CS161
Discussion 3
Uninformed Search Algorithms

Shirley Chen

10/11/2019

Search Problem

- Before an agent can start searching for solutions, a **goal** must be identified and a well-defined **problem** must be formulated

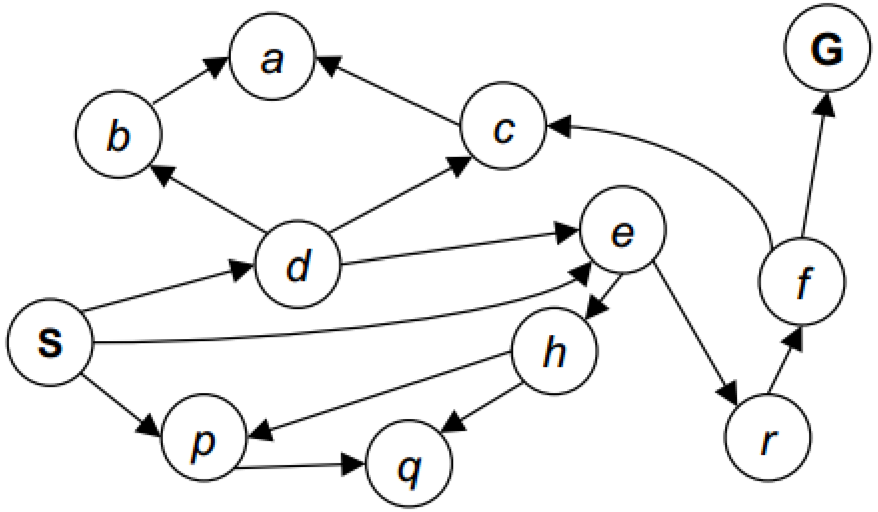
- Search problem formulation
    - **Initial State**
    - **A state space**
    - **Actions:** a set of possible actions
    - **Successor function** (transition model): $F(s_t, a_t) = s_{t+1}$ sometimes with a **path cost function**
    - **Goal test**: determine if solution is achieved.

    - A **solution**: a sequence of actions (a ***path***) that transform the initial state to a goal state

State space graph

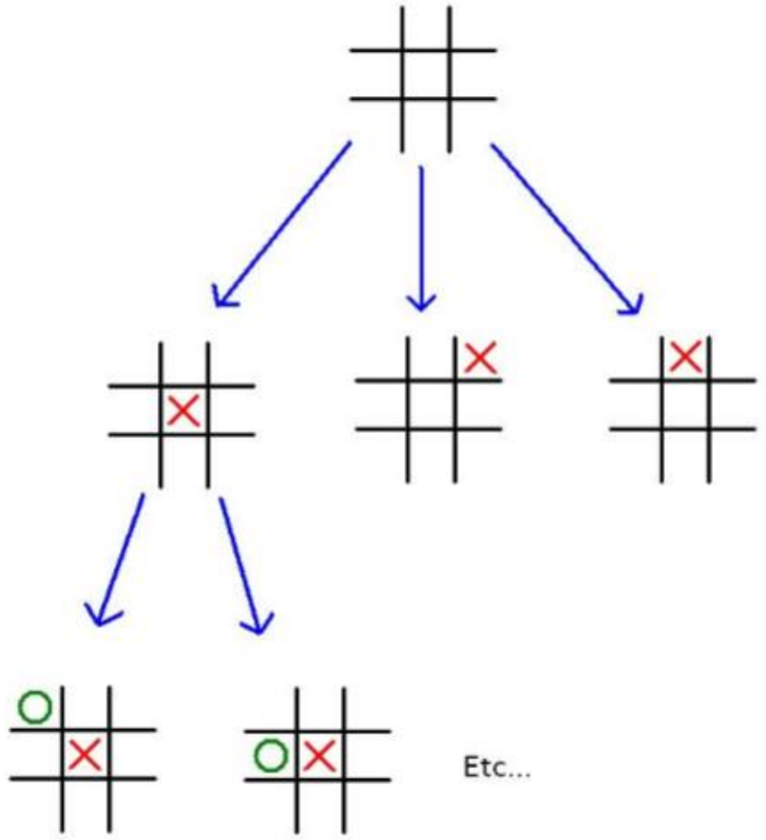
- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- Goal test: one or a set of goal nodes

- Each state occurs only once!



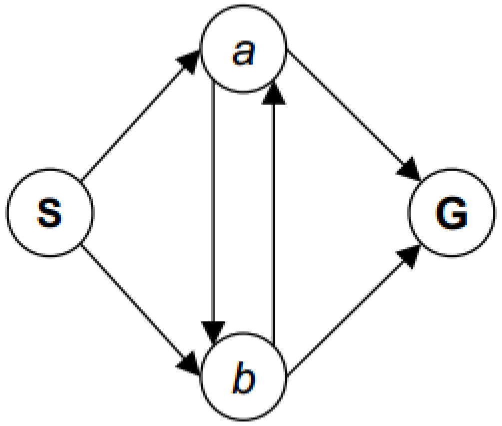
Search Tree

- A "what if" tree of plans and their outcomes
- Root: initial state
- Children: correspond to successors



State Space Graph vs Search Tree

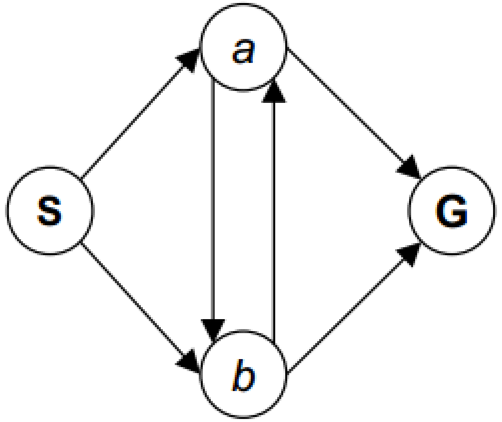
4-state graph



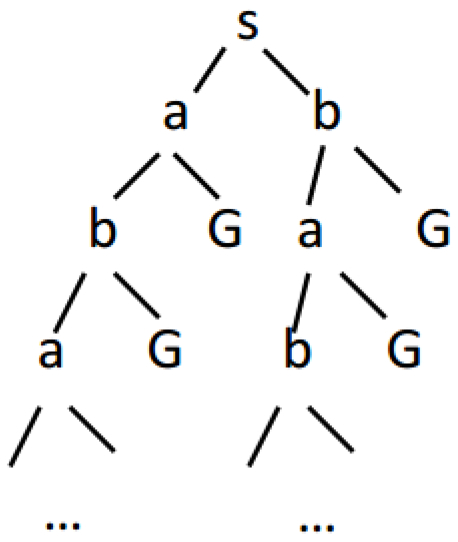
How big is the search tree?

State Space Graph vs Search Tree

4-state graph



How big is the search tree? ∞



- Expand out potential tree nodes
- Maintain a fringe of partial plans under consideration
- Try to expand as few tree nodes as possible

How to evaluate search algorithms

- **completeness**
- **optimality**
- **time complexity**
- **space complexity**

Key concepts

Frontier (nodes to expand)

- Expansion
  - pop a node from frontier and expand
- Generation

Uninformed Search

- BFS
- Uniform-cost search
- DFS, Depth-Limited Search
- Iterative Deepening

BFS

- BFS
  - Expands <u>shallowest</u> nodes first
  - Complete
  - Optimal for unit step costs
  - Time complexity (*exponential*): # of generated nodes
    - $\mathrm{b} + \mathrm{b}^2 + \cdots + \mathrm{b}^\mathrm{d} = O(b^d)$ (*Goal test on generation* by default)
    - Goal test on expansion: $O(b^{d+1})$
  - Space complexity (*exponential*): $O(b^d)$
    - Explored $1 + \mathrm{b} + \mathrm{b}^2 + \cdots + \mathrm{b}^{\mathrm{d}-1} = O(b^{d-1})$
    - Fringe $O(b^d)$

Uniform-cost Search

- Expands the node with lowest path cost
- Uniform-cost search keeps going after a goal node has been generated.
    - Terminate after a goal node is expanded

- Optimal in general
    - Infinite loop if there is a path with an infinite sequence of zero-cost actions
- Complete
    - (If all step cost > a small positive constant ϵ)

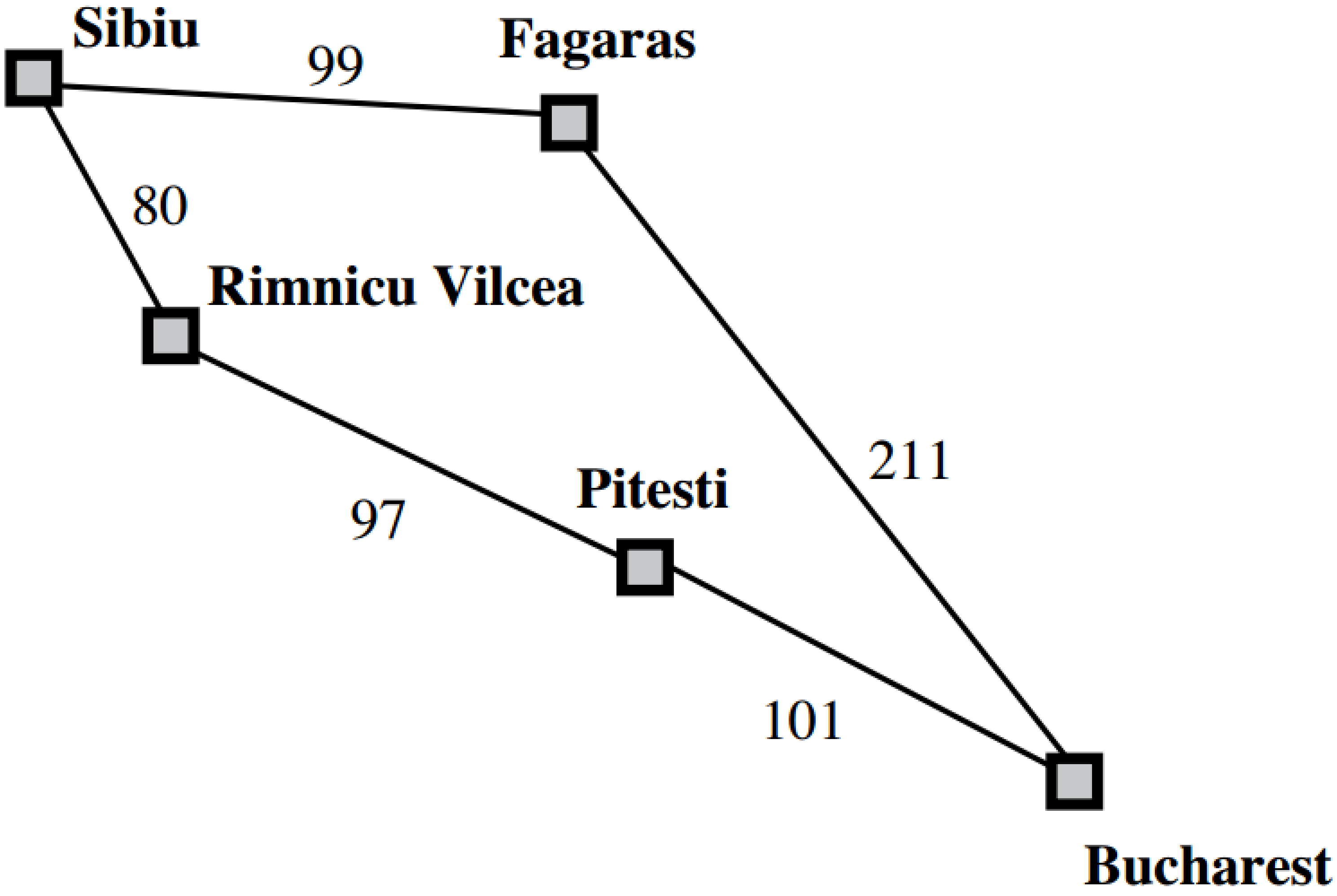
Uniform-Cost Search

- Time Complexity (Worst Case)
  - C: cost of optimal solution
  - Every action costs at least ϵ
  - Time complexity: $O(b^{1+\lfloor C/\epsilon \rfloor})$
    - When all step costs are equal: $O(b^{1+d})$ (test on expansion)
- Space Complexity (Worst Case)
  - Fringe: priority queue (priority: cumulative cost)
  - Worst case: roughly the last tier, $O(b^{1+\lfloor C/\epsilon \rfloor})$

Uniform-Cost Search

- Uniform-cost search and BFS
  - BFS stops after a goal node is generated (unless otherwise specified)
  - Uniform-cost search keeps going after a goal node has been generated

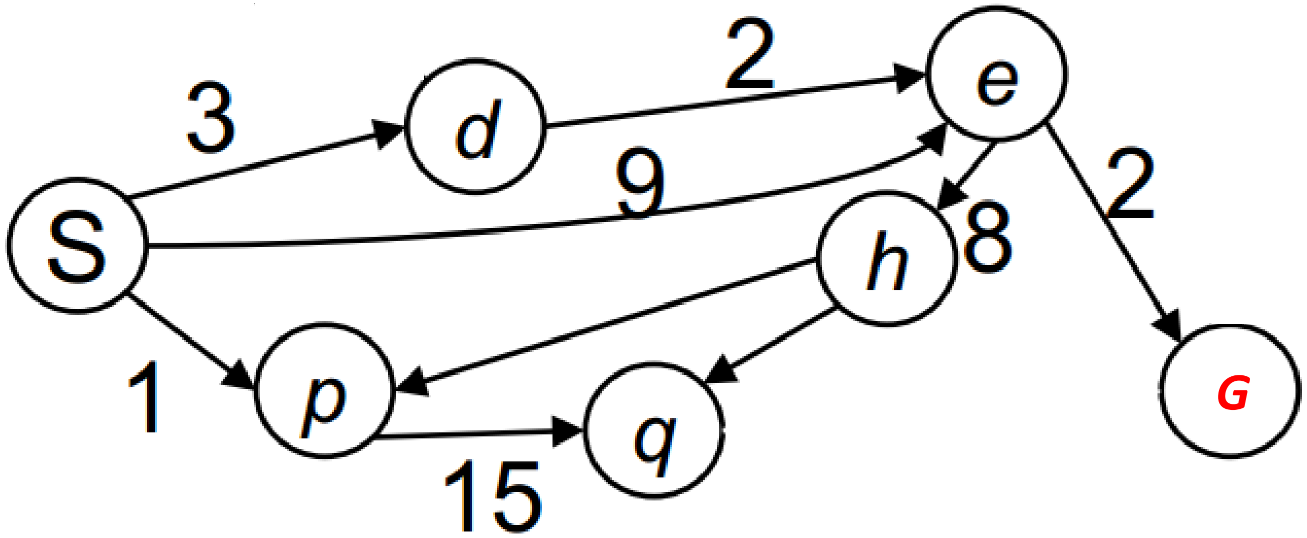
Uniform-Cost Search - Example



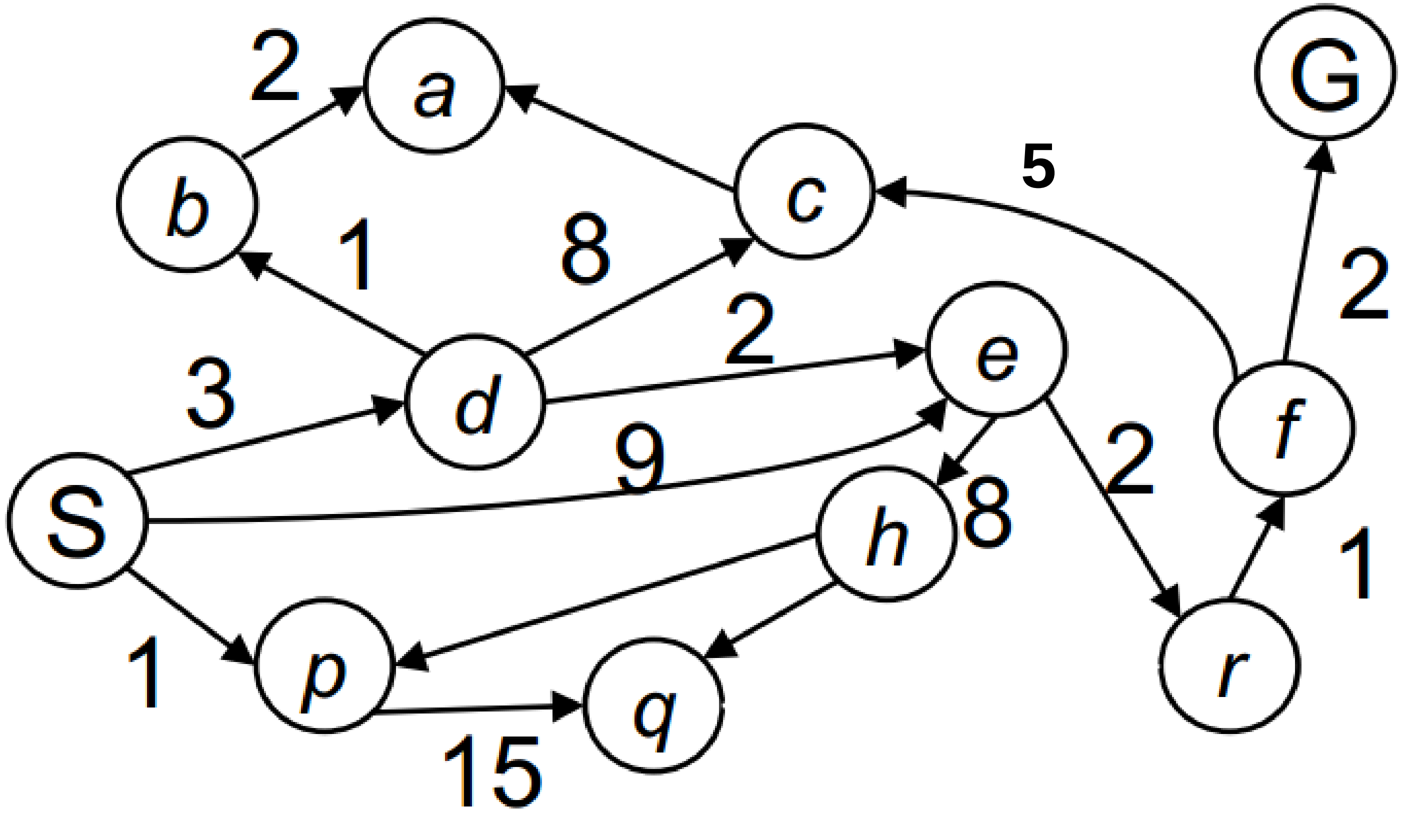
Uniform-Cost Search – Exercise 1

Use uniform-cost search

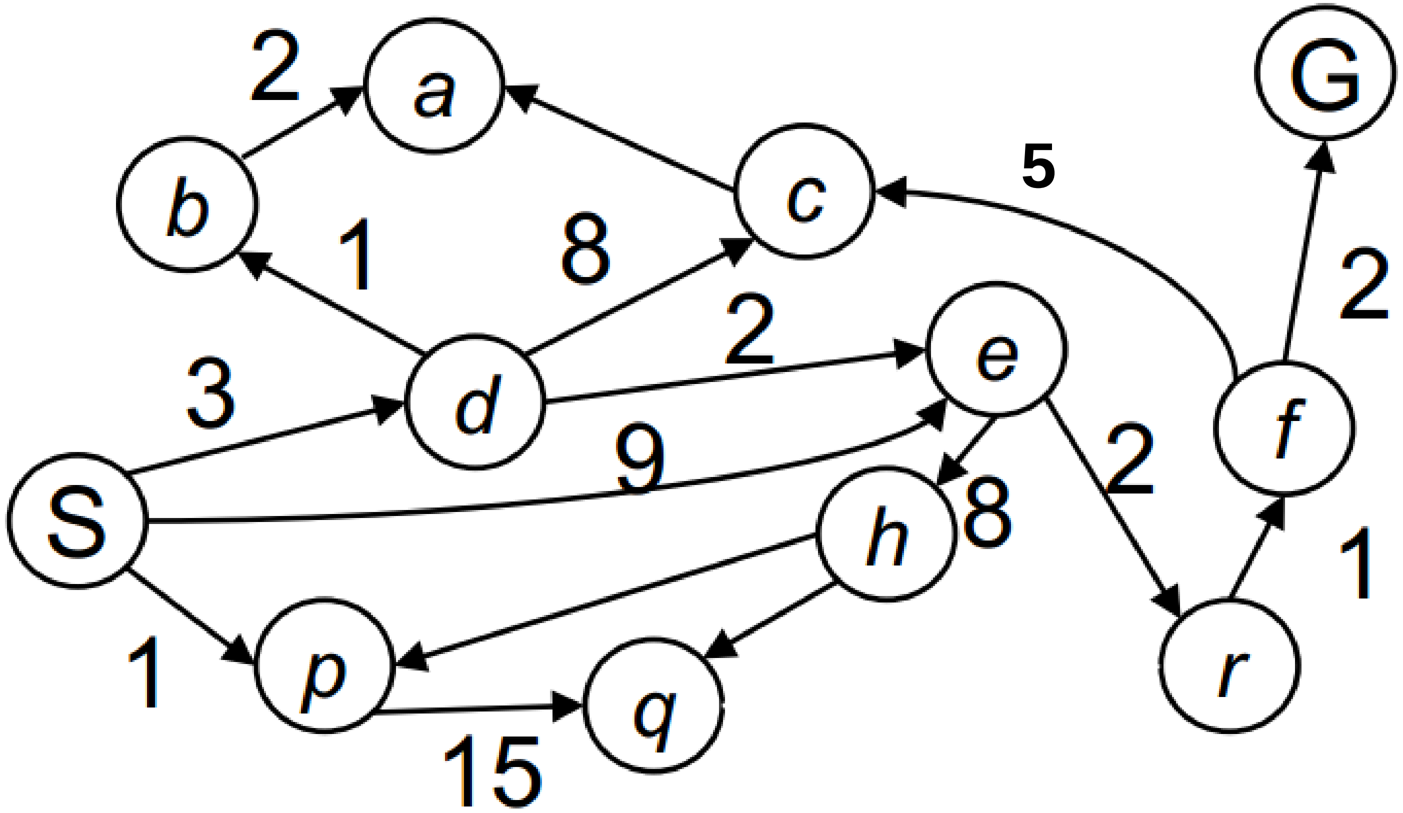
- Give the generated (partial) search tree
- Show in what order we expand nodes
- Return the optimal solution (path)



Uniform-Cost Search – Exercise 2

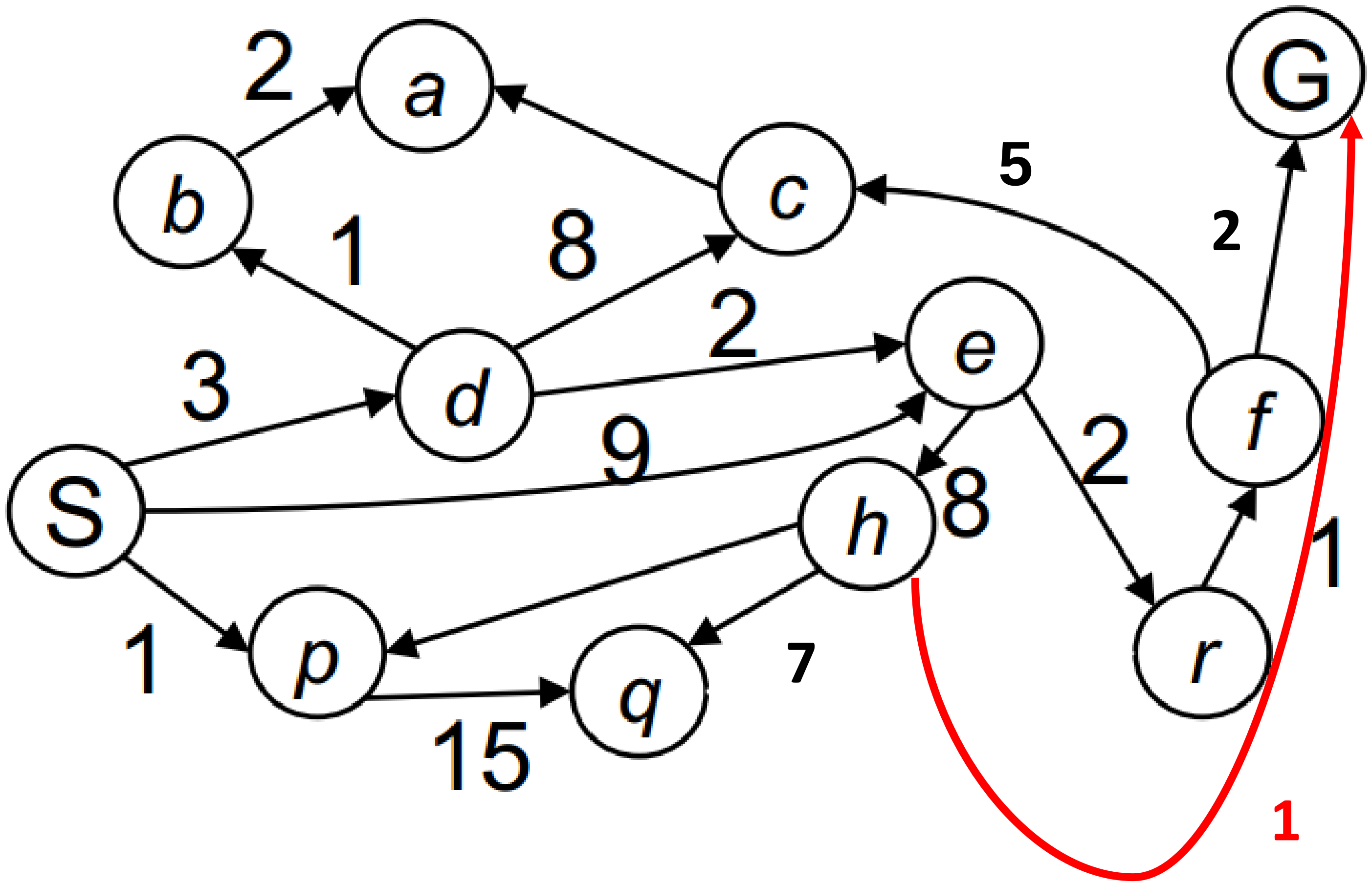


Uniform-Cost Search – Exercise 2

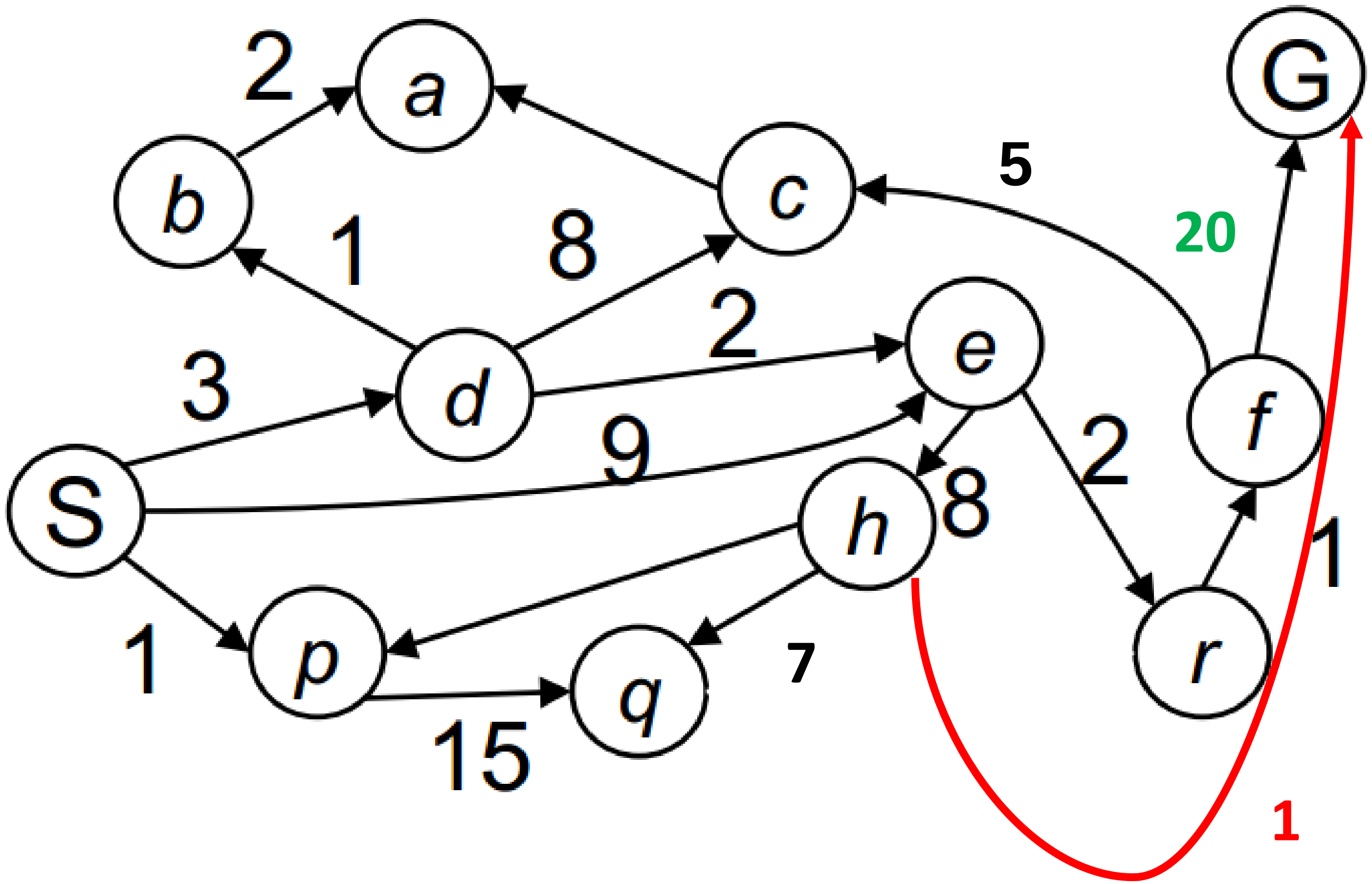


Think:
When does search terminate?

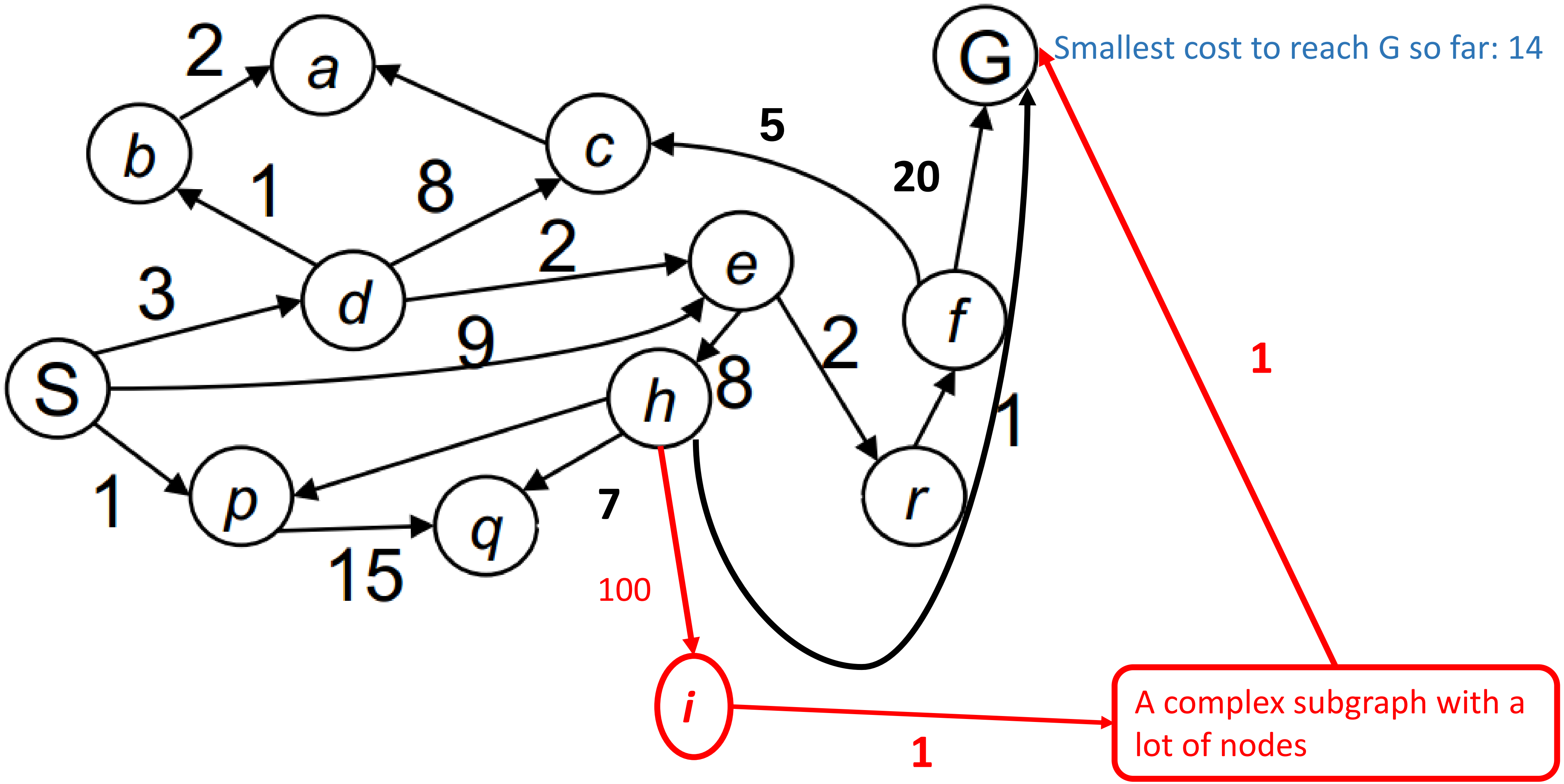
Uniform-Cost Search – Exercise 3



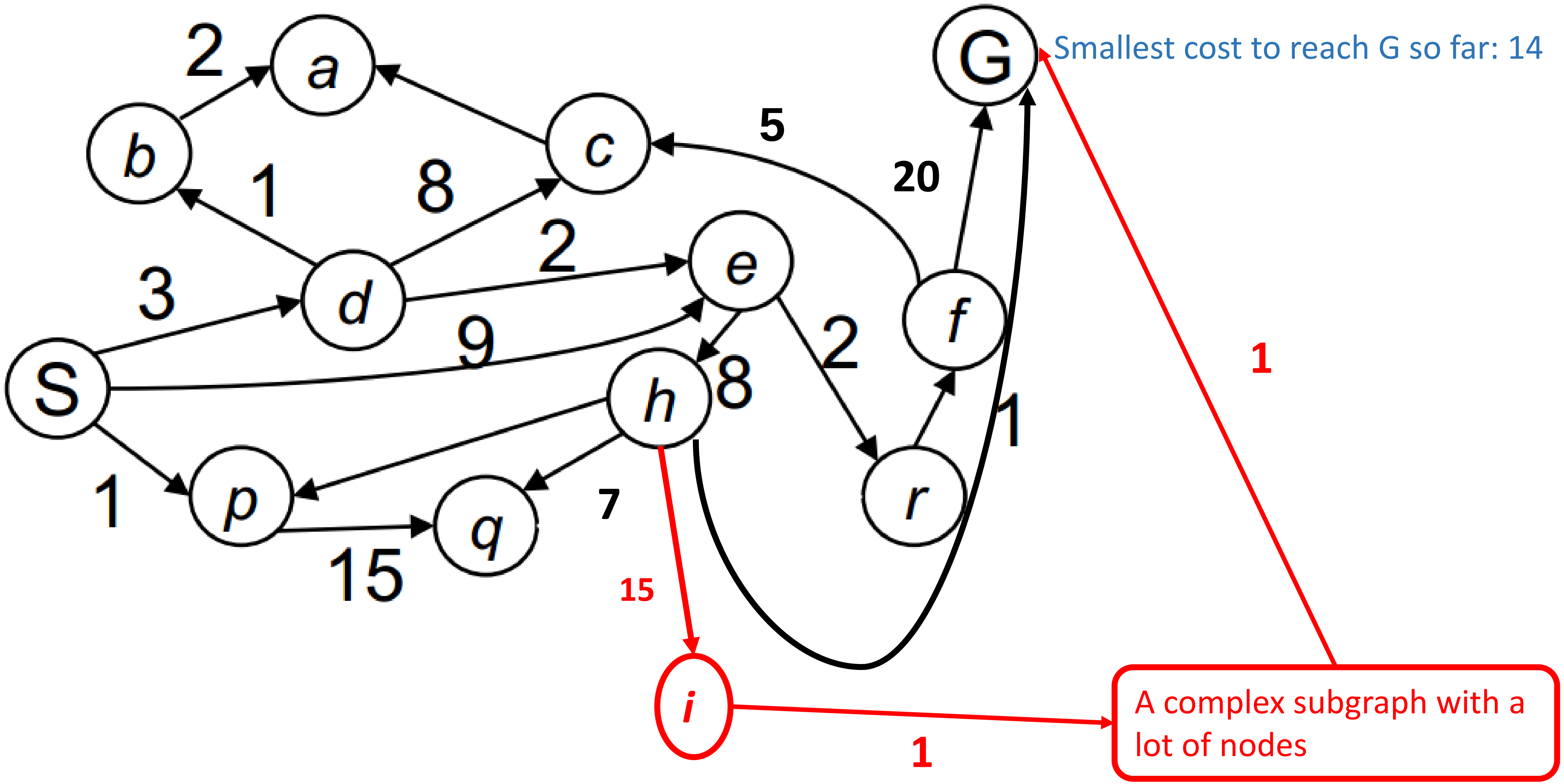
Uniform-Cost Search – Exercise 3



Uniform-Cost Search – Exercise 4



Uniform-Cost Search – Exercise 4

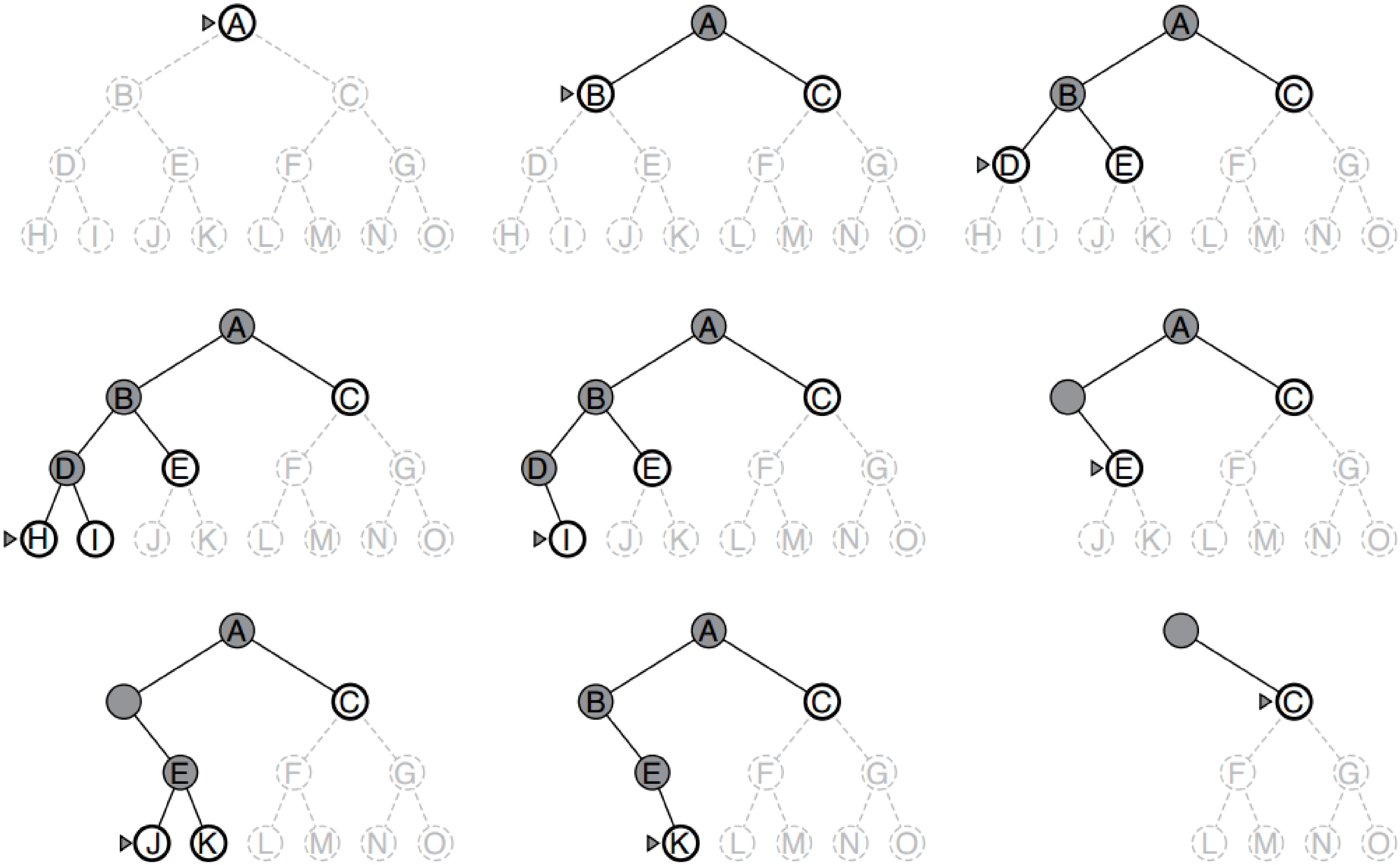


DFS and Depth-Limited Search

- DFS
  - Not optimal
  - Not complete
  - Time complexity $O(b^m)$ (m: maximum depth)
  - Space complexity $O(bm)$ Fringe: path and siblings along the path

- Depth-Limited search: add a depth bound l
  - Complete
  - Not optimal
  - Time complexity $O(b^l)$
  - Space complexity $O(bl)$

DFS


A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G I J K L M N O
A C E F G J K L M N O
A C E F G J K L M N O
A B C E F G K L M N O
C F G L M N O

Comparison

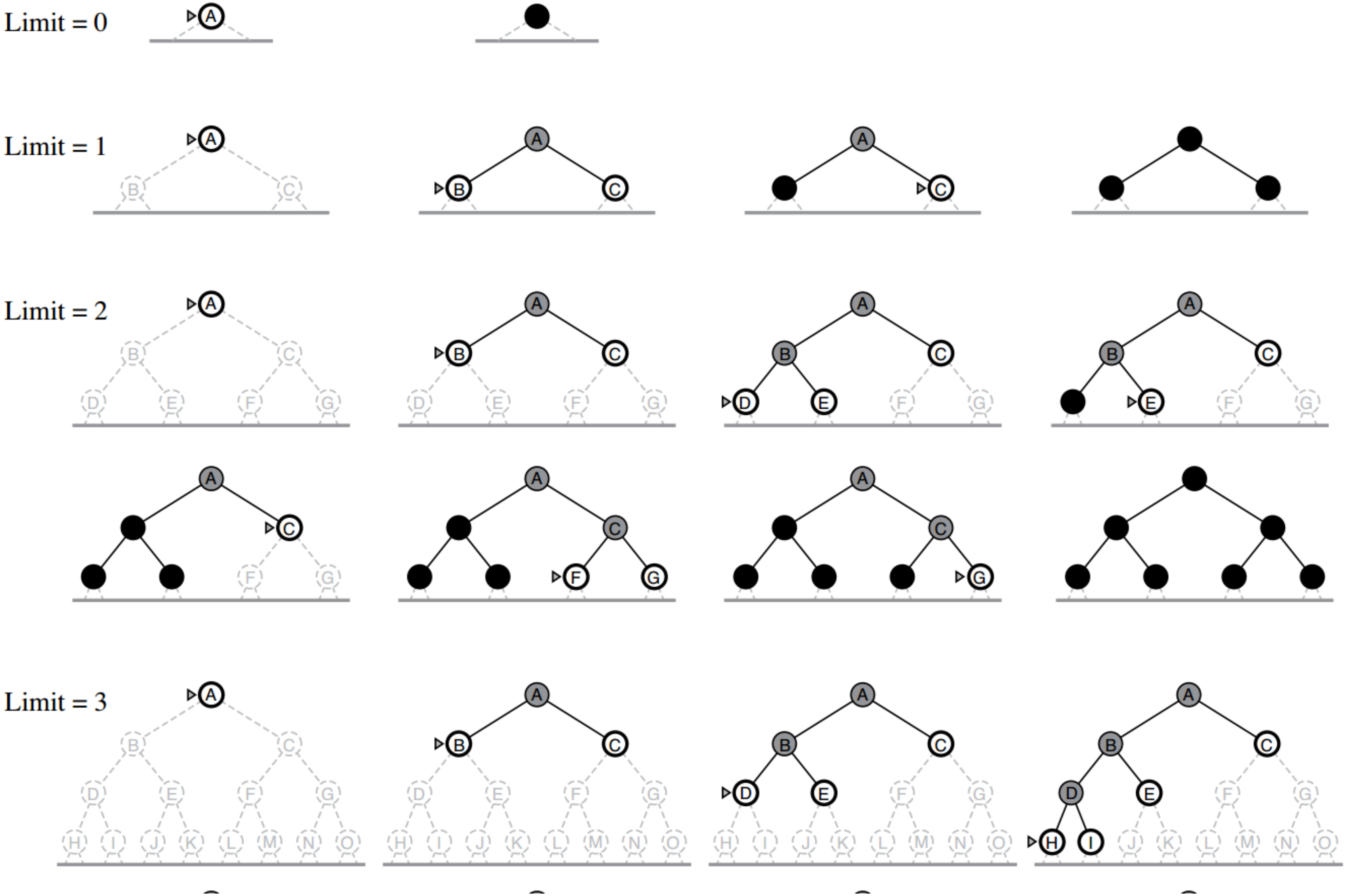
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Iterative Deepening Search

- Depth-first search. Increase depth limits until a goal is found
- Complete; Optimal for unit step costs
- Space complexity of DFS: $O(bd)$ (d: depth of the shallowest solution)
- Time complexity comparable to BFS
  - (Analysis: see lecture slides)

Iterative Deepening



Bidirectional Search

- Run two simultaneous searches (BFS/Iterative Deepening)
  - One forward from the initial state
  - The other backward from the goal
- Replacing Goal test: Check whether the frontiers of the two searches intersect
  - The first found solution may not be optimal
    - Additional search is required to make sure there isn't short-cut across the gap! (Will show later)

Bidirectional Search (cont'd)

- What if we have multiple goal states?
  - For explicitly listed goal states: construct a new dummy goal state
    - Dummy goal state's immediate predecessors are all the actual goal states
  - For abstract description (e.g. “no queen attacks another queen”)
    - Bidirectional search is difficult to use
- Time complexity & Space Complexity (Using two BFS)
  - $O(b^{d/2})$

Exercise - Word Ladder

- Input:
  - *beginWord*
  - *endWord* (endWord != beginWord)
  - A dictionary
- Transformation:
  - wordA -> wordB (e.g. "hit"-> "hot")
  - Only one letter can be changed at a time.
  - wordB must be in dictionary
- Question:
  - Transform *beginWord to endWord*
  - <u>How many transformations we need at least?</u>
    - Return -1 if no such sequence
- **How do you solve it?**

Exercise – Word Ladder

Example

- *beginWord* = "hit"
- *endWord* = "cog",
- *dictionary* = ["hot","dot","dog","lot","log","cog"]
- Output: 4
  - "hit" -> "hot" -> "dot" -> "dog" -> "cog"

Bidirectional Search (cont'd)

We mentioned

- The first found solution may not be optimal
  - Additional search is required to make sure there isn't short-cut across the gap!

- When does this happen?
  - What if "hot" and "log" are connected in the word ladder example?