CS387 – 1st Recitation

Uninformed Search

2/10/2020

Search

Search permeates ALL of AI.

Which choices do we make?

- Problem solving (15 puzzle)
  Move 1, then move 3, then move 2, then move 2, ...
- Natural language
  Ways to map words to parts of speech
- Computer vision
  Ways to map features with object model
- Machine learning
  Possible concepts that fit examples seen so far
- Motion planning
  Sequence of moves to reach goal destination

In search, an intelligent agent is trying to find a set or sequence of actions that will achieve a goal

We are thus looking at a type of *goal-based agent*

Problem-Solving Agent

```
SimpleProblemSolvingAgent(percept)
        state = UpdateState(state, percept)
        if sequence is empty then
                goal = FormulateGoal(state)
                problem = FormulateProblem(state, g)
                sequence = Search(problem)
        action = First(sequence)
        sequence = Rest(sequence)
        return action
```

Assumptions

Static or dynamic?

Assumptions

Environment is static

Assumptions

Environment is static

Fully or partially observable?

Assumptions

Environment is static

Environment is fully observable

Discrete or continuous?

Assumptions

Environment is static

Environment is fully observable

Environment is discrete

Deterministic or stochastic?

Assumptions

Environment is static

Environment is fully observable

Environment is discrete

Environment is deterministic

Search Example

On holiday in Romania, currently in Arad. The flight leaves tomorrow from Bucharest.

Formulate goal:

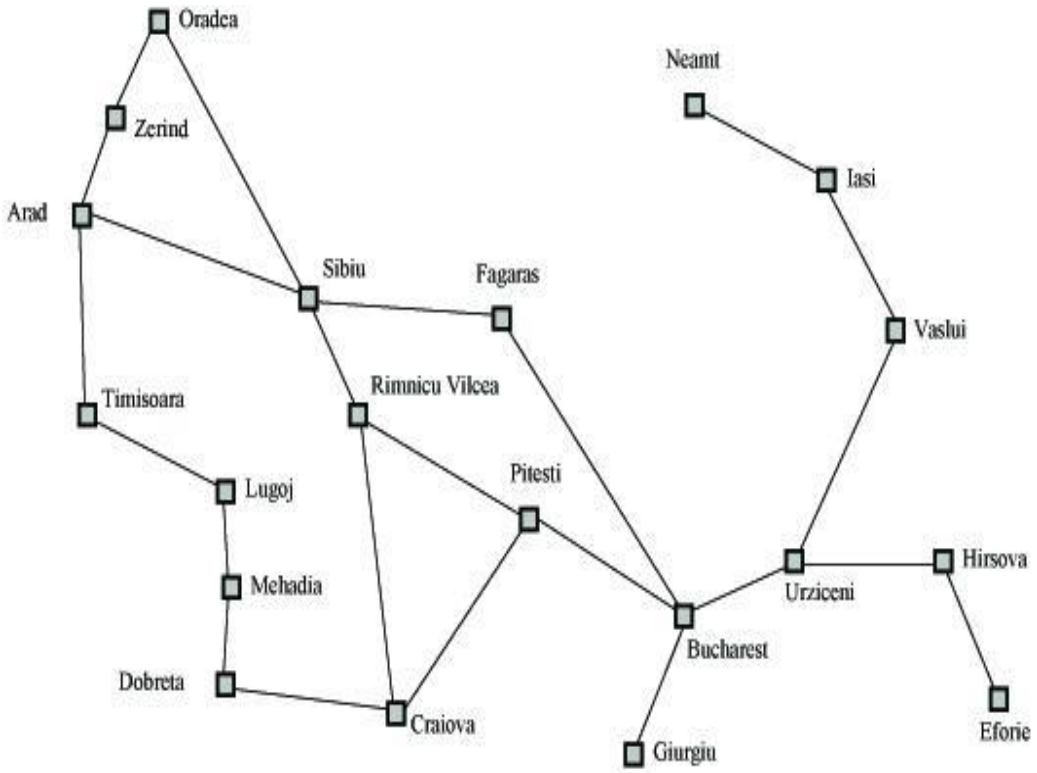
- Be in Bucharest

Formulate problem:

- states are cities, operators are to drive between the pairs of cities

Find solution:

- find the sequence of cities (e.g., Arad, Sibiu, Fagaras, Bucharest) that leads from current state to state meeting goal condition



Search Space Definitions

World State

- A description of a possible state of the world
  (includes all features of the world that are pertinent to the problem)

Initial State

- A description of all pertinent aspects of the world state in which the agent starts the search

Goal Test

- Conditions trying to meet
  - (Have $1,000,000)

Goal State

- Any world state which meets the goal conditions
  - (Thursday, have $1,000,000, live in NYC)
  - (Friday, have $1,000,000, live in Valparaiso)

Action

- Function from a state to a state (transitions from one state to another)

Search Space Definitions

Problem Formulation

- Describe a general problem as a search problem

Solution

- Sequence of actions that transitions the world from the initial state to a goal state

Solution Cost (additive)

- Sum of distances, number of operators, cost of operators, etc.

Search

- Process of looking for a solution
- Search algorithm - algorithm that takes problem as input and returns solution
- We are searching through a space of possible world states

Execution

- Process of executing sequence of actions that comprises problem solution

Problem Formulation

A single-state search problem is defined by the

- Initial state (e.g., Arad)
- Operators (Arad → Zerind, Arad → Sibiu, etc.)
- Goal test (e.g., at Bucharest)
- Solution cost (path cost)

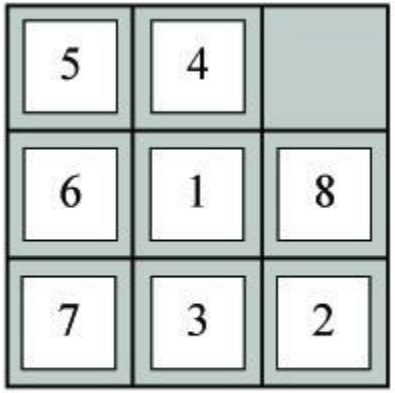
Search Process

3 main steps

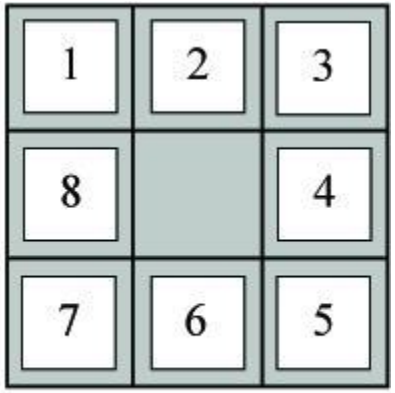
1. Formulate
2. Search
3. Execute

Sample Search Problems

Eight puzzle problem



Start State



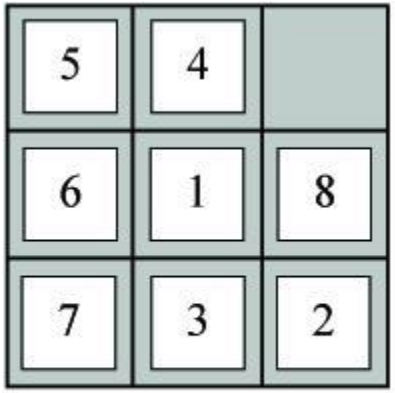
Goal State

What is the problem formulation?

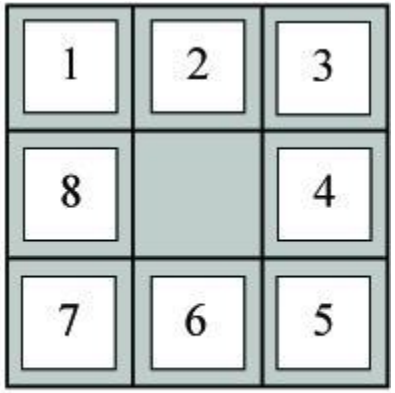
States:
Initial state:
Operators:
Goal test:
Path cost:

Sample Search Problems

Eight puzzle problem



Start State



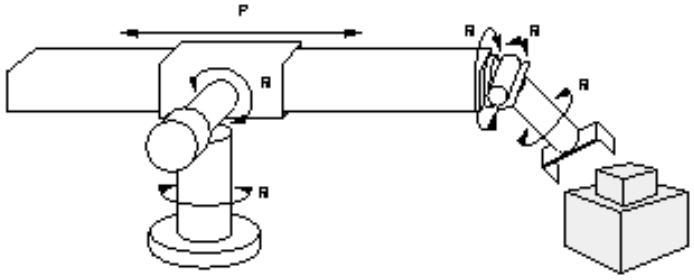
Goal State

What is the problem formulation?

States: tile locations
Initial state: one specific state as shown in picture (henceforth drop if obvious)
Operators: move blank left, right, up, down
Goal test: match goal state (shown in picture)
Path cost: 1 per move

Sample Search Problems

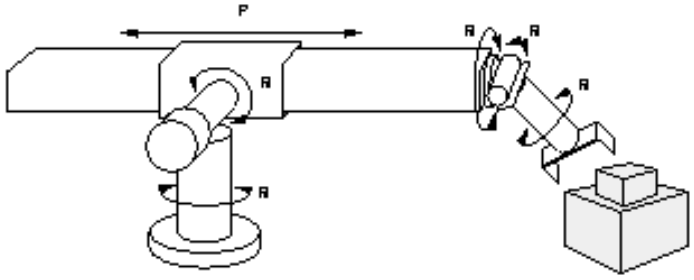
Robotic assembly



States:
Operators:
Goal test:
Path cost:

Sample Search Problems

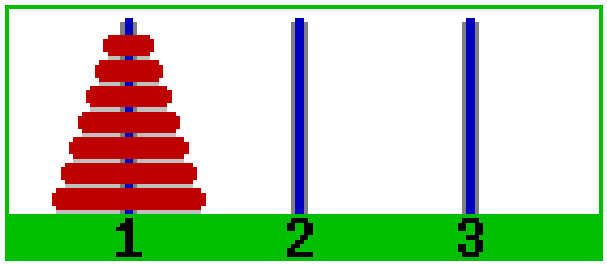
Robotic assembly



States: real-valued coordinates of robot joint angles parts of the object to be assembled
Operators: continuous motions of robot joints
Goal test: complete assembly
Path cost: time to execute

Sample Search Problems

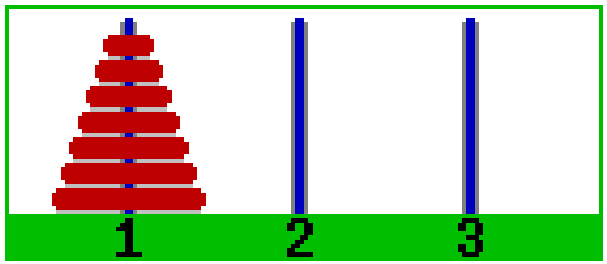
Towers of Hanoi problem



States:
Operators:
Goal test:
Path cost:

Sample Search Problems

Towers of Hanoi problem



States: combinations of poles and disks
Operators: move disk x from pole y to pole z subject to constraints
Goal test: disks from smallest to largest on goal pole
Path cost: 1 per move

Sample Search Problems

Rubik's Cube

States:
Operators:
Goal:
Path Cost:

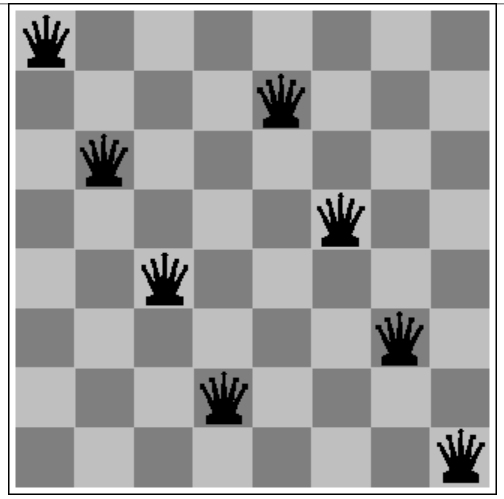
Sample Search Problems

Rubik's Cube

States: list of colors for each cell on each face
Operators: rotate row x or col y on face z direction a
Goal: each face has all one color
Path Cost: 1 per action

Sample Search Problems

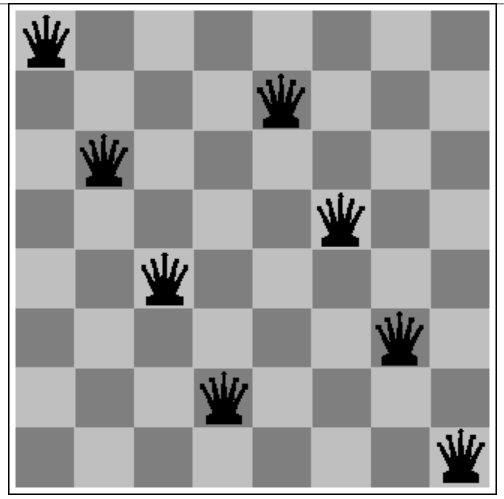
Eight queens problem



States:
Operators:
Goal:
Path Cost:

Sample Search Problems

Eight queens problem



States: locations of 8 queens on chess board
Operators: move queen x to row y and column z
Goal: no queen can attack another
Path Cost: 0 per move

Sample Search Problems

Missionaries and Cannibals

States:
Operators:
Goal:
Path Cost:

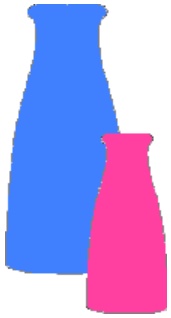
Sample Search Problems

Missionaries and Cannibals

States: number of missionaries, cannibals, and boat on near river bank
Operators: move boat with x missionaries and y cannibals to other side of river (constraints are no more cannibals than missionaries in any location, boat holds at most m occupants)
Goal: all missionaries, cannibals, and boat on far river bank
Path Cost: 1 per river crossing

Sample Search Problems

Water jug problem

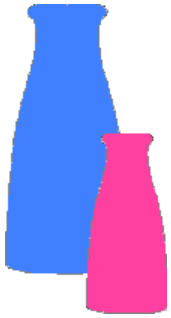


Given a 4-litre and 3-litre water jug and a faucet with unlimited water, fill the 4-litre jug with exactly two litres of water.

States:
Operators:
Goal:
Path Cost:

Sample Search Problems

Water jug problem



Given a 4-litre and 3-litre water jug and a faucet with unlimited water, fill the 4-litre jug with exactly two litres of water.

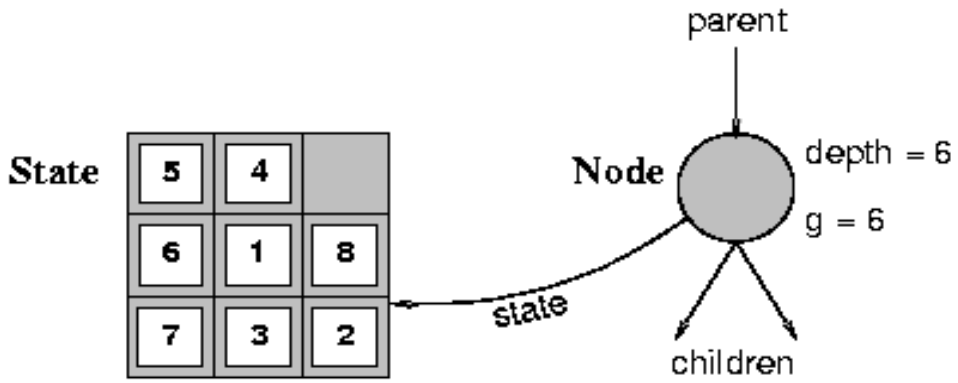
States: Contents of 4-litre jug, contents of 3-litre jug
Initial state is thus (0,0)
Operators: Fill jug x from faucet, pour contents of jug x into jug y, dump contents of jug x down drain
Goal: (2,n)
Path Cost: 1 per fill

Sample Search Problems

- graph coloring problem
- protein folding problem
- game playing
- airline travel
- proving algebraic equalities
- robot motion planning

View Search Space as a Tree

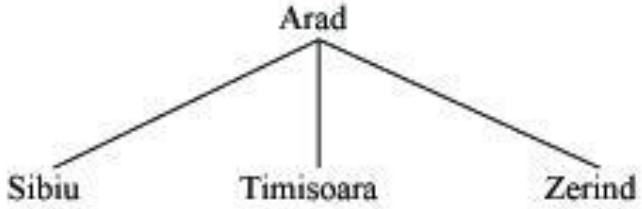
States are nodes
Actions are arcs
Initial state is root
Solution is path from root to goal node
Arcs sometimes have associated costs
Possible resulting states are children of a node



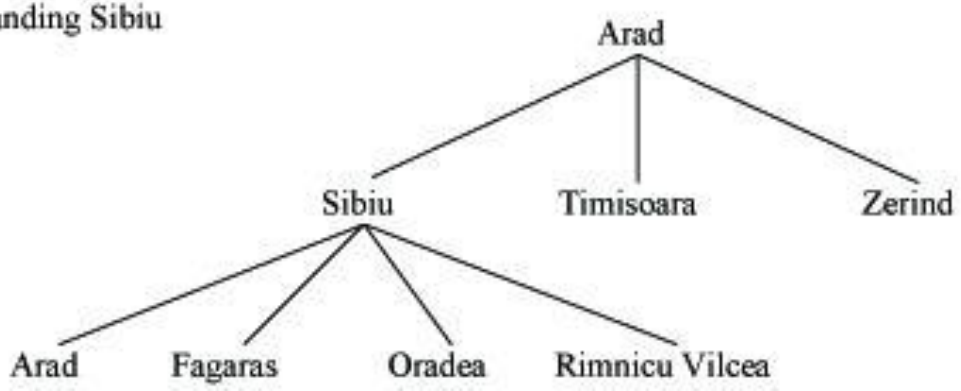
General Search Example

(a) The initial state

(b) After expanding Arad



(c) After expanding Sibiu



Search Function

We will first look at search algorithms that are uninformed

Note: use of the word "enqueued" for "open list"

```
; SEARCH function
open = (list initial)          ; Keep track of the open list, all generated
                               ; states that have not been expanded
while open                     ; One iteration of search algorithm
   state = (first open)        ; Let first item be current world state
   open  = (rest open)         ; Remove current state from open list
   if (goal state)             ; Test to see if current state satisfies goal
      then SUCCEED             ; Search is complete
                               ; Expand the current state (generate children)
   else                        ; Reorder list according to search strategy
      open = (queueing-fn open (expand state))
FAIL
```

Search Strategies

Search strategies differ only in queueing-fn portion of algorithm

Issues to keep in mind

- **Completness** (always find solution)
- **Cost of search** (time and space)
- **Cost of solution, optimal solution**
- **Make use of knowledge of the domain**

  ``uniformed search'' vs. ``informed search''

Breadth-First Search

Generate children of a state, queueing-fn adds the children to the end of the open list

Net effect is all nodes at one level are expanded before any nodes at the next level

Level-by-level search

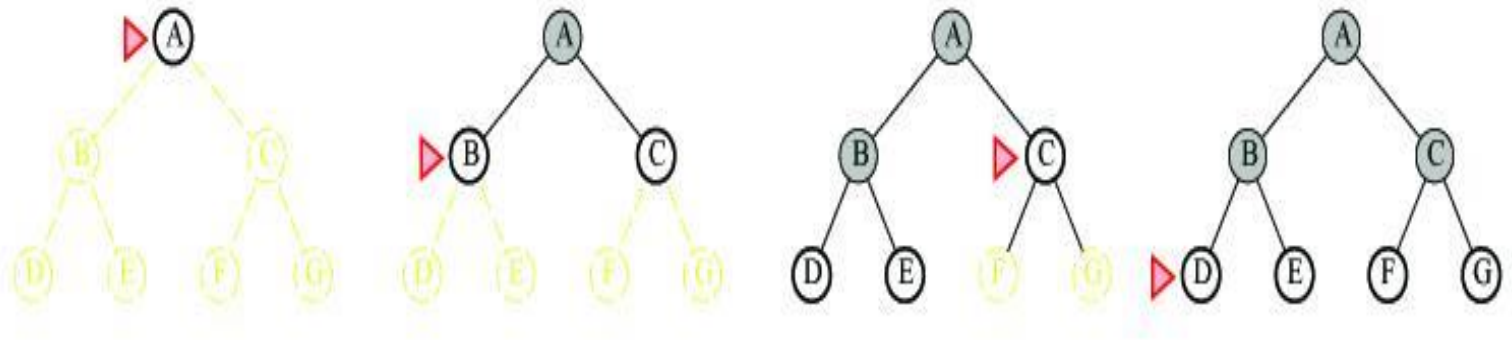
Order in which children are inserted is arbitrary (4 children, which is first?)

In tree, assume children are considered left-to-right unless ordered

Number of children is ``branching factor'' b

Examples

Breadth-first expansion of search tree with a branching factor of 2



Analysis

Assume the solution is at level **d** and branching factor is **b**

Time complexity (number of nodes considered) is

$$1 \text{ (first level)} + b \text{ (second level)} + b^2 \text{ (third level)} + \ldots + b^d \text{ (solution level)} + (b^{d+1} - b) = O(b^{d+1})$$

This assumes solution is on the far right of the solution level This also assumes a constant branching factor b

$$\text{Space complexity (at most a majority of nodes at } d \text{ level} + \text{majority of nodes at } d+1 \text{ level}) = O(b^{d+1})$$

This means exponential time and space

Benefits

- Simple to encode
- Always find solution if one exists (reach any point in space in finite time)
- Least-LENGTH solution
  Not necessarily least-cost solution, unless all operators (actions) have equal cost

Depth-First Search

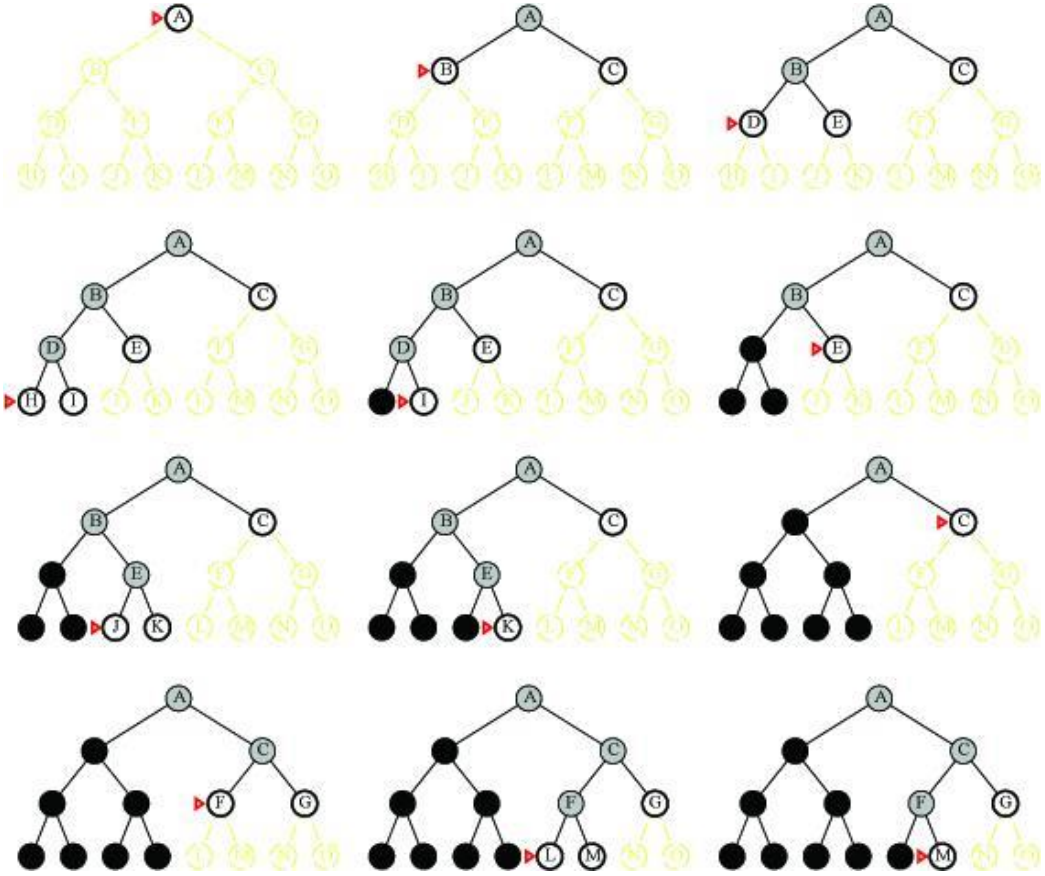
queueing-fn puts children at front of open list

BFS uses FIFO queue, DFS uses LIFO stack

Net effect is to follow leftmost path to the bottom, then incrementally backtrack - also expand deepest node first

Examples

Depth-first expansion of search tree with a branching factor of 2



Analysis

Time complexity
In the worst case, have to search entire search space
Solution may be at level d, but tree may go to level m, $m >= d$, so run time is $O(b^m)$

Particularly bad if tree is infinitely deep

Space complexity
Only have to save 1 set of children at each level
$1 + b + b + \ldots + b$ (m levels total) = $O(bm)$

For previous case, DFS requires 118 kilobytes instead of 10 petabytes at d=12 (10 billion times less)

May not always find solution
Solution it finds is not always least-length or least-cost

If many solutions, can find one quickly (quickly moves to depth d)

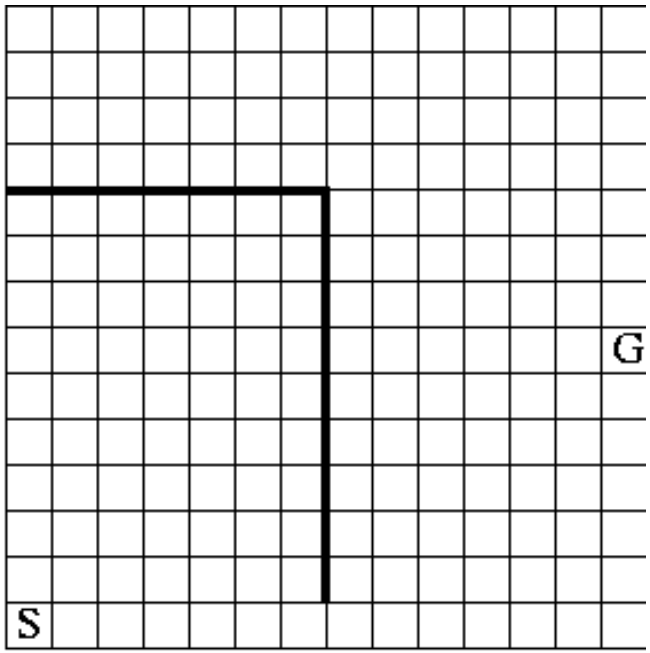
Simple to implement
Space usually bigger constraint than time, so more usable than BFS for large problems

Avoiding Repeated States

- Do not return to the parent state (e.g., in 8 puzzle problem, do not allow the Up move right after a Down move)
- Do not create solution paths with cycles
- Do not generate any repeated states (need to store and check a potentially large number of states)
- This is done by keeping a list of "expanded states" i.e., states whose daughters have already been put on the enqueued list.
  This entails removing states from the "enqueued list" and placing them on an "expanded list" (In the standard algorithm literature, the list of expanded states is called the "closed list ", thus, we would move states from the open list to the closed list)

Maze Example

Imagine states are cells in a maze, and you can move N, E, S, or W. What would BFS do, assuming it always expanded the E child first, then N, then W, then S? What about DFS? What if the order changed to N, E, S, W, and loops are prevented?



Uniform Cost Search (Branch and Bound Search)

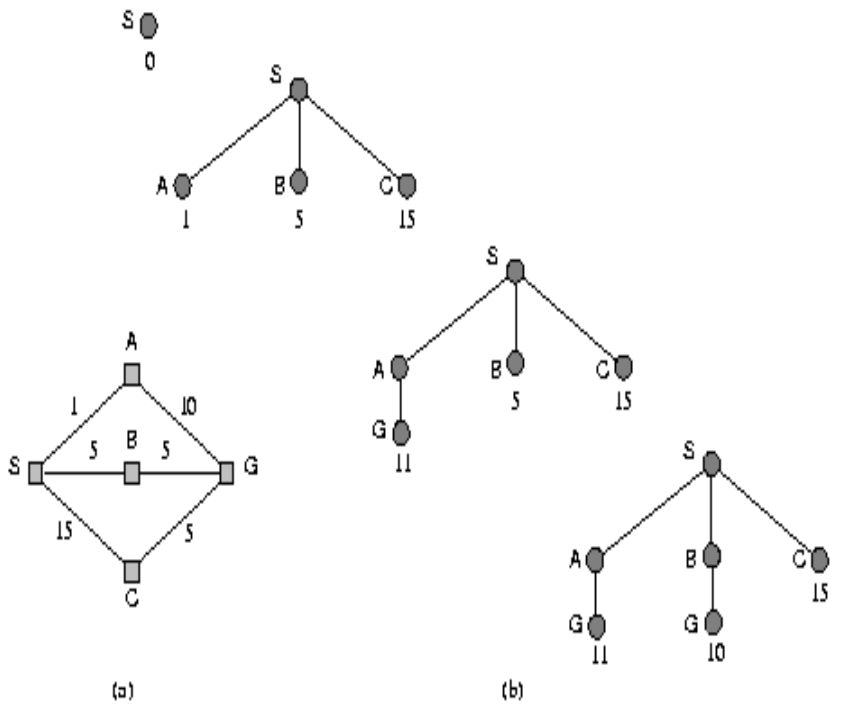
queueing-fn is sort-by-cost-so-far

Cost from root node to current node n is g(n), which adds up individual action costs along path from root to n

Because cheapest path length is always picked until solution is reached, first solution found is least-cost (optimal) solution

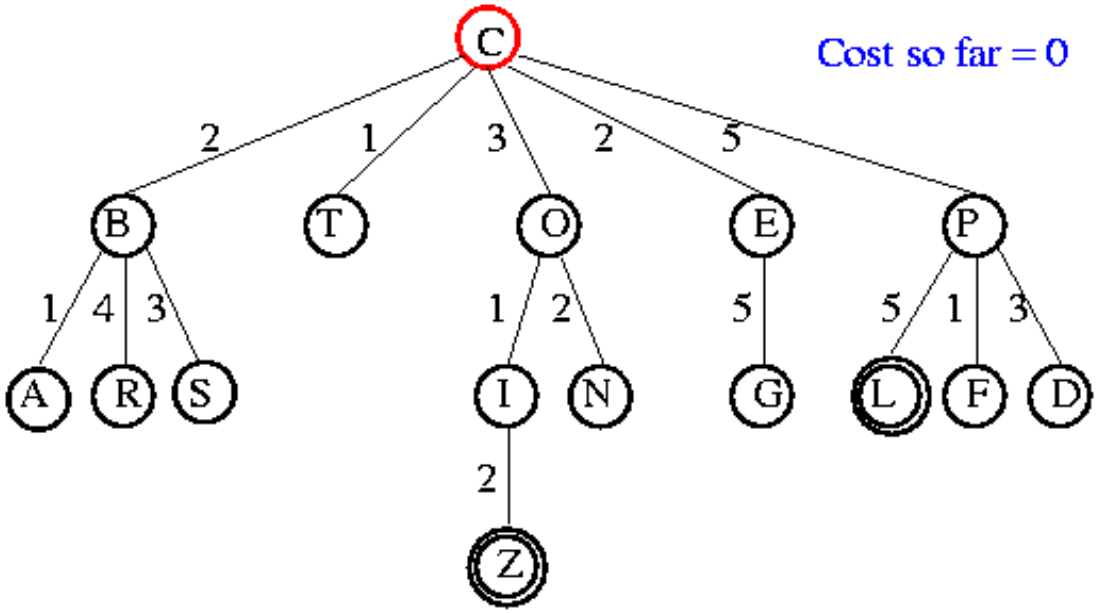
Space and time complexities can be exponential, because large subtrees with inexpensive steps can be explored before useful paths with costly steps.

If costs are equal, space and time are $O(b^d)$. Otherwise, exponent is related to cost of optimal solution.



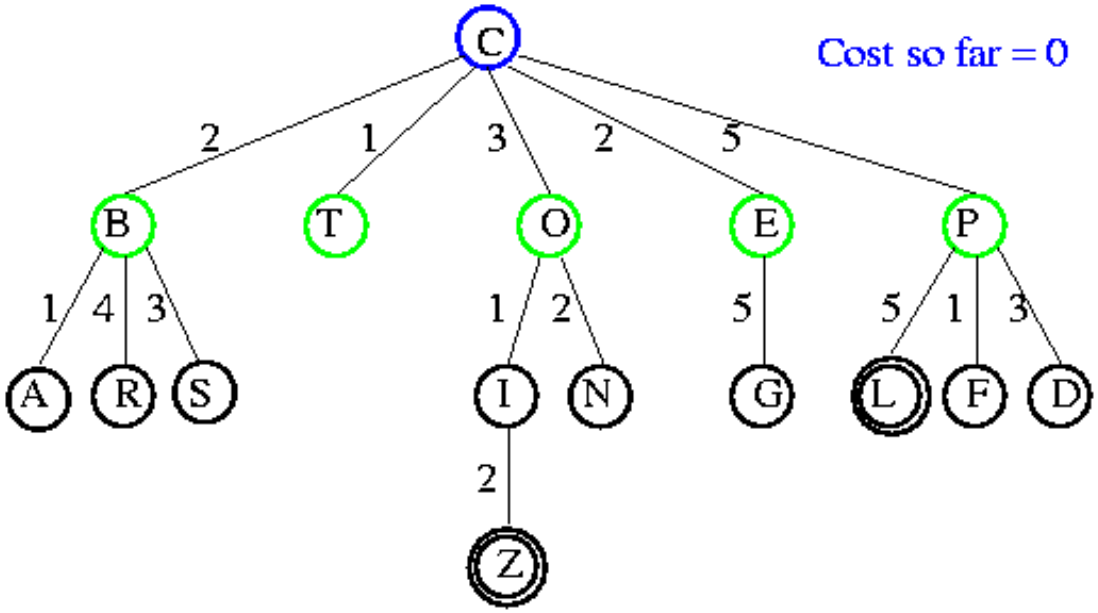
Example

Resolve ties alphabetically and based on depth.



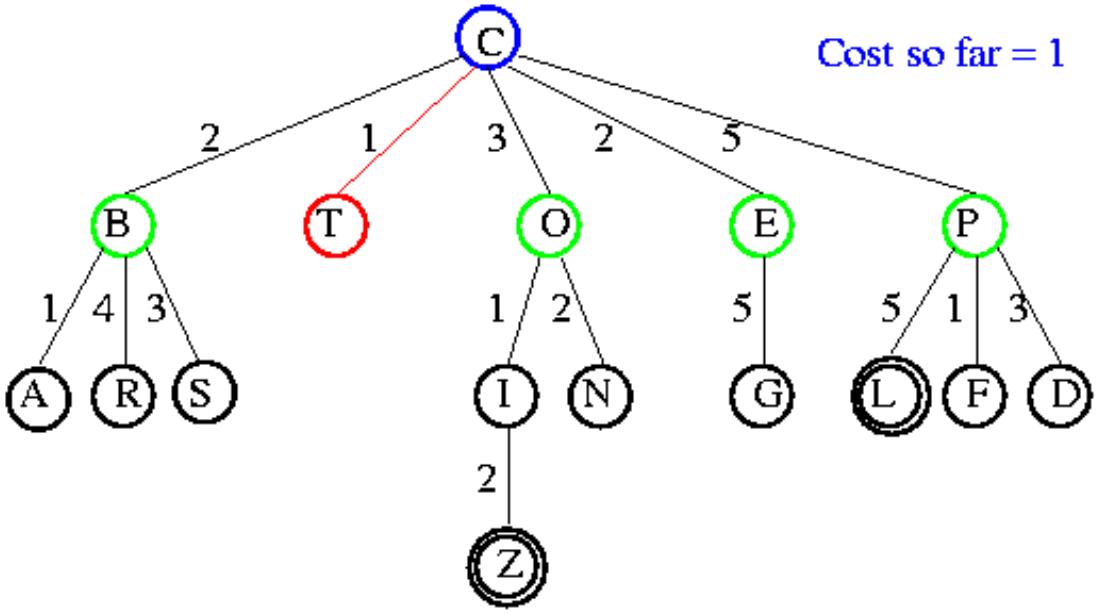
Example

Resolve ties alphabetically and based on depth.



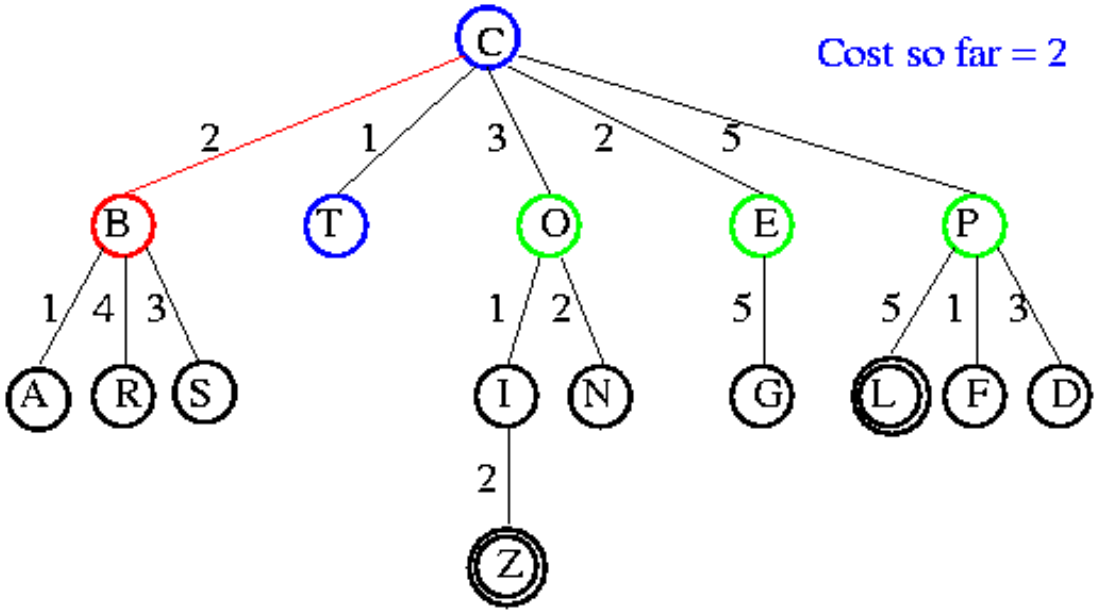
Example

Resolve ties alphabetically and based on depth.



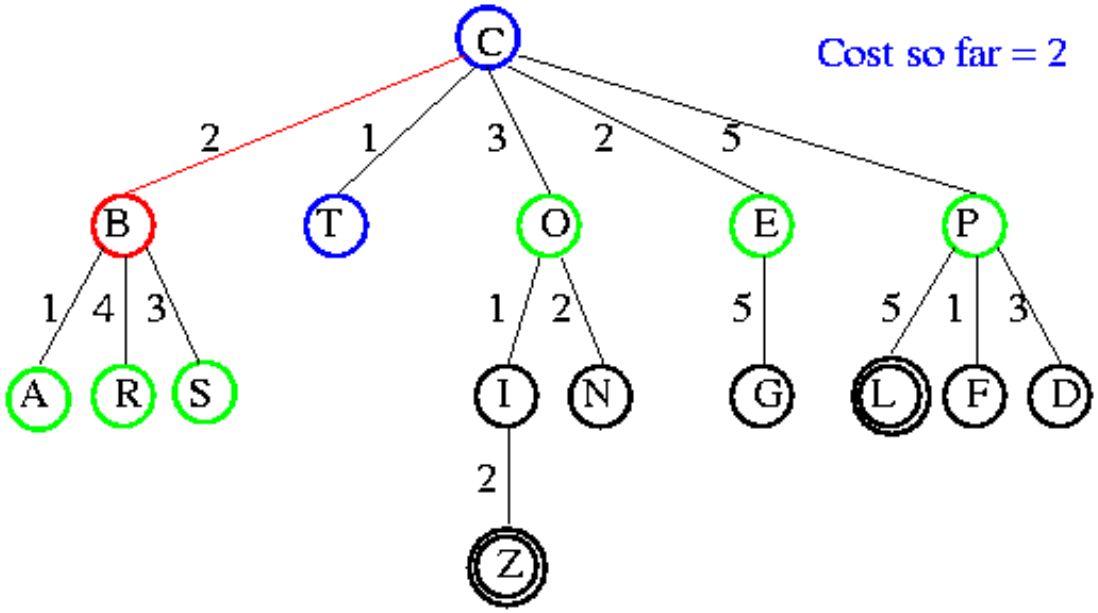
Example

Resolve ties alphabetically and based on depth.



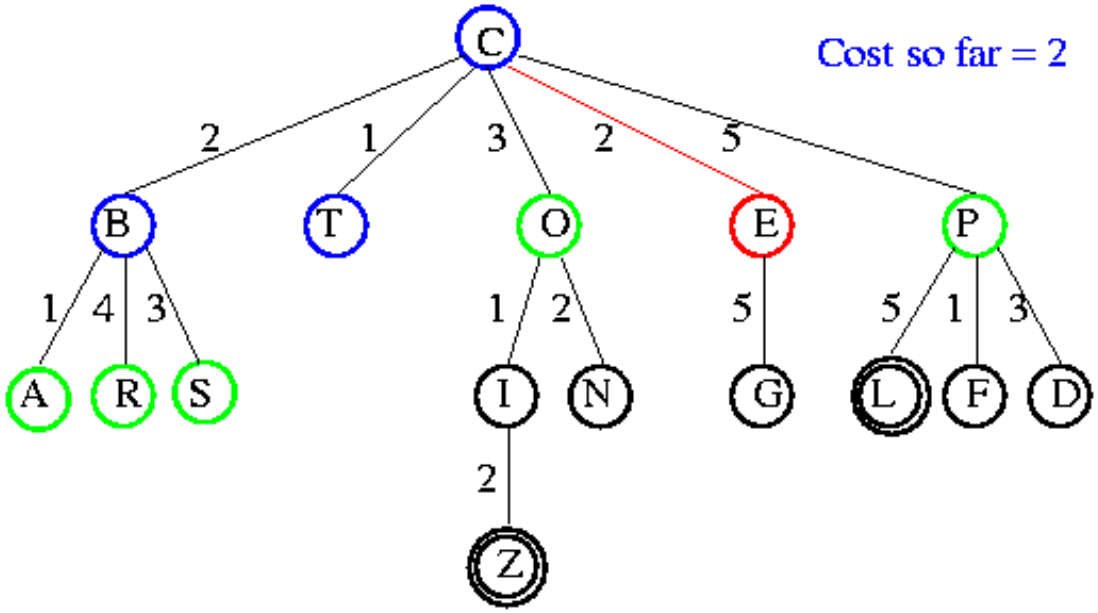
Example

Resolve ties alphabetically and based on depth.



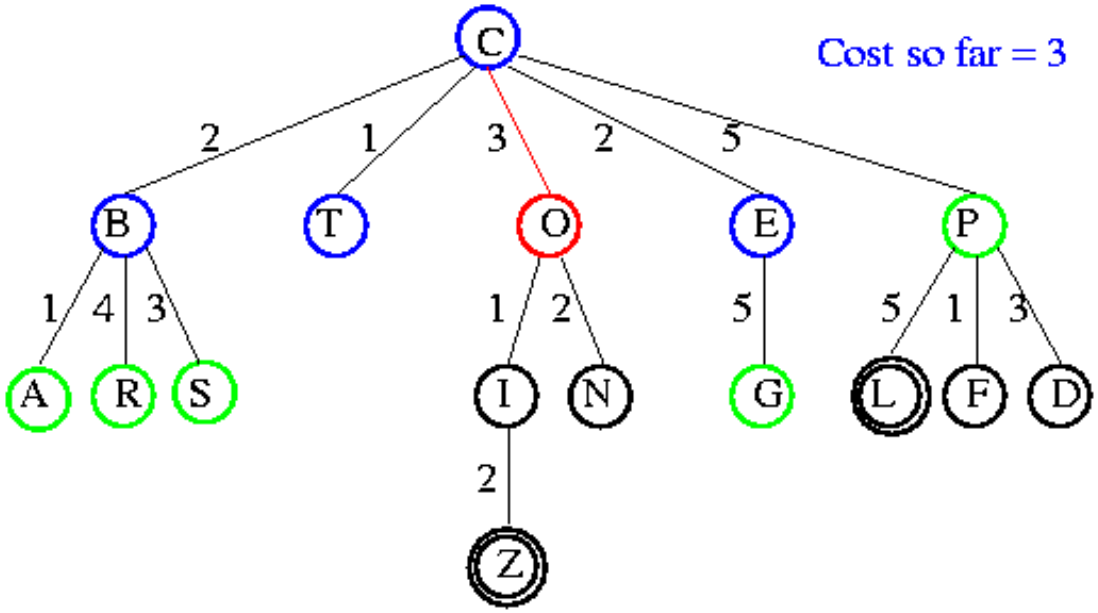
Example

Resolve ties alphabetically and based on depth.



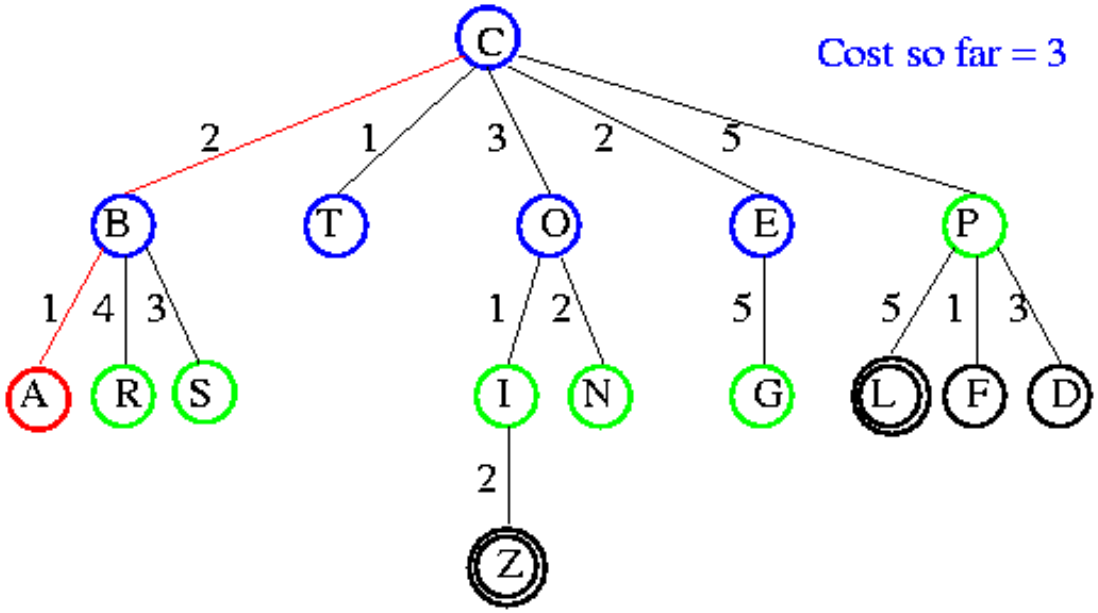
Example

Resolve ties alphabetically and based on depth.



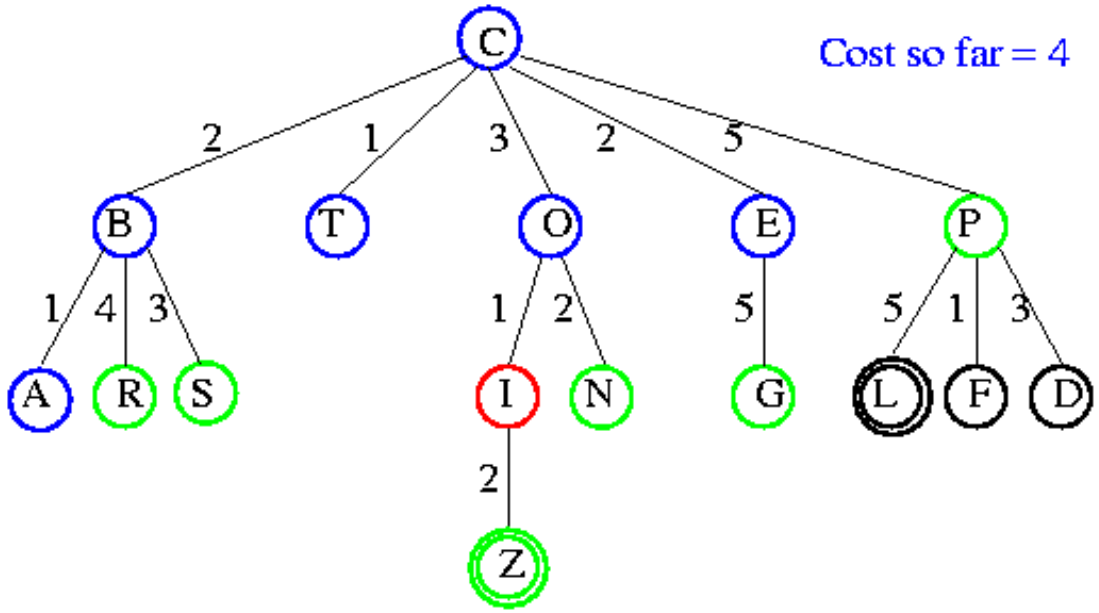
Example

Resolve ties alphabetically and based on depth.



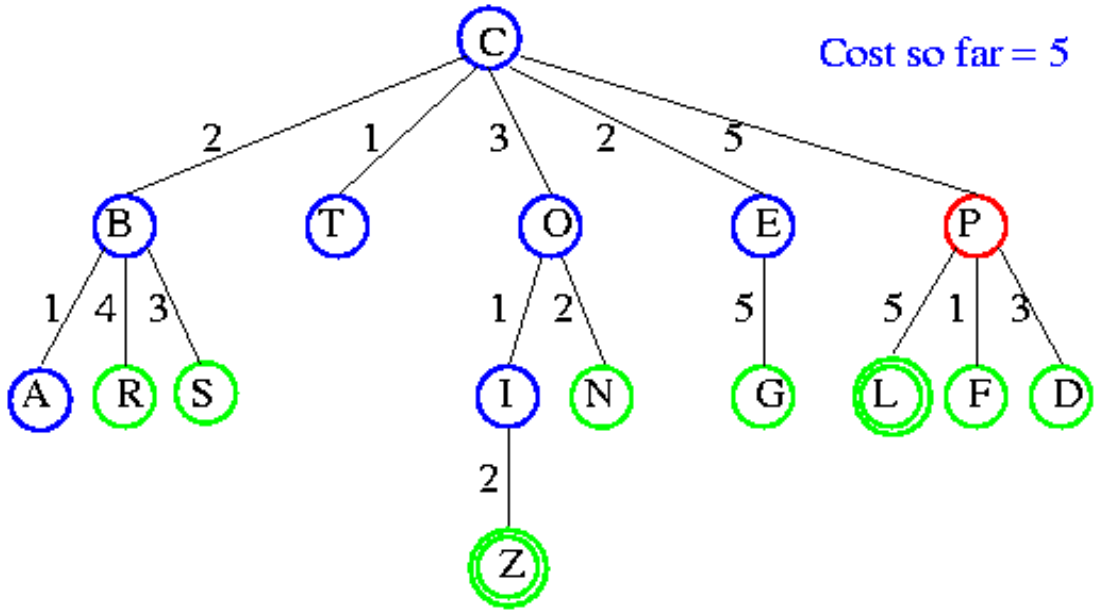
Example

Resolve ties alphabetically and based on depth.



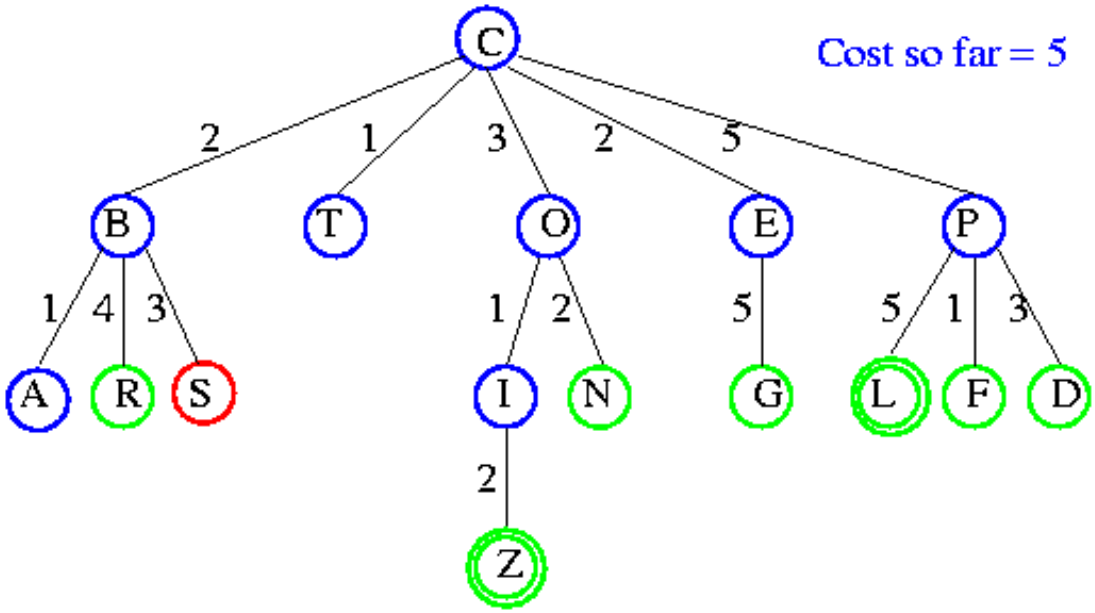
Example

Resolve ties alphabetically and based on depth.



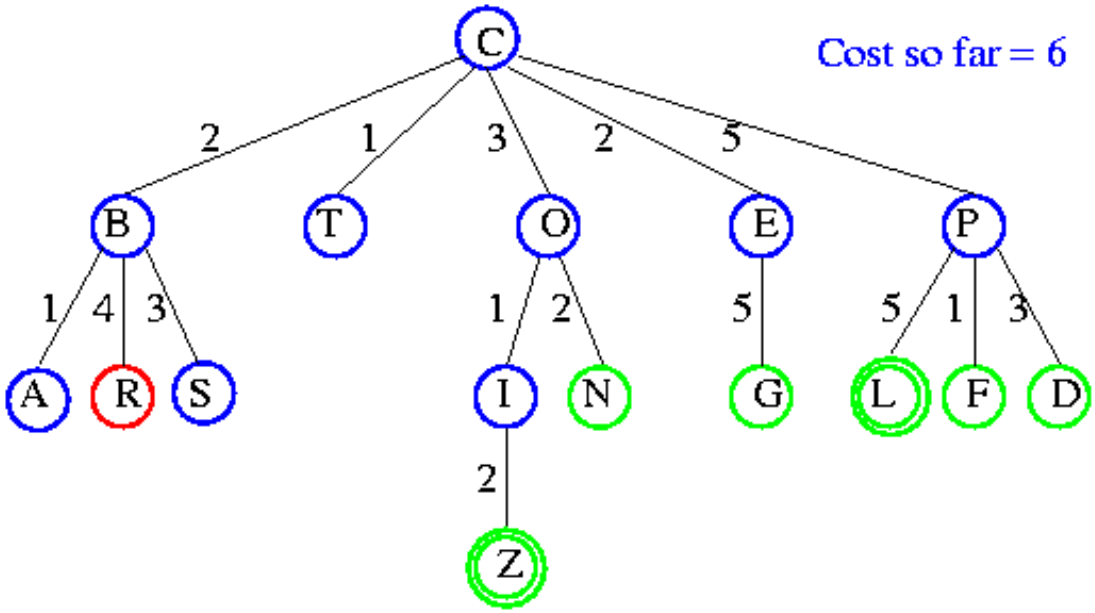
Example

Resolve ties alphabetically and based on depth.



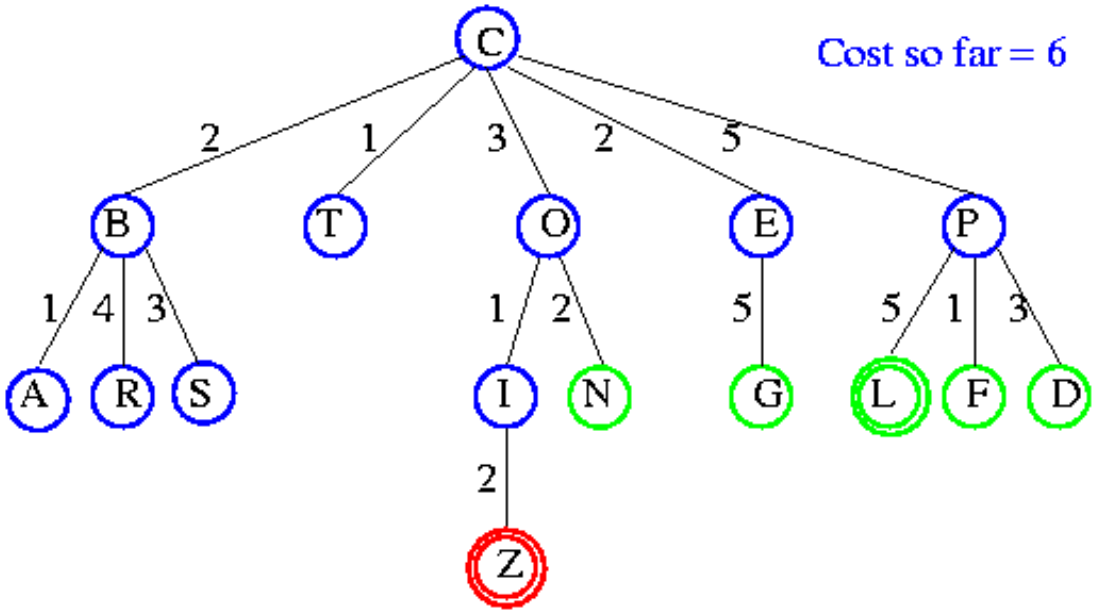
Example

Resolve ties alphabetically and based on depth.



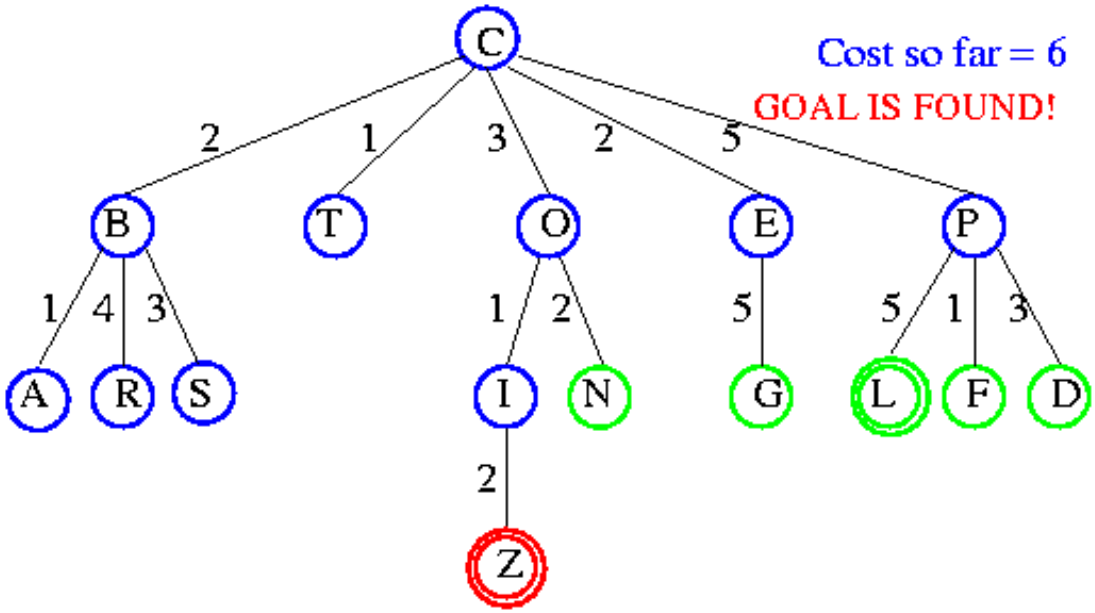
Example

Resolve ties alphabetically and based on depth.



Example

Resolve ties alphabetically and based on depth.



Iterative Deepening Search

DFS with cutoff bound

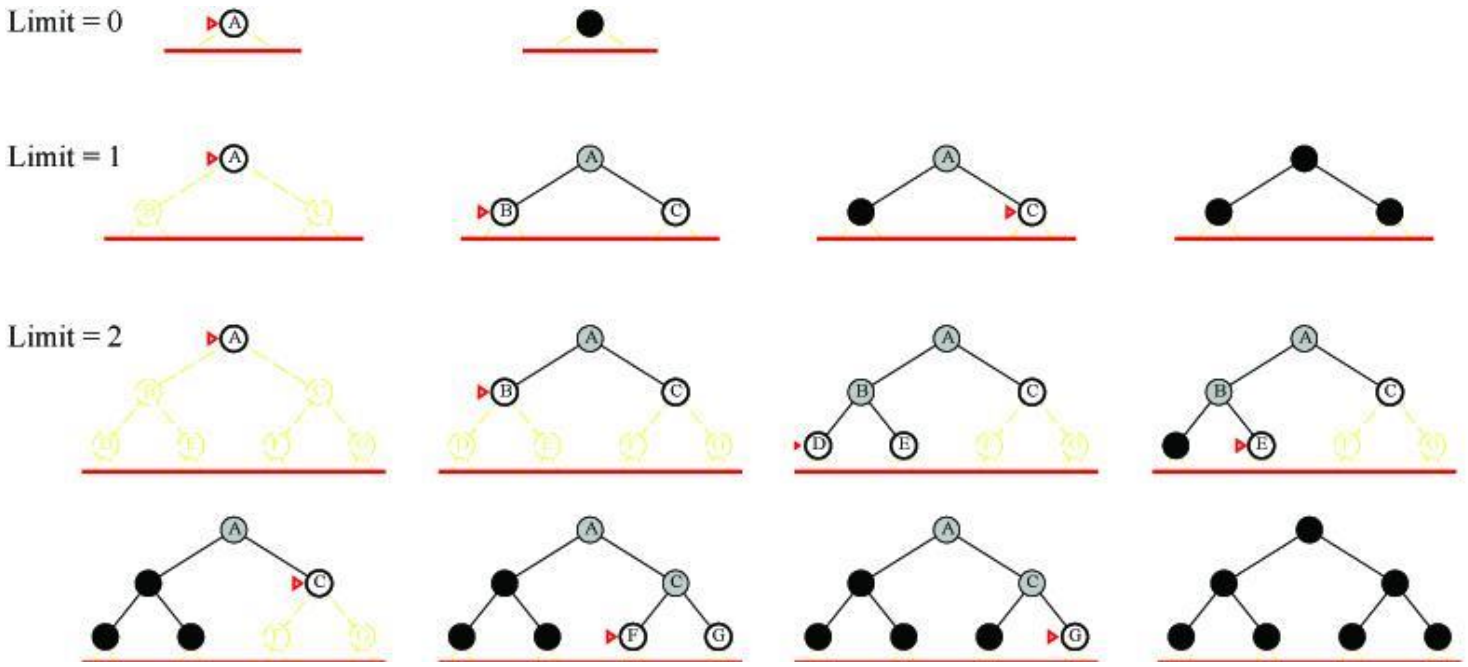
queueing-fn is enqueue-at-front as before, but (expand state) ONLY returns children where depth(children) ≤ threshold

This prevents DFS from going down infinite path

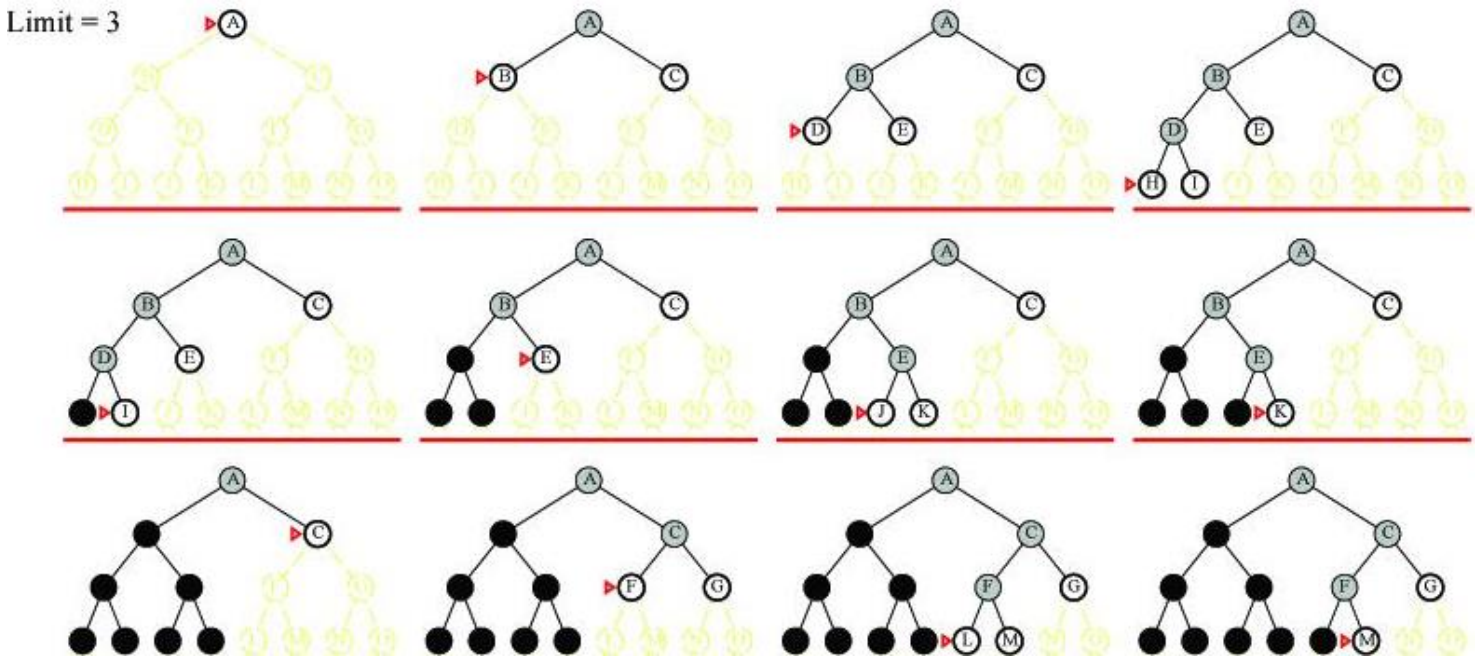
However, we may not find a solution at this threshold!

Try one threshold - if no solution, increase threshold and start again from root

Examples



Examples


Limit = 3
A
A
B
C
A
B
C
D
E
A
B
C
D
E
H
I
A
B
C
D
E
I
A
B
C
E
A
B
C
E
J
K
A
B
C
E
K
A
C
A
C
F
G
A
C
F
G
L
M
A
C
F
G
M

Analysis

Repeat some work, but repeated work is only fraction of work on last (unrepeated) level

$$[1] + [1+b] + [1+b+b^2] + \cdots + [1+b+b^2+\ldots+b^d]$$

Repeated work is approximately $1/b$ of total work (negligible)

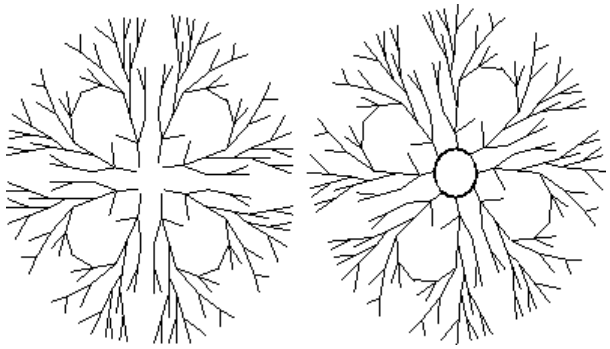
Least-length solution but not least-cost solution

Is there a better way to decide thresholds? Yes!

IDA* search (Like A* search but we avoid the space issues with A* by running it iteratively - more later)

Bidirectional Search

Search forward from initial state to goal AND backward from goal to initial state



Can prune many options

Must consider:

- Which goal state to use (set of goal states)?
- How to determine when two searches overlap
- Which search to use for each direction (DFS probably not good choice)
- Figure shows two breadth-first searches

Run time and space of bidirectional search using a uninformed searches is $O(b^{d/2})$

Exercise 3.8

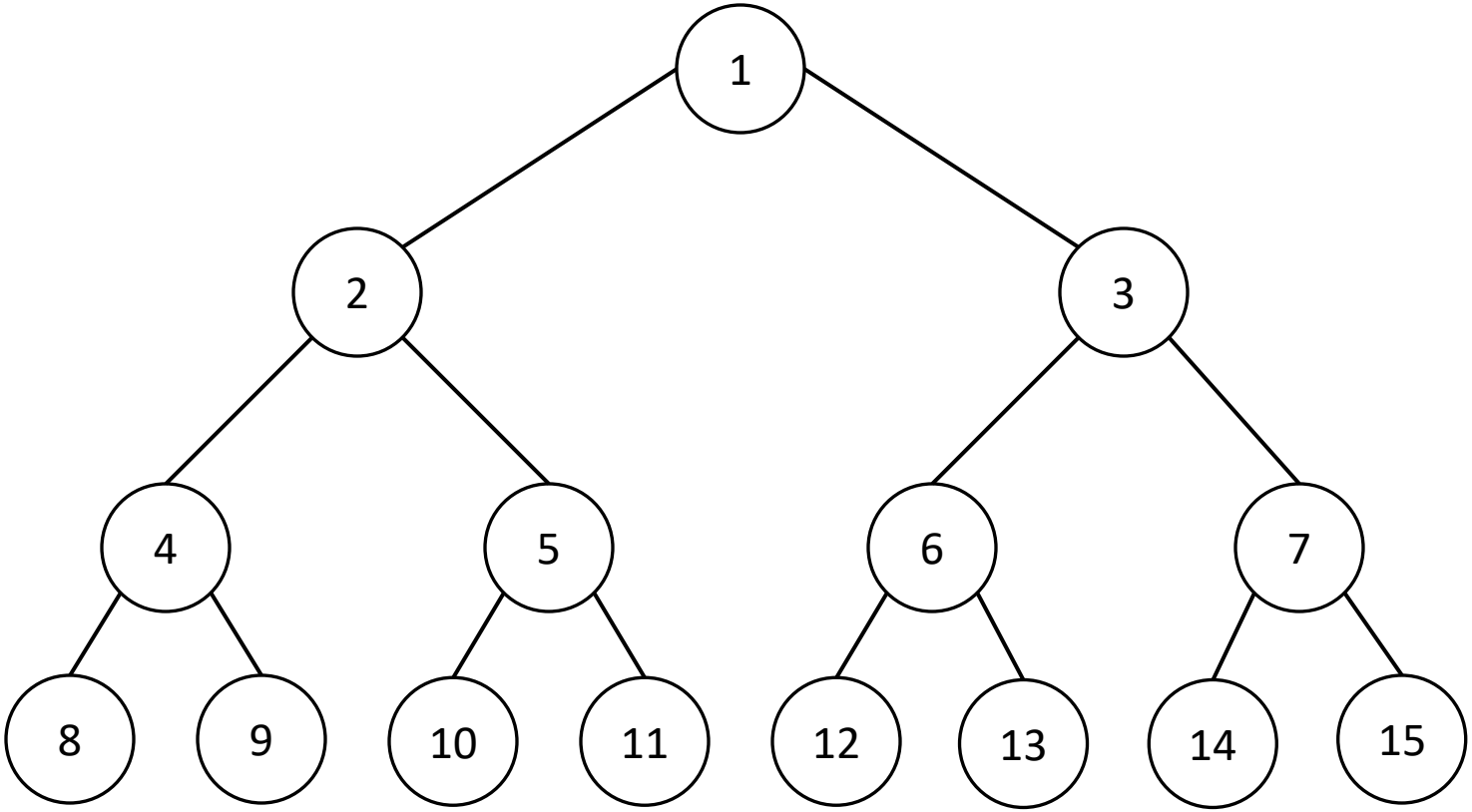
Consider a state space where the start state is number 1 and the successor function for state n returns two states, numbers 2n and 2n+1.

Exercise 3.8

a) Draw the portion of the state space for states 1 to 15.

Exercise 3.8

a) Draw the portion of the state space for states 1 to 15.



Exercise 3.8

b) Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

Exercise 3.8

b) Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

Breadth-first:

1 2 3 4 5 6 7 8 9 10 11

Depth-limited:

1 2 4 8 9 5 10 11

Iterative-deepening:

1, 1 2 3, 1 2 4 5 3 6 7, 1 2 4 8 9 5 10 11

Exercise 3.8

c) Would bidirectional search be appropriate for this problem? If so, describe in detail how it would work.

Exercise 3.8

c) Would bidirectional search be appropriate for this problem? If so, describe in detail how it would work.

Yes, it is appropriate, as it is simple to defined the predecessor of a state.

The predecessor is floor(n/2).

Exercise 3.8

d) What is the branching factor in each direction of the bidirectional search?

Exercise 3.8

d) What is the branching factor in each direction of the bidirectional search?

Forward direction: 2

Backward direction: 1

Exercise 3.8

e) Does the answer to c) suggest a reformulation of the problem that would allow you to solve the problem of getting state 1 to a given goal state with almost no search?

Exercise 3.8

e) Does the answer to c) suggest a reformulation of the problem that would allow you to solve the problem of getting state 1 to a given goal state with almost no search?

Yes, since starting at the goal there is only one possible state to move to. This can be applied recursively to reach the initial state.

Exercise 3.12

a) Prove that uniform-cost search and breadth-first search with constant step costs are optimal when used with the GRAPH-SEARCH algorithm.

b) Show a state space with constant step costs in which GRAPH-SEARCH using iterative deepening finds a suboptimal solution.

Exercise 3.12

Notice that uniform-cost search and breadth-first search are equivalent for constant step costs.

So it suffices to show that either one finds an optimal solution. We will show this for breadth-first search.

Exercise 3.12

Let C be the step cost, S the starting node and V any node reachable from S.

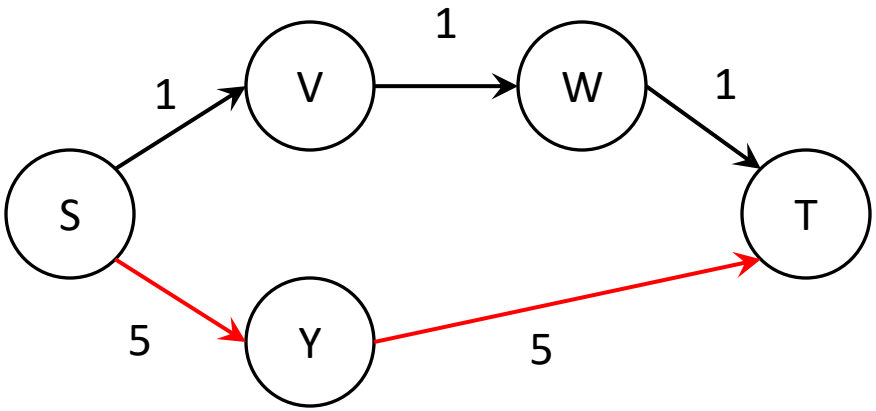
Assume that the first visit to V is after k expansions, with a path cost of $c \cdot k$.

Now assume that there is another path from S to V, with a length of m, where $m \geq k$.

The GRAPH-SEARCH algorithm will discard this path, since V is already visited, without sacrificing optimality, since the path cost will be $c \cdot m \geq c \cdot k$.

Exercise 3.12

Example for suboptimal solution of iterative deepening search:



Exercise 3.17

On page 62, we said that we would not consider problems with negative path costs. In this exercise, we explore this in more depth.

Exercise 3.17

a) Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.

Exercise 3.17

a) Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.

Any path may lead to arbitrarily large rewards, and cannot be discarded. So, the only way to find the best one is to search all possible paths.

Exercise 3.17

b) Does it help if we insist that step costs must be greater or equal to some negative constant c? Consider both trees and graphs.

Exercise 3.17

b) Does it help if we insist that step costs must be greater or equal to some negative constant c? Consider both trees and graphs.

Trees: if the tree is finite and the depth is known it would help, because we can compute an upper bound of the remaining possible improvement.

Graphs (with loops): it doesn’t help, because it is possible go around loops and constantly improve the path cost.

Exercise 3.17

c) Suppose that there is a set of operators that form a loop, so that executing the set in some order results in no net change to the state if all of these operators have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

Exercise 3.17

c) Suppose that there is a set of operators that form a loop, so that executing the set in some order results in no net change to the state if all of these operators have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

The agent should go around this loop forever.

Exercise 3.17

d) One can easily imagine operators with high negative costs, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive round scenic loops indefinitely, and explain how to define the state space of operators for route finding so that artificial agents can also avoid looping.

Exercise 3.17

Each time one revisits a place, the value decreases. Something may be exciting the first time, but will not be as exciting after the 100^{th} visit.

To do this one would have to remember how often a place was visited, and the cost should be a decreasing function of the number of times the place was visited.