

Uninformed Search

Note: this material was originated from the slides provided by Prof. Padhraic Smyth

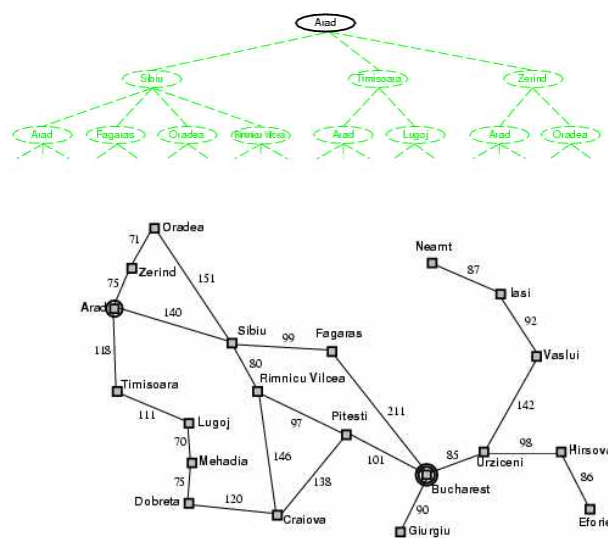
Search Algorithms

- Uninformed Blind search
 - Breadth-first
 - depth-first
 - Iterative deepening depth-first
 - uniform cost
- Informed Heuristic search
 - Greedy search, Heuristics, hill climbing,
- Important concepts:
 - Completeness
 - Time complexity
 - Space complexity
 - Quality of solution

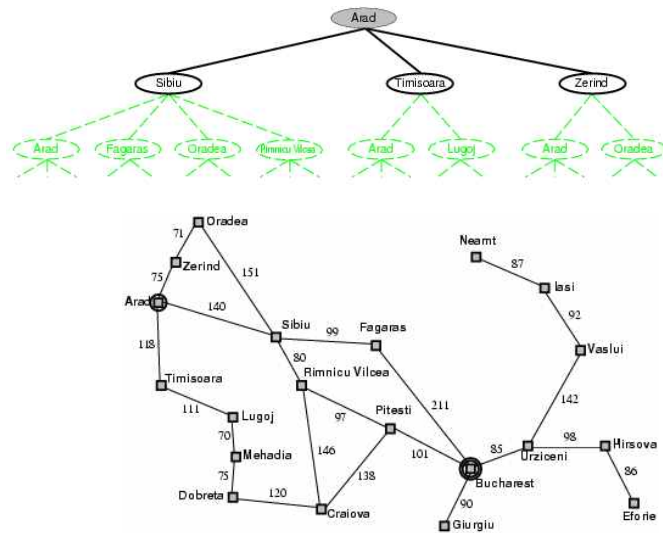
Tree-based Search

- Basic idea:
 - Exploration of state space by generating successors of already-explored states (a.k.a. expanding states).
 - Every state is evaluated: *is it a goal state?*

Tree search example



Tree search example



Search Tree for the 8 puzzle problem

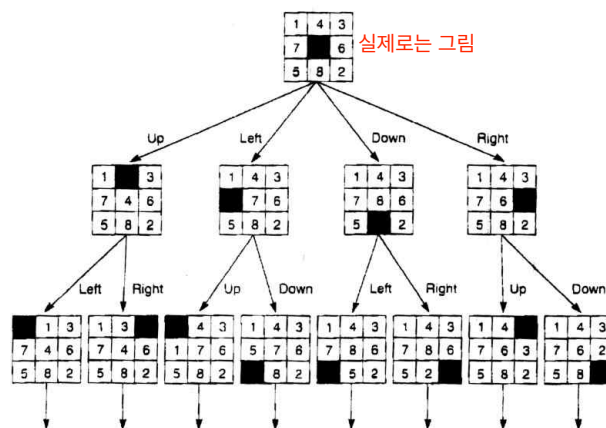


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

Search Strategies

- A **search strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
 - completeness: does it always find a solution if one exists?
 - time complexity: number of nodes generated
 - space complexity: maximum number of nodes in memory
 - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
 - b : maximum branching factor of the search tree
 - d : depth of the least-cost solution
 - m : maximum depth of the state space (may be ∞)

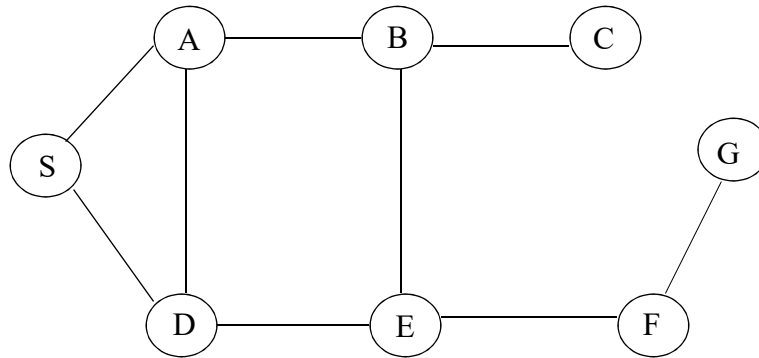
Breadth-First Search (BFS)

- Expand shallowest unexpanded node
- Fringe: nodes waiting in a queue to be explored, also called **OPEN**
- Implementation:
 - For BFS, *fringe* is a first-in-first-out (FIFO) queue
 - new successors go at end of the queue
- Repeated states?
 - Simple strategy: do not add parent of a node as a leaf

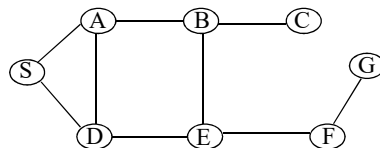
Example: Map Navigation

State Space:

S = start, G = goal, other nodes = intermediate states, links = legal transitions



BFS Search Tree



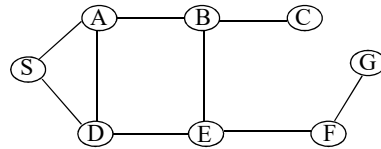
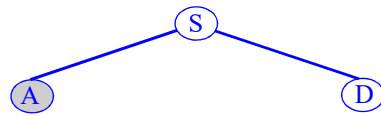
Queue = {S}

Select S

Goal(S) = true?

If not, Expand(S)

BFS Search Tree



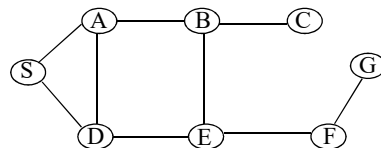
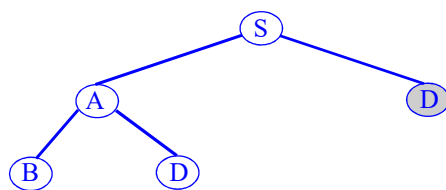
Queue = {A, D}

Select A

Goal(A) = true?

If not, Expand(A)

BFS Search Tree



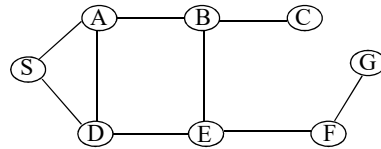
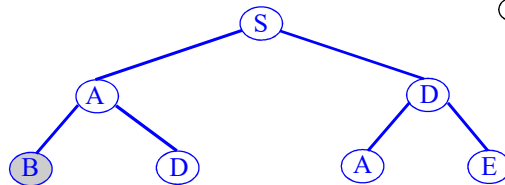
Queue = {D, B, D}

Select D

Goal(D) = true?

If not, expand(D)

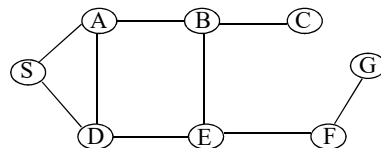
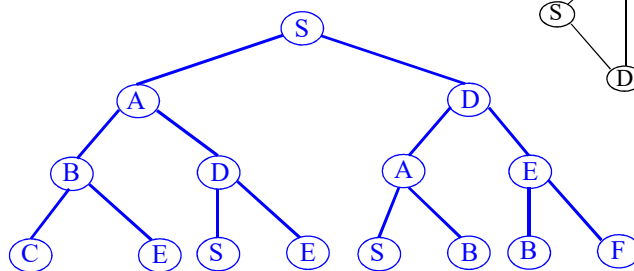
BFS Search Tree



Queue = {B, D, A, E}

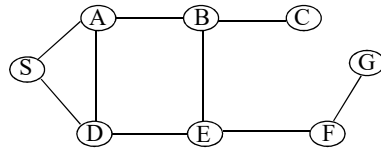
Select B
etc.

BFS Search Tree



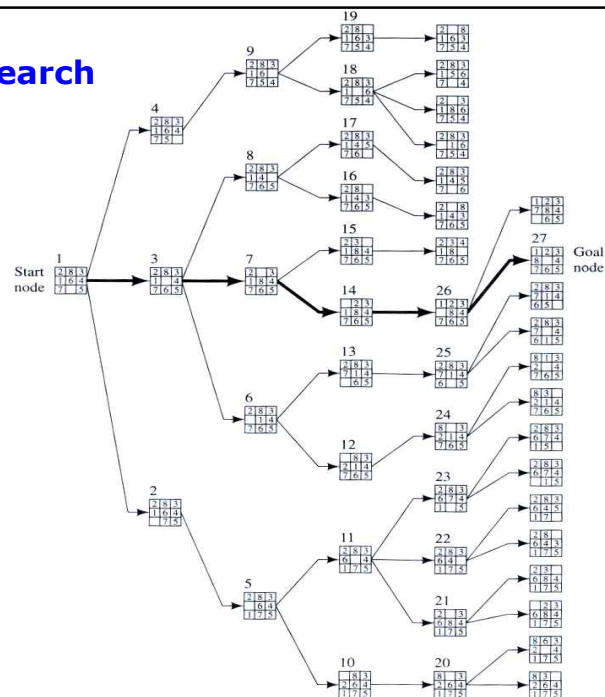
Level 3
Queue = {C, E, S, E, S, B, B, F}

BFS Search Tree



Level 4
Expand queue until G is at front
Select G
Goal(G) = true

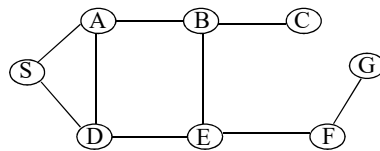
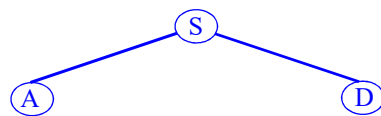
Breadth-First Search



Depth-First Search (DFS)

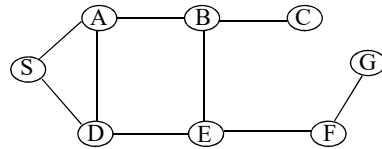
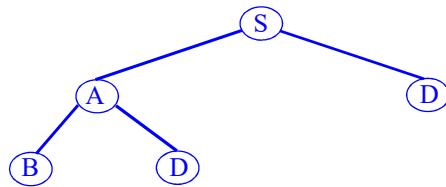
- Expand deepest unexpanded node
- Implementation:
 - For DFS, *fringe* is a Last-in-first-out (LIFO) stack
 - new successors go at beginning of the stack
- Repeated nodes?
 - Simple strategy: Do not add a state as a leaf if that state is on the path from the root to the current node

DFS Search Tree



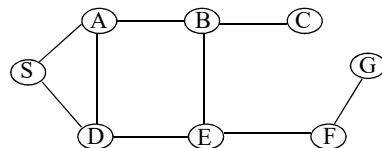
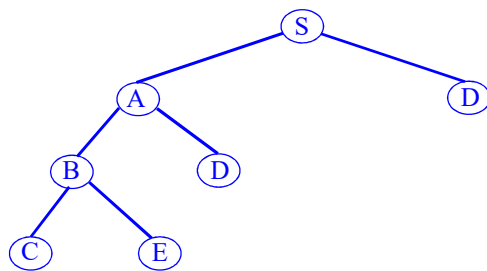
Stack = {A,D}

DFS Search Tree



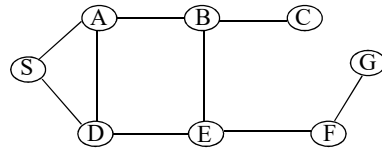
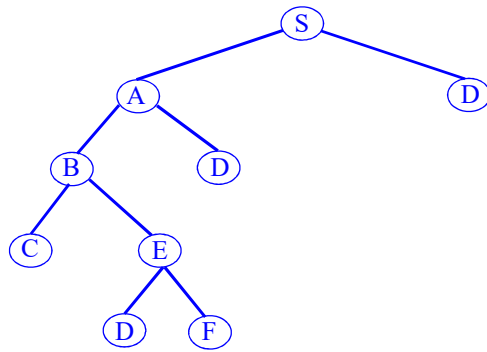
Stack = {B,D,D}

DFS Search Tree



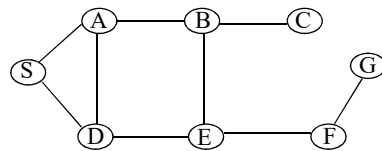
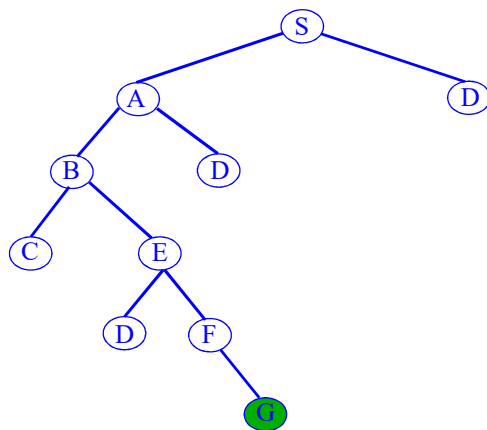
Stack = {C,E,D,D}

DFS Search Tree



Stack = {D,F,D,D}

DFS Search Tree



Stack = {G,D,D}

Evaluation of Search Algorithms

- Completeness
 - does it always find a solution if one exists?
- Optimality
 - does it always find a least-cost (or min depth) solution?
- Time complexity
 - number of nodes generated (worst case)
- Space complexity
 - number of nodes in memory (worst case)
- Time and space complexity are measured in terms of
 - b : maximum branching factor of the search tree
 - d : depth of the least-cost solution
 - m : maximum depth of the state space (may be ∞)

Breadth-First Search (BFS) Properties

- Complete? Yes
- Optimal? Yes
- Time complexity $O(b^d)$
- Space complexity $O(b^d)$
- Main practical drawback? exponential space complexity

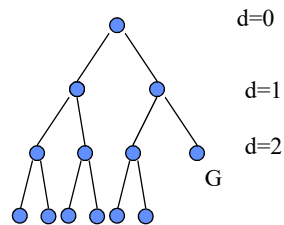
Complexity of Breadth-First Search

- **Time Complexity**

- assume (worst case) that there is 1 goal leaf at the RHS at depth d
- so BFS will generate

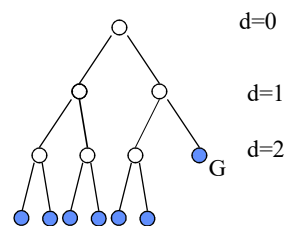
$$= b + b^2 + \dots + b^d + b^{d+1} - b$$

$$= O(b^{d+1})$$



- **Space Complexity**

- how many nodes can be in the queue (worst-case)?
- at depth d there are b^{d+1} unexpanded nodes in the Q = $O(b^{d+1})$



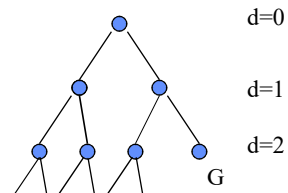
Examples of Time and Memory Requirements for Breadth-First Search

Assuming $b=10$, speed = 10000 nodes/sec, node_size = 1kbyte/node

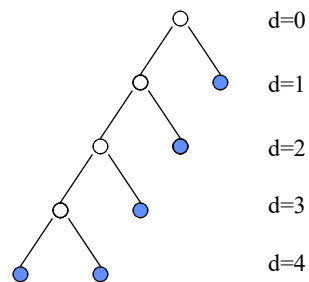
Depth of Solution	Nodes Generated	Time	Memory
2	1100	0.11 seconds	1 MB
4	111,100	11 seconds	106 MB
8	10^9	31 hours	1 TB
12	10^{13}	35 years	10 PB

What is the Complexity of Depth-First Search?

- Time Complexity
 - maximum tree depth = m
 - assume (worst case) that there is 1 goal leaf at the RHS at depth d
 - so DFS will generate **$O(b^m)$**



- Space Complexity
 - how many nodes can be in the queue (worst-case)?
 - at depth m we have b nodes
 - and $b-1$ nodes at earlier depths
 - total = $b + (m-1)*(b-1) = O(bm)$



Examples of Time and Memory Requirements for Depth-First Search

Assuming $b=10$, $m = 12$, speed=10000 nodes/sec, node_size=1kbyte/node

Depth of Solution	Nodes Generated	Time	Memory
2	10^{12}	3 years	120kb
4	10^{12}	3 years	120kb
8	10^{12}	3 years	120kb
12	10^{12}	3 years	120kb

Depth-First Search (DFS) Properties

- Complete?
 - Not complete if tree has unbounded depth
- Optimal?
 - No
- Time complexity?
 - Exponential
- Space complexity?
 - Linear

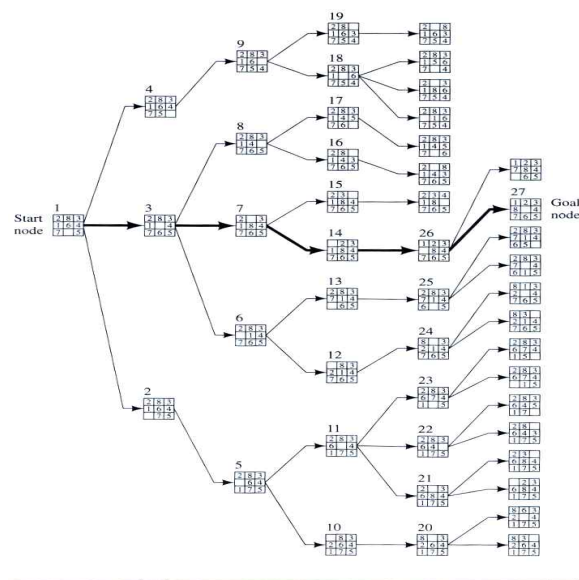
Comparing DFS and BFS

- Time complexity: same, but
 - In the worst-case BFS is always better than DFS
 - Sometime, on the average DFS is better if:
 - many goals, no loops and no infinite paths
- BFS is much worse memory-wise
 - DFS is linear space **stack이 완만하게 쌓인다**
 - BFS may store the whole search space. **감당 못할 정도로 커지기도 한다**
- In general
 - BFS is better if goal is not deep, if infinite paths, if many loops, if small search space
 - DFS is better if many goals, not many loops, **DFS가 메모리 방면에서 현실적**
 - DFS is much better in terms of memory

DFS with a depth-limit L

- Standard DFS, but tree is not explored below some depth-limit L
- Solves problem of infinitely deep paths with no solutions
 - But will be incomplete if solution is below depth-limit
- Depth-limit L can be selected based on problem knowledge
 - E.g., diameter of state-space:
 - E.g., max number of steps between 2 cities
 - But typically not known ahead of time in practice

Depth-First Search with a depth-limit, L = 5



Iterative Deepening Search (IDS)

- Run multiple DFS searches with increasing depth-limits

Iterative deepening search

- $L = 1$
- While no solution, do
 - DFS from initial state S_0 with cutoff L
 - If found goal,
 - stop and return solution,
 - else, increment depth limit L

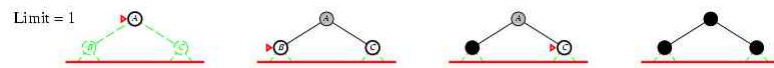
depth를 ++++ 하면서 찾을 때까지 돌린다

Iterative deepening search $L=0$

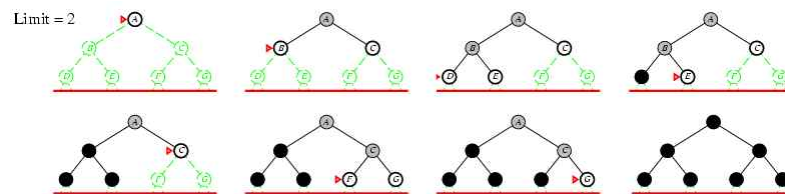
Limit = 0



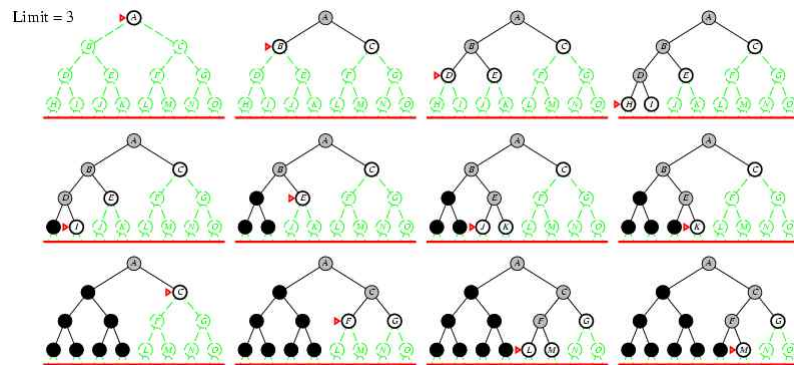
Iterative deepening search $L=1$



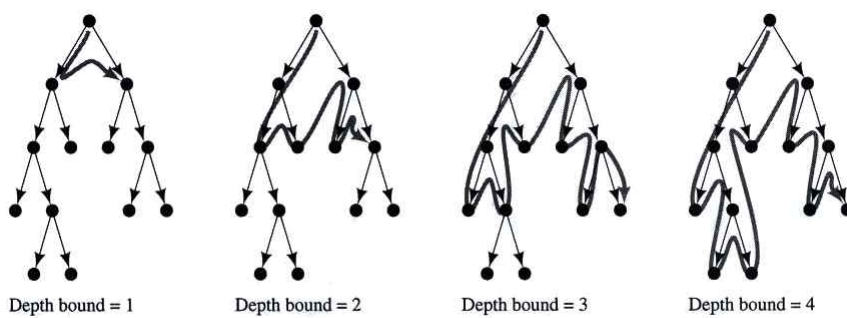
Iterative deepening search $L=2$



Iterative Deepening Search $L=3$



Iterative deepening search



Stages in Iterative-Deepening Search

Properties of Iterative Deepening Search

- Space complexity = $O(bd)$
 - (since its like depth first search run different times, with maximum depth limit d)
- Time Complexity **함해봤자 $(b + \dots b^d)$ 너무 크다**
 - $b + (b+b^2) + \dots (b+\dots b^d) = O(b^d)$
(i.e., asymptotically the same as BFS or DFS to limited depth d in the worst case)
 $1/(b-1)$ 만큼 증가
- Complete?
 - Yes
- Optimal
 - Only if path cost is a non-decreasing function of depth
- IDS combines the small memory footprint of DFS, and has the completeness guarantee of BFS

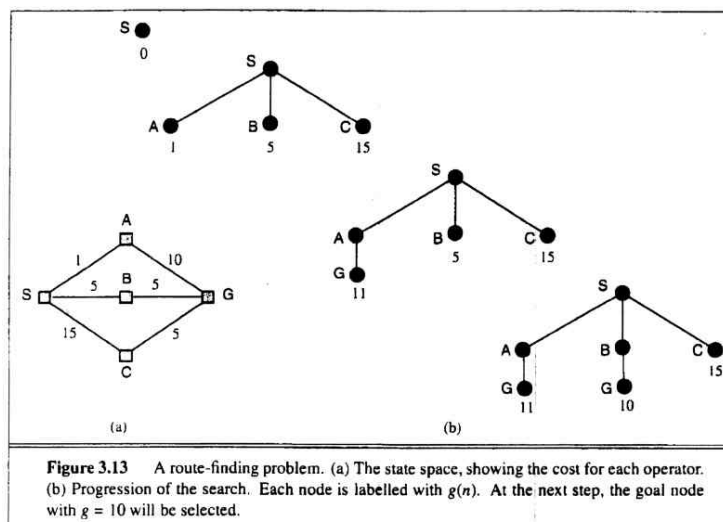
IDS in Practice

- Isn't IDS wasteful?
 - Repeated searches on different iterations
 - Compare IDS and BFS:
 - E.g., $b = 10$ and $d = 5$
 - $N(\text{IDS}) \sim db + (d-1)b^2 + \dots b^d = 123,450$
 - $N(\text{BFS}) \sim b + b^2 + \dots b^d = 111,110$
 - Difference is only about 10%
 - Most of the time is spent at depth d , which is the same amount of time in both algorithms
- In practice, IDS is the preferred uniform search method with a large search space and unknown solution depth

Uniform Cost Search

- Optimality: path found = lowest cost
 - Algorithms so far are only optimal under restricted circumstances
- Let $g(n)$ = cost from start state S to node n
- Uniform Cost Search:
 - Always expand the node on the fringe with minimum cost $g(n)$
 - Note that if costs are equal (or almost equal) will behave similarly to BFS

Uniform Cost Search



Optimality of Uniform Cost Search?

- Assume that every step costs at least $\epsilon > 0$
- Proof of Completeness:
Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach it in a finite number of steps.
- Proof of Optimality given Completeness:
 - Assume UCS is not optimal.
 - Then there must be a goal state with path cost smaller than the goal state which was found (invoking completeness)
 - However, this is impossible because UCS would have expanded that node first by definition.
 - Contradiction.

Complexity of Uniform Cost

- Let C^* be the cost of the optimal solution
- Assume that every step costs at least $\epsilon > 0$
- Worst-case time and space complexity is:

$$O(b^{1 + \lfloor C^*/\epsilon \rfloor})$$

Why?

$\lfloor C^*/\epsilon \rfloor \sim$ depth of solution if all costs are approximately equal

Comparison of Uninformed Search Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Summary

- A review of search
 - a search space consists of states and operators: it is a graph
 - a search tree represents a particular exploration of search space
- There are various strategies for “uninformed search”
 - breadth-first
 - depth-first
 - iterative deepening
 - Uniform cost search
- Various trade-offs among these algorithms
 - “best” algorithm will depend on the nature of the search problem
- Next up – heuristic search methods