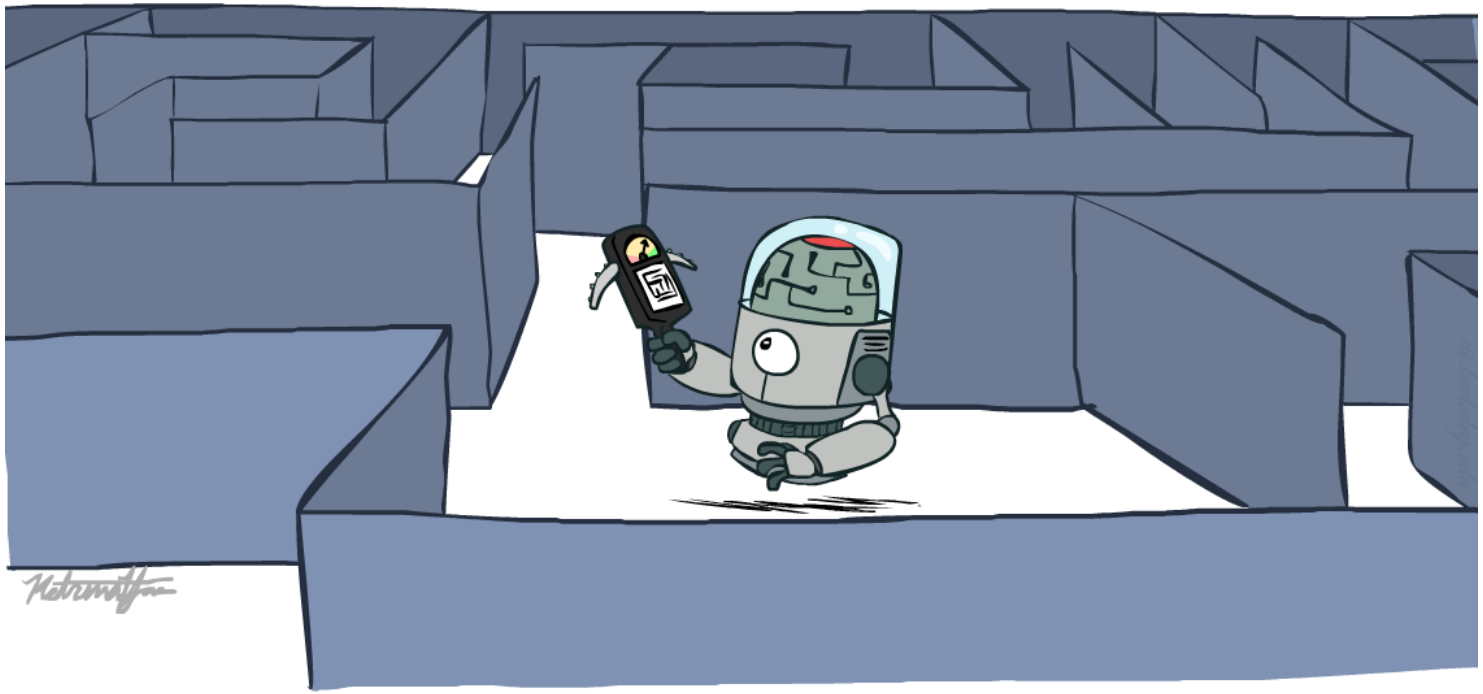
40417: Artificial Intelligence

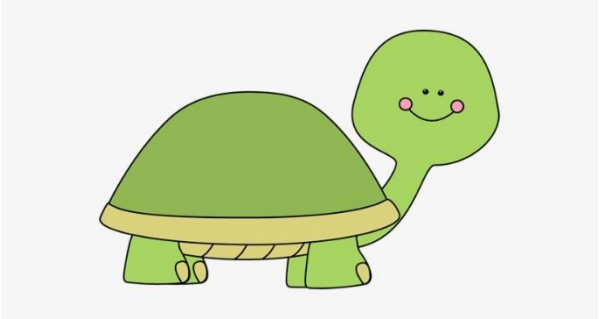
Informed Search (Part-II)



Instructor: Gholamreza Ghassem-Sani

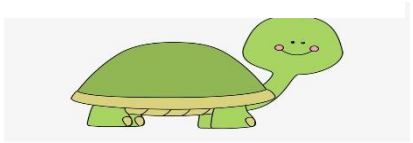
Sharif University of Technology

A* Search



UCS

Greedy



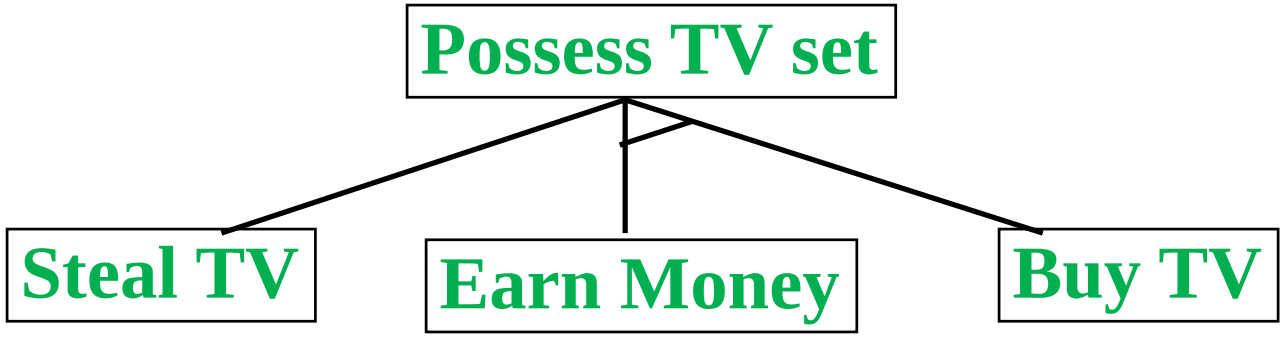
A*

Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

AND/OR Trees

- Some problems are best represented as achieving subgoals, some of which achieved simultaneously and independently (AND nodes)
- Up to now, we only dealt with OR nodes



Searching AND/OR Trees

- A solution in an AND-OR tree is a *sub tree* whose *leafs* are included in the goal set

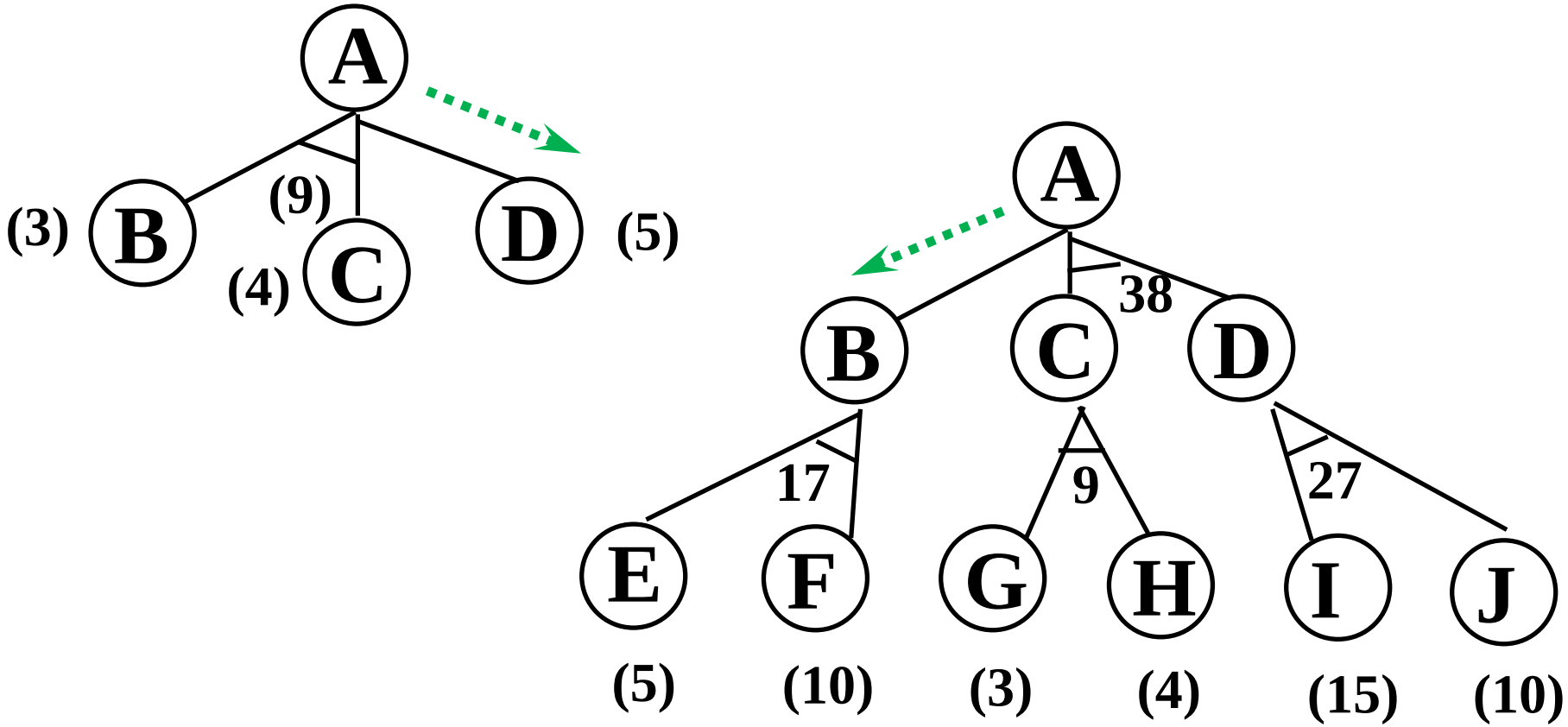
- Cost function: sum of costs in AND node

$$f(n) = f(n_1) + f(n_2) + ... + f(n_k)$$

- How can we extend A* to search AND/OR trees?
  - The AO* algorithm.

AND/OR Tree Search

- We must examine several nodes simultaneously when choosing the next move

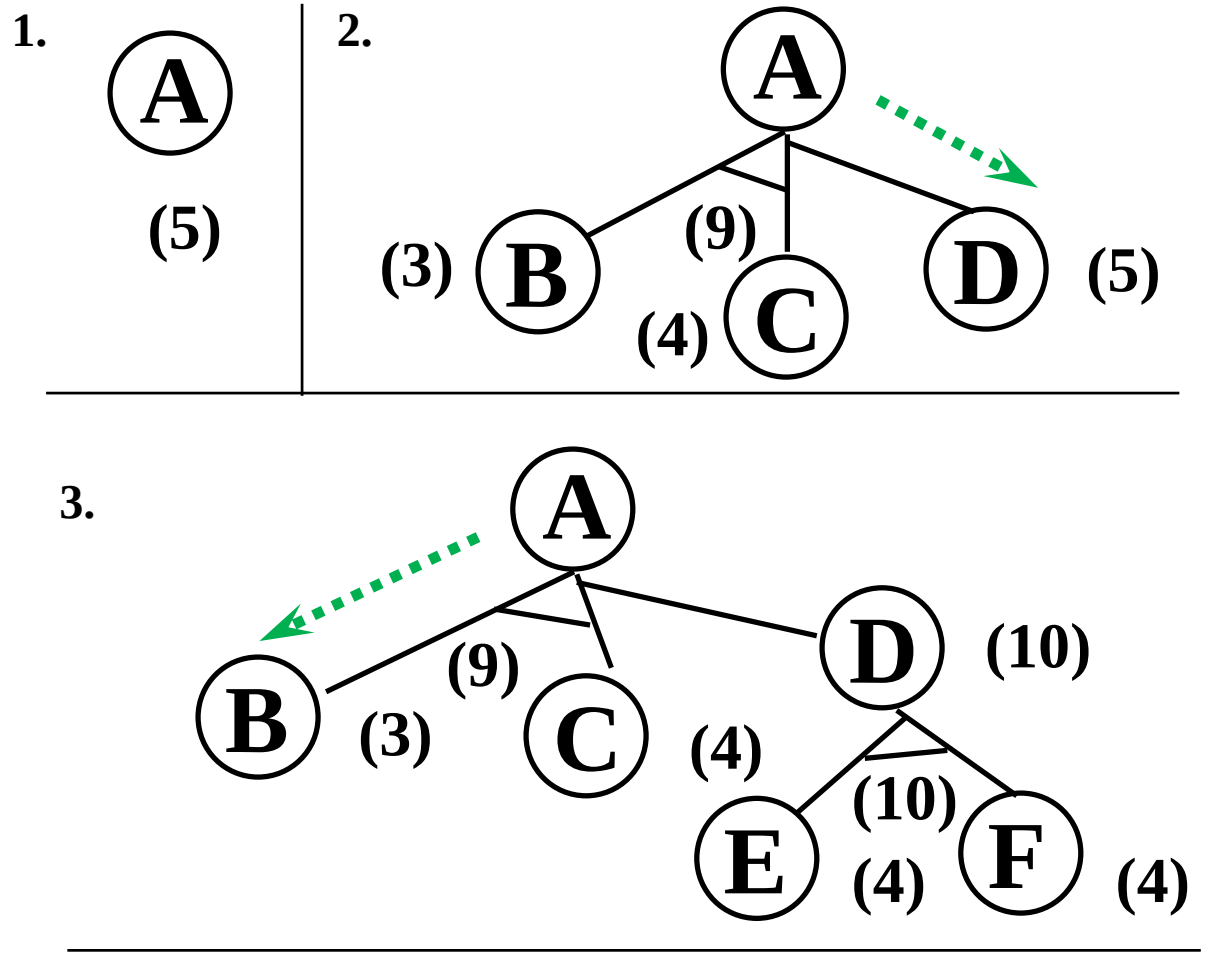


[40-417, SUT]

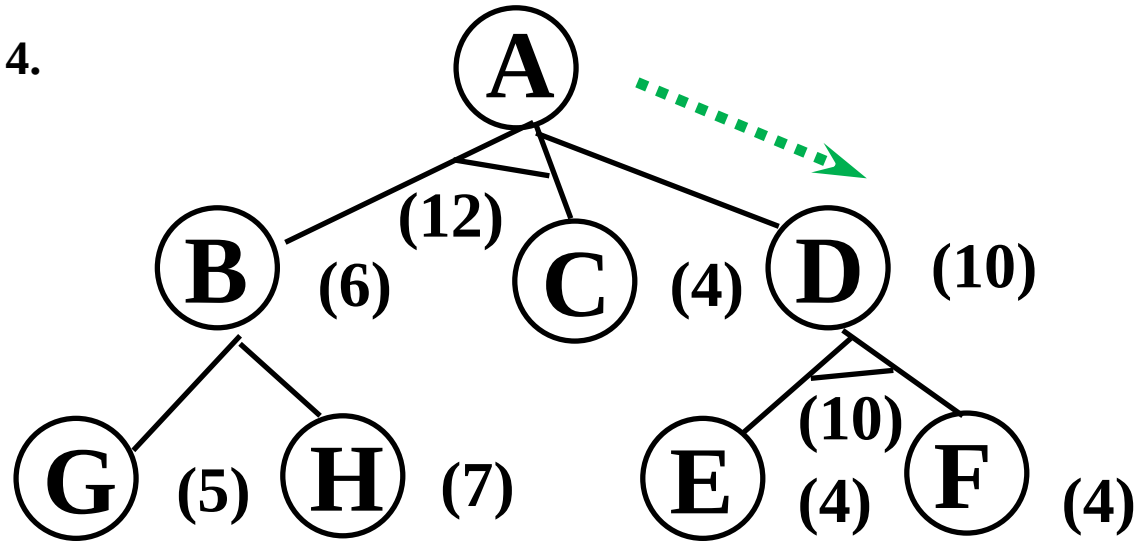
AND/OR Best-First-Search

- Traverse the graph (from the initial node) following the best current path.
- Pick one of the unexpanded nodes on that path and expand it. Add its successors to the graph and compute *f* for each of them
- Change the expanded node's *f*-value to reflect its successors. Propagate the change up the graph.
- Reconsider the current best solution and repeat until a solution is found

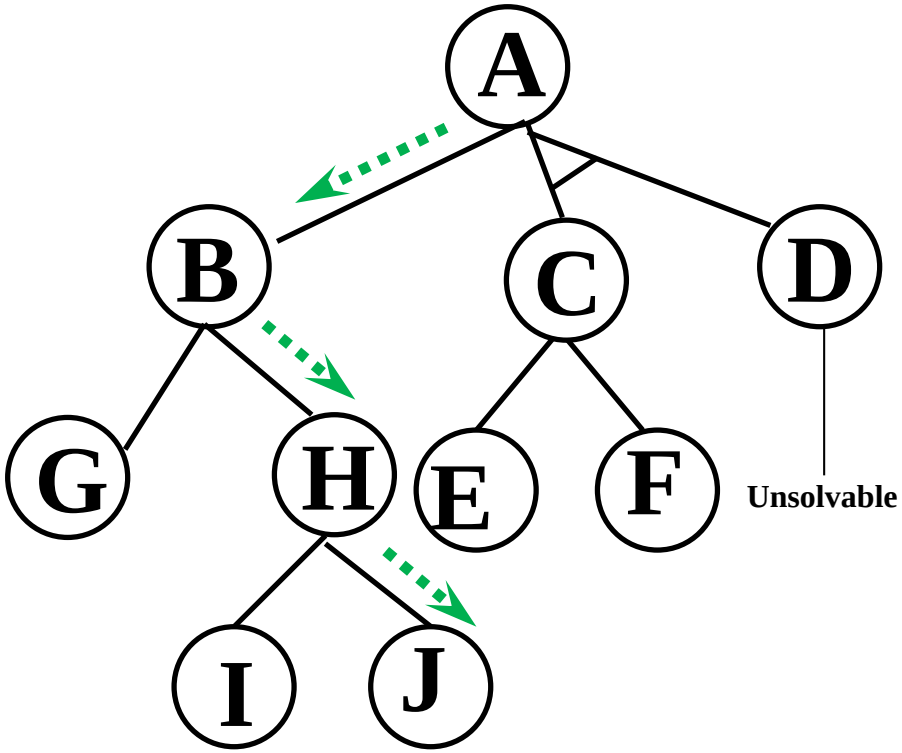
AND/OR Search example (Cont.)

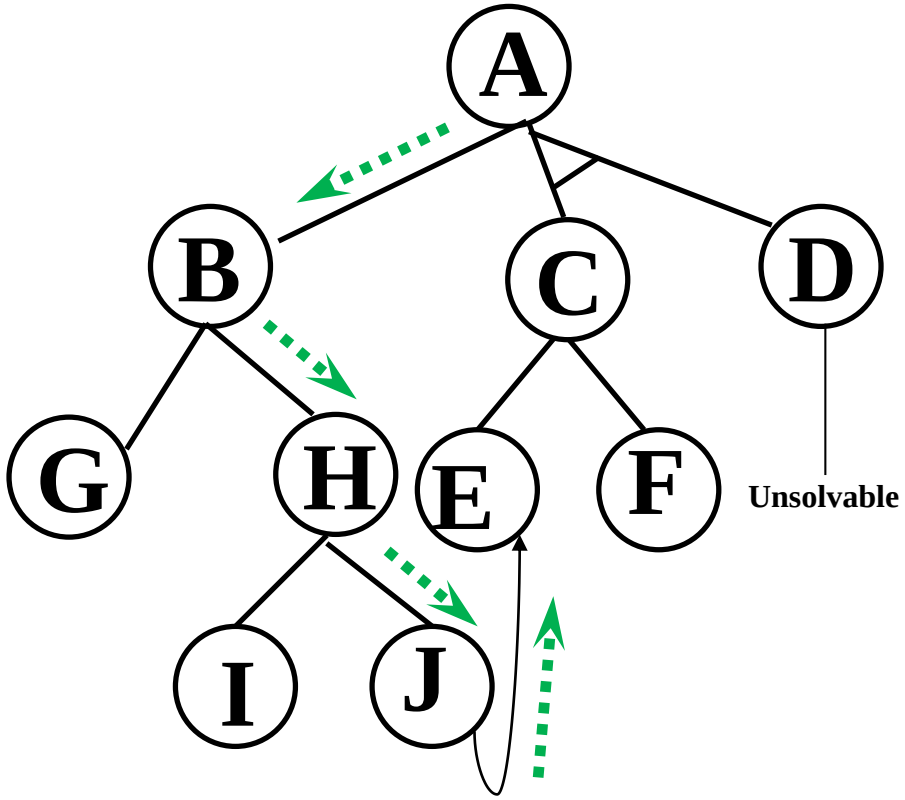


AND/OR Search example



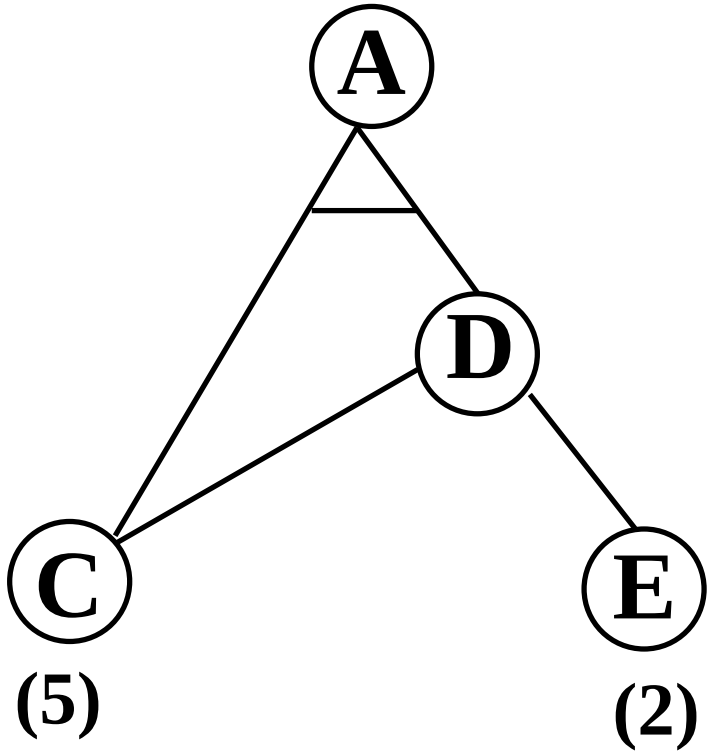
A Longer Path May Be Better





[40-417, SUT]

Interacting Sub Goals



[40-417, SUT]

AO* Algorithm

1. Let G be a graph with only starting node *INIT.*
2. Repeat the followings until *INIT* is labeled SOLVED or h(*INIT*) > FUTILITY
   a) Select an unexpanded node N from the most promising path from INIT
   b) Generate successors of N. If there are none, set h(N) = FUTILITY; otherwise for each SUCCESSOR that is not an ancestor of N:
      i. Add SUCCESSSOR to G.
      ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.
      iii. If SUCCESSOR is not a terminal node, compute its h

AO* Algorithm (Cont.)

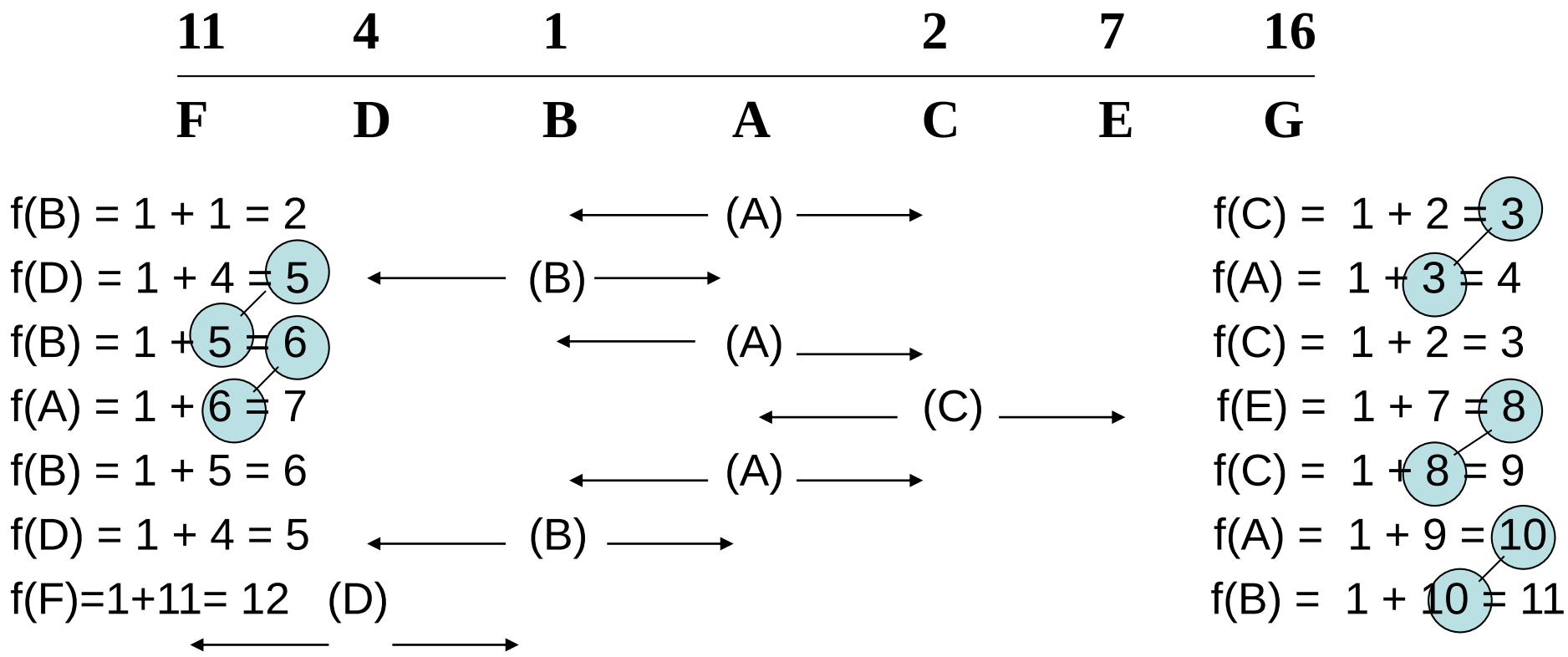
c) let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to N. Until S is empty repeat the followings:

i. Remove a node from S and call it C.
ii. Compute the cost of each of the arcs emerging from C. Assign minimum cost of its successors as its h.
iii. Mark the best path out of C by marking the arc that had the minimum cost in step ii
iv. Mark C as SOLVED if all of the nodes connected to it through new labeled arc have been labeled SOLVED
v. If C has been labeled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of C to S.

Real Time A* (Online Search)

- *Backtracking* has a uniform (or actual) cost
- Considers the cost (> 0) for switching from **one** branch to another (i.e., *backtracking*) in the search
- Uses a heuristic function similar to that of A*
  - $f(n) = g(n) + h(n)$
- But, g is calculated from the current state, instead of the initial state

Example: path finding in real life



[40-417, SUT]

Example 2

Current State = S

f(A) = 3 + 5 = 8

f(B) = 2 + 4 = 6

Current State = B

f(S) = 2 + 8 = 10

f(A) = 4 + 5 = 9

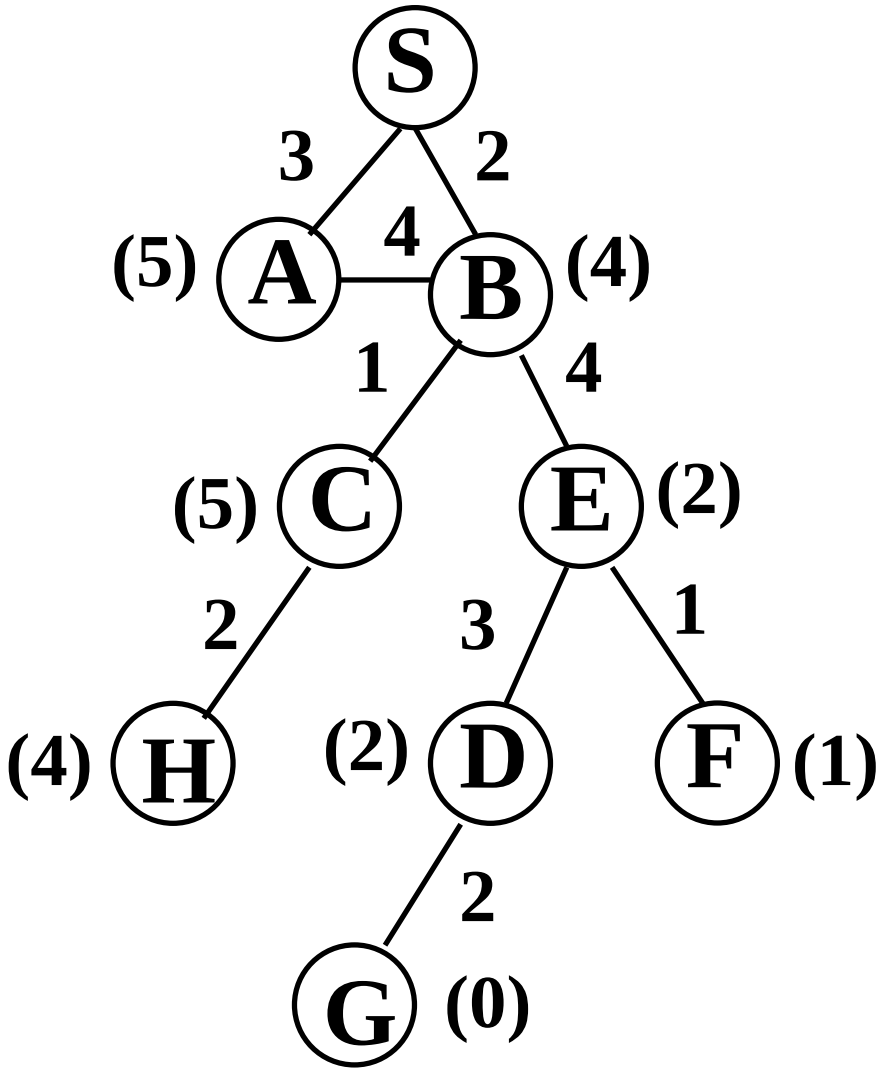
f(C) = 1 + 5 = 6

f(E) = 4 + 2 = 6

Current State = C

f(H) = 2 + 4 = 6

f(B) = 1 + 6 = 7



[40-417, SUT]

Example 2

Current State = H

f(C) = 2 + 7 = 9

Current State = C

f(B) = 1 + 6 = 7

f(H) = ∞

Current State = B

f(S) = 2 + 8 = 10

f(A) = 4 + 5 = 9

f(E) = 4 + 2 = 6

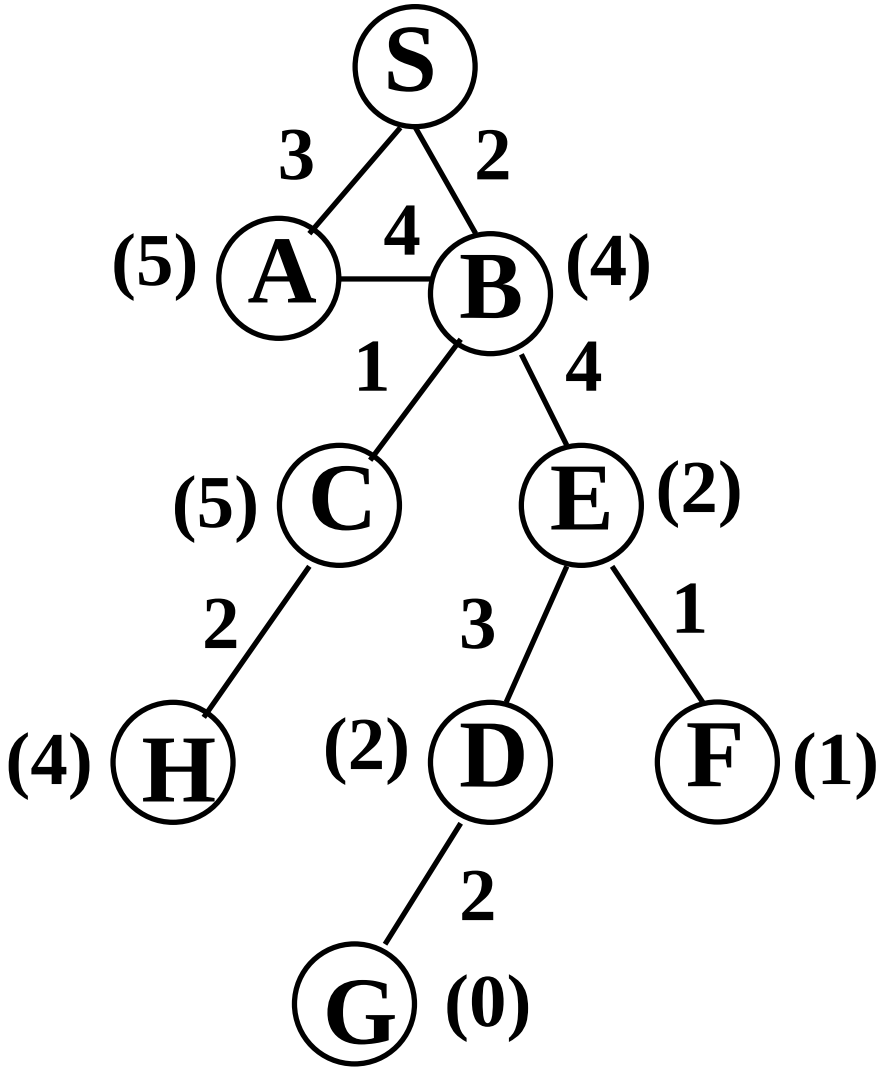
f(C) = ∞

Current State = E

f(B) = 4 + 9 = 13

f(D) = 3 + 2 = 5

f(F) = 1 + 1 = 2



Example 2

Current State = F

f(E) = 2 + 5 = 7

Current State = E

f(D) = 3 + 2 = 5

f(B) = 4 + 9 = 13

f(F) = ∞

Current State = D

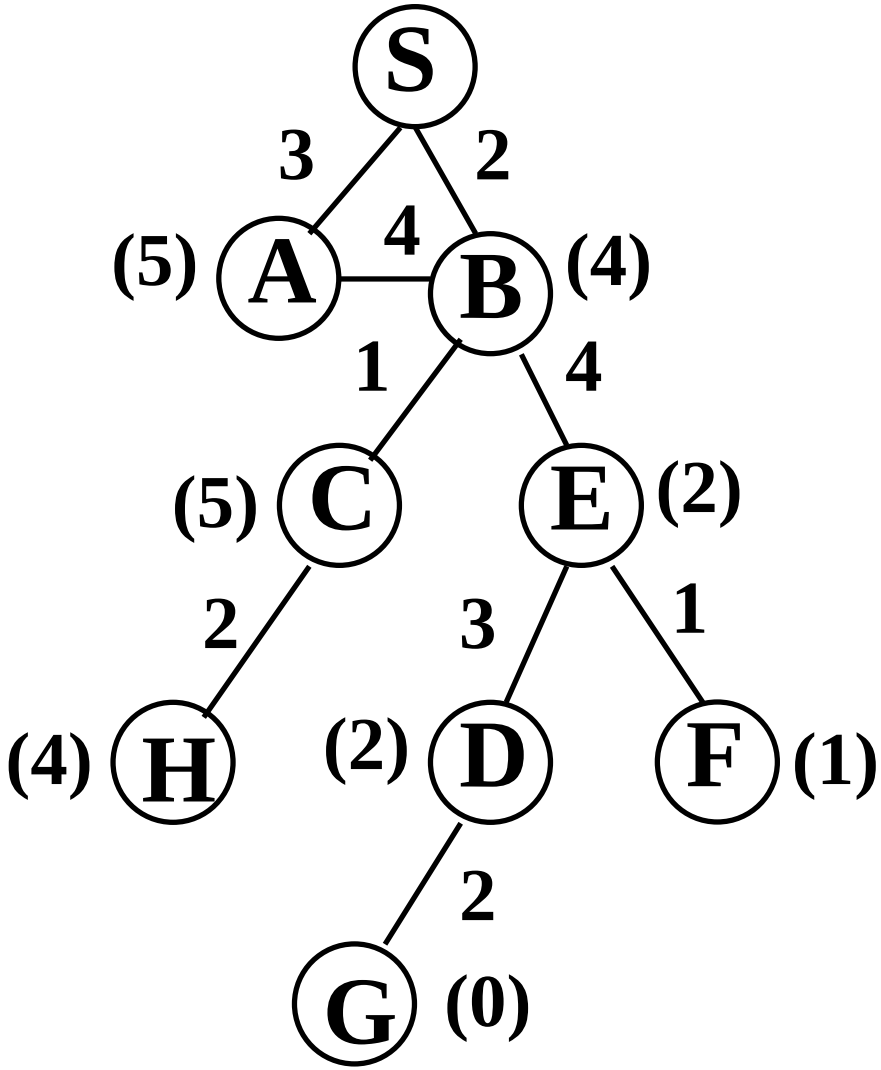
f(G) = 2 + 0 = 2

f(E) = 3 + 13 = 16

Visited Nodes =

S, B, C, H, C, B, E, F, E, D, G

Path = S, B, E, D, G



Real Time A* (Online Search)

- Maintains in a hash table a list of those states/nodes that have been visited by an *actual move* in the real world of the problem solver;
- At each cycle in the real-world, the current state is expanded and the heuristic function, possibly augmented by look-ahead search, is applied to each successor state which is **not in the hash table**;
- The f value of each neighboring state is computed by adding the h value plus the cost of the link to the current state;
- The neighbor with the minimum f value is chosen for the current state;
- The **second best f value is stored in the hash table** for the current state
  - Represents the estimated h cost of solving the problem by returning to this state
  - Second best avoids loops

[V. Lesser, CS683]