Introduction to Computational Intelligence

Lecture 4

Announcement

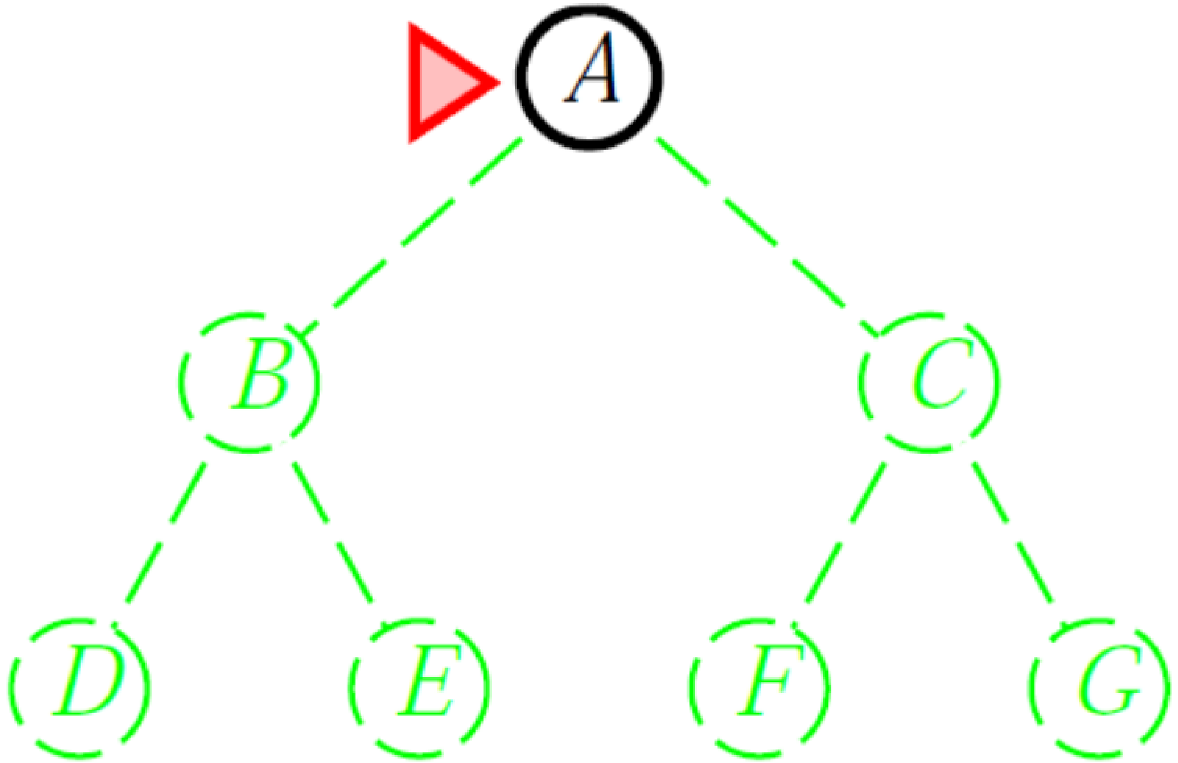
- Project initial submission
- Lab task (next Monday)

Outline

- Searching for solutions
- Properties of search strategies
- Uninformed search strategies
  - BFS
  - Uniform cost search
  - Depth first search
  - Iterative deepening search
- Informed search strategies
  - Greedy best-first search
  - A* search

BFS – FIFO Implementation

- What would be the progress of breadth-first search if its frontier is implemented using a FIFO queue?



- Maker indicate the node to be expanded next.

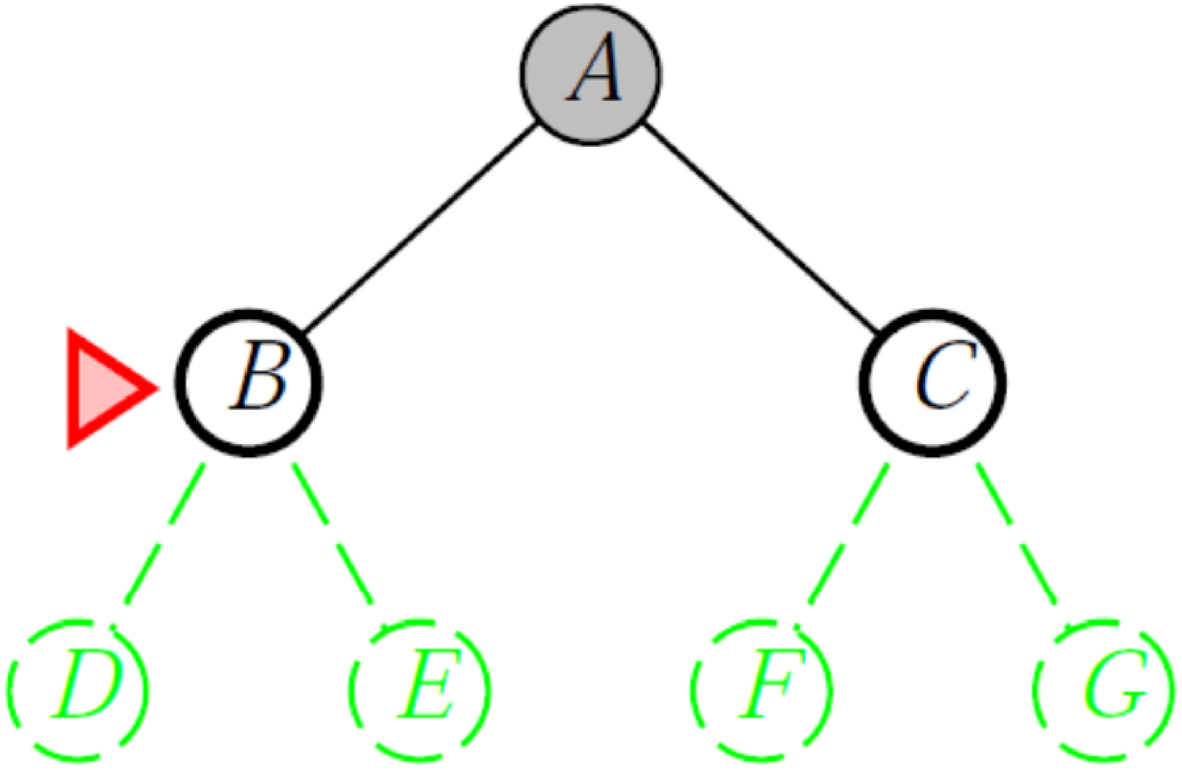
A			

➢ Frontier queue

Image adopted from Nilufer Onder

BFS – FIFO Implementation Cont.

- What would be the progress of breadth-first search if its frontier is implemented using a FIFO queue?



- Maker indicate the node to be expanded next.
- The nodes that are already explored are gray.
- The nodes with dashed lines are not generated yet.

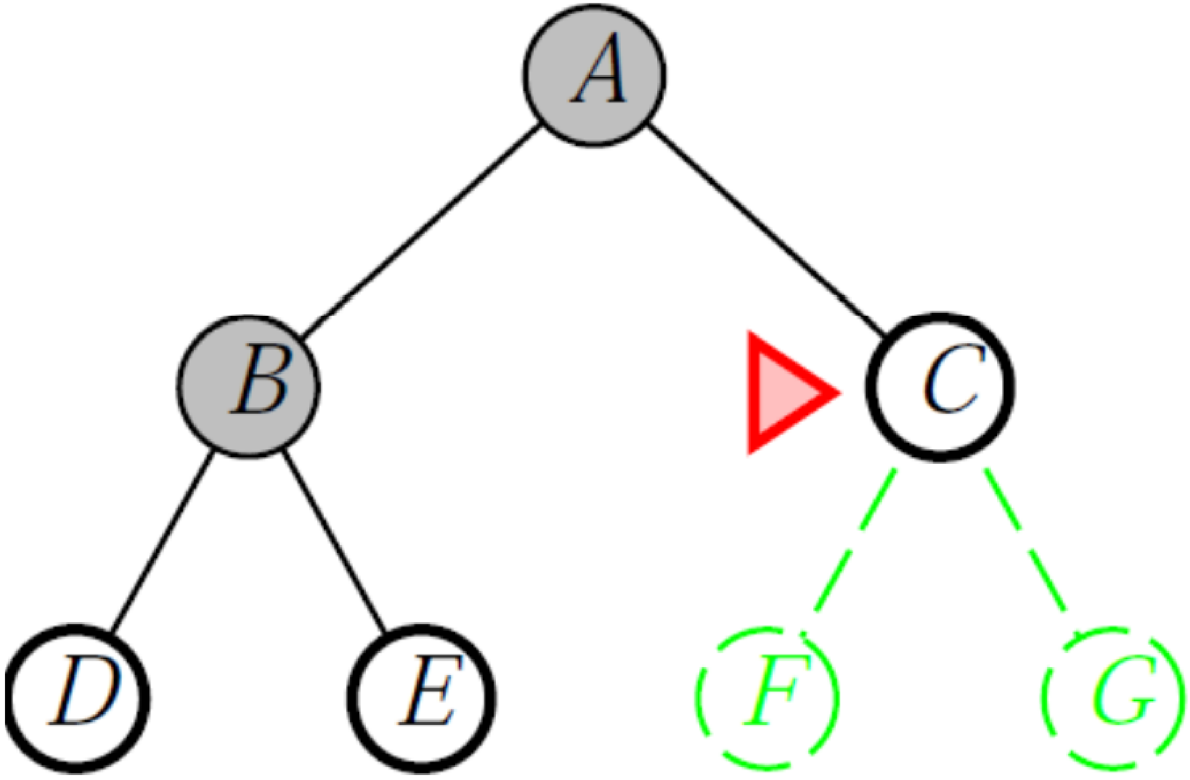
B	C		

- ➢ As the node (A) is explored, it is removed from the queue
- ➢ And the successors (newly generated nodes: B, C) are added to the frontier's end queue

Animation adopted from Nilufer Onder

BFS – FIFO Implementation Cont.

- What would be the progress of breadth-first search if its frontier is implemented using a FIFO queue?



- Maker indicate the node to be expanded next.
- The nodes that are already explored are gray.
- The nodes with dashed lines are not generated yet.

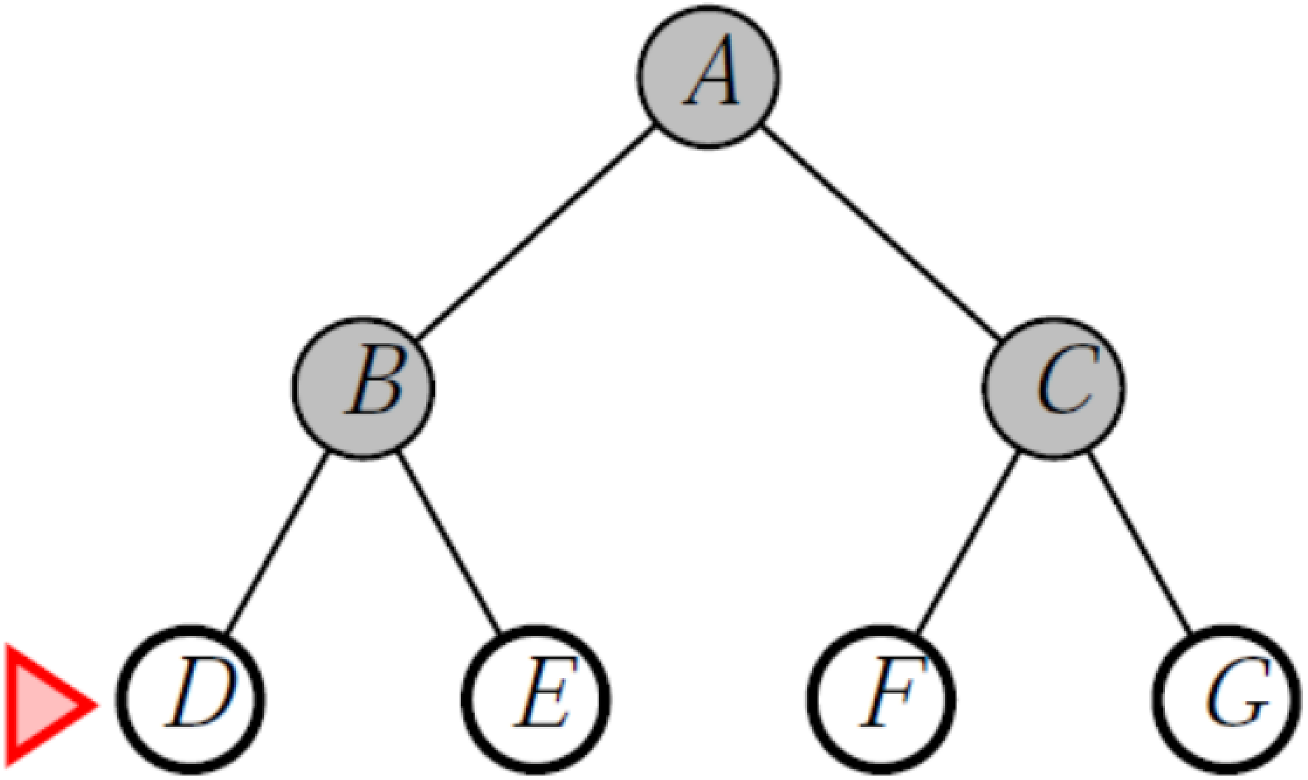
C	D	E	

- ➢ As the node (B) is explored, it is removed from the queue
- ➢ And the successors (newly generated nodes: D, E) are added to the frontier's end queue

Animation adopted from Nilufer Onder

BFS – FIFO Implementation Cont.

- What would be the progress of breadth-first search if its frontier is implemented using a FIFO queue?



- Maker indicate the node to be expanded next.
- The nodes that are already explored are gray.
- The nodes with dashed lines are not generated yet.

D	E	F	G

- ➢ As the node (C) is explored, it is removed from the queue
- ➢ And the successors (newly generated nodes: F, G) are added to the frontier's end queue

Image adopted from Nilufer Onder

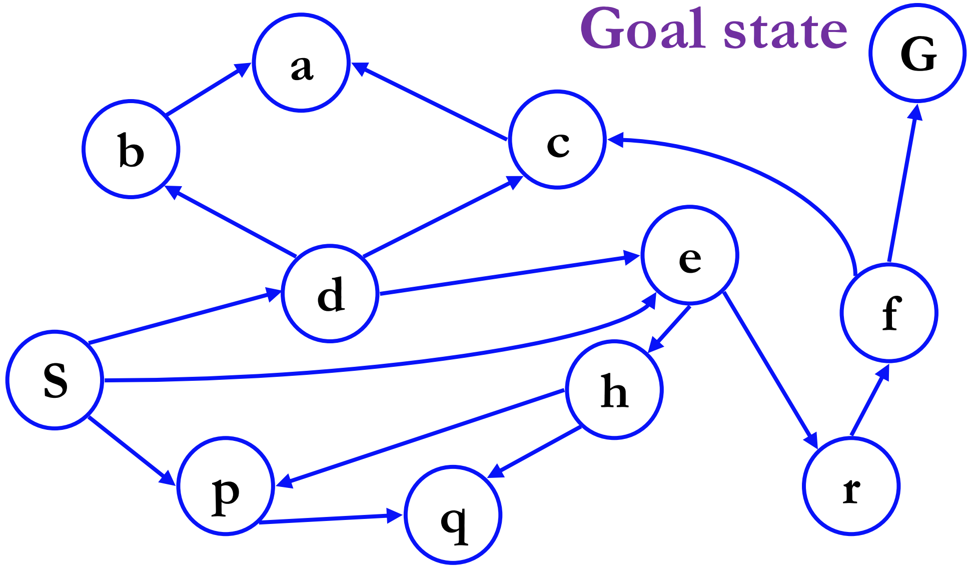
Breadth-First Search Cont.

- What is the order/uniformity of expanding the search tree using BFS?
  - expanding the **lowest-depth** node in the frontier

- How about if we expand the next node with the lowest-cost?
  - expanding nodes in the order of **lowest path cost** → Uniform Cost Search

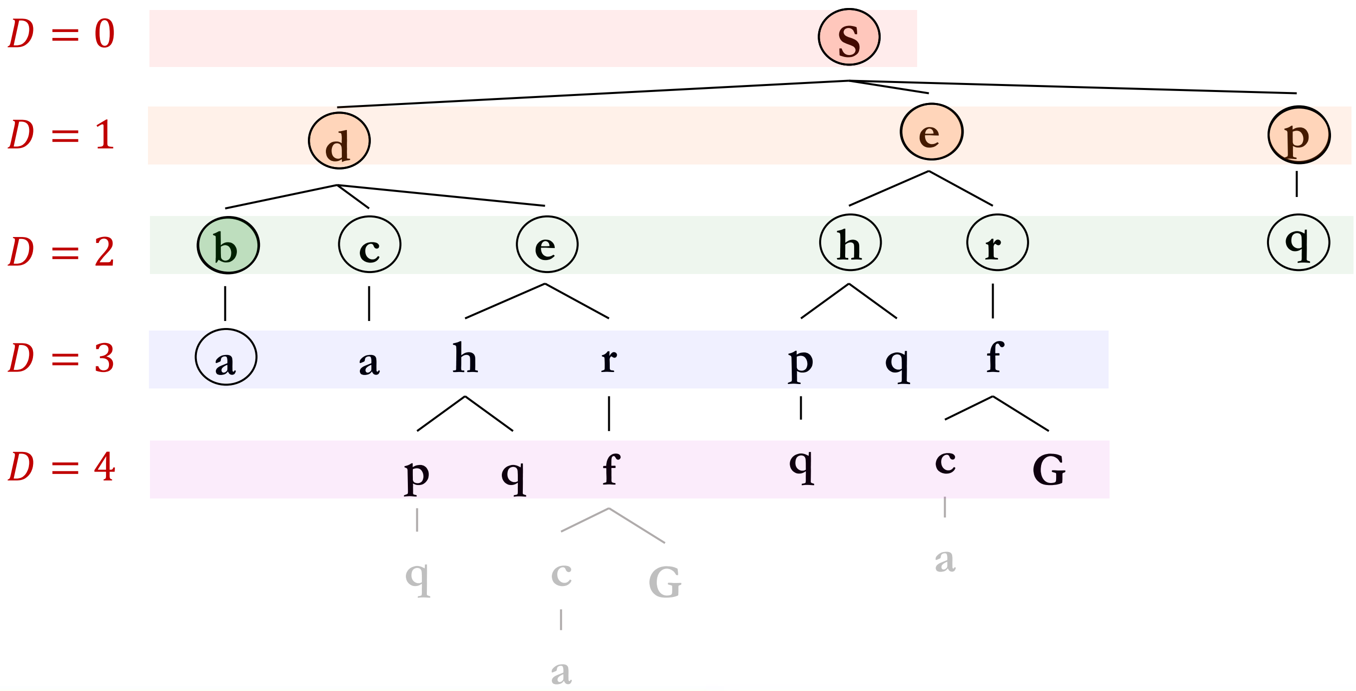
Cost-Sensitive Search: BFS vs UCS

- An example:

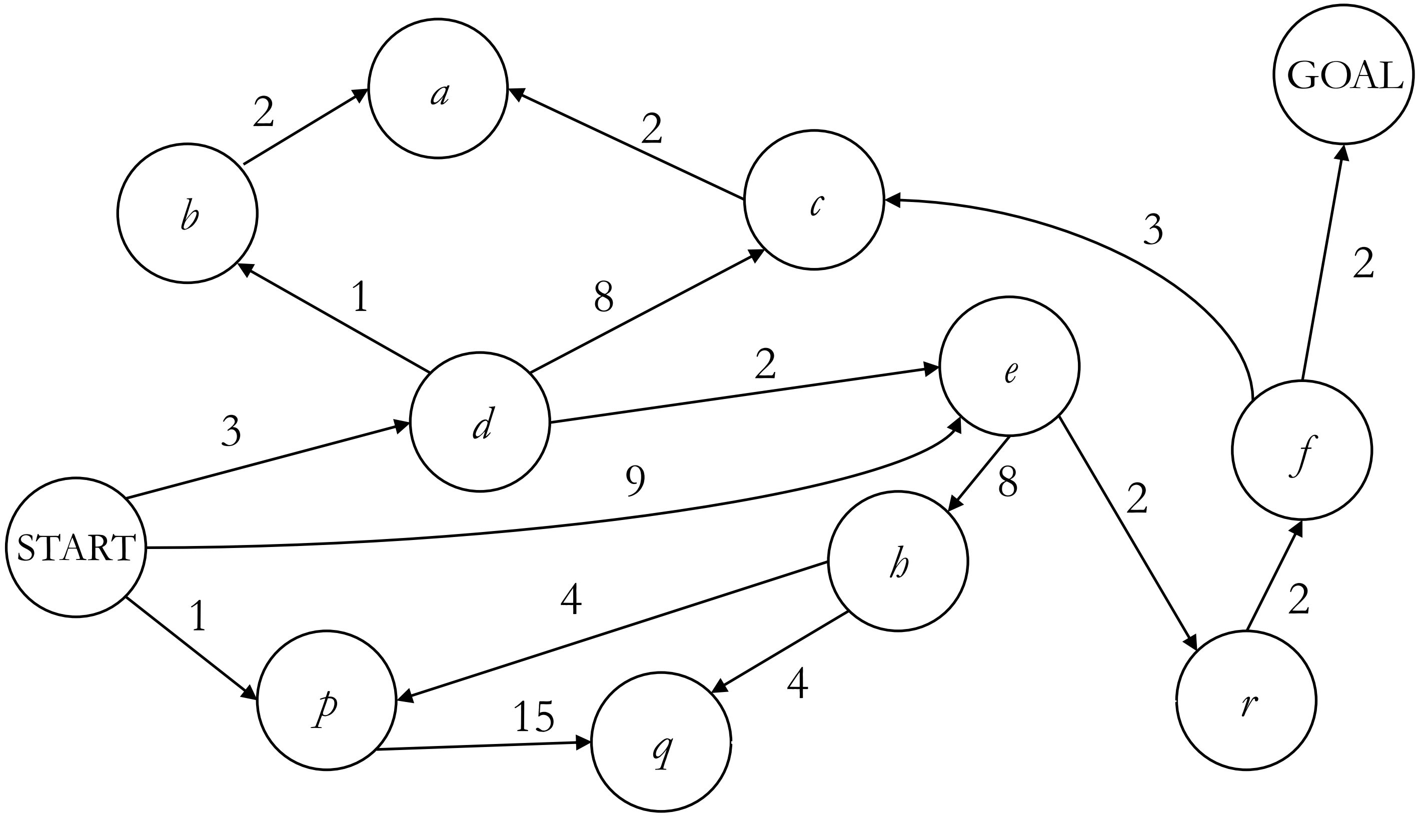


State Space Graph

Search Tree



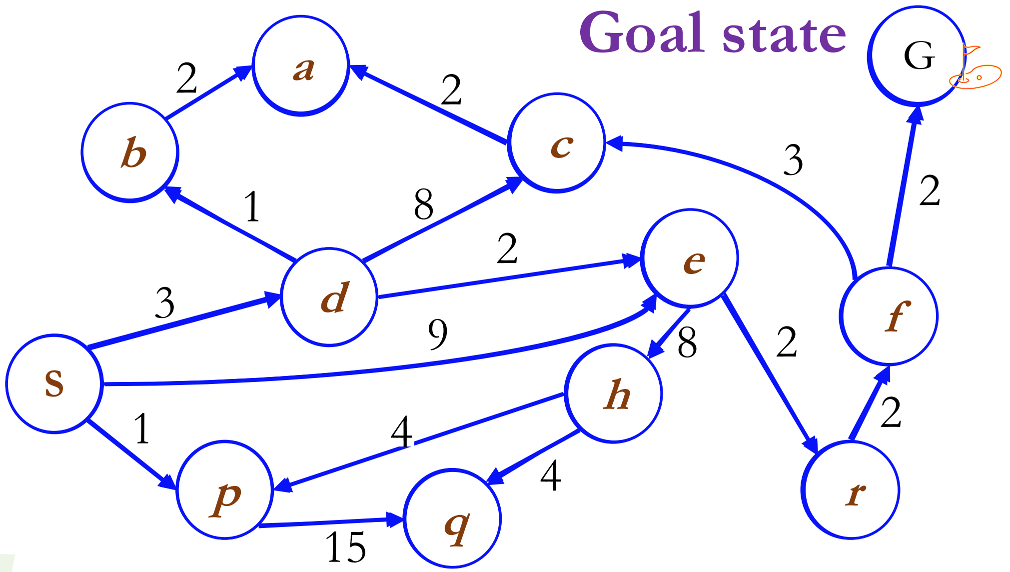
Cost-Sensitive Search: BFS vs UCS Cont.



- BFS finds the shortest path in terms of number of actions: S→E→R→F→G
- The total cost = 9+2+2+2 = **15**

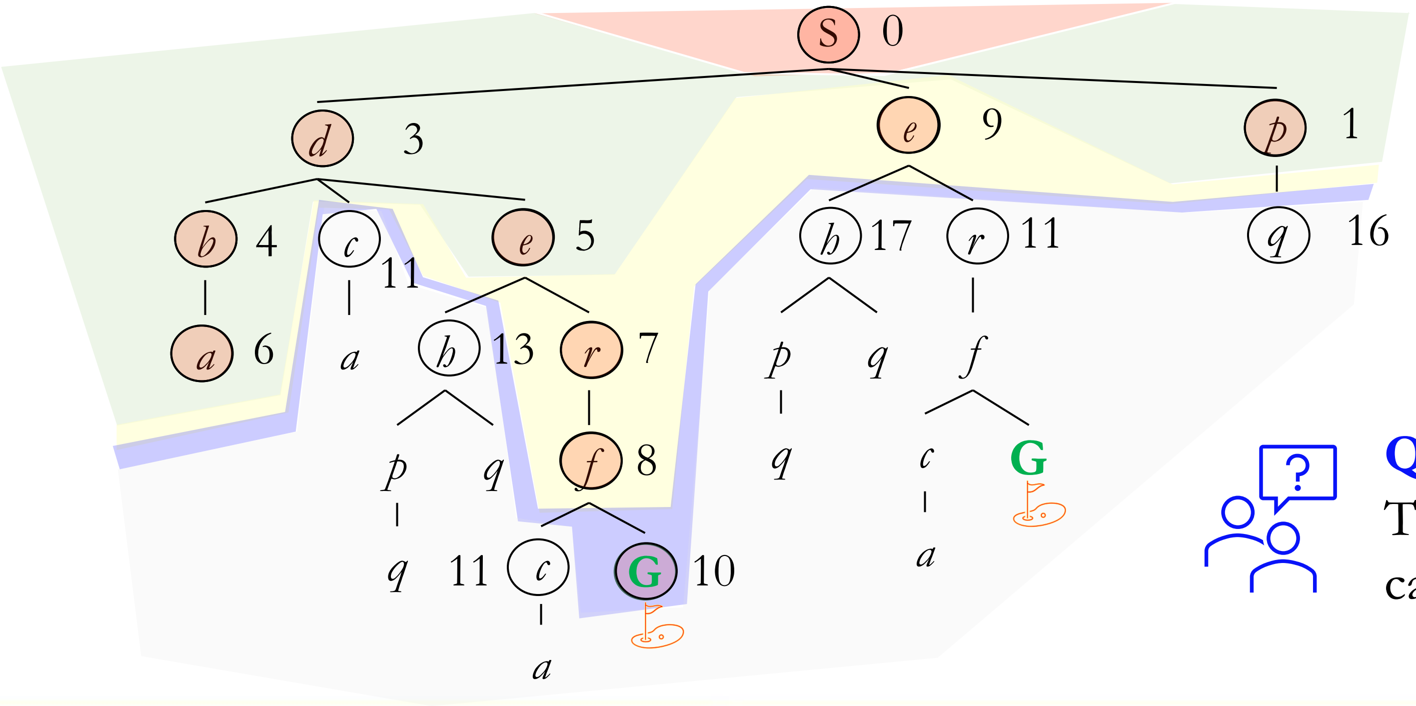
Cost-Sensitive Search: BFS vs UCS – Example # 1

- Like breadth first let's explore the nodes layer by layer in the order of cumulative cost.



State Space Graph

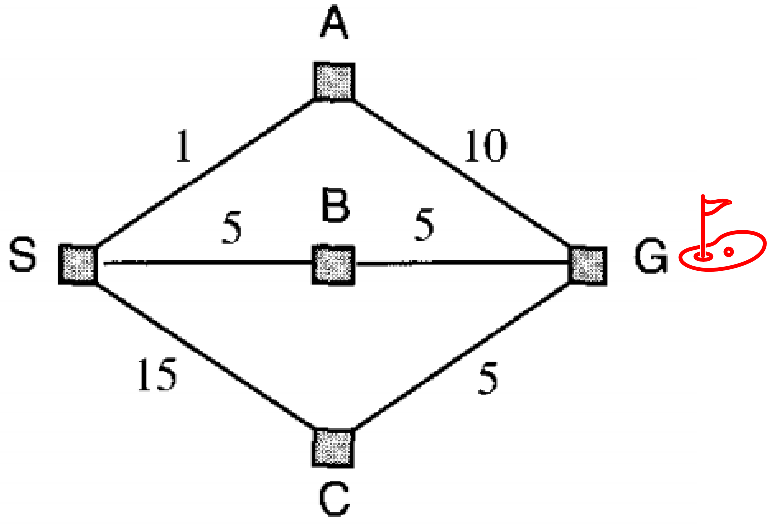
Search Tree



Question:
There is one more path to the Goal state, so can't it be the optimal (lowest cost) solution?

Uniform Cost Search – Example # 2

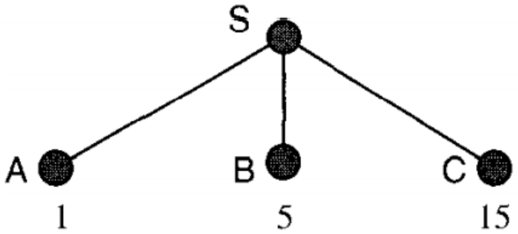
- A state space, showing the cost for each operator:



Problem: get from S to G.

Solution: The UCS will return the path, SBG.

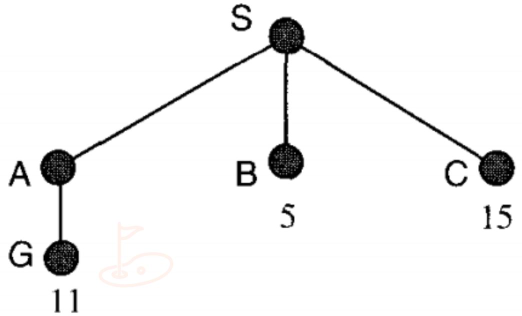
Expands the initial state



A is cheapest, it is expanded next.

Priority queue fringe:

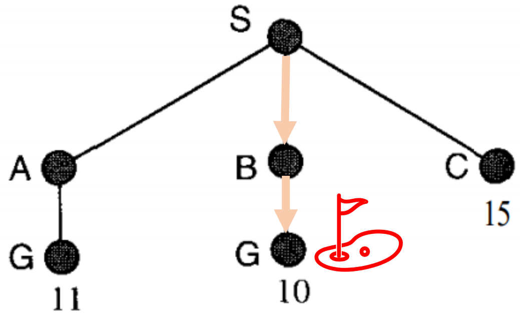
A
B
C



Generating the path SAG.

Priority queue fringe:

B
G
C



B is expanded next, generating the path SBG.

SBG is now the cheapest path remaining in the queue.
It is returned as the solution

UCS – Properties

- **Complete ?**

  Yes, if step cost is greater than some positive constant ε

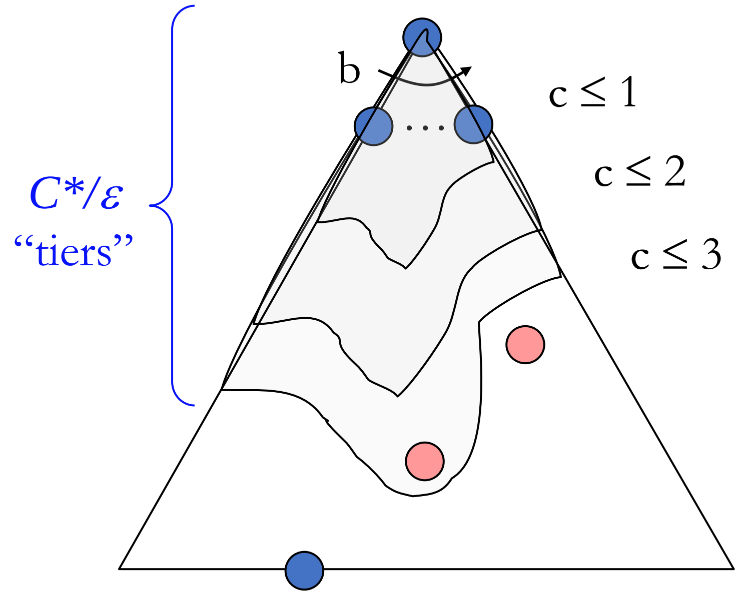
- **Optimal ?**

  Yes – nodes expanded in increasing order of path cost

- **Time:**
  - If C^* is the cost of the optimal solution, and every action costs at least e, then the "**effective depth**" is roughly C^*/ε
  - It will have a timing complexity of $O(b^{C^*/\varepsilon})$ exponential in effective depth

- **Space:**

  $O(b^{C^*/\varepsilon})$



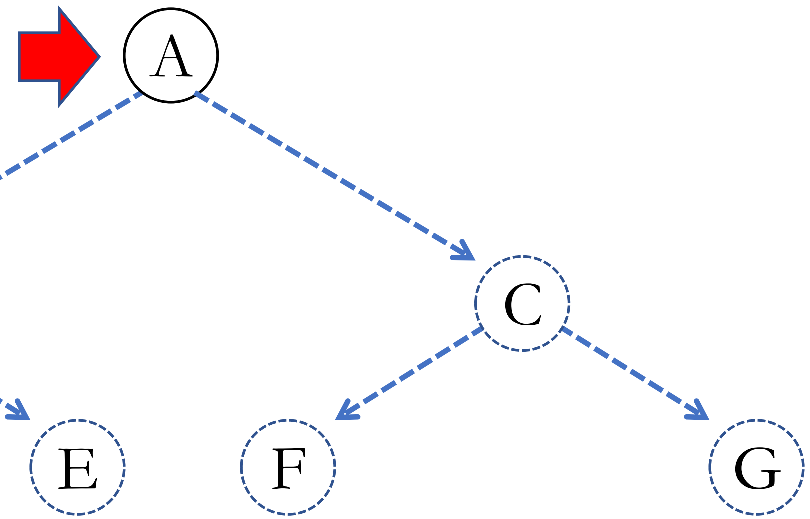
Animation Adopted from Anca Dragan, University of California, Berkeley

UCS – Summary

- **Strategy:** Expand least-cost unexpanded node
- **Implementation:** Fringe (the frontier) is a queue ordered by path cost (priority queue)
- Note: Equivalent to breadth-first if step costs are all equal

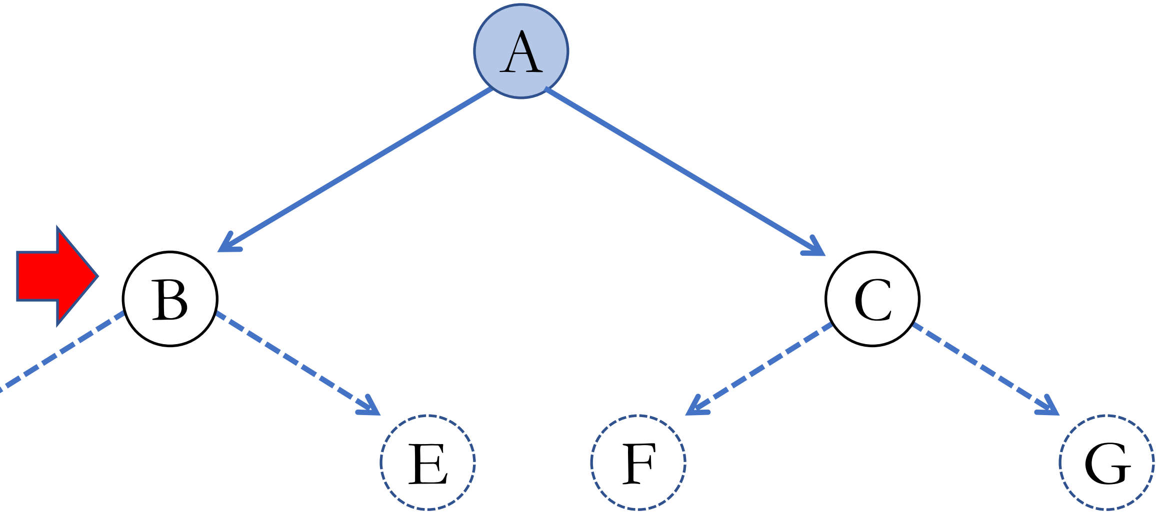
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = **LIFO** queue, i.e., put successors at front



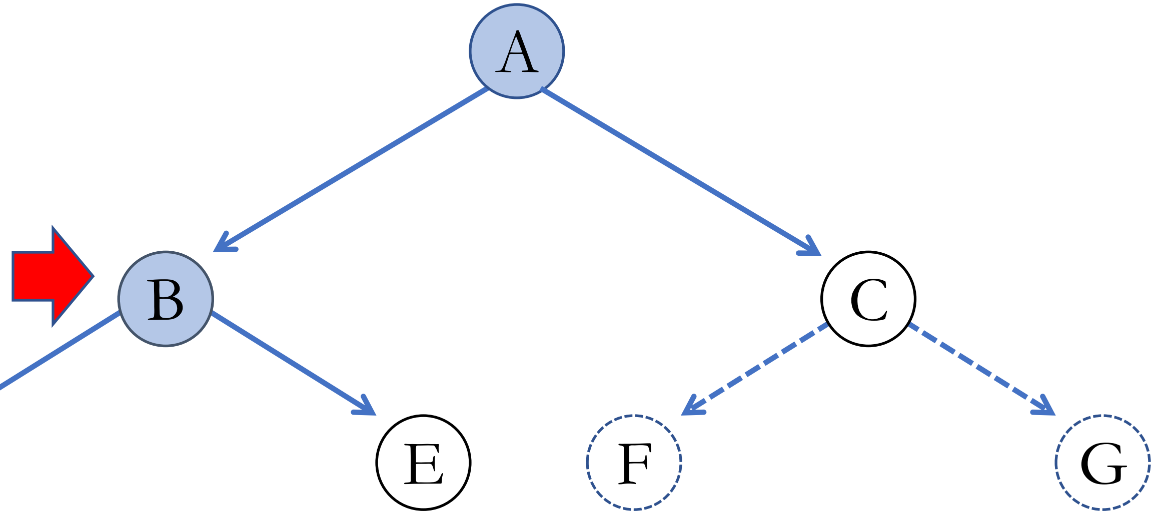
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



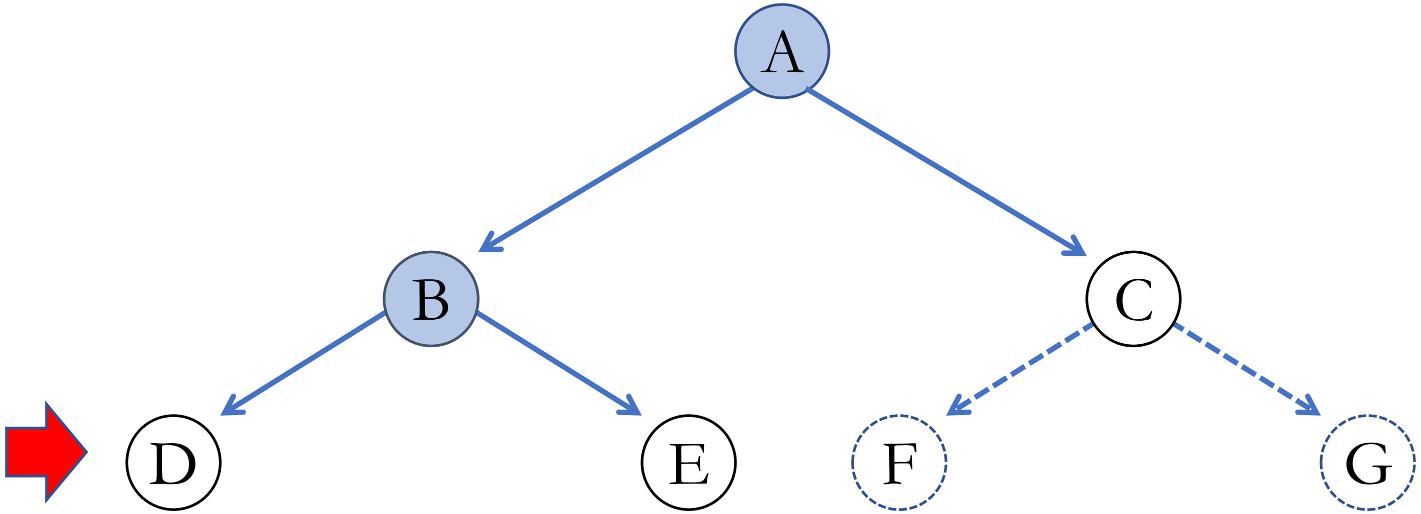
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



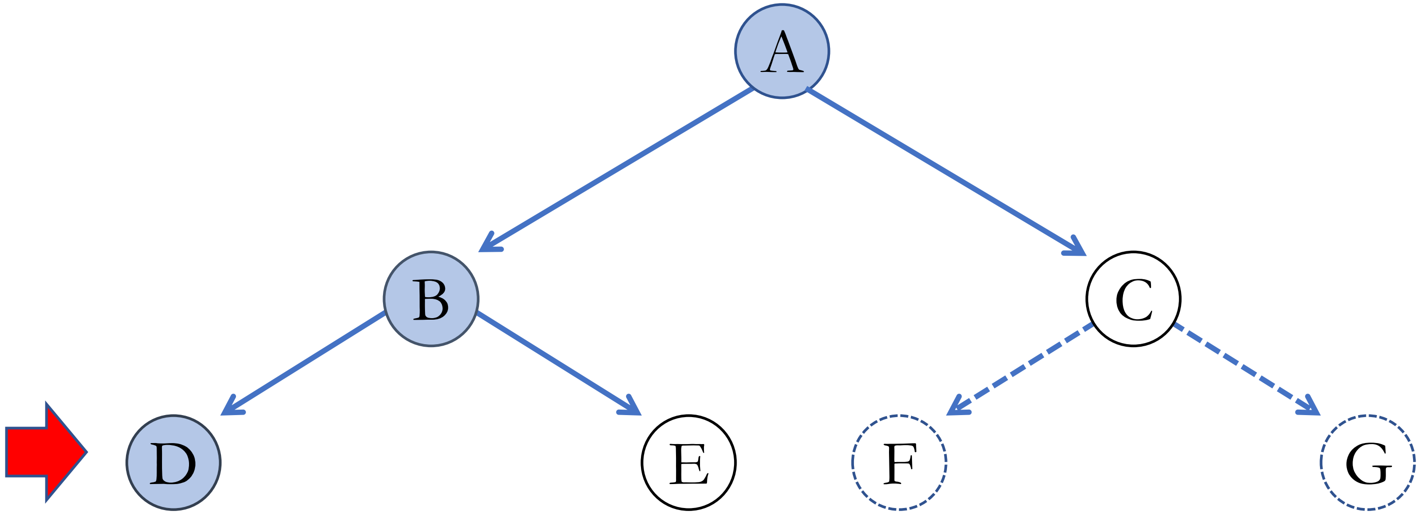
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



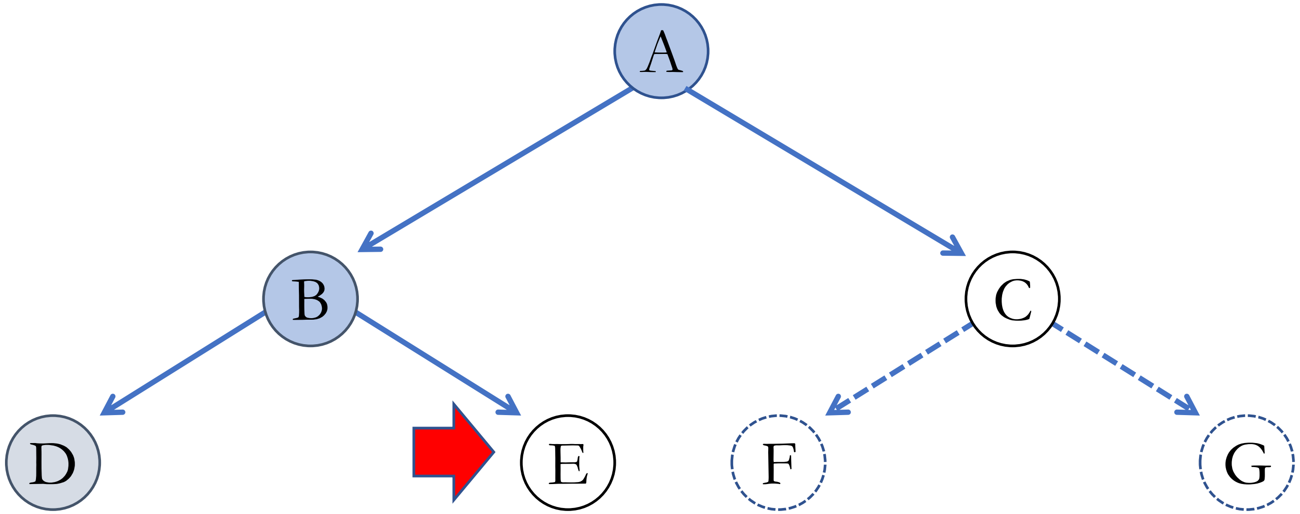
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



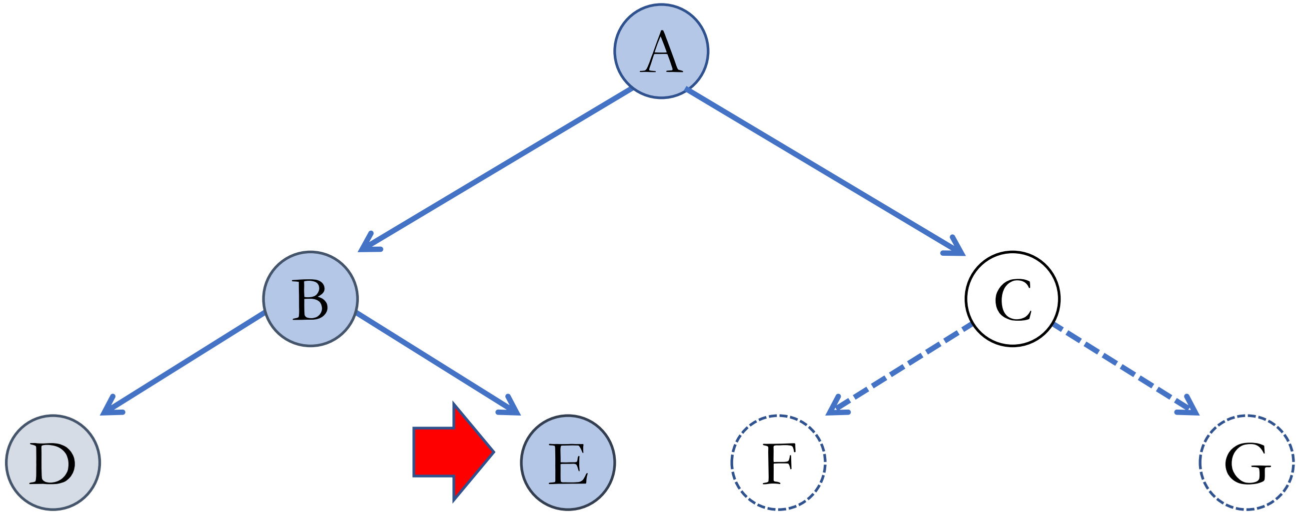
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



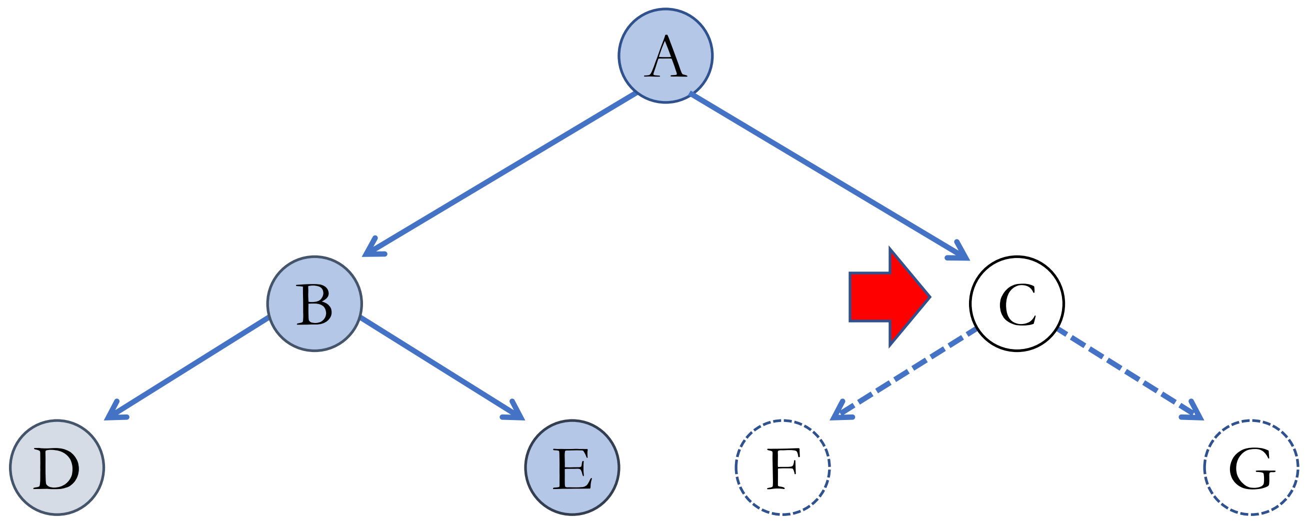
Depth-first search

- Expand deepest unexpanded node
- Implementation:
    - *fringe* = LIFO queue, i.e., put successors at front



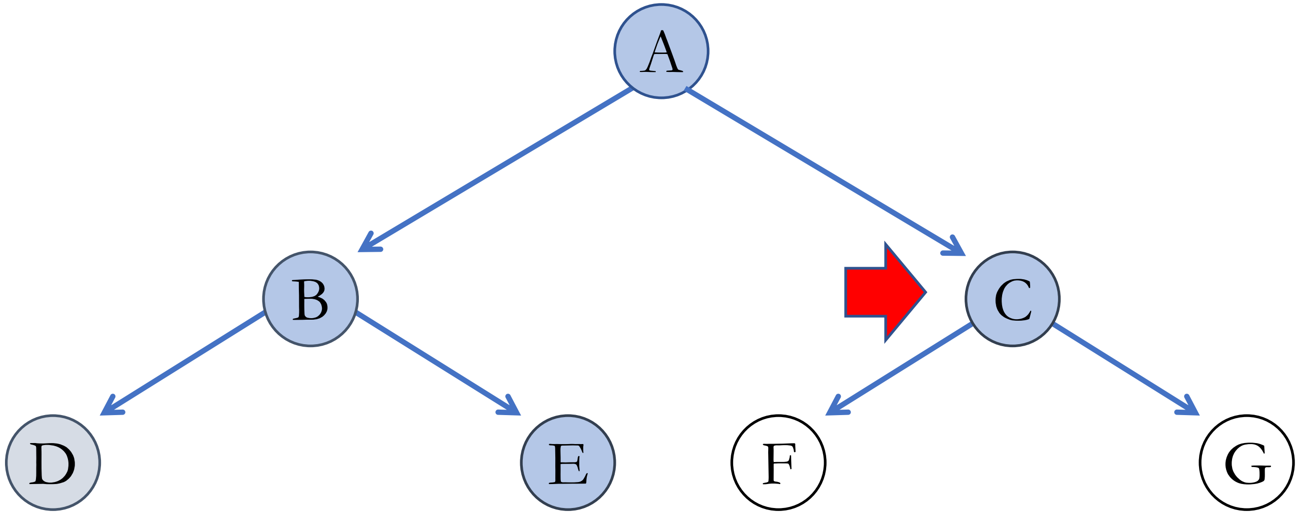
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front

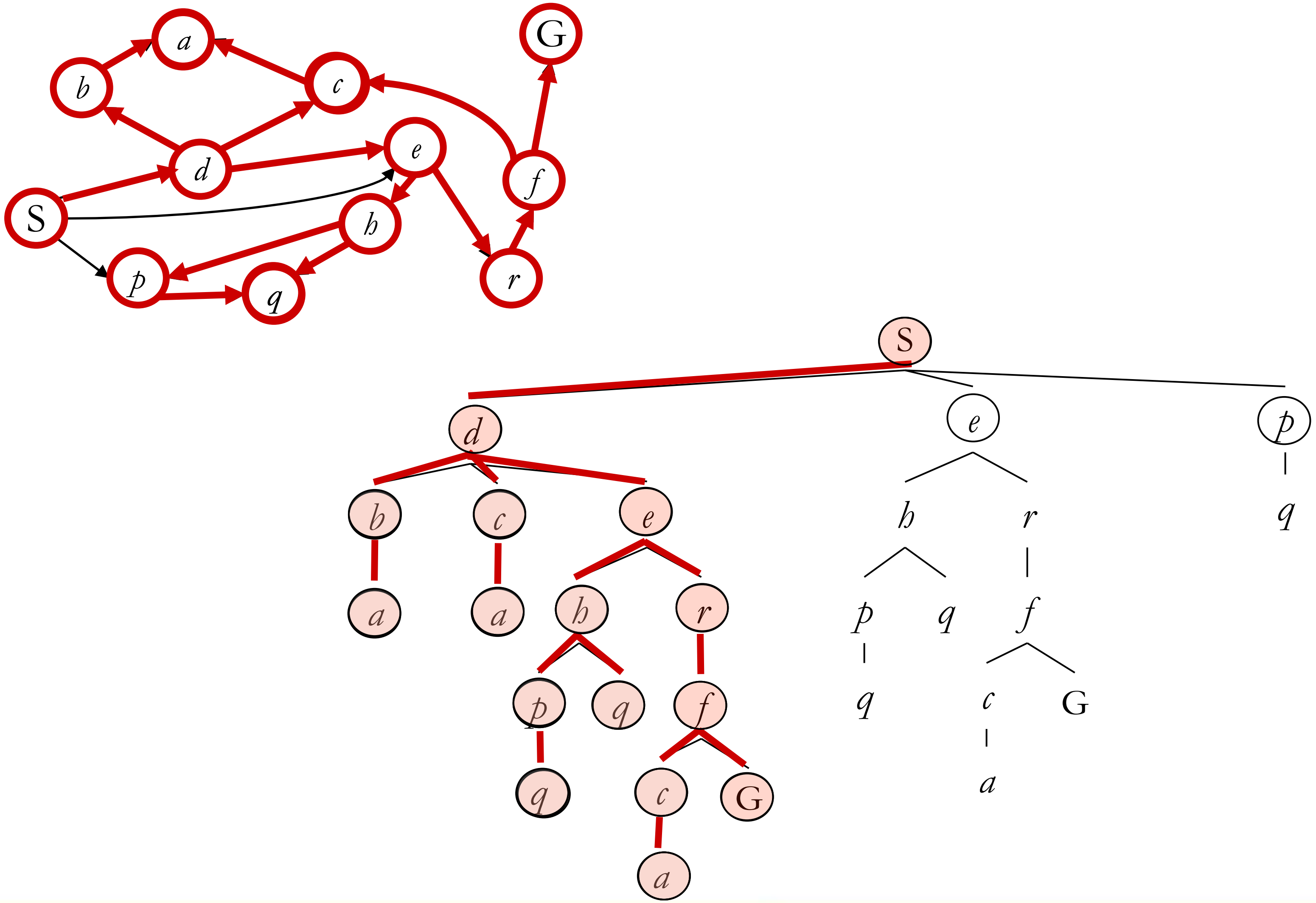


Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front



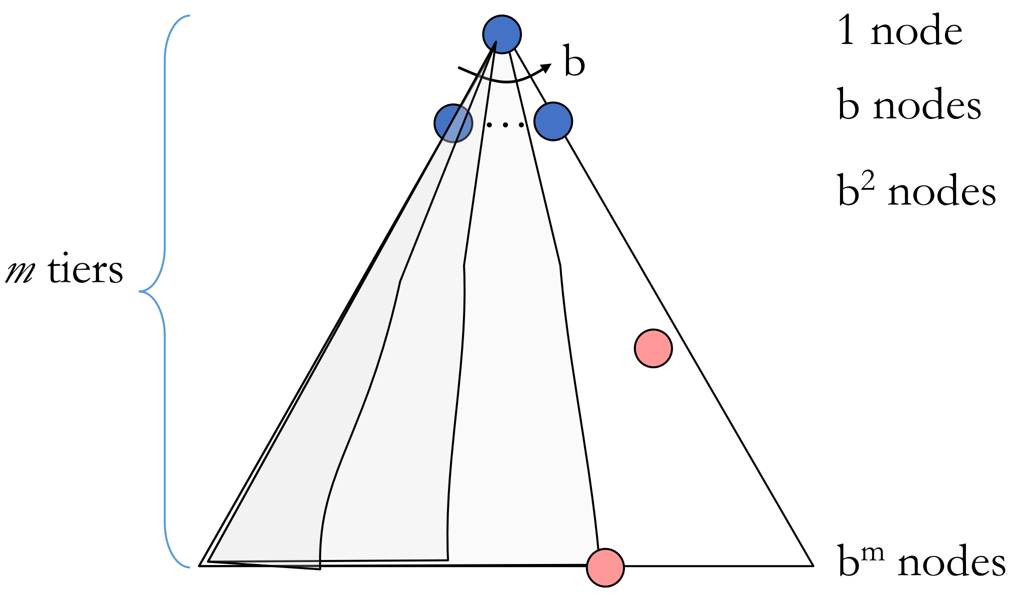
Depth-first search – Working Example



Animation Adopted from Anca Dragan, University of California, Berkeley

Properties of depth-first search

- **Complete?**
  - Fails in infinite-depth spaces (stage spaces with loops)
  - Modify to avoid repeated states along path
    - ✓Complete in finite spaces



- **Optimal?**
  - No - regardless of the depth or cost,
    it returns the first solution it finds,

- **Time?**
  - What could be the **time to reach a solution** at **maximum depth** ***m***: $O(b^m)$
  - Terrible if m is much larger than d (the depth of the shallowest solution)
  - Infinite if the tree is unbounded

- **Space?**
  - $O(bm)$, i.e., linear space!