CS 161 Intro. To Artificial Intelligence

Week 2, Discussion 1C

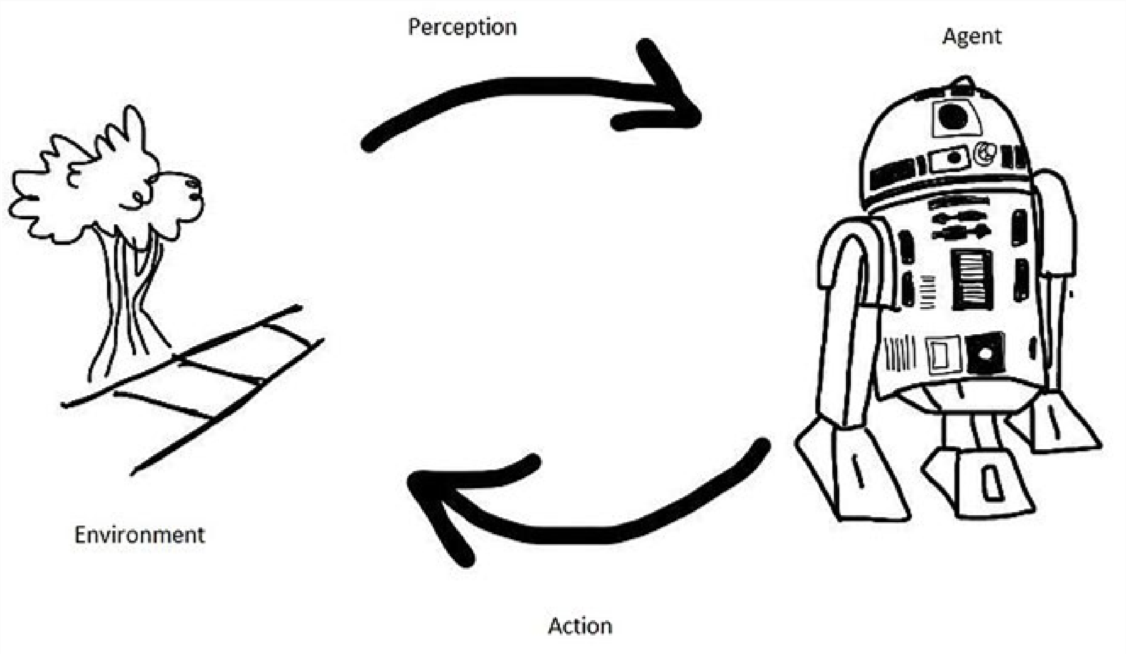
Li Cheng Lan

Today's Topics:

- Agents
- Search Problem Formulation
- State Space Graph & Search Tree
- Search Algorithm Evaluation
- Uninformed Search
    - Breadth-first Search (BFS)
    - Uniform-cost Search (UCS)
    - Depth-first Search (DFS)
    - Depth-limited Search
    - Iterative Deepening Search

Agents

In AI, an intelligent agent **perceives** its **environment** through **sensors** and **acts** upon it through **actuators**.



Agents

2 types of agents:

- **Rational agents**:
  - Choose actions based on maximized expected utility
    - Goal-based agents (eg. reach a goal with lowest cost)
    - Utility-based agents (eg. reinforcement learning)
- **Reflex agents:**
  - Choose actions based on current percept of the environment
  - Does not consider future consequence of actions
  - If-else condition-action: if current condition ⇒ action
    - Eg. a mail sorting robot (see an address ⇒ put letter into a right bag)

Search Problem Formulation

A search problem consists of:

- **Initial state**
- **State space:** $S = \{s_1, s_2, \ldots, s_d\}$
- **Actions (operators)**: a set of possible actions
- **Transition model (successor function)**: $F(s_t, a_t) = s_{t+1}$, sometimes with a path cost function
- **Goal test**: determine if the solution/goal is achieved

A **solution**: a sequence of actions that transform the initial state to a goal state

Problem formulation: the process of deciding what actions and states to consider, given a goal.

Search Problem Formulation - Example

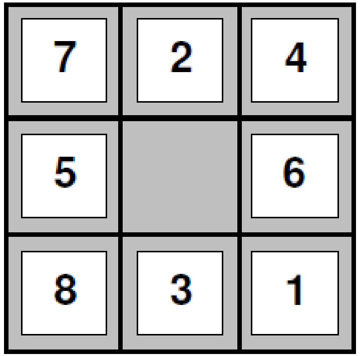
Travel from CA to MA:

- Initial State: CA
- State space: 50 states in U.S.
- Actions: go to adjacent state (cost=distance)
  - Eg. Actions(CA)= {GoTo(OR), GoTo(AZ), GoTo(NV)}
- Successor function (transition model):
  - Eg. Result(In(CA), GoTo(AZ)) = In(AZ)
- Path cost function:
  - Eg. sum of dist, # of actions, none, etc.
- Goal test: state == MA?

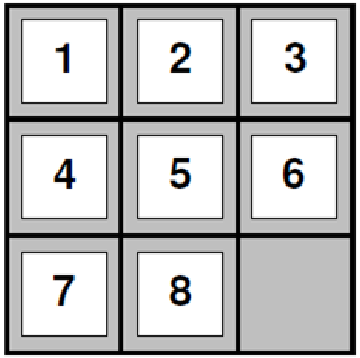
Search Problem Formulation - Example

The 8-puzzle:

- Initial State: some integer locations of tiles
- State space: integer locations of tiles (ignore intermediate positions)
- Actions: move a tile left, right, up, down (ignore blocking, etc.)
- Successor function (transition model): exchange a tile with another adjacent tile in its left/right/up/down direction
- Path cost function: 1 per move
- Goal test: state == goal state (given in picture)



Start State

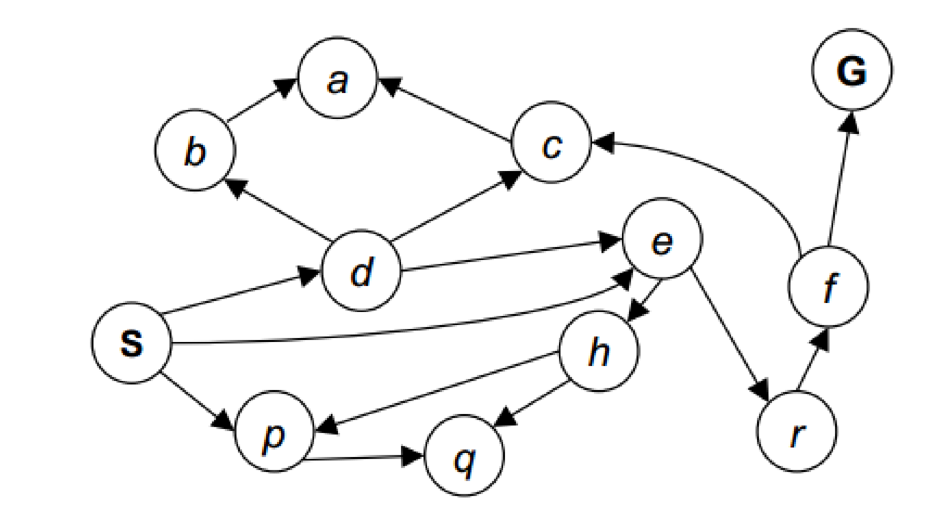


Goal State

State Space Graph

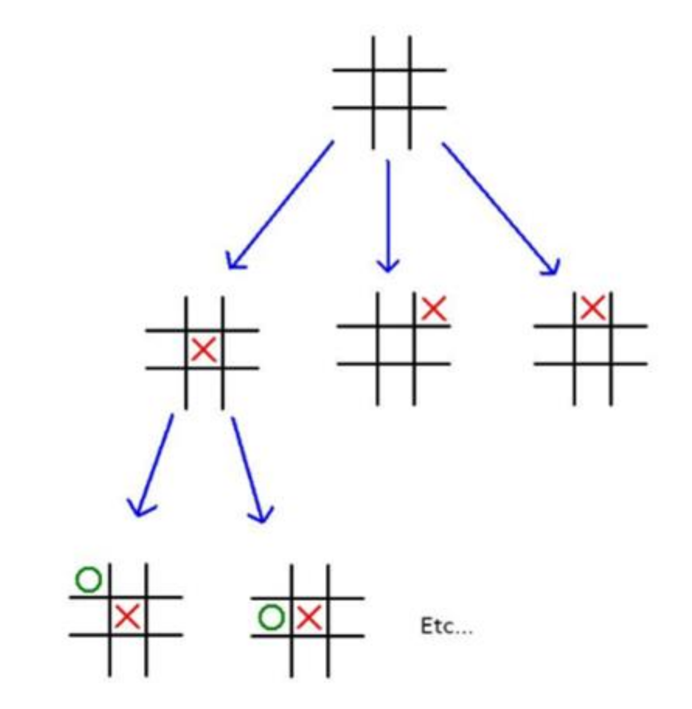
State space graph is a mathematical representation of a search problem

- Nodes are (abstracted) world configurations, represent states
- Arcs represent successors (action results)
- Goal test: <u>one or a set of</u> goal nodes
- Each state occurs only once!



Search Tree - Basics

- A "what if" tree of plans and their outcomes
- Root is initial state
- Children correspond to successor states (correspond to plans to those states)



Search Tree Algorithm

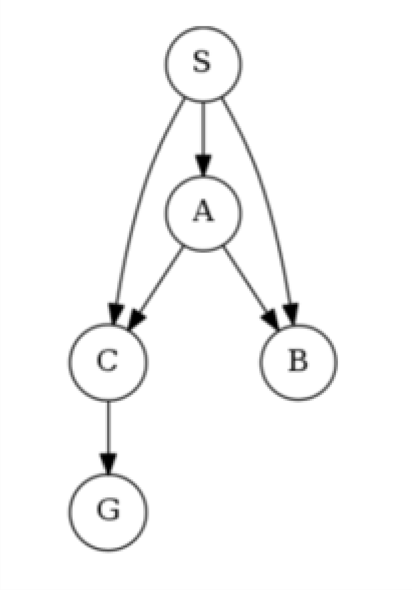
- **Basic idea:** offline, simulated exploration of state space by generating successors of already-explored states
- Strategy: Breadth-first Search (BFS), Depth-first Search (DFS), etc. (will go through these later)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Goal test

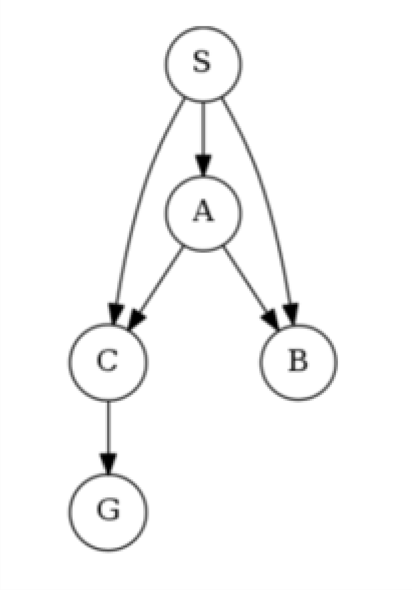
State Space Graph & Search Tree - Example

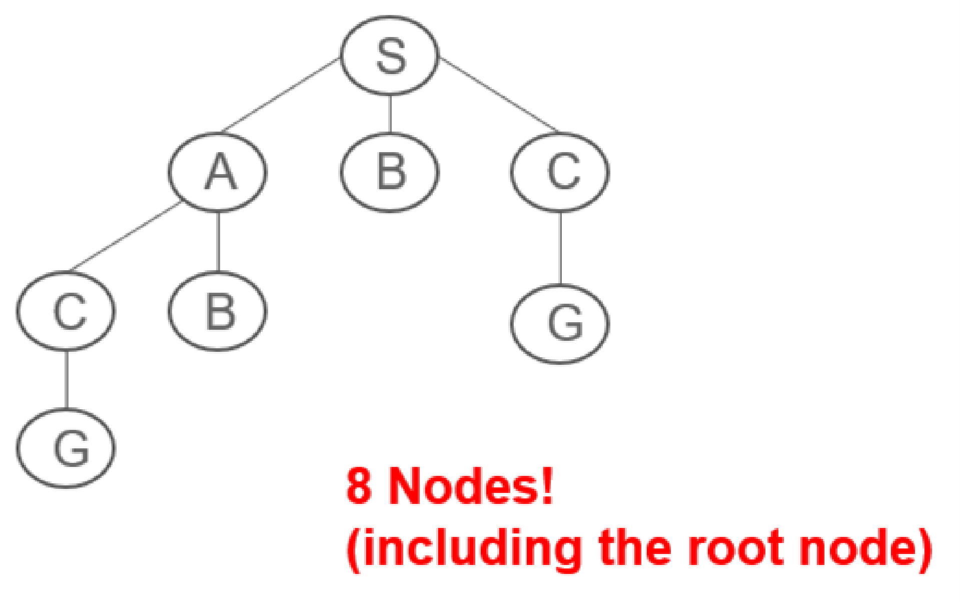
Given the following state graph, start from S and end with G, how many nodes are there in this search tree?



State Space Graph & Search Tree - Example

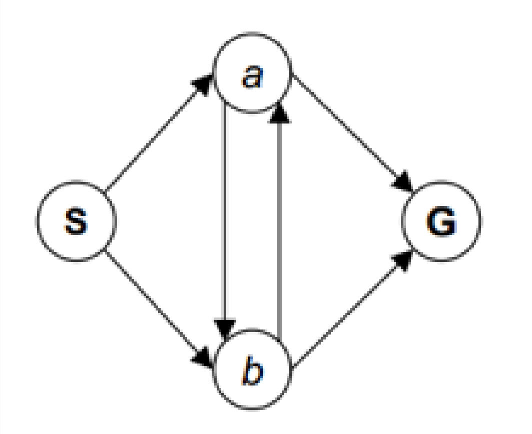
Given the following state graph, start from S and end with G, how many nodes are there in this search tree?





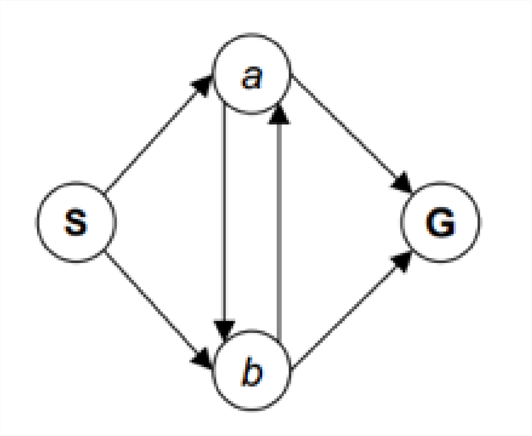
State Space Graph & Search Tree - Example

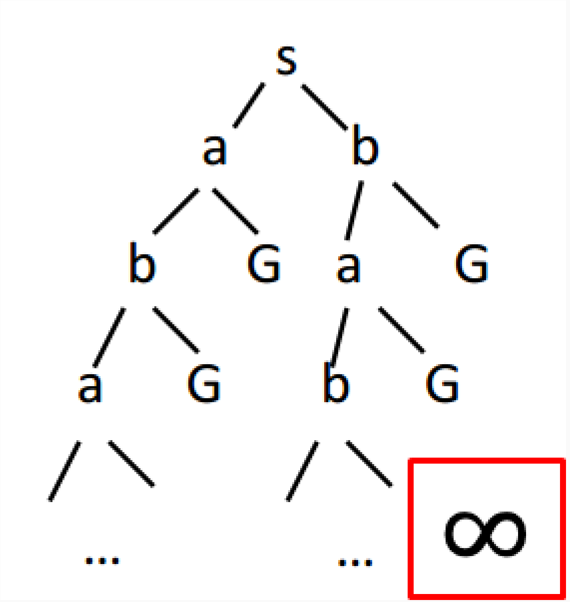
Given the following 4-state graph, start from S and end with G, how many nodes are there in this search tree?



State Space Graph & Search Tree - Example

Given the following 4-state graph, start from S and end with G, how many nodes are there in this search tree?





Search Algorithm Evaluation

- **Completeness**
  - Does it always find a solution if one exists?
- **Optimality**
  - Does it always find a least-cost solution?
    - In search trees, this is often in terms of depth of the solution if no path cost given
    - For some algorithm (eg. Uniform-cost Search), it can be in terms of path cost
- **Time Complexity**
  - Number of nodes generated/expanded
- **Space Complexity**
  - Maximum number of nodes in memory

Search Algorithm Evaluation

Time and space complexity are measured in terms of:

- **b**: maximum # of children nodes for one parent node, also called **branching factor**
- **d**: depth of the least cost solution
- **m**: maximum depth of the state space (may be infinity)

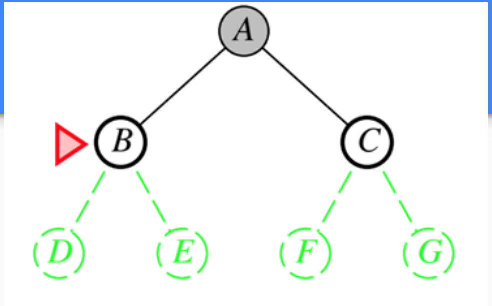
Other useful terms:

- **Frontier/Fringe**: the (generated) nodes to be expanded
- **Generation**: create a node and have it in memory
- **Expansion**: pop a node from frontier and generate its children (expand)

Uninformed (Blind) Search

- Breadth-first Search (BFS)
- Uniform-cost Search (UCS)
- Depth-first Search (DFS)
- Depth-limited Search
- Iterative Deepening Search

Breadth-first Search (BFS)



Expands **shallowest** nodes first

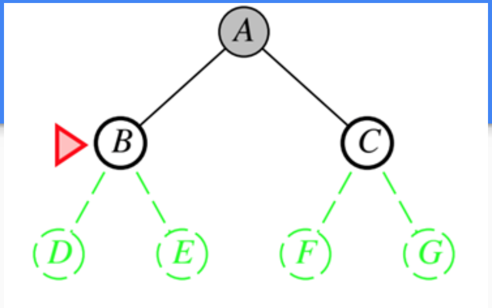
We can do goal test with generation or expansion:

- If goal test with **generation**:
  - root node A is give, it is not count into one layer
  - when expand A, we generate B and C
  - then we **goal test B and C**
  - then we expand B and generate D and E
  - then we **goal test D and E**
- If goal test with **expansion**:
  - root node A is give, it is not count into one layer
  - when expand A, we generate B and C
  - then we **goal test A**
  - then we expand B and generate D and E
  - then we **goal test B**

Suppose our goal is C:

- When goal test with generation, after we generate C, we stop
- When goal test with expansion, after we generate C, it's not enough, we need to continue after we generate F and G
- **By default, we do goal test with generation for BFS**

Breadth-first Search (BFS)



Evaluation:

- Complete (if b is finite)
- Optimal for unit step costs (i.e. each step's cost is same)
- Time complexity:
  - If goal test during the generation: $b+b^2+...+b^d=\mathbf{O(b^d)}$ → by default
  - If we goal test on expansion: $b+b^2+b^3+...+b^{d+1}=O(b^{d+1})$
- Space complexity is $\mathbf{O(b^d)}$
  - we have to keep all nodes in memory
  - $1+b+...+b^d$ (including root node)

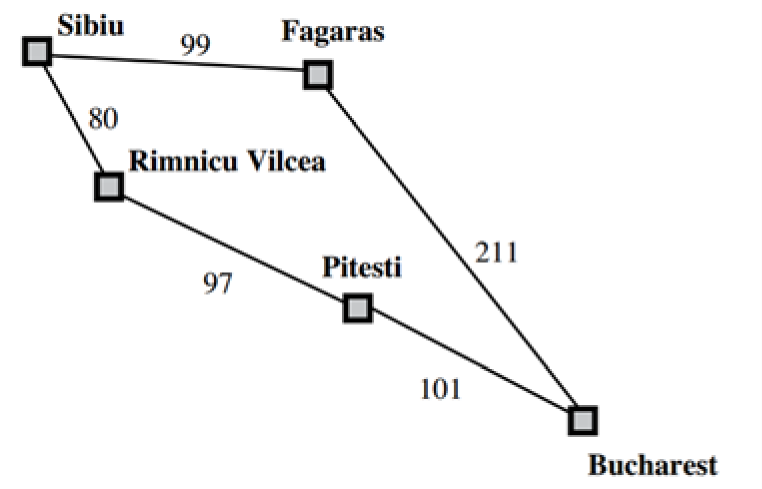
Uniform-cost Search (UCS)

Expands the node with **lowest path cost**

UCS keeps going after goal node has been generated, and **terminate after a goal node is expanded.**

Example: From Sibiu (S) to Bucharest (B)

- Start with S, expand it and get RV and F
- **Goal test S**
- Then, since RV cost 80, F cost 99, so we expand RV and get P
- **Goal test RV**
- Then, since F cost 99, P cost 80+97=177, so we expand F and get B (**and we continue!!**)
- **Goal test F**
- Then, since B cost 99+211=310, P cost 80+97=177, so we expand P and get B again
- **Goal test P**
- Then we expand and do goal test for B. It has 2 path, we choose the path with minimum cost



Uniform-cost Search (UCS)

Evaluation:

- Complete (if b is finite and the step costs has a positive lower bound ϵ)
- Optimal in general (**cost = cumulative path cost instead of depth for UCS**)
- Time complexity:
  - C: cost (NOT depth) of optimal solution
  - Every action costs at least ϵ
  - Worst case time complexity: $\boldsymbol{O(b^{1+\lfloor C/\epsilon \rfloor})}$
    - If all step costs are equal: $O(b^{1+d})$ (similar to BFS but test on expansion)
- Space complexity:
  - Fringe: priority queue (where priority is cumulative cost)
  - Worst case space complexity: roughly the last tier, $\boldsymbol{O(b^{1+\lfloor C/\epsilon \rfloor})}$

Uniform-cost Search & BFS

BFS:

- Stops after a goal node is generated (unless otherwise specified)
- Optimal when we have unit step cost

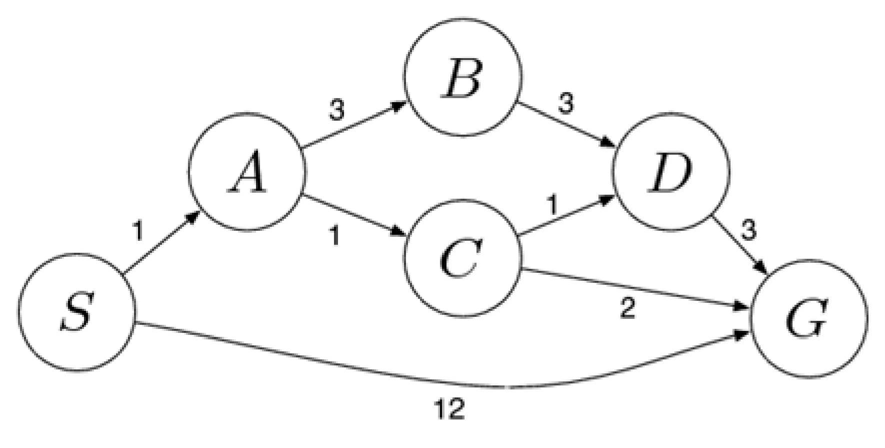
UCS:

- Keeps going after a goal node has been generated (since it do goal test when expansion)
- Can deal with the case that all step costs are not equal
- When we have unit step cost, UCS is similar to BFS (except the timing of goal test)

Uniform-cost Search (UCS) - Example

Use uniform-cost search (assume S is initial state, G is goal state)

- Give the generated search tree?
- Show in what order we expand nodes (choose left-to-right when have tie)?
- Return the optimal solution/path?



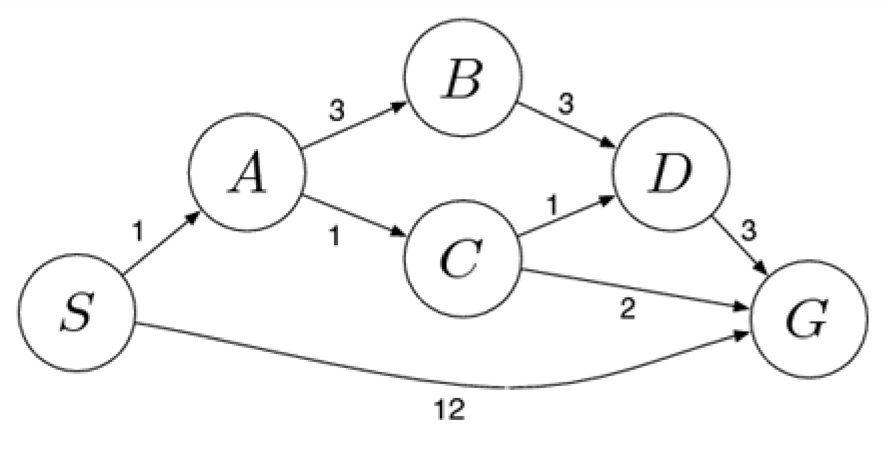
Uniform-cost Search (UCS) - Example

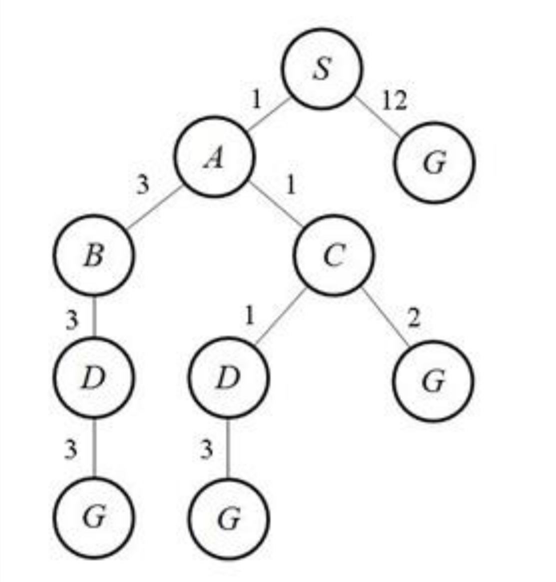
Use uniform-cost search (assume S is initial state, G is goal state)

- Give the generated search tree?
- Show in what order we expand nodes (choose left-to-right when have tie)?

  A: S->A->C->D->B->G
- Return the optimal solution/path?

  A: S->A->C->G, cost is 4



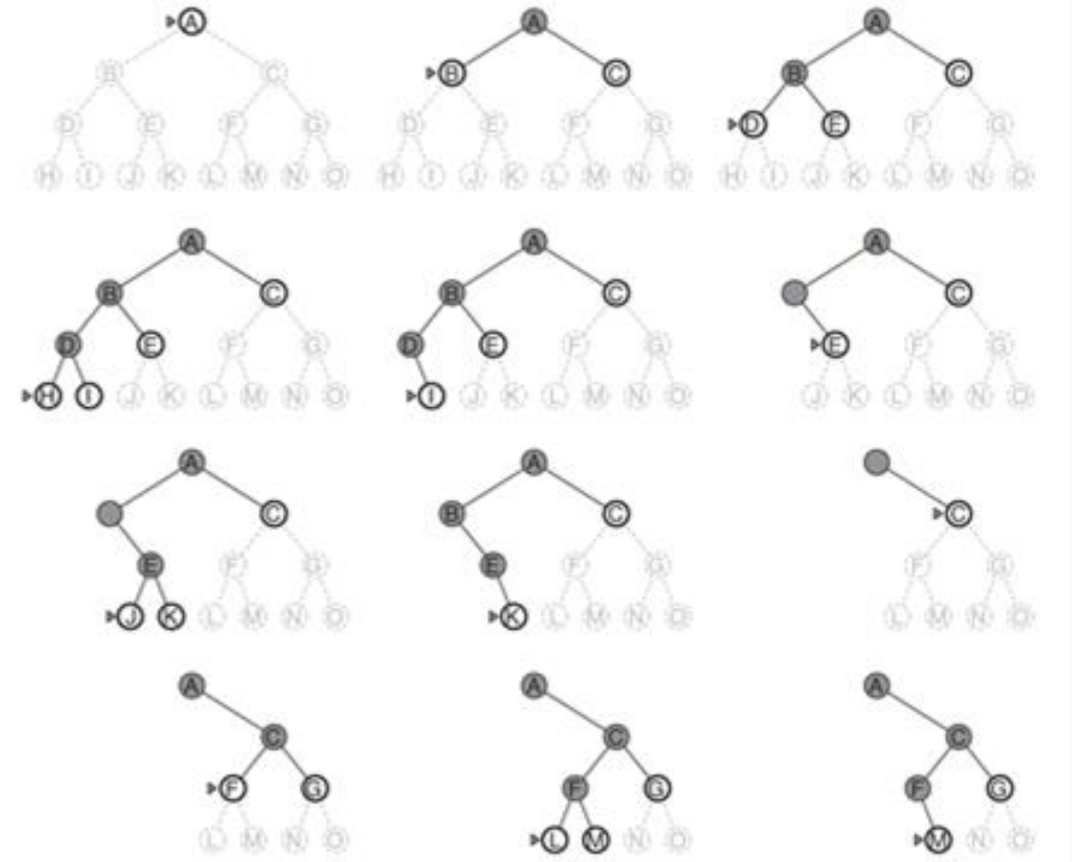


Depth-first Search (DFS)

Expand **deepest** unexpanded node

Example:

- M is target, shallowest solution depth d = 3
- State space max depth m = 3
- Light grey nodes: unexplored region
- Grey nodes: expanded nodes
- White nodes: generated but not expanded node ⇒ frontier / fringe
- Removed nodes: explored nodes with no descendants in the frontier



Depth-first Search (DFS)

Evaluation:

- Not complete (a tree can be unbounded)
- Not optimal (if **m** > **d**)
- Time complexity: $\boldsymbol{O(b^m)}$**,** where m is maximum depth
  - $1+b+b^2+\ldots b^m$
  - If tree is unbounded, m is infinite
- Space complexity: $\boldsymbol{O(bm)}$ → linear space!
  - Fringe = nodes on current search path + siblings along the path
  - # nodes keep in memory: b+b+...+b (m times)

Depth-limited Search

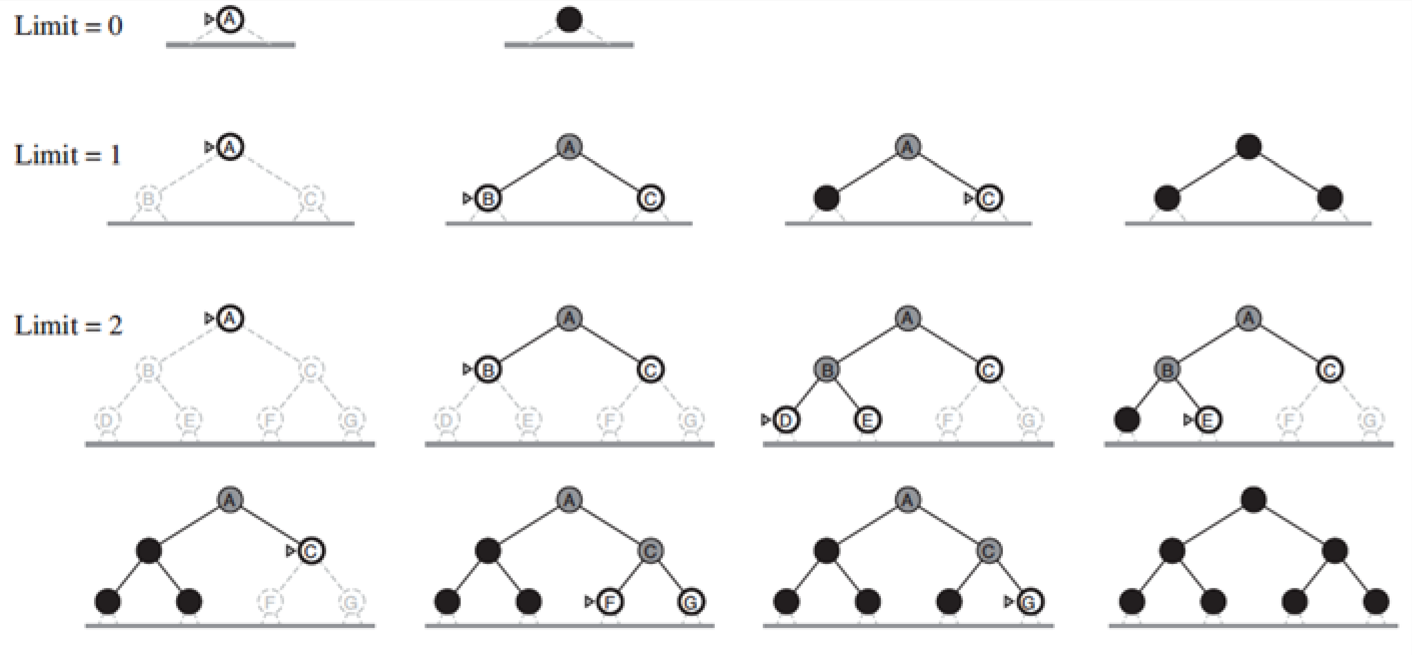
- Compare to DFS, it add a depth limit L (often written in lowercase l)
- Even if the tree can be further expanded, we stopped when reach depth L

Evaluation:

- Not complete in general, only complete if L >= d
- Not optimal (same as DFS)
- Time complexity: **$O(b^L)$**, where L is the depth limit
- Space complexity is **$O(bL)$**

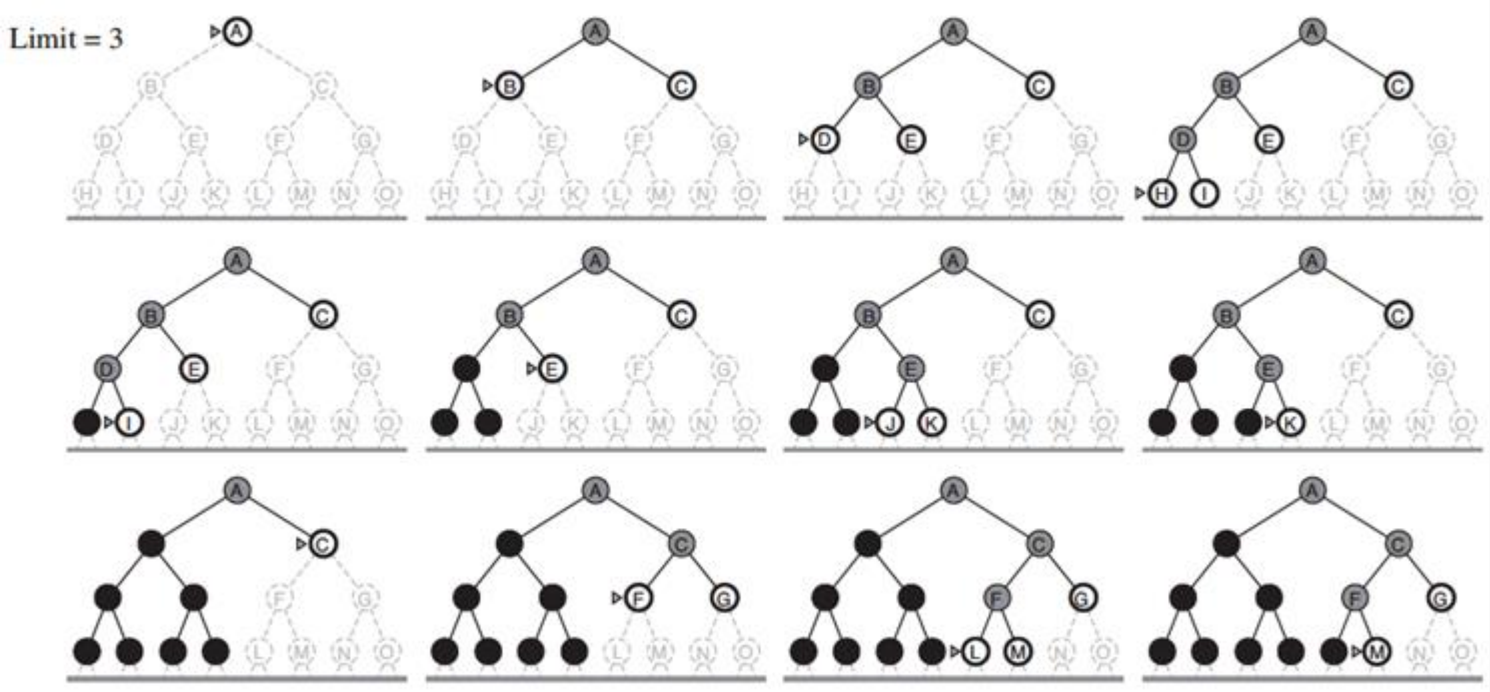
Iterative Deepening Search (ID / IDS)

- Similar to DFS and Depth-limited Search given a particular depth limit
- **Increase depth limits** until a goal is found



Iterative Deepening Search (ID / IDS)

- Similar to DFS and Depth-limited Search, but **increase depth limits** until a goal is found
- Has redundancy but won't matter (see details in Lecture-4-B video from ~38:20)



Iterative Deepening Search (ID / IDS)

Evaluation:

- Complete (if b is finite)
- Optimal for unit step costs (i.e. each step's cost is same)
- Time complexity: $\mathbf{O(b^d)}$, where d is the depth of the shallowest solution
  - $db+(d-1)b^2+...b^d$, where the last item b^d dominates
  - This is because the children of root have been generated d times (when depth limit l goes from 1 to d, we need to generate them), then the children of those children of root have been generated d-1 times, and so on...
- Space complexity is $\mathbf{O(bd)}$

Summary of Uninformed Search

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

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.

Exercise - Word Ladder

- Input:
  - A beginWord
  - An 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
  - How many transformations we need **at least**?
    - Return 0 if no such sequence

How do we solve it?

- Assumptions:
  - All words have the same length
  - All words contain only lowercase alphabetic characters
  - No duplicates in the word list
  - beginWord and endWord are non-empty and are not the same

Exercise - Word Ladder

Example:

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

Questions?

- My slides take the following materials as references:
  - Shirley Chen’s slides
  - Yewen Wang’s (Winter 2020’s TA) slides
  - Prof. Quanquan Gu’s (Winter 2020) slides

Thank you!