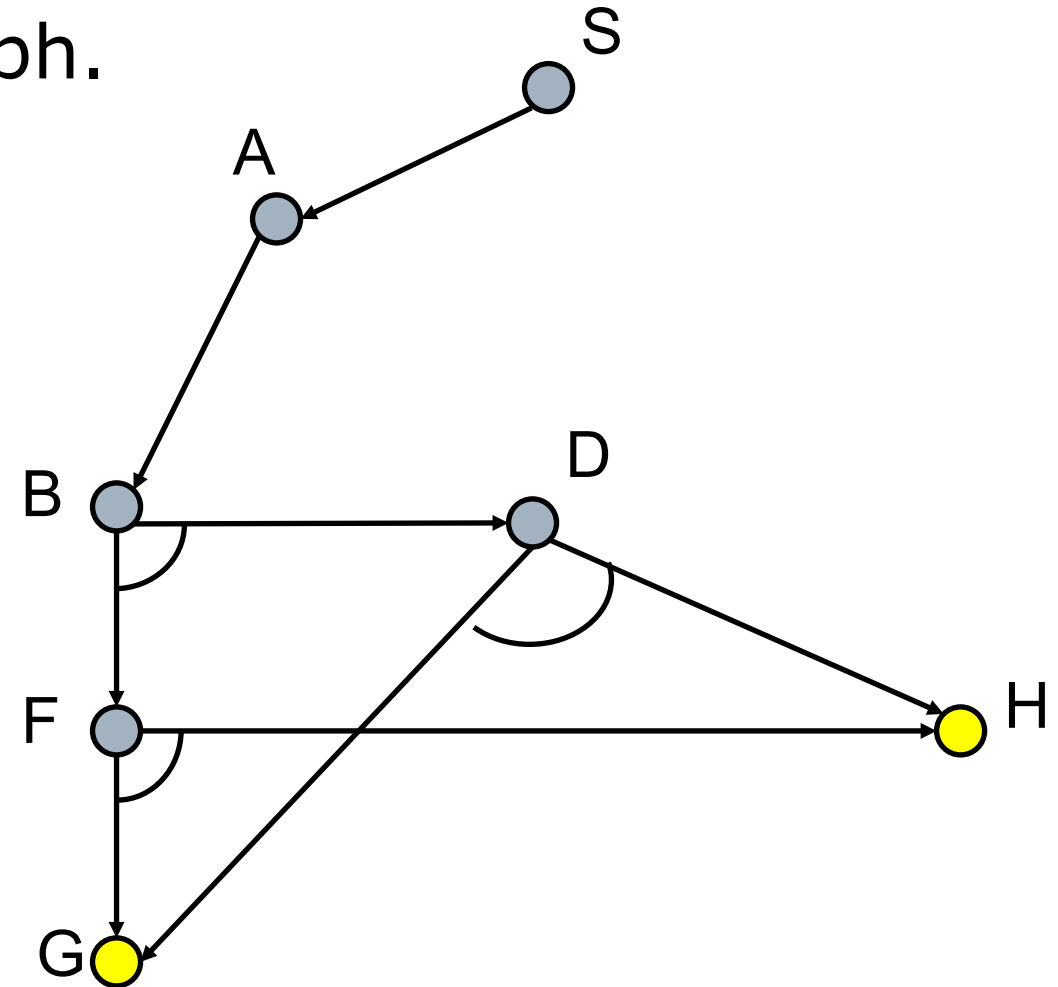




# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

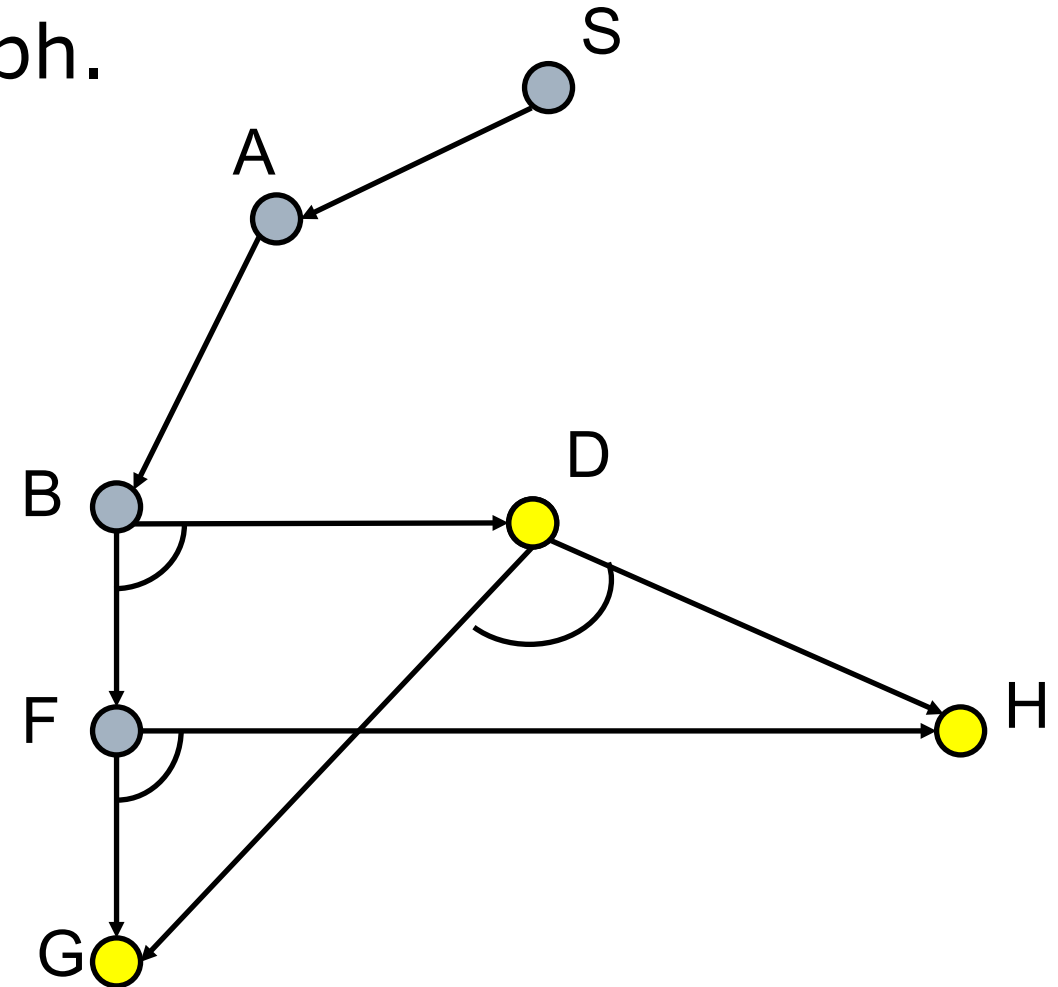
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

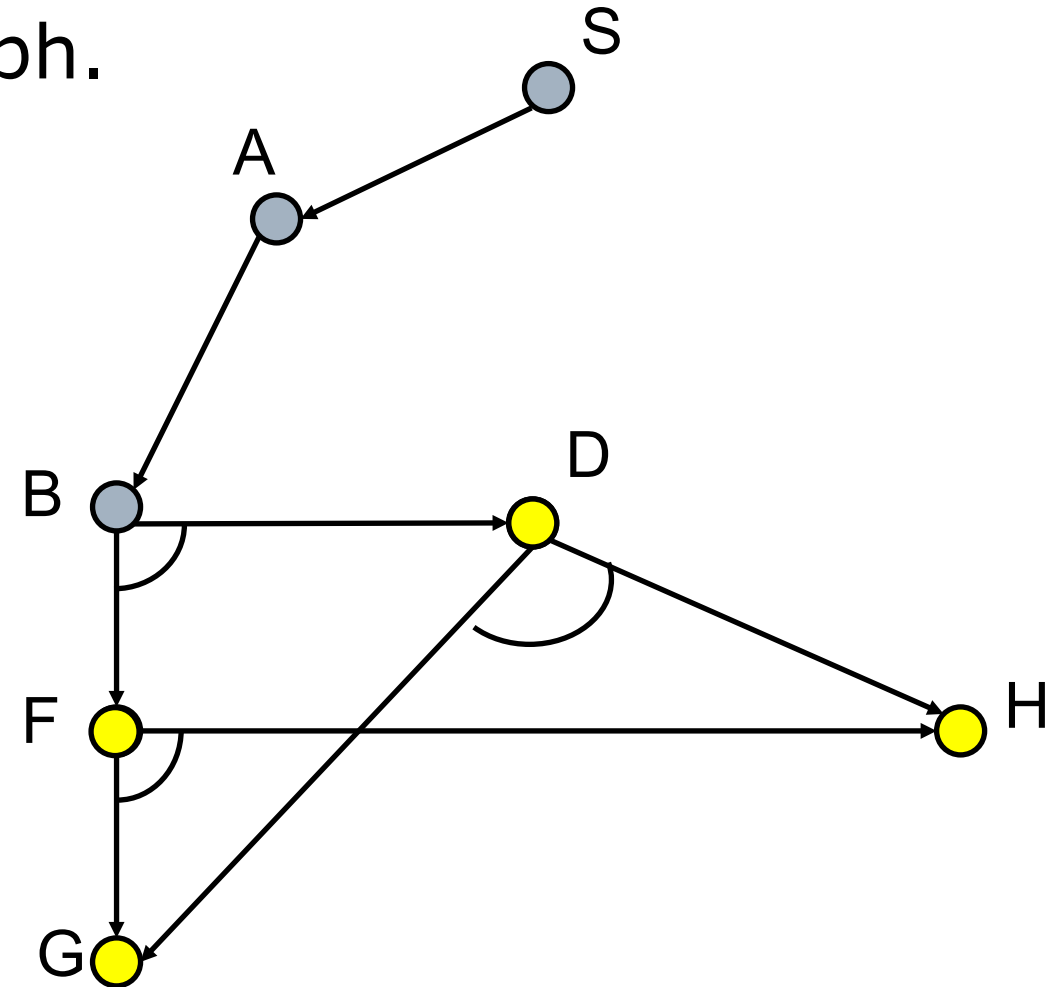
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

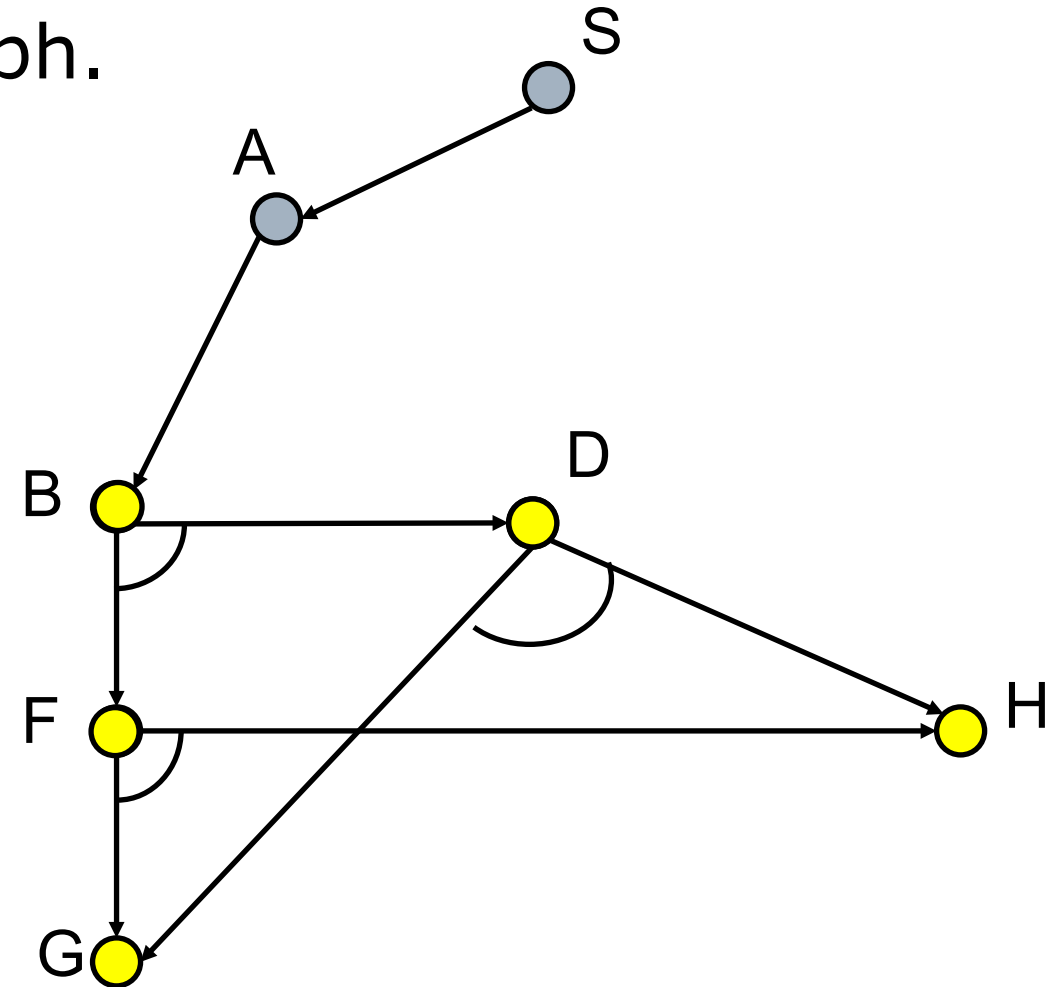
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

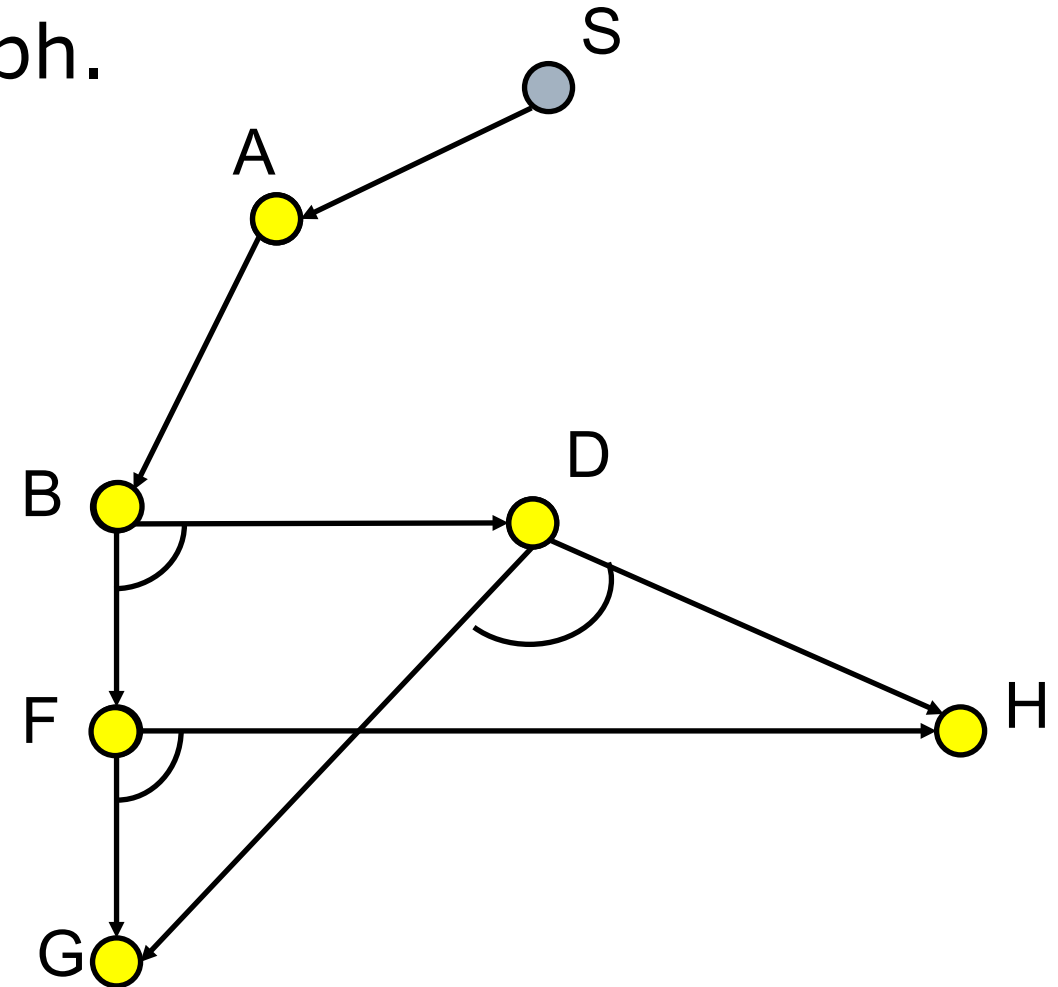
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector.**
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

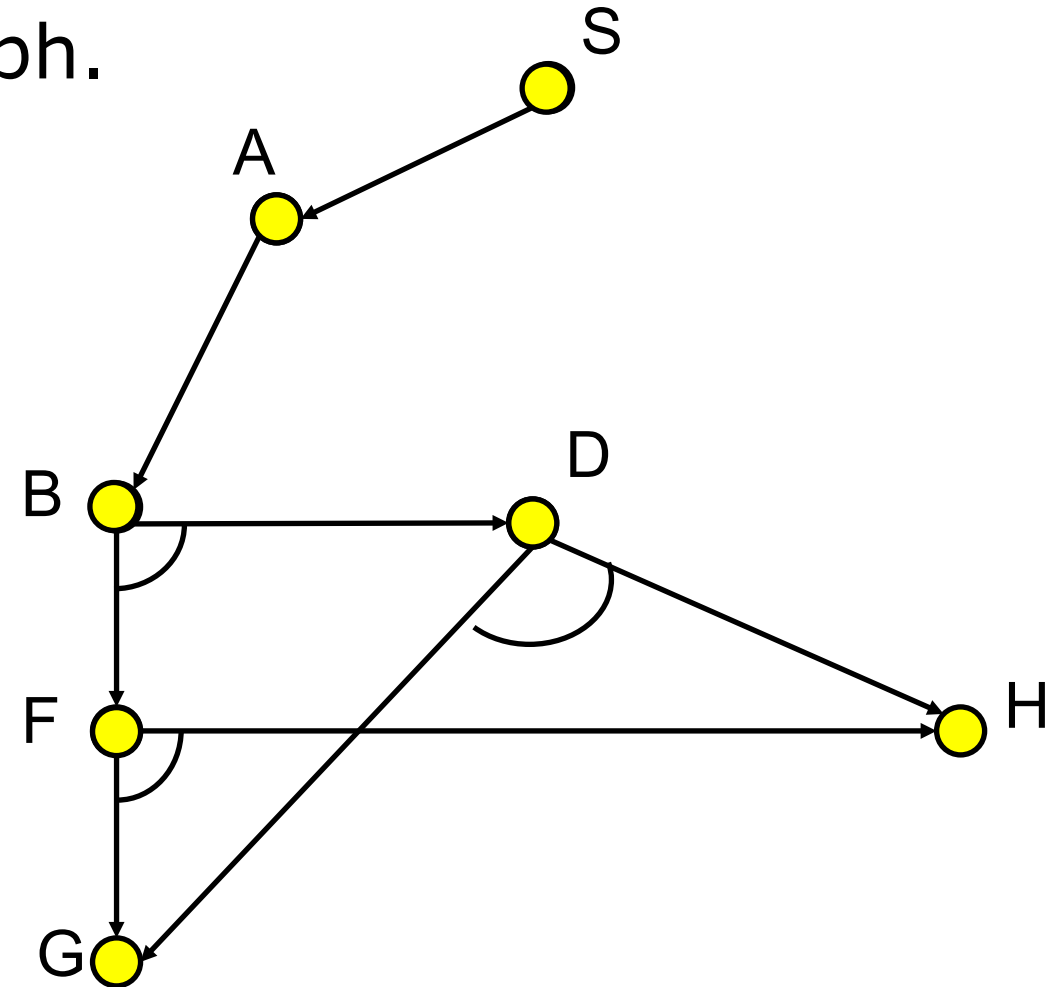
- Solution graph from  $n$  to  $N$   
 $n = S; N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

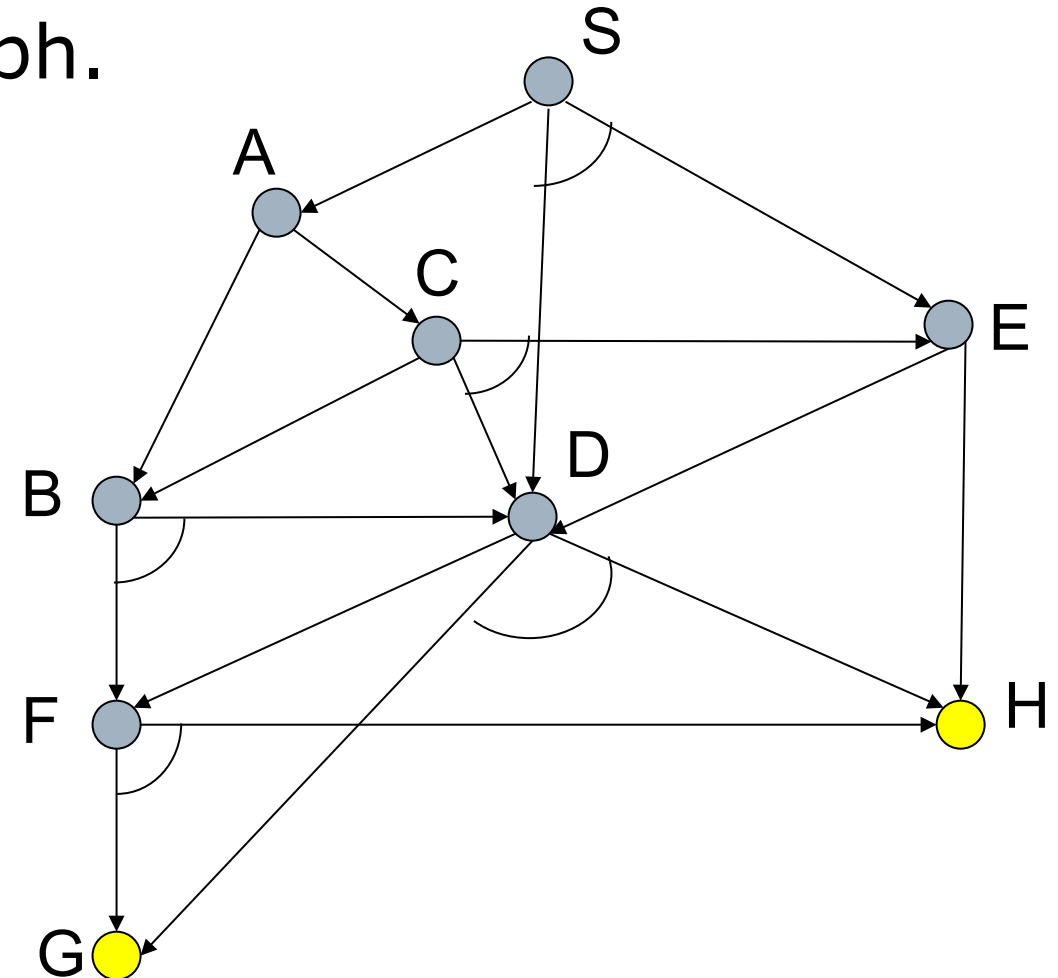
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

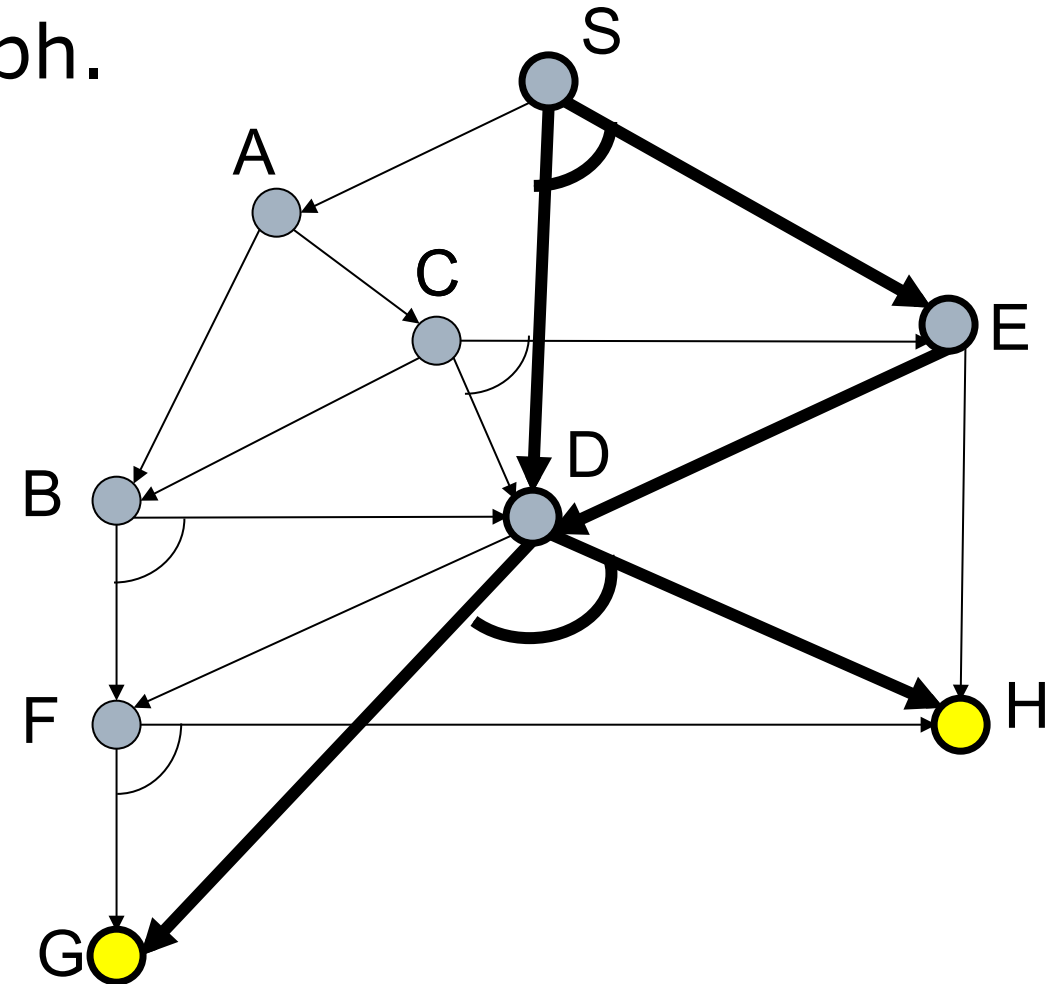
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**

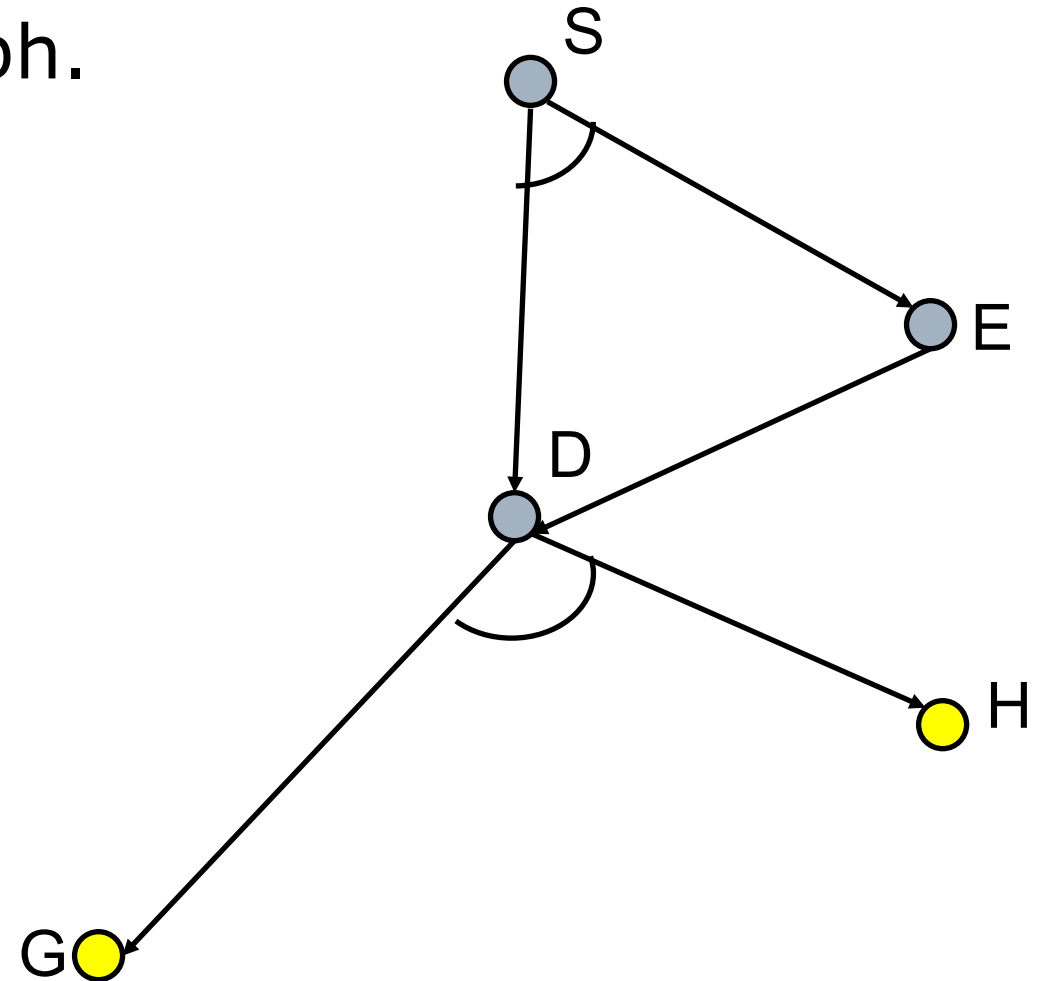




# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

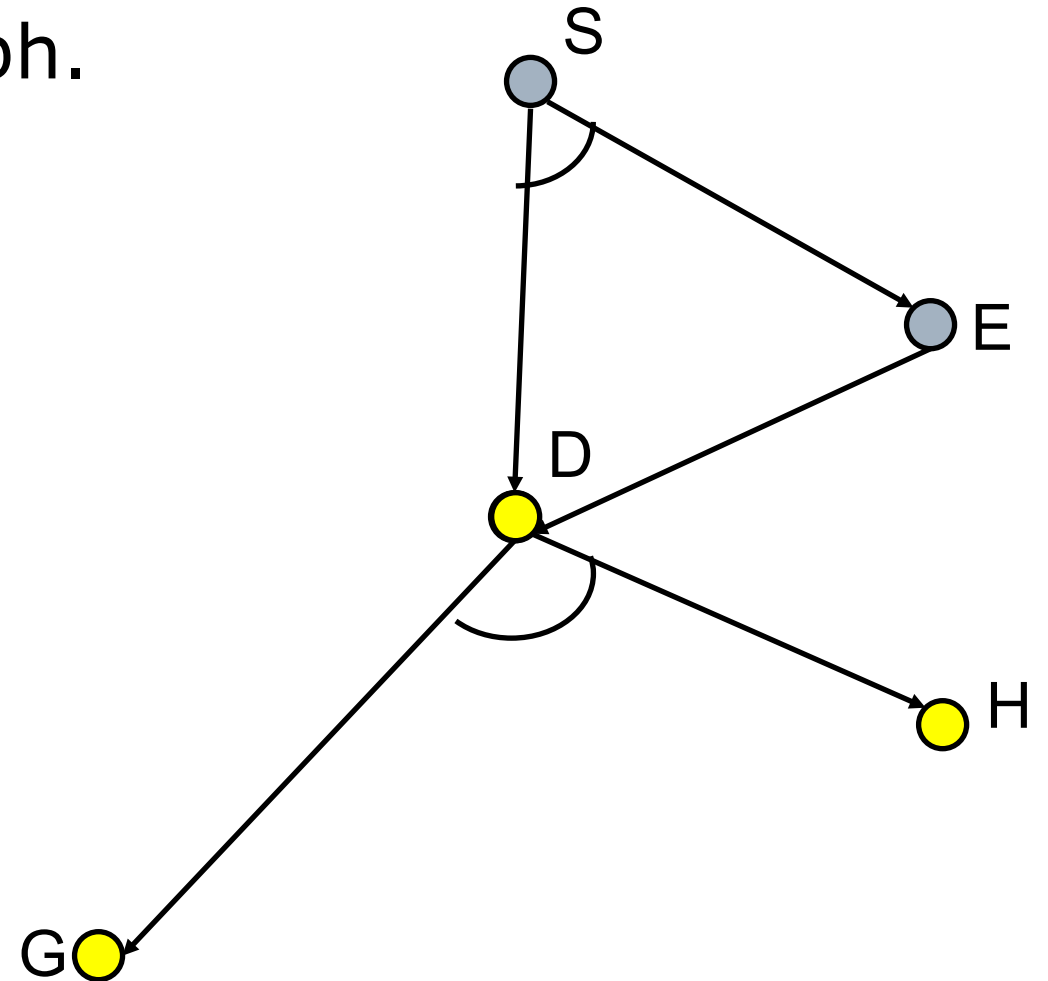
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector.**
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

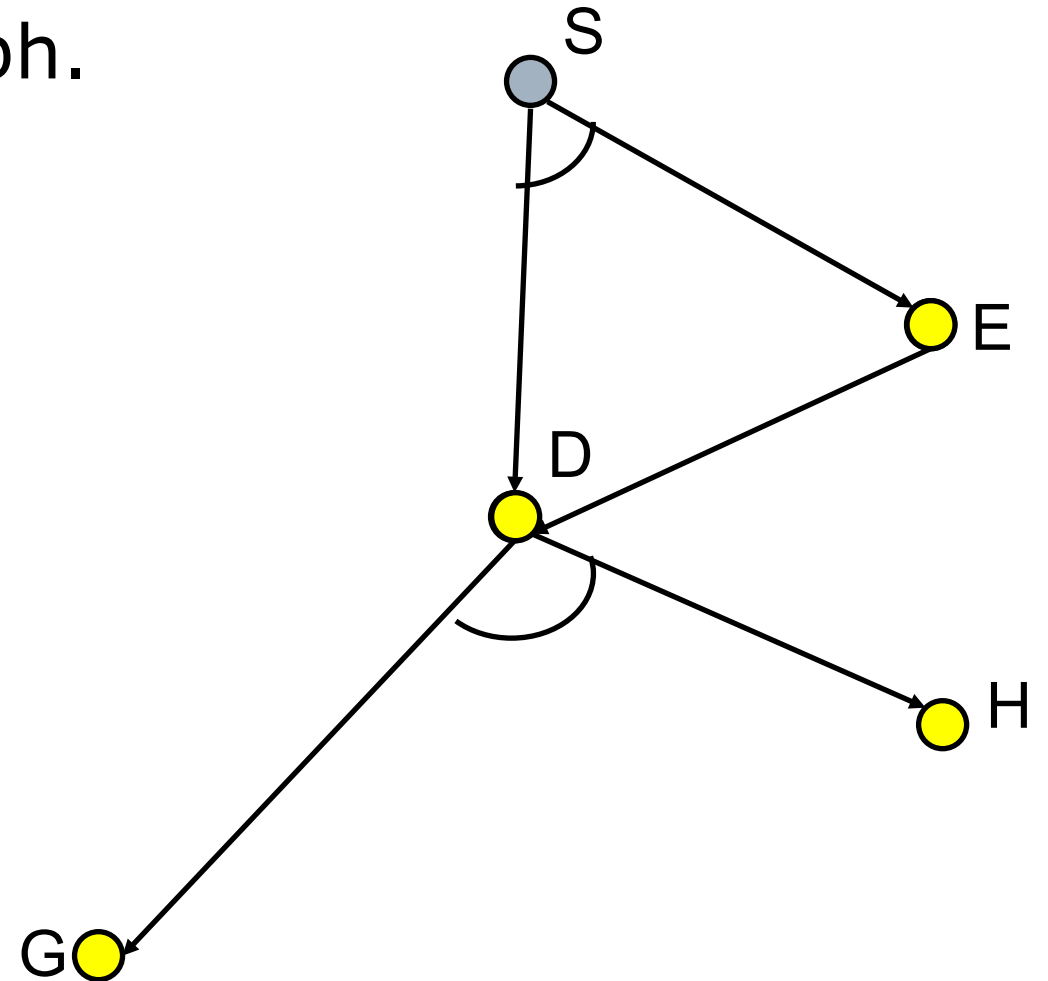
- Solution graph from  $n$  to  $N$   
 $n = S; N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector**.
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

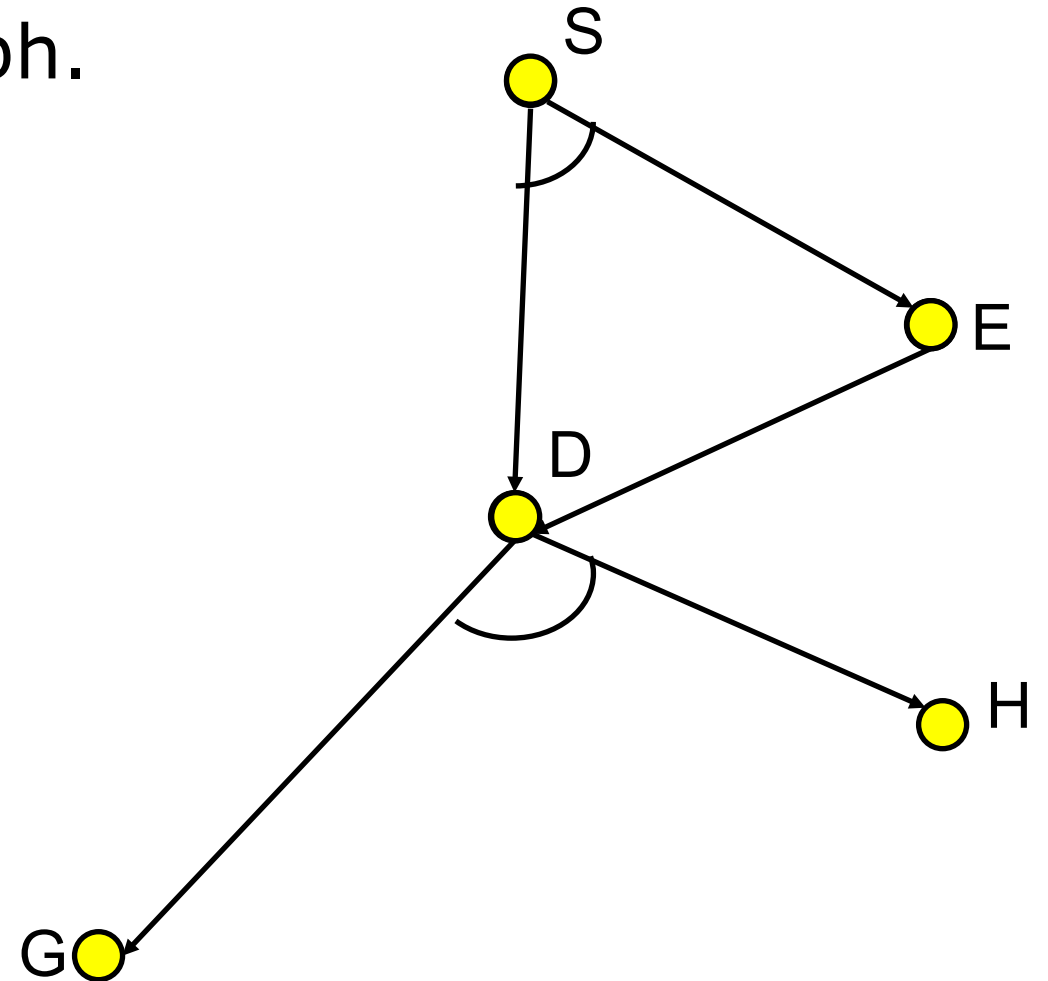
- Solution graph from  $n$  to  $N$   
 $n = S$ ;  $N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector.**
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Solution Graph

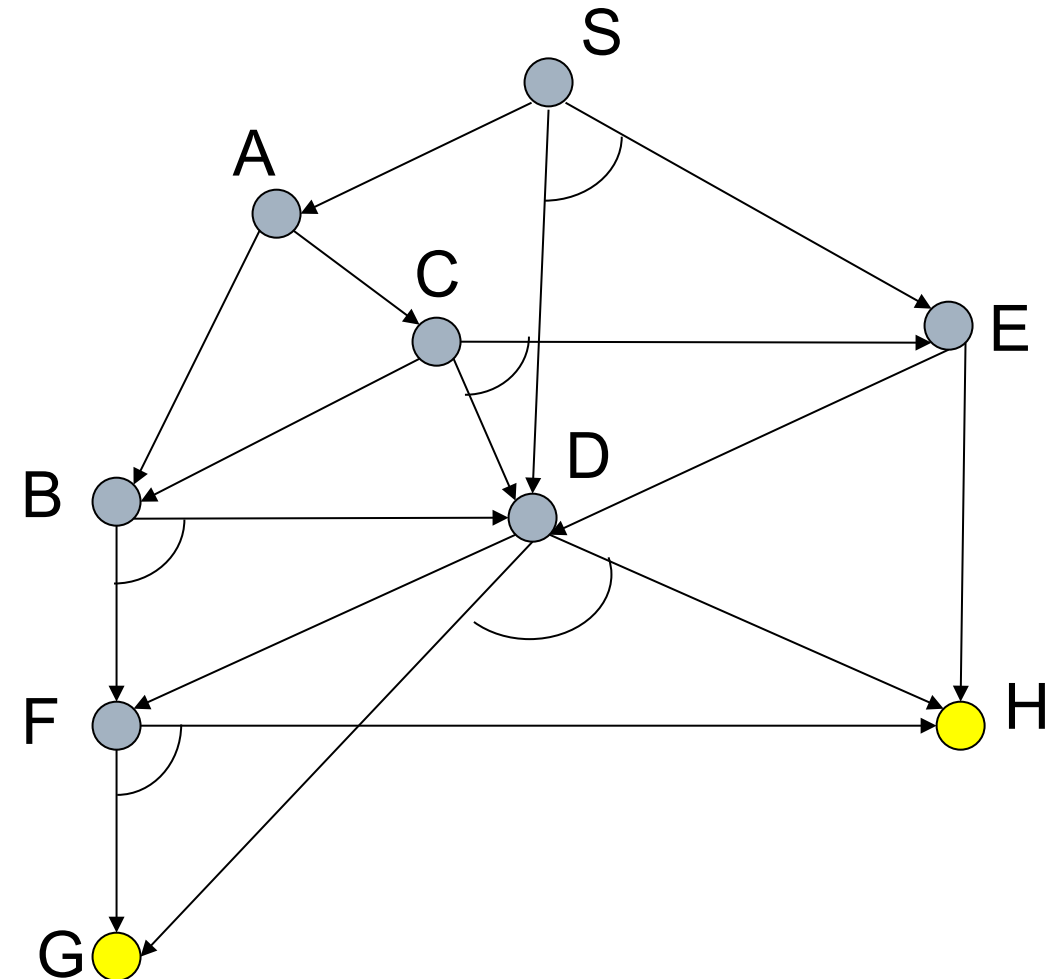
□ A **solution graph** of an AND-OR graph is **analogous to a path** in an ordinary graph.

- Solution graph from  $n$  to  $N$   
 $n = S; N = \{G, H\}$
- Starting from node  $S$ ; **select exactly one connector.**
- From **each successor** node to which the connector is directed, **select an outgoing connector** and so on.
- Eventually every successor is an **element of  $N$**



# Searching AND-OR Graph

- To find solutions, we need an algorithm similar to Best-First Search
  - But with the ability to handle the AND arcs appropriately.
- Algorithm should find a path from the start node to a set of nodes representing the solution states.
  - May be required to reach to more than one solution state; Each arm of an AND arc must lead to a solution node.



# Best-First Algorithm ?

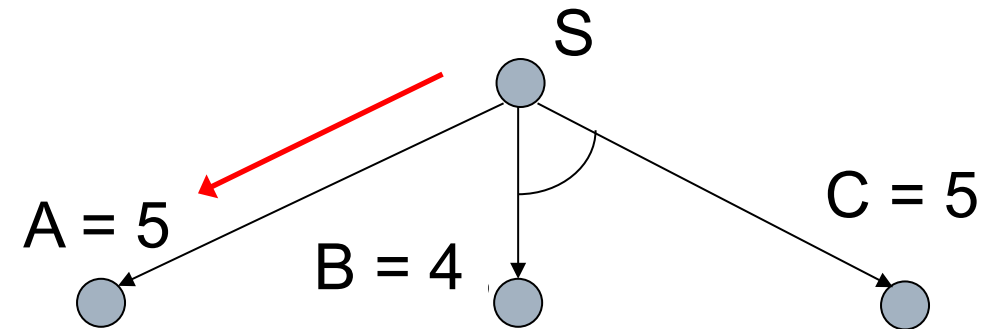
- To find solution in an AND-OR graph we need an algorithm similar to best-first search; but with the ability to handle the AND arc appropriately.

Edge cost = 1 unit.

Best-first Search ? Node B

Involving B =  $4 + 5 + 2 = 11$

Path S-A =  $5 + 1 = 6$  ; Better



B is part of an AND arc;  
If we choose to use B, we must also use C.

Assumption: Each k-connector has a cost of 1 for each of its k successors.  
AND arc with TWO successors have a cost of 2.

# Best-First Algorithm ?

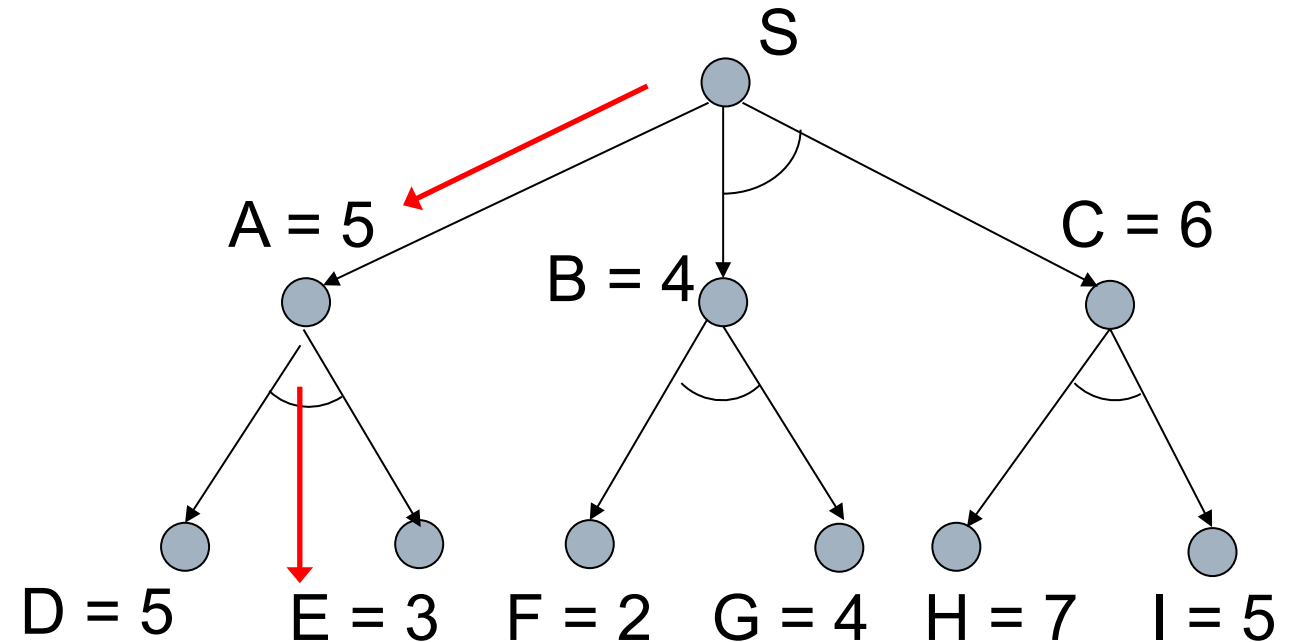
- To find **solution** in an **AND-OR** graph we need an **algorithm** similar to **best-first search**; but with the ability to **handle the AND arc** appropriately.

Edge cost = 1 unit.

**Best-first Search ? Edge B-FG**

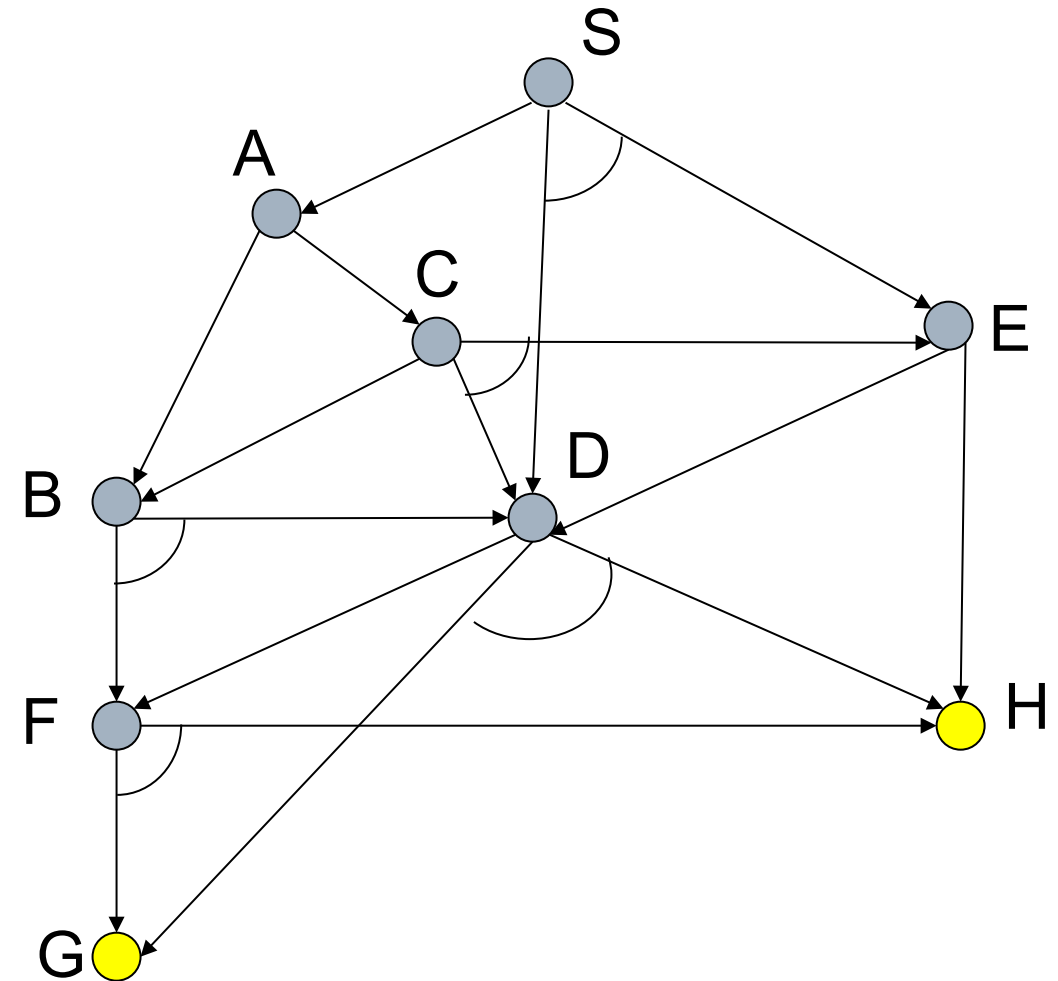
Involving B-FG; Also C-HI

**Path S-A; A-DE Better**



# Searching AND-OR Graph

- To find solutions, we need an algorithm similar to Best-First Search
  - But with the ability to handle the AND arcs appropriately.
- Algorithm should find a **path from the start node to a set of nodes** representing the solution states.
  - May be required to reach to more than one solution state; Each arm of an AND arc must lead to a solution node.





# AO\*: A Heuristic Search Procedure

---

Use a heuristic function  $h(n)$ ; estimate of the cost of an optimal solution from node  $n$  to set of terminal nodes.

Each node will have an associated  $h$  value;  
serve as measure of goodness of the node.

Rather than two lists – OPEN and CLOSED - used for A\* algorithm, the AO\* algorithm use a single structure - a graph - representing part of the search graph that has been explicitly generated so far.

Two major operations:

1. Top-down, graph-growing
2. Bottom-up, cost-revising, connector-marking, SOLVE-labelling.

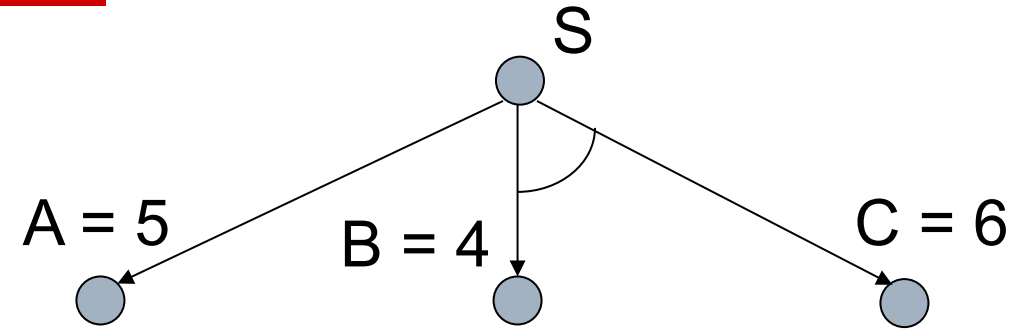
Not store a value for the cost to the current node; such a value is not necessary because of the top down traversal of the best-known path.

# AO\*: A Heuristic Search Procedure

After graph G is initialized to S

Follow which hyperedge?

Alternatives – Path S-A; Path S-BC



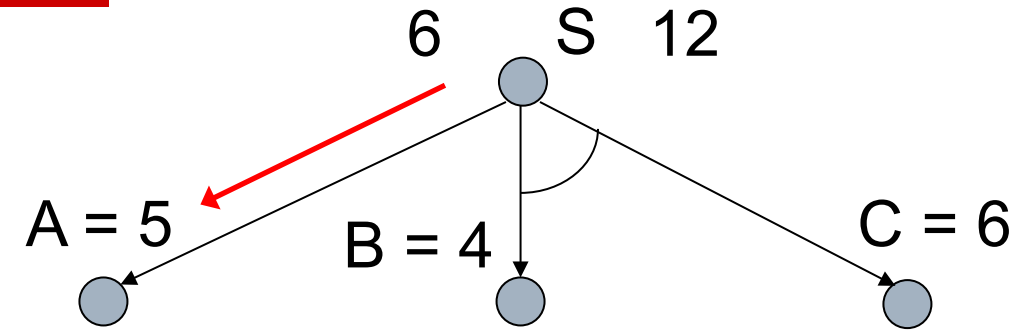
Edge cost = 1 unit.

# AO\*: A Heuristic Search Procedure

After graph G is initialized to S

Follow which hyperedge?

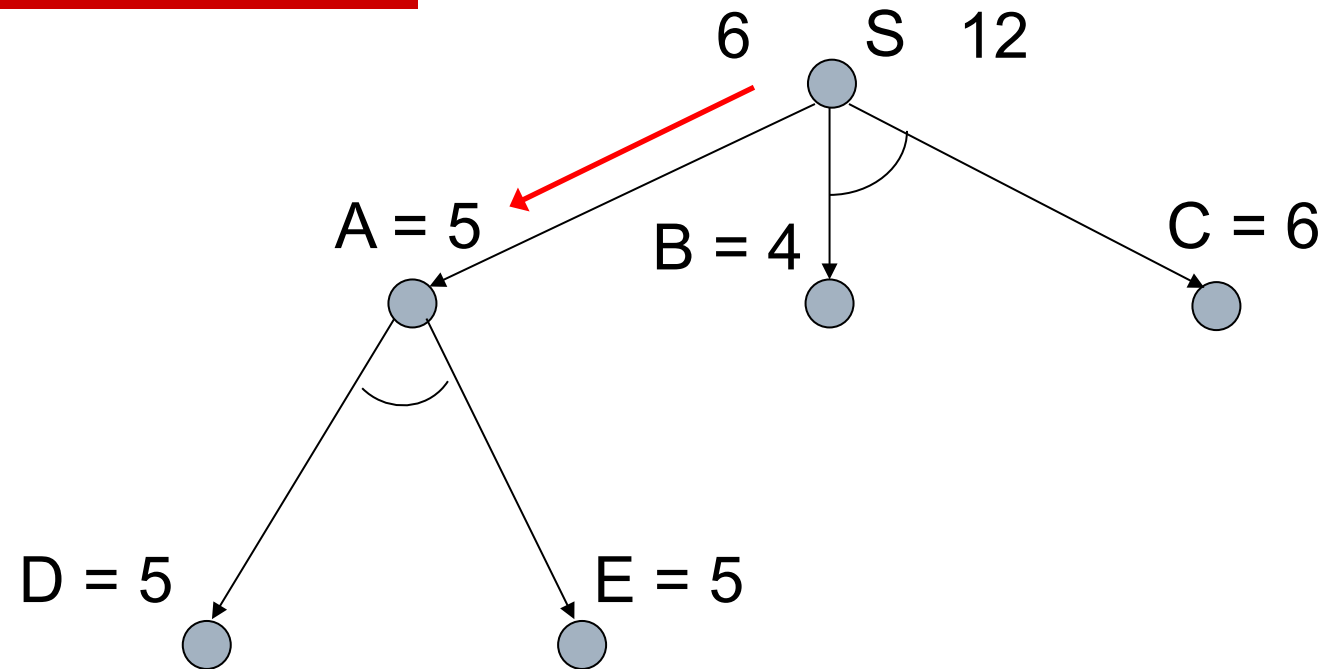
Alternatives – Path S-A; Path S-BC



Edge cost = 1 unit.

# AO\*: A Heuristic Search Procedure

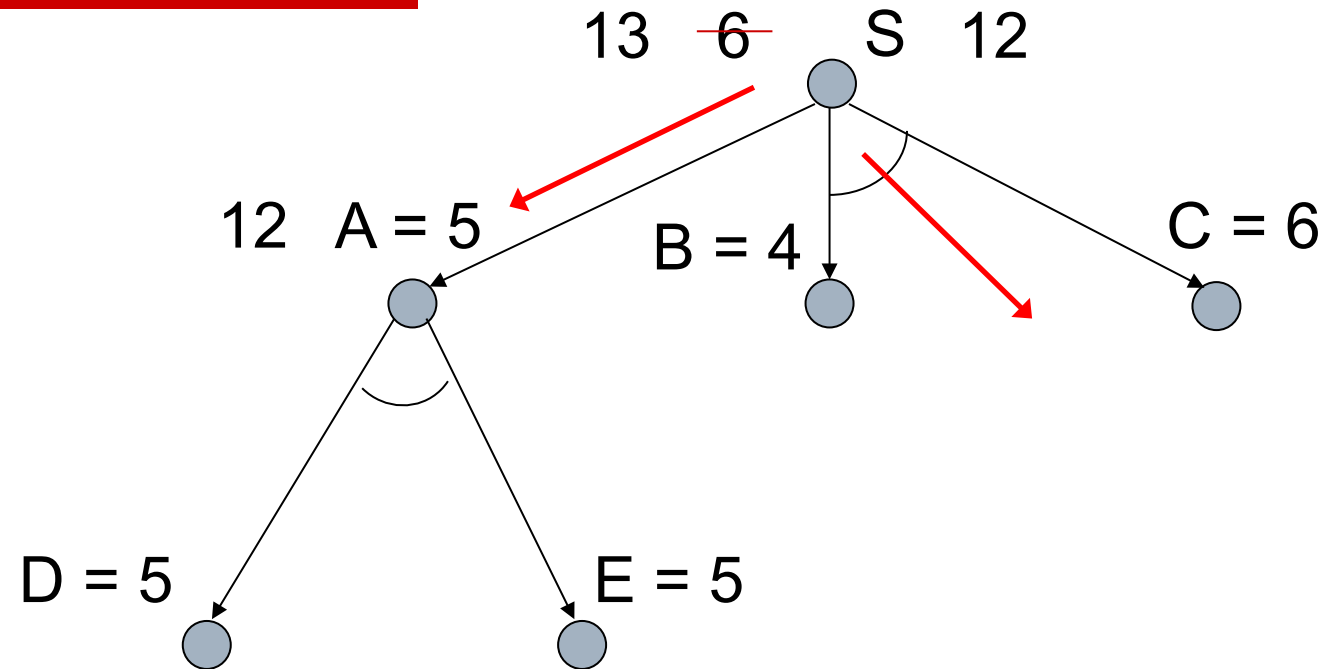
After graph A is expanded;  
Nodes – D and E  
How promising?  
Evaluate h values – back-up



Edge cost = 1 unit.

# AO\*: A Heuristic Search Procedure

After graph A is expanded;  
Nodes – D and E  
How promising?  
Evaluate h values – back-up



Edge cost = 1 unit.

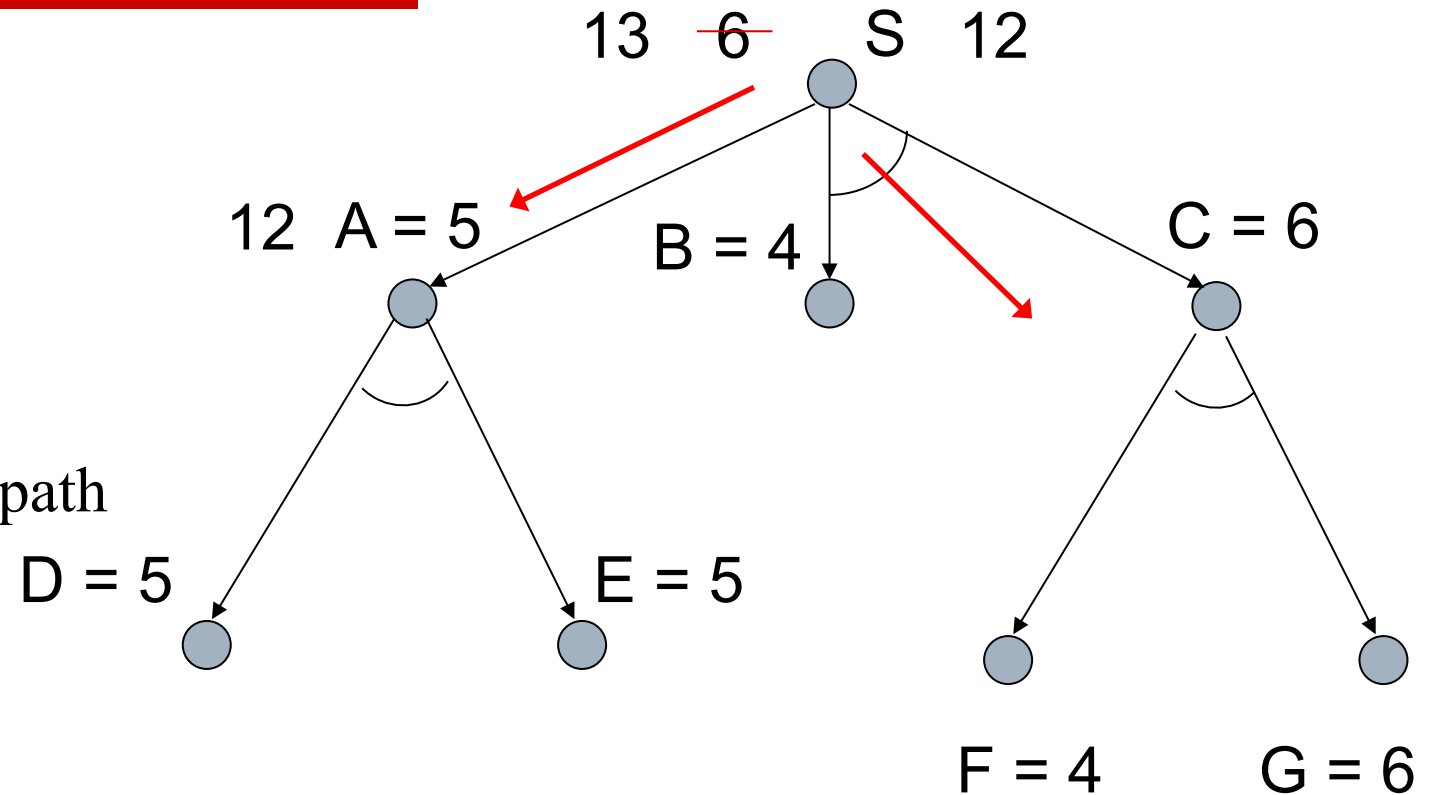
# AO\*: A Heuristic Search Procedure

More promising?

After node C is expanded;  
Nodes – F and G

How promising?

Evaluate h values – revise on path



Edge cost = 1 unit.

# AO\*: A Heuristic Search Procedure

More promising?

After node C is expanded;  
Nodes – F and G

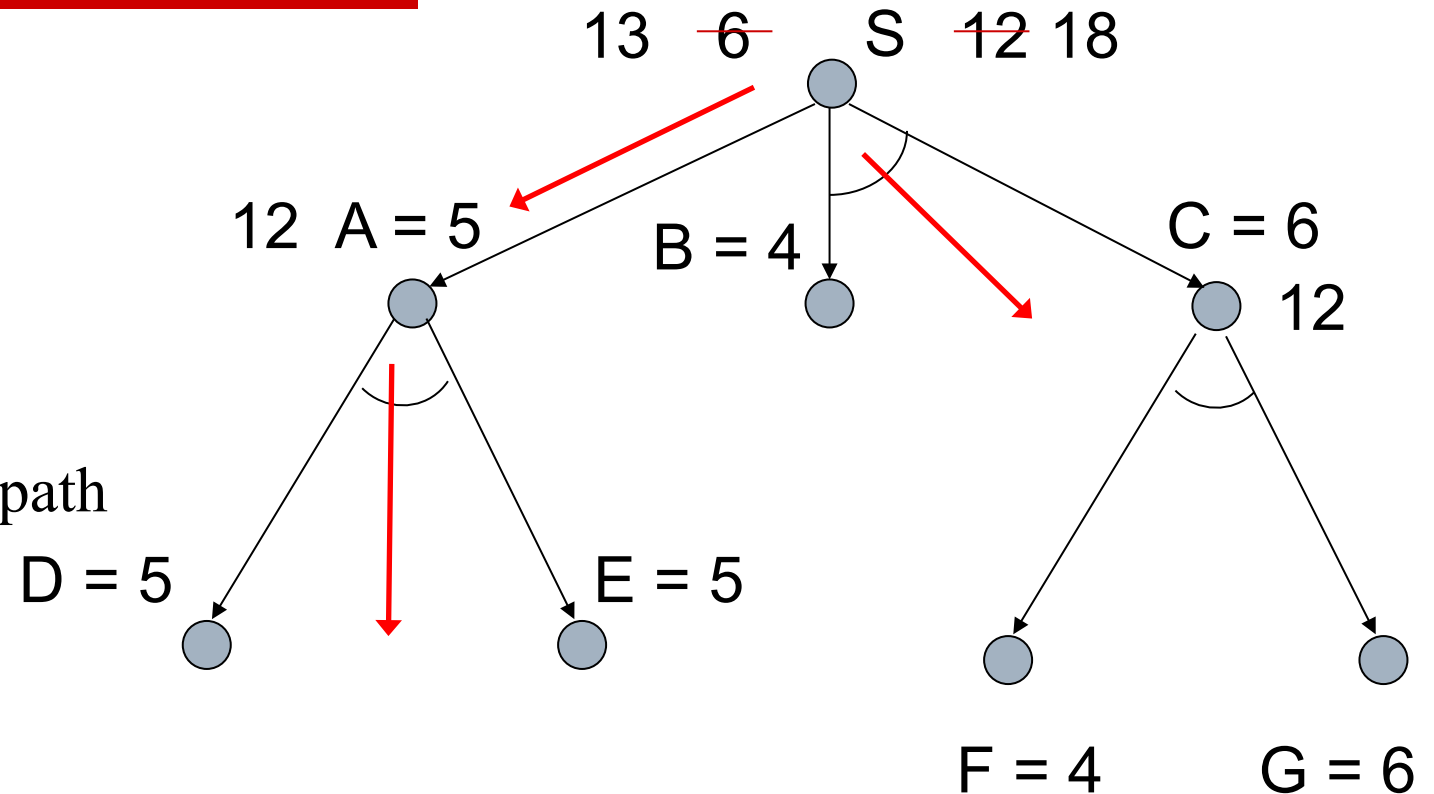
How promising?

Evaluate h values – revise on path

Nodes – D and E

How promising?

Evaluate h values – back-up



Edge cost = 1 unit.

# Basic Idea of AO\*

---



## □ Top-down graph growing

- Picks out **best available partial solution sub-graph** from explicit graph by **tracing down *marked* connectors**.
  - Here *marked* – indicates the **current best partial solution graph** from each node in the search graph.
- One of the **nonterminal leaf nodes** of this **best partial solution graph** is expanded.
  - **cost is assigned** to its successors.

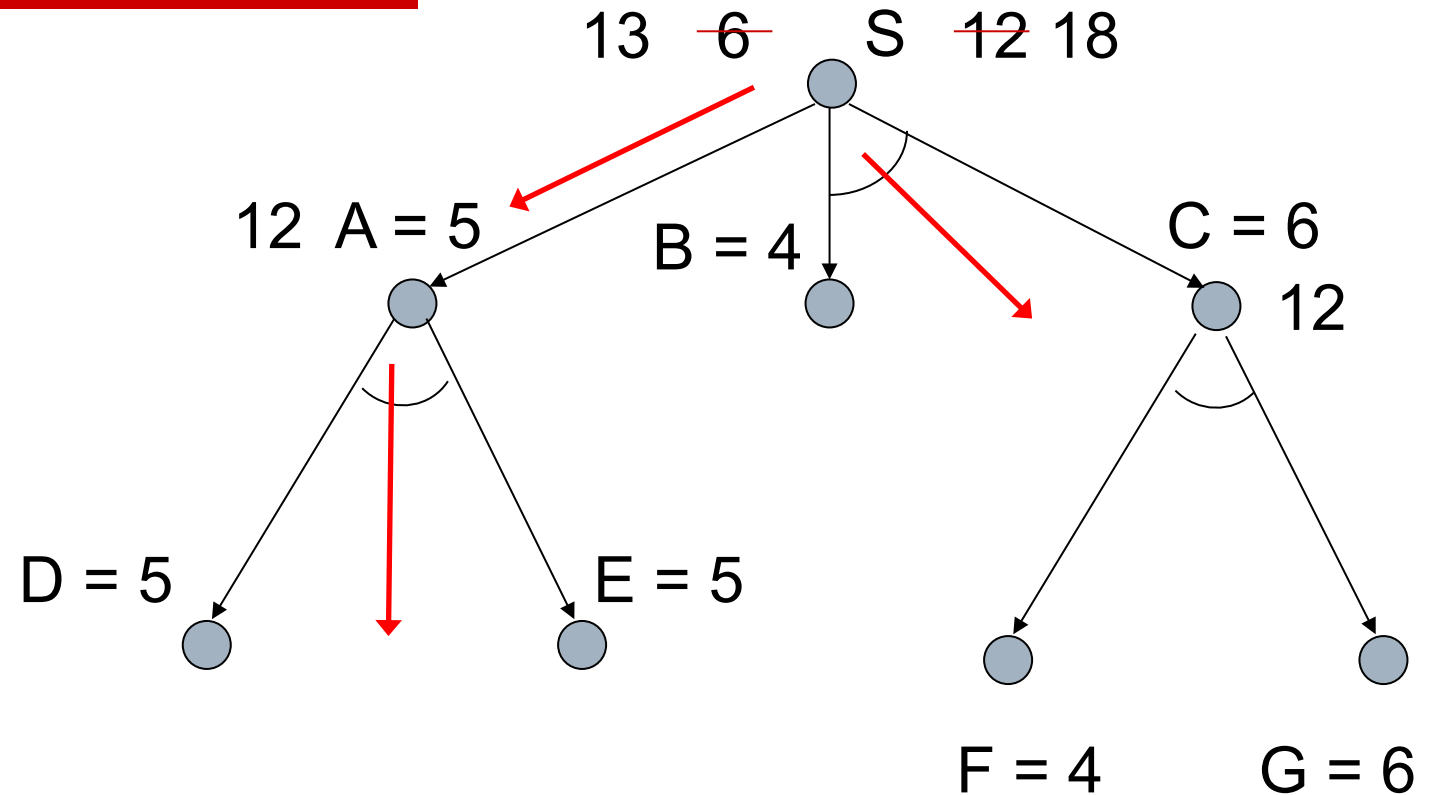


# AO\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – D and E

How promising?

Evaluate h values – back-up



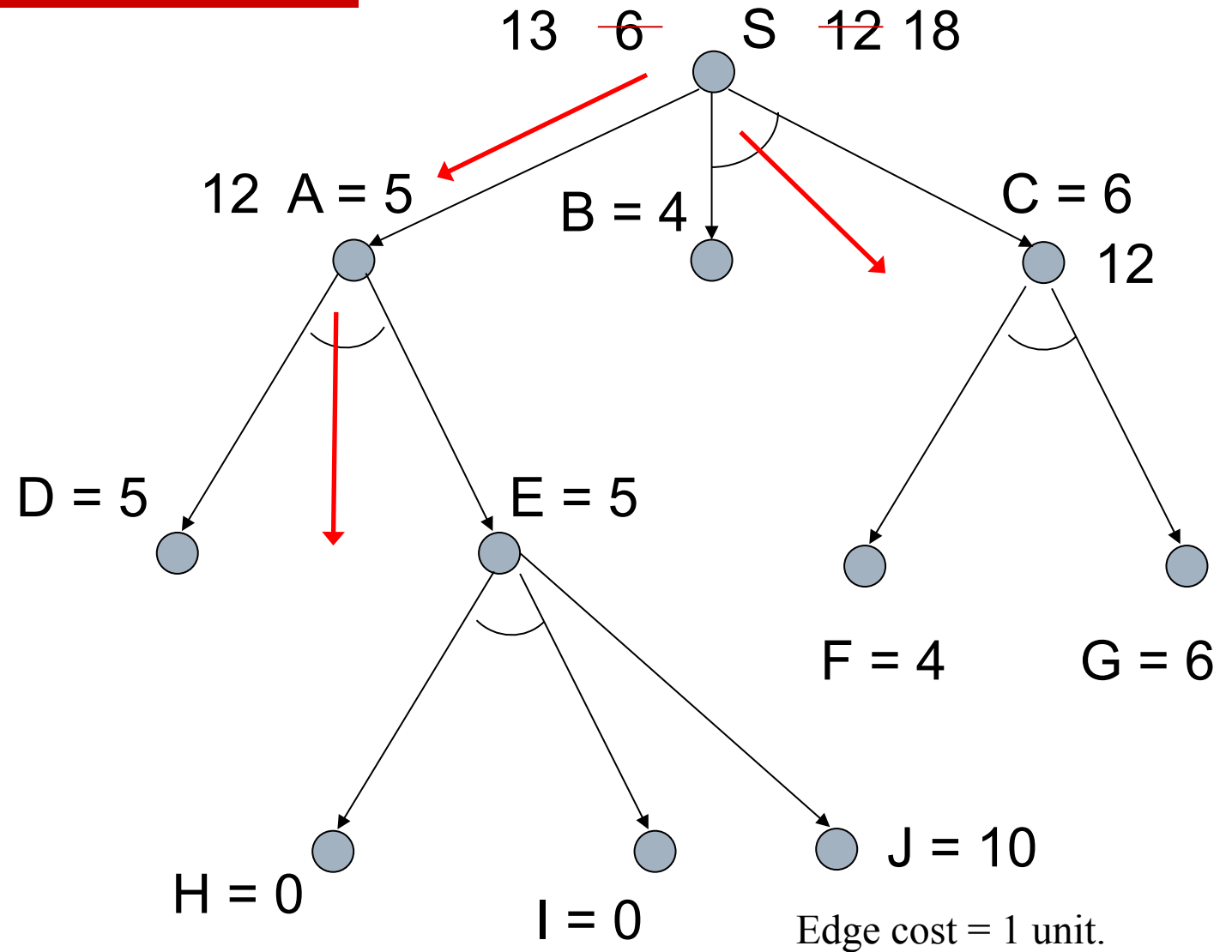
Edge cost = 1 unit.

# AO\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – H-I; J

How promising?

Evaluate h values – back-up

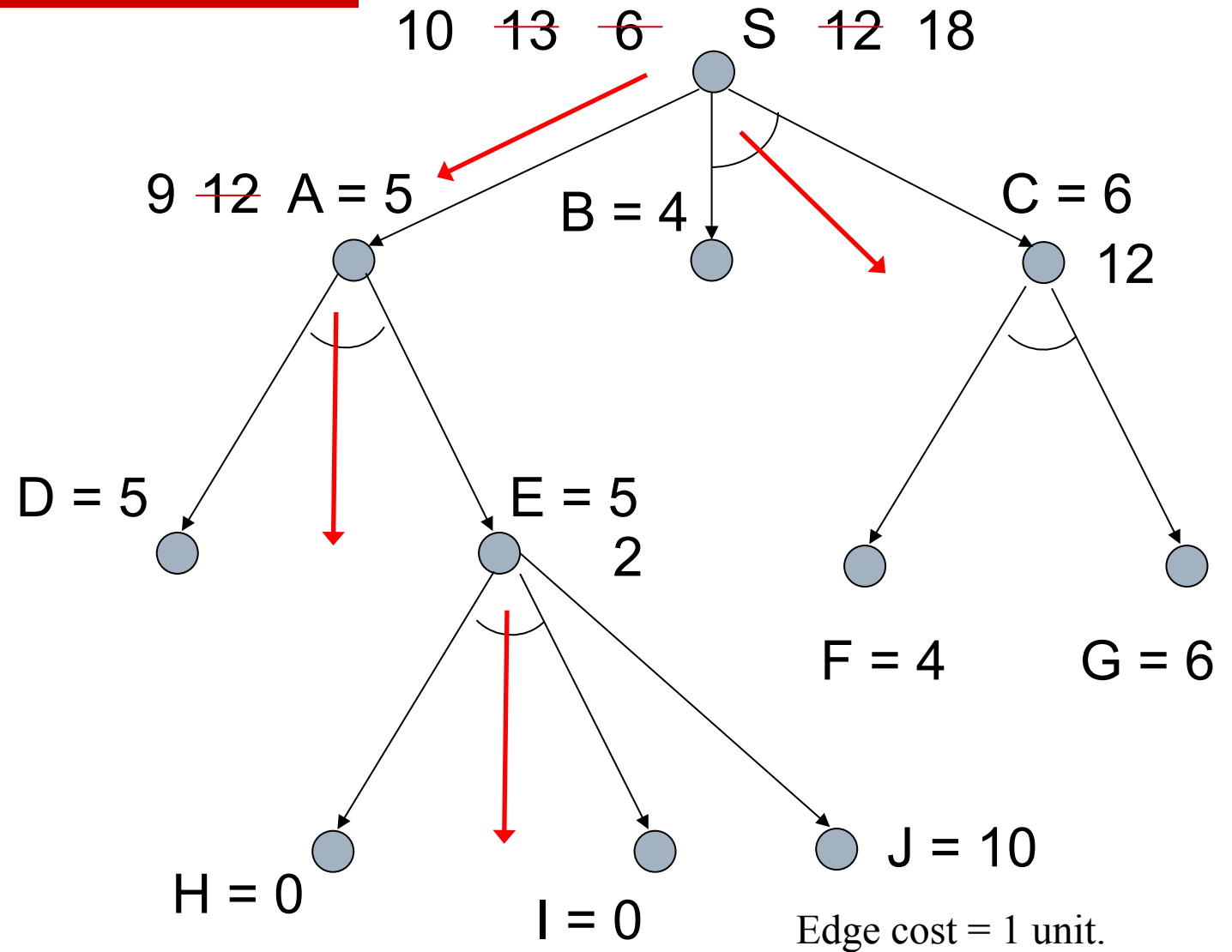


# AO\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – H-I; J

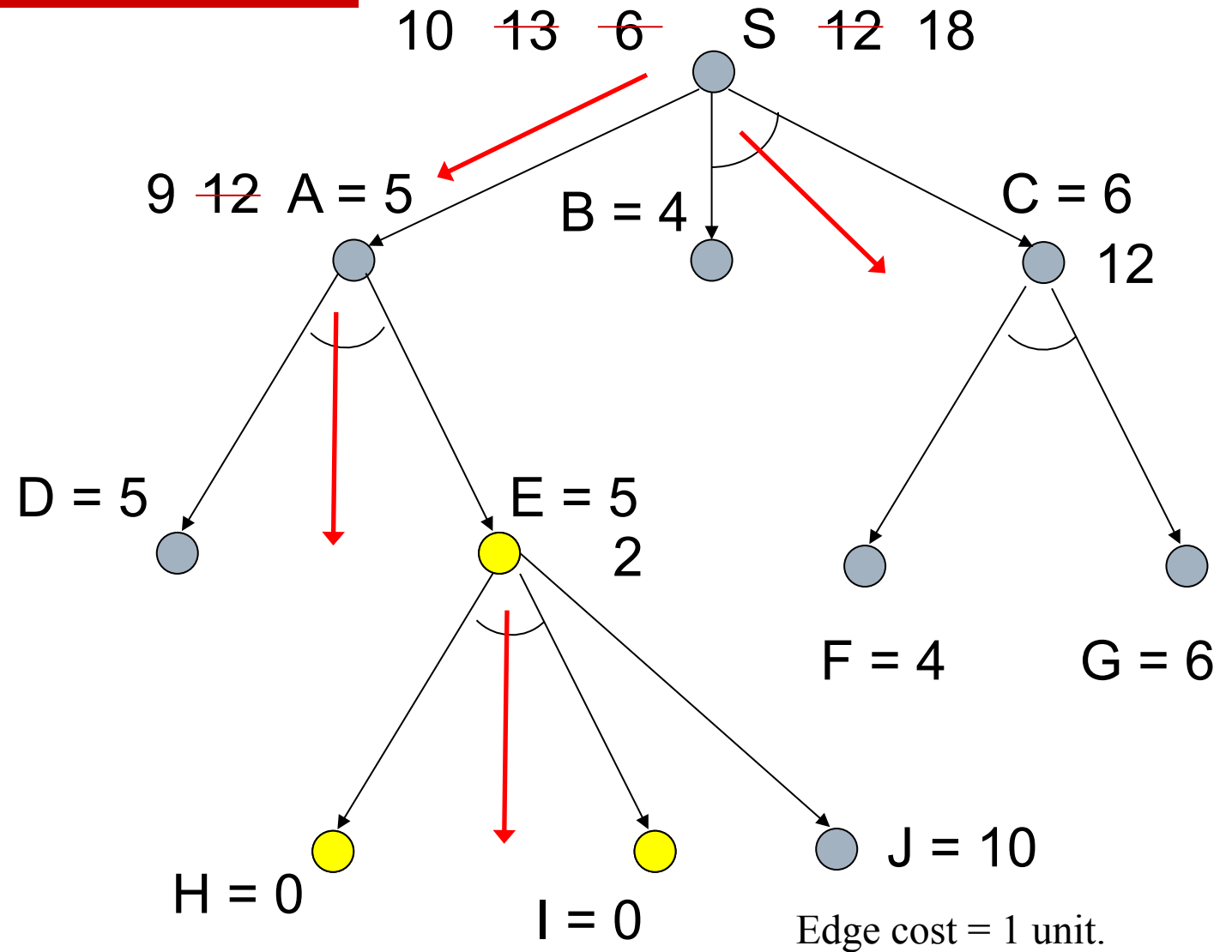
How promising?

Evaluate h values – back-up



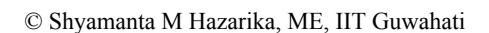
# AO\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – H-I; J  
SOLVE - Labelling



## SOLVE - Labelling

## Sub-tree at Node D SOLVED!

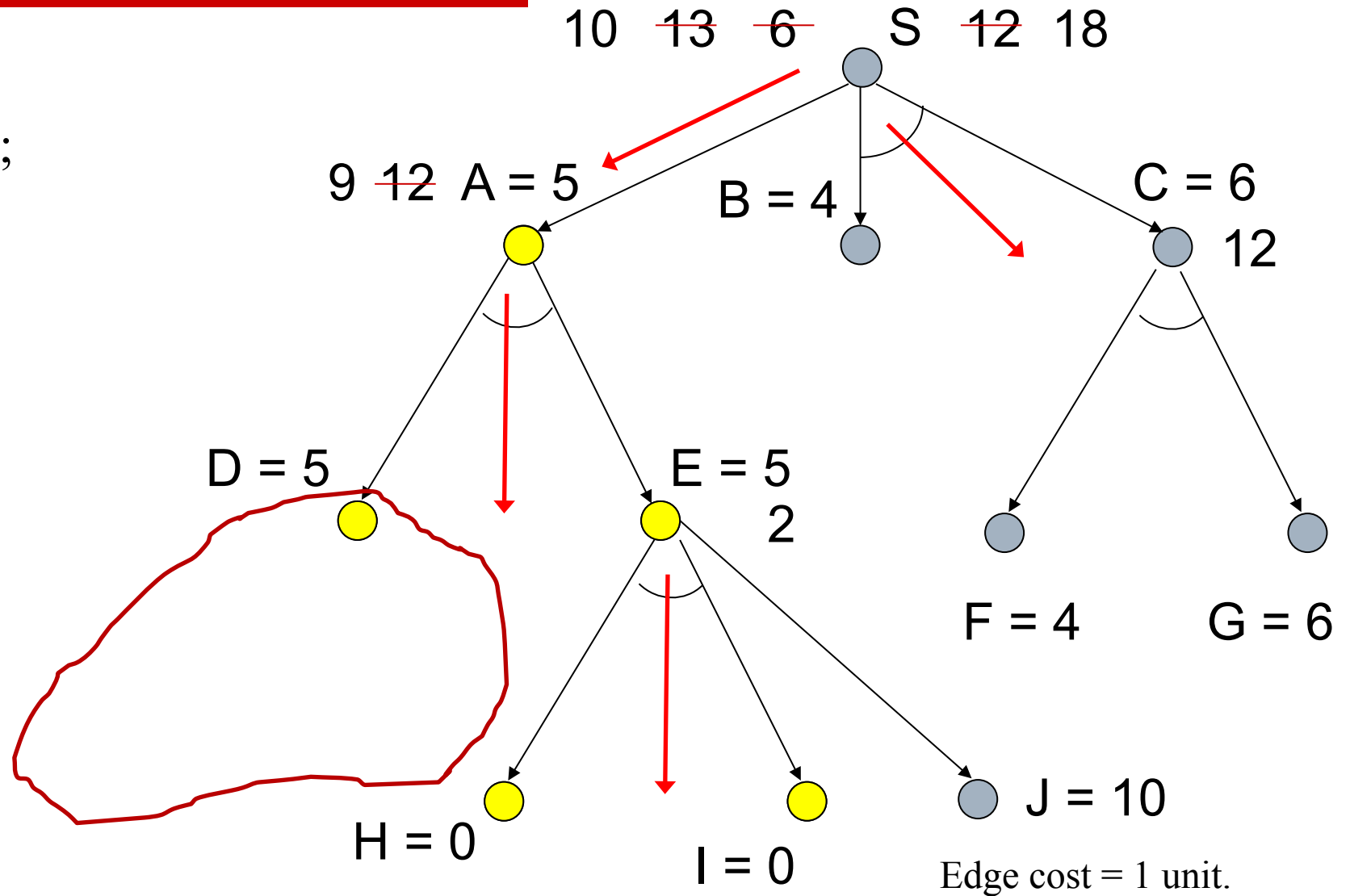


# A0\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – H-I; J

# SOLVE - Labelling

## Sub-tree at Node D SOLVED!

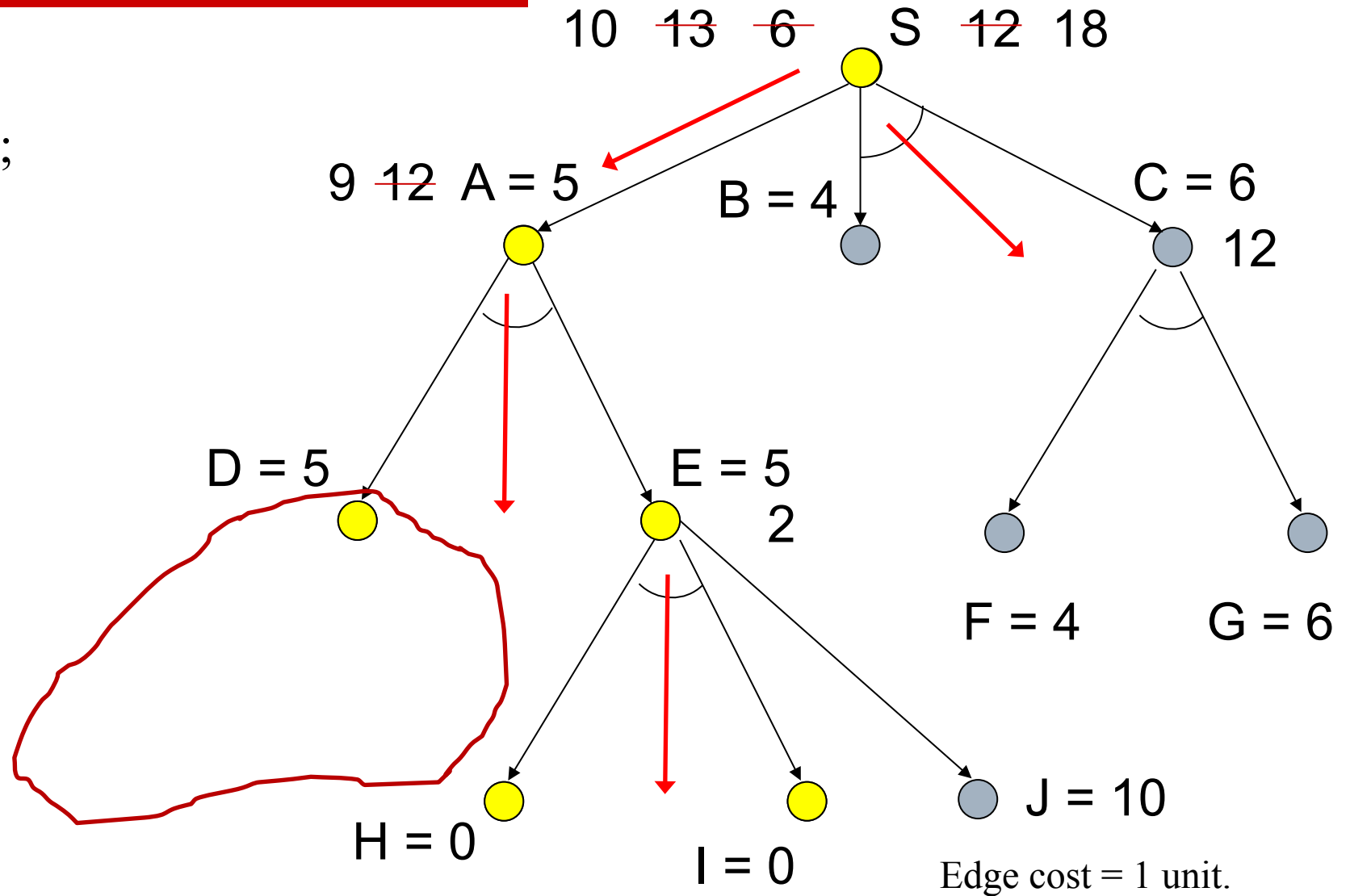


# AO\*: A Heuristic Search Procedure

Node E is expanded;  
Nodes – H-I; J

SOLVE - Labelling

Sub-tree at Node D  
SOLVED!



# Basic Idea of AO\*



- Bottom-up cost-revising, connector-marking
  - SOLVE-labeling.
  - If a direction has **all successors SOLVED** then **n is marked SOLVED**.
  - Starting with the node just expanded, the **procedure revises its cost** (using the newly computed cost of its successors).
    - Marks the outgoing connector on the estimated best path to terminal nodes.
    - Propagated upward in the graph.



# AO\* Algorithm



Create a search graph  $G = \langle s \rangle$  ;  $q(s) = h(s)$  ; If  $s$  is a terminal node, mark  $s$  SOLVED

Until  $s$  labeled SOLVED do:

begin

Top-down graph growing - picks out best available partial solution sub-graph

- a. **Compute  $G'$  partial solution graph in  $G$**  by tracing down marked connectors in  $G$  from  $s$ .
- b. **Select any nonterminal leaf node  $n$  of  $G'$ .**
- c. **Expand  $n$** , place successors in  $G$ , For each successor not already in  $G$  let  $q(\text{successor}) = h(\text{successor})$ . Label SOLVED all successors that are terminal nodes.

d. Let  $S := \{n\}$ .

Until  $S$  is empty do :

begin

a. Remove a node,  $m$ , from  $S$  which has no descendent in  $G$  also in  $S$  (minimal node).

b. Revise cost for  $m$

$$q(m) = \min [c + q(n_1) + \dots + q(n_k)]$$

**Mark chosen connector.**

If all successors through the connectors are SOLVED then mark  $m$  SOLVED.

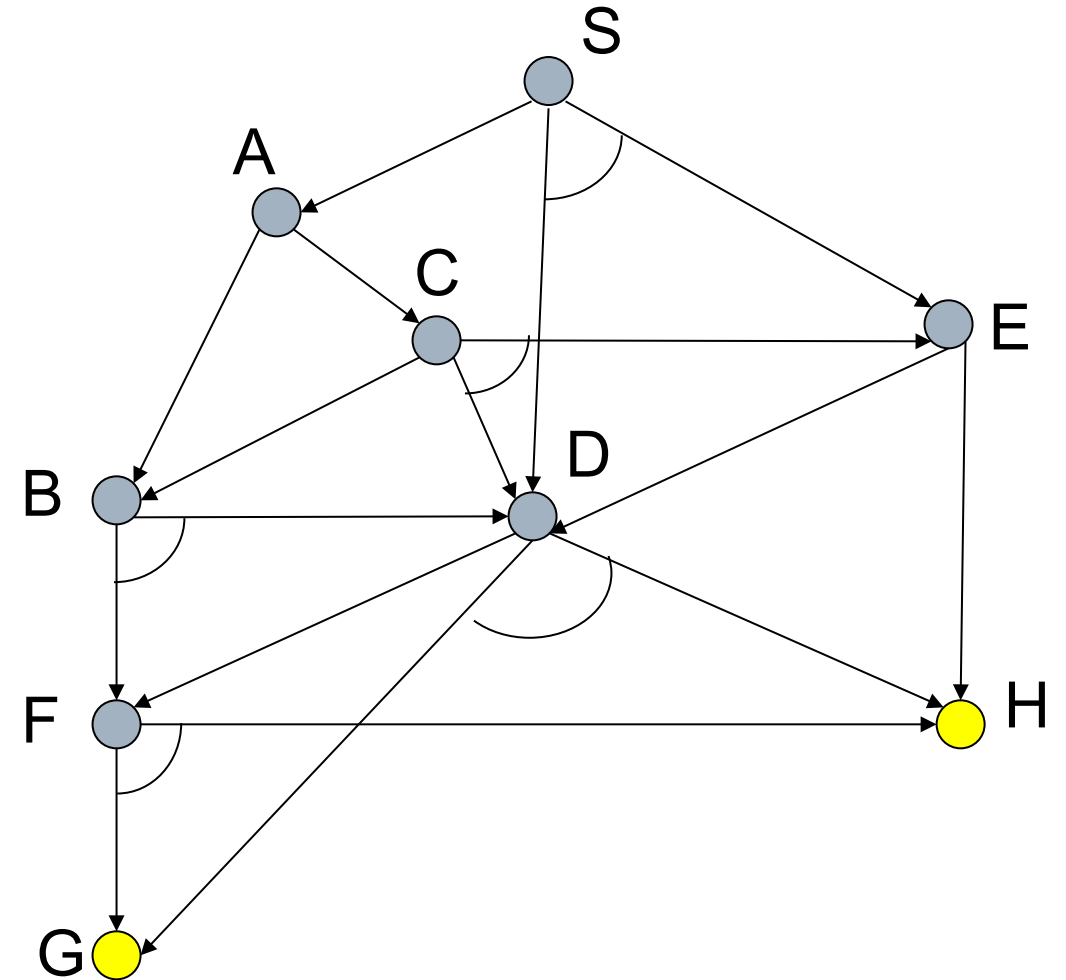
c. If  $m$  SOLVED or changed  $q(m)$  then add to  $S$  all “preferred” parents of  $m$ .

End.

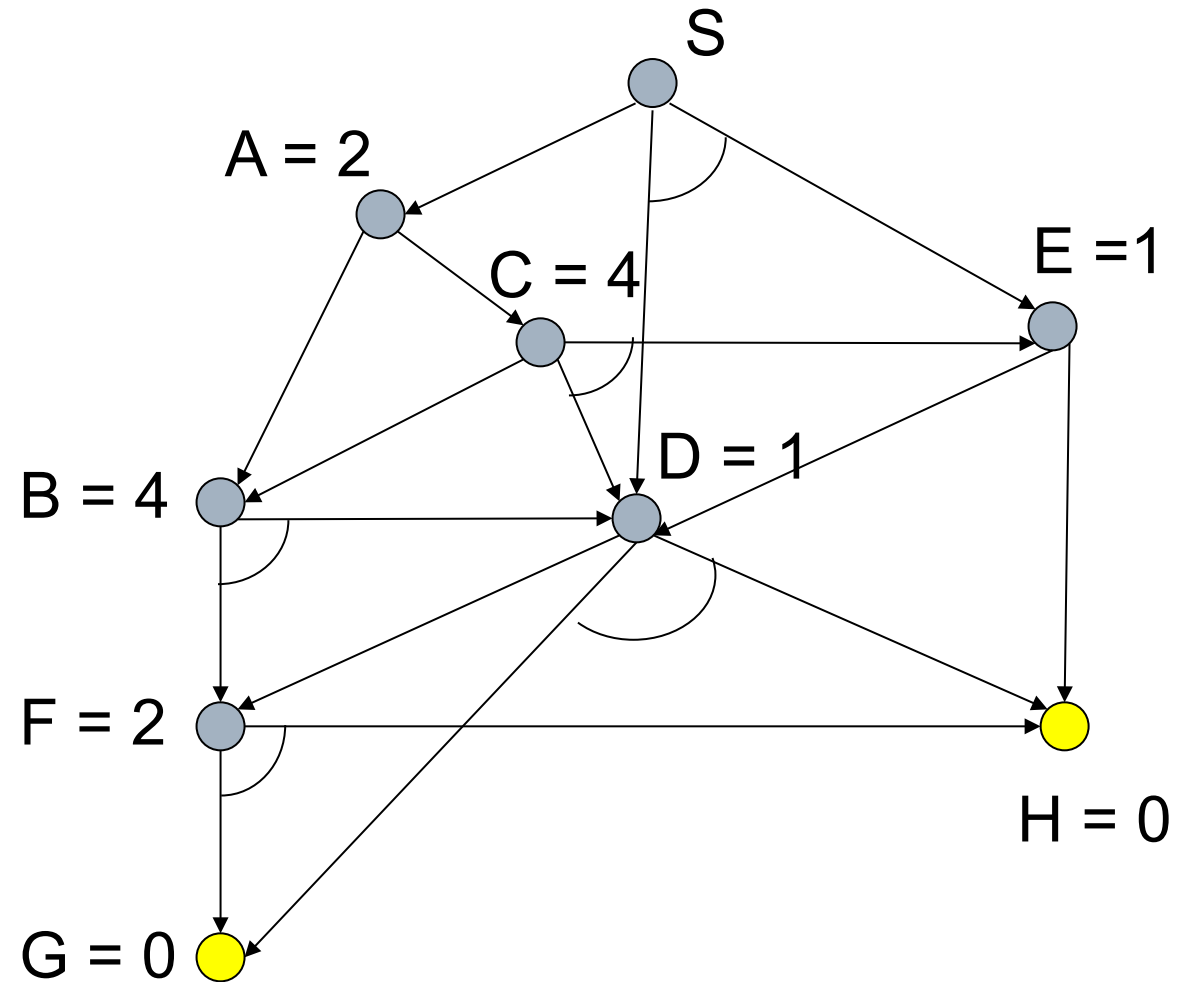
End.

Bottom-up, cost-revising, connector marking; SOLVED labelling procedure

# Tracing A0\*

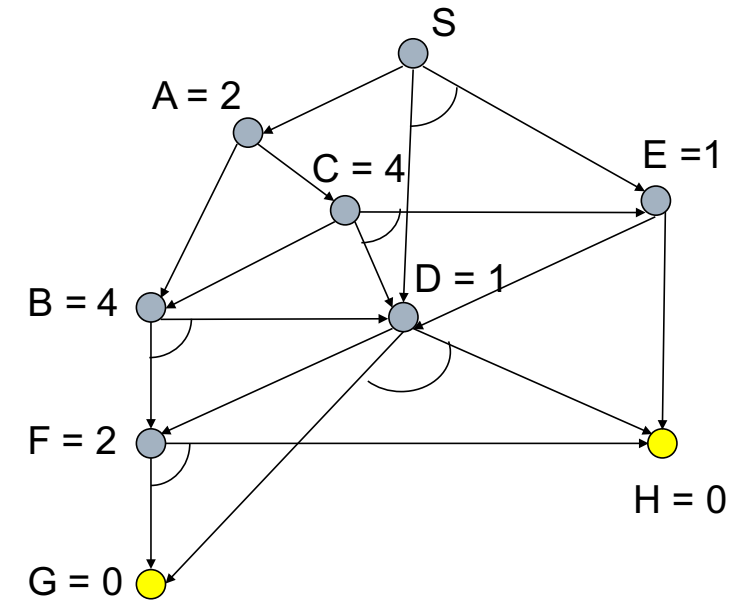
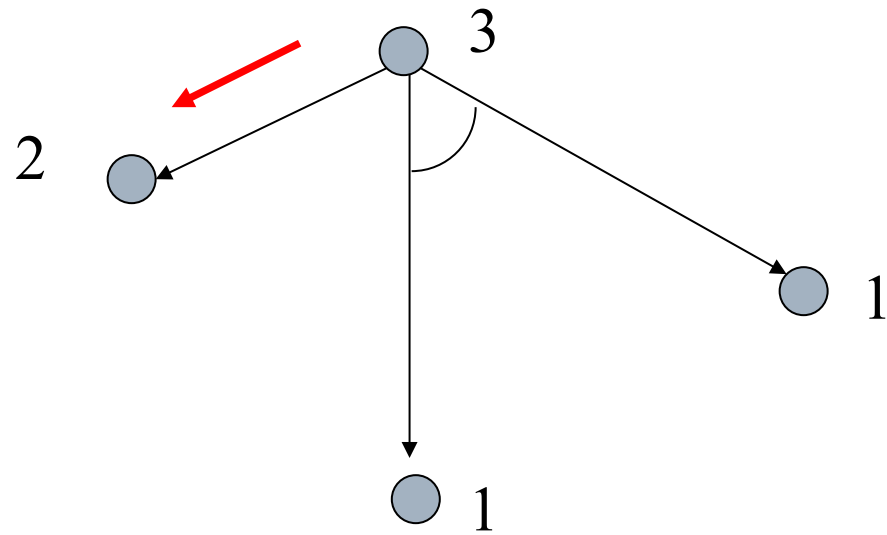


# Tracing A0\*

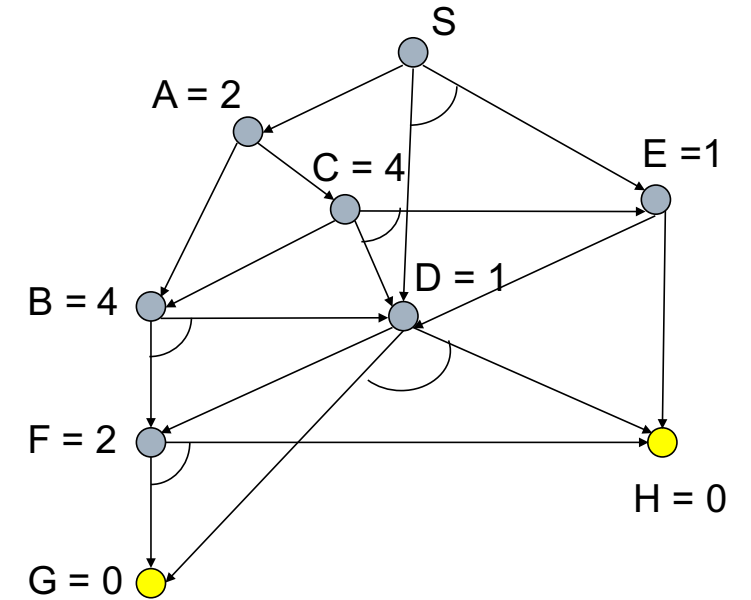
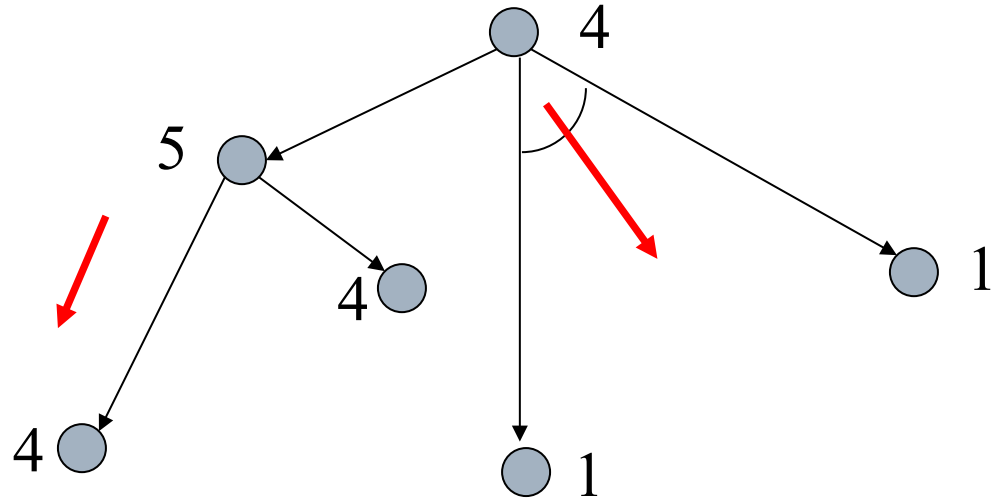


Nils J. Nilsson, Principles of Artificial Intelligence, Narosa.  
Chapter 3, Pages 107-108.

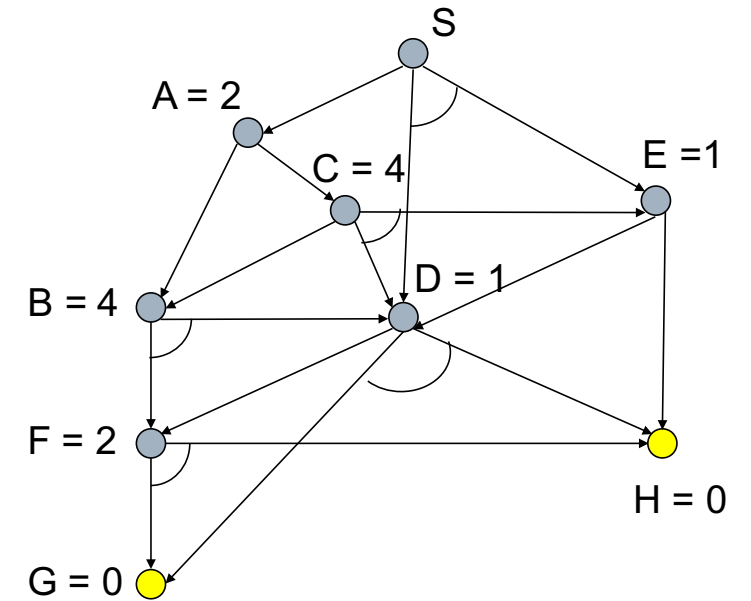
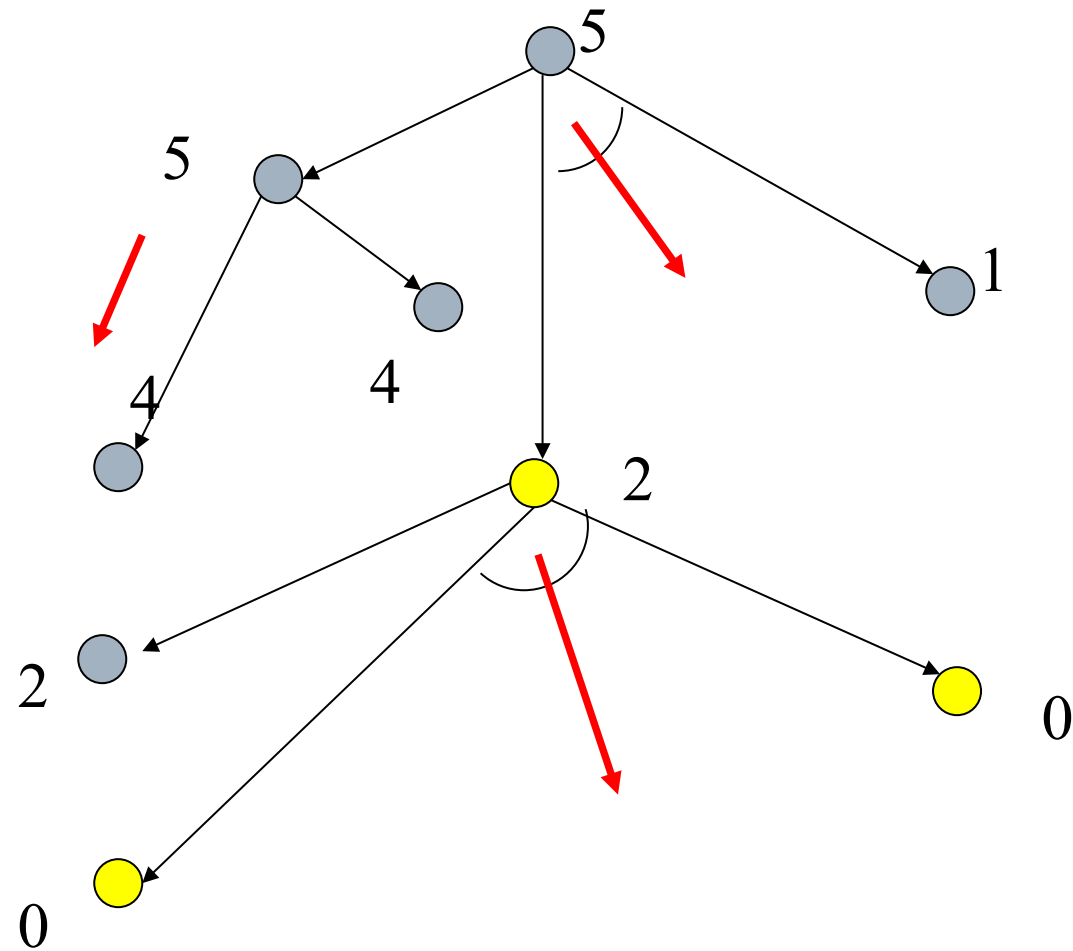
# Tracing AO\*



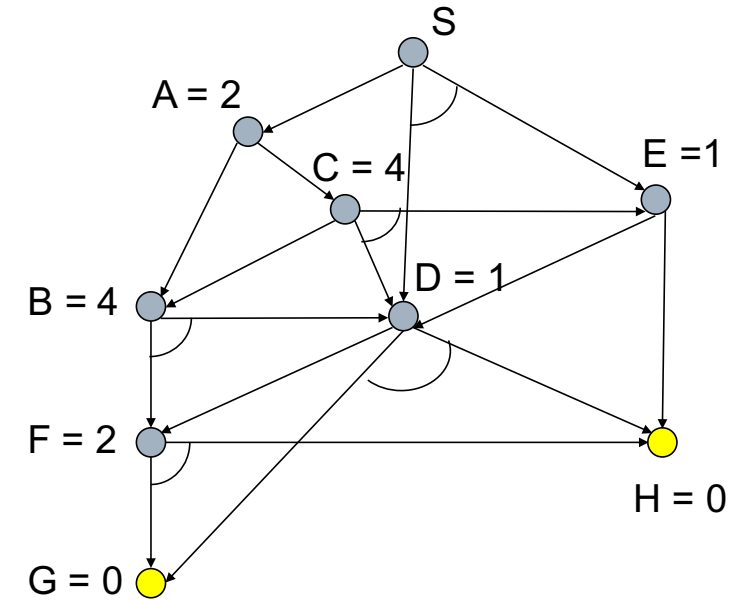
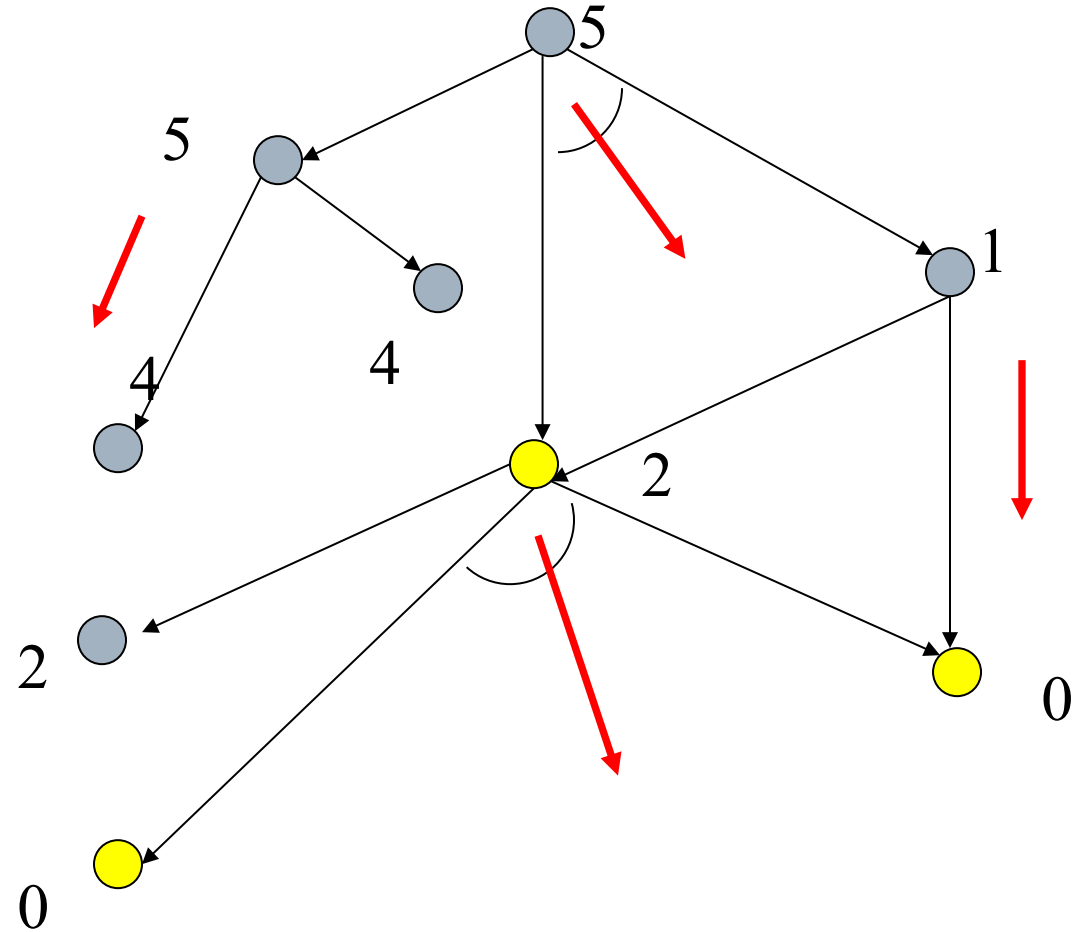
# Tracing AO\*



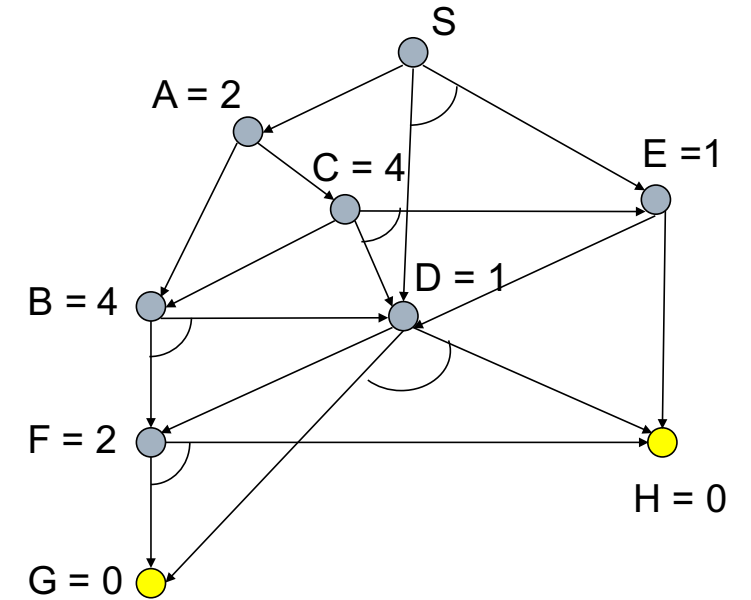
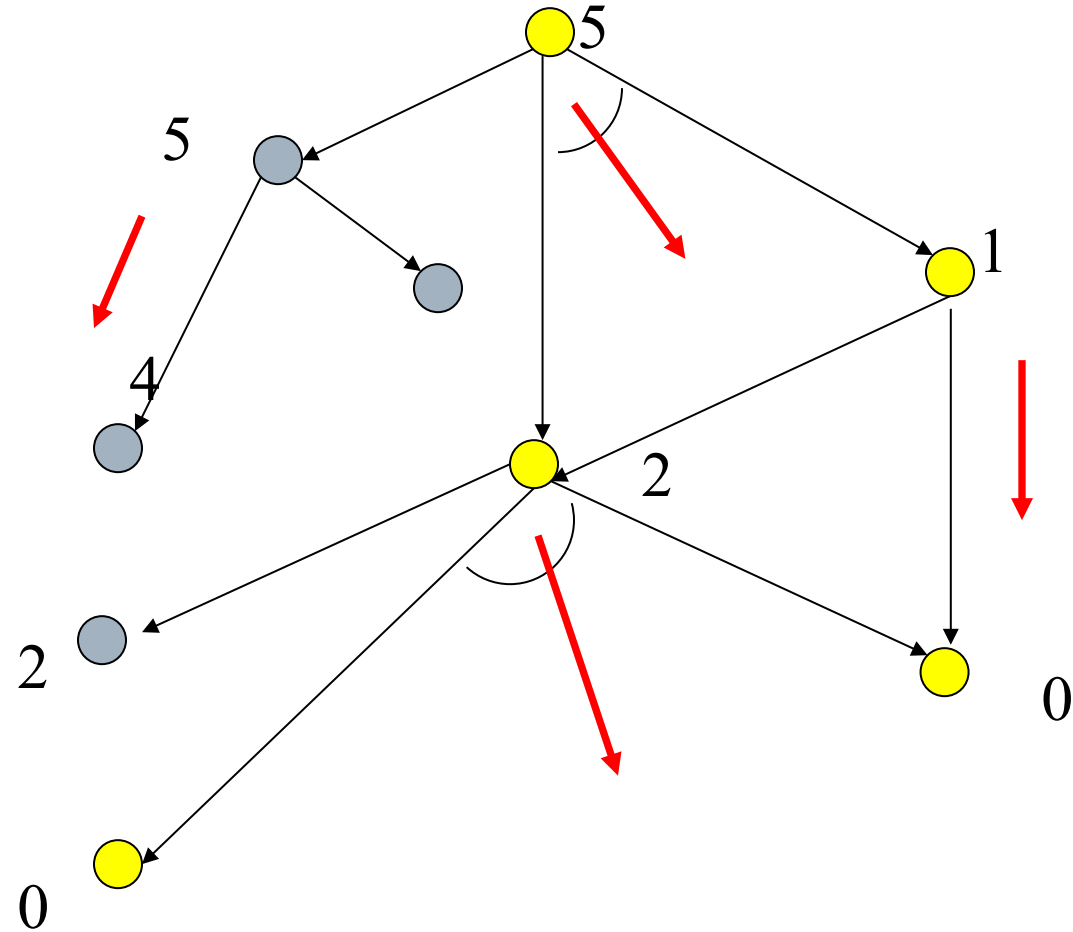
# Tracing A0\*



# Tracing A0\*



# Tracing A0\*





# AO\* Modifications

- AO\* may be modified in a variety of ways to render it more practical.
  - Rather than recompute a new estimated partial solution graph after every node expansion; expand one or more leaf nodes and some number of their descendants all at once. Recompute an estimated best partial solution graph.
    - This would reduce the overhead of frequent bottom-up operations.
- Staged-search strategy may be used for AND-OR graphs
  - Identify few partial solution graphs having largest estimated cost; these can be discarded periodically.
    - Risk of discarding one that might turn out to be the top of an optimal solution graph.