Artificial Intelligence

Lesson 1

Ram Meshulam 2004

About

- Lecturer: Prof. Sarit Kraus
- TA: Galit Haim: haimga@cs.biu.ac.il
- (almost) All you need can be found on the course website:
  - **http://u.cs.biu.ac.il/~haimga/Teaching/AI/**

Course Requirements 1

- The grade is comprised of 70% exam and 30% exercises.
- 3 programming exercises will be given. Work individually.
- All the exercises are counted for the final grade.
- 10% each.
- Exercises will be written in C++ or JAVA only. It should be compiled and run on planet machine, and will be submitted via “submit”. Be precise!

Course Requirements 2

- Exercises are not hard, but work is required. Plan your time ahead!
- When sending me mail **please include the course number (89-570) in the header**, to pass the automatic spam filter.
- You (probably) will be required to participate in AI experiments.
- See other general rules in:
  http://u.cs.biu.ac.il/~haimga/Teaching/AI/assignments/general-rules.pdf

Course Schedule

- Lesson 1:
  - Introduction
  - Transferring a general problem to a graph search problem.
- Lesson 2
  - Uninformed Search (BFS, DFS etc.).
- Lesson 3
  - Informed Search (A*,Best-First-Search etc.).

Course Schedule – Cont.

- Lesson 4
  - Local Search (Hill Climbing, Genetic algorithms etc.).
- Lesson 5
  - "Search algorithms" chapter summary.
- Lesson 6-7
  - Game-Trees: Min-Max & Alpha-Beta algorithms.

Course Schedule – Cont.

- Lesson 8-9
  - Planning: STRIPS algorithm
- Lesson 10-11-12
  - Learning: Decision-Trees, Neural Network, Naïve Bayes, Bayesian Networks and more.
- Lesson 13: Robotics
- Lesson 14
  - Questions and exercise.

AI – Alternative Definitions

- **Elaine Rich and Kevin Knight:** AI is the study of how to make computers do things at which, at the moment, people are better.
- **Stuart Russell and Peter Norvig:** [AI] has to do with smart programs, so let's get on and write some.
- **Claudson Bornstein:** AI is the science of common sense.
- **Douglas Baker:** AI is the attempt to make computers do what people think computers cannot do.
- **Astro Teller:** AI is the attempt to make computers do what they do in the movies.

AI Domains

- Games – chess, checkers, tile puzzle.
- Expert systems
- Speech recognition and Natural language processing, Computer vision, Robotics.

AI & Search

- "The two most fundamental concerns of AI researchers are *knowledge representation* and *search*"
- "*knowledge representation* … addresses the problem of capturing in a language…suitable for computer manipulation"
- "*Search* is a problem-solving technique that systematically explores a space of problem states". Luger, G.F. Artificial Intelligence: Structures and Strategies for Complex Problem Solving

Solving Problems with Search Algorithms

- Input: a problem *P*.
- Preprocessing:
  - Define *states* and a *state space*
  - Define *Operators*
  - Define a *start* state and *goal* set of states.
- Processing:
  - Activate a Search algorithm to find a *path* from start to one of the goal states.

Example - Missionaries & Cannibals

- State space – [M,C,B]
- Initial State – [3,3,1]
- Goal State – [0,0,0]
- Operators – adding or subtracting the vectors [1,0,1], [2,0,1], [0,1,1], [0,2,1] or [1,1,1]
- Path – moves from [3,3,1] to [0,0,0]
- Path Cost – river trips
- http://www.plastelina.net/game2.html
- http://www.youtube.com/watch?v=W9NEWxabGmg

Breadth-First-Search Pseudo code

- Intuition: Treating the graph as a tree and scanning top-down.
- Algorithm:

BFS(Graph graph, Node start, Vector Goals)
1. L→ make_queue(start)
2. While L not empty loop
    1. n ← L.remove_front()
    2. If goal (n) return true
    3. S ← successors (n)
    4. L.insert(S)
3. Return false

Breadth-First-Search Attributes

- Completeness – yes $(b < \infty, d < \infty)$
- Optimality – yes, if graph is un-weighted.
- Time Complexity: $\boxed{O(b^{d+1})}$
- Memory Complexity: $\boxed{O(b^{d+1})}$
  - Where *b* is branching factor and *d* is the solution depth
- See water tanks example:
- http://u.cs.biu.ac.il/~haimga/Teaching/AI/lessons/lesson1&2b-%20WaterTank-BFS_DFS.pdf

Artificial Intelligence

Lesson 2

Uninformed Search

- Uninformed search methods use only information available in the problem definition.
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Iterative DFS (IDA)
  - Bi-directional search
  - Uniform Cost Search (a.k.a. Dijkstra alg.)

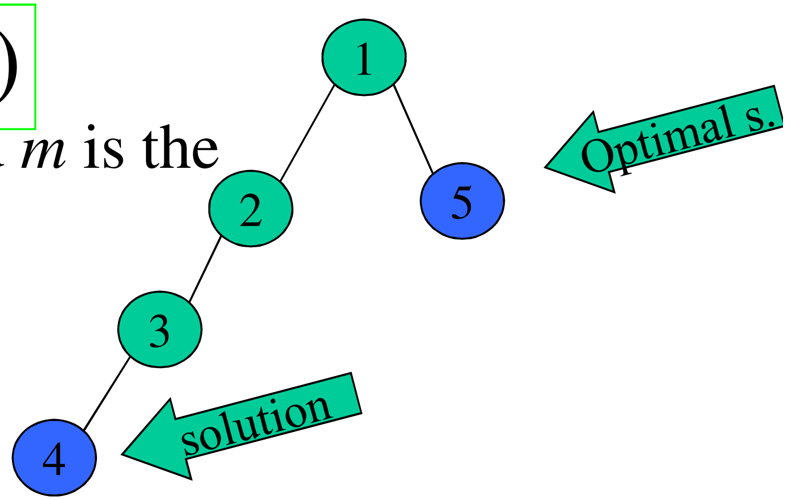
Depth-First-Search Pseudo code

DFS(Graph graph, Node start, Vector Goals)

1. L← make_stack(start)
2. While L not empty loop
    - 2.1 n ← L.remove_front()
    - 2.2 If goal (n) return true
    - 2.3 S ← successors (n)
    - 2.4 L.insert(S)
3. Return false

Depth-First-Search Attributes

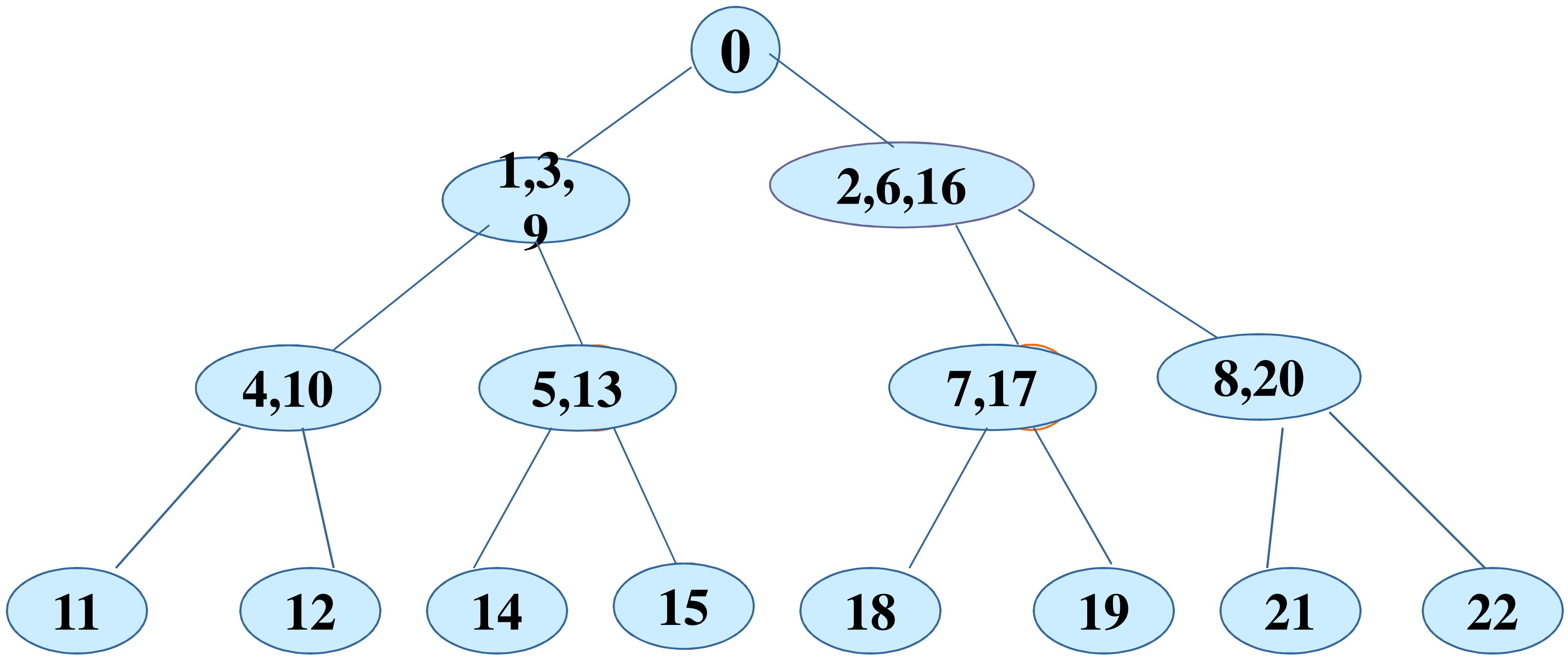
- Completeness – No. Infinite loops or Infinite depth can occur.
- Optimality – No.
- Time Complexity: $\boxed{O(b^m)}$
- Memory Complexity: $\boxed{O(bm)}$
  - Where *b* is branching factor and *m* is the maximum depth of search tree
- See water tanks example



Limited DFS Attributes

- Completeness – Yes, if d≤l
- Optimality – No.
- Time Complexity: $\boxed{O(b^l)}$
  - If d<l, it is larger than in BFS
- Memory Complexity: $\boxed{O(bl)}$
  - Where *b* is branching factor and *l* is the depth limit.

Depth-First Iterative-Deepening



The numbers represent the order generated by DFID

Iterative-Deepening Attributes

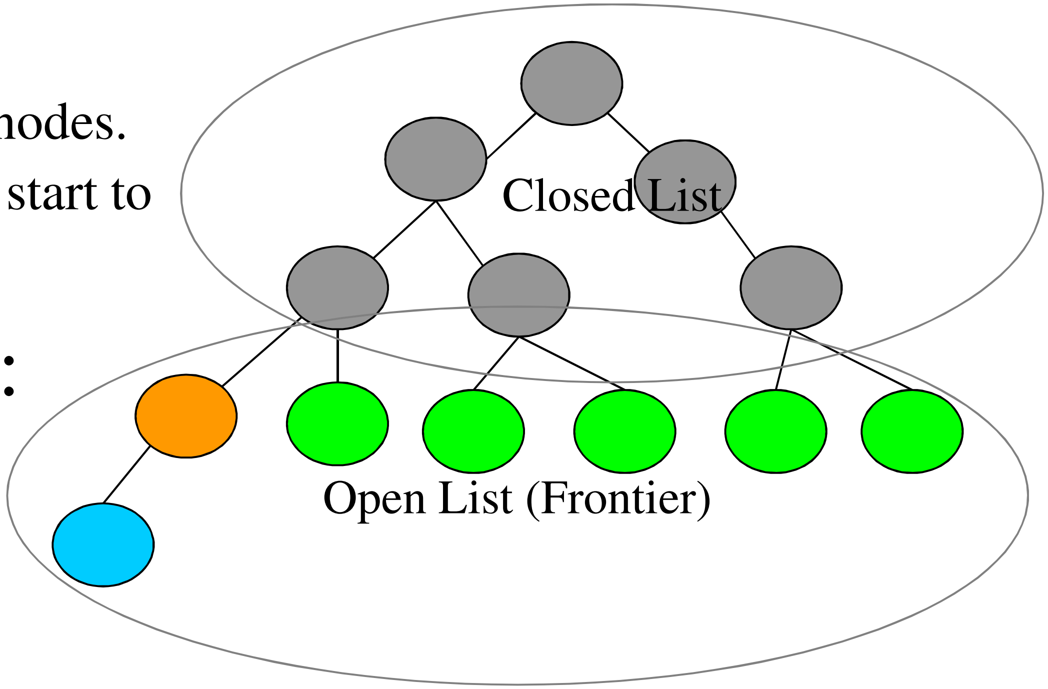
- Completeness – Yes
- Optimality – yes, if graph is un-weighted.
- Time Complexity:

$$\boxed{O((d)b + (d-1)b^2 + \ldots + (1)b^d) = O(b^d)}$$

- Memory Complexity: $\boxed{O(db)}$
  - Where *b* is branching factor and *d* is the maximum depth of search tree

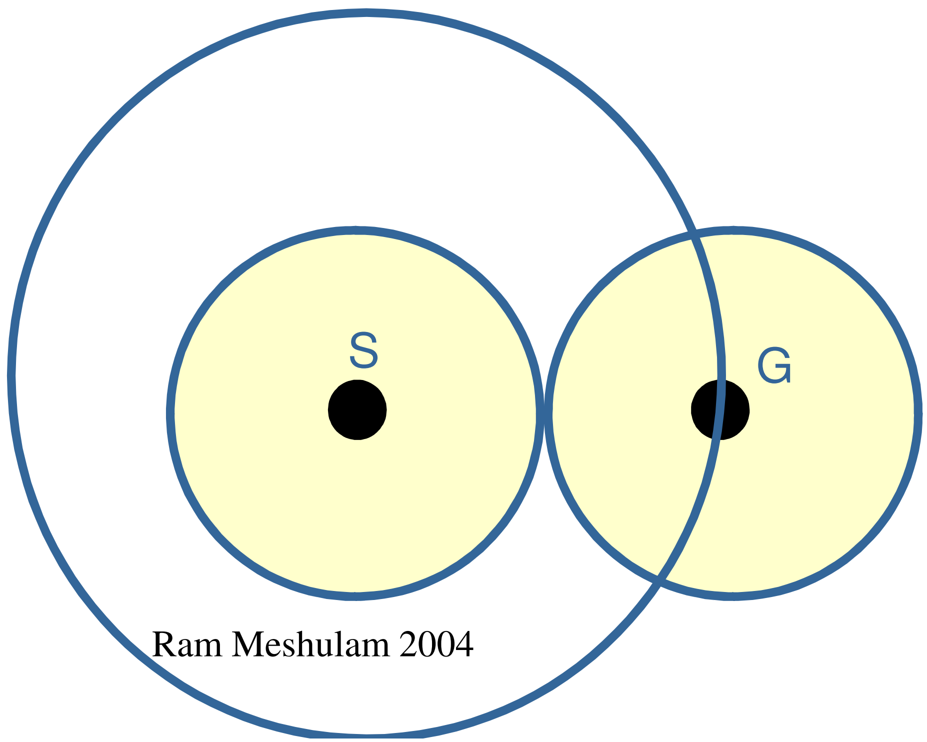
State Redundancies

- Closed list - a hash table which holds the visited nodes.
  - Prevent re-exploring of nodes.
  - Hold solution path from start to goal
- For example BFS:



Bi-directional Search

- Search both from initial state to goal state.
- Operators must be symmetric.



Bi-directional Search Attributes

- Completeness – Yes, if both directions use BFS
- Optimality – yes, if graph is un-weighted and both directions use BFS.
- Time and memory Complexity: $\boxed{O(b^{d/2})}$
- Pros.
  - Cuts the search tree by half (at least theoretically).
- Cons.
  - Frontiers must be constantly compared.

Minimum cost path

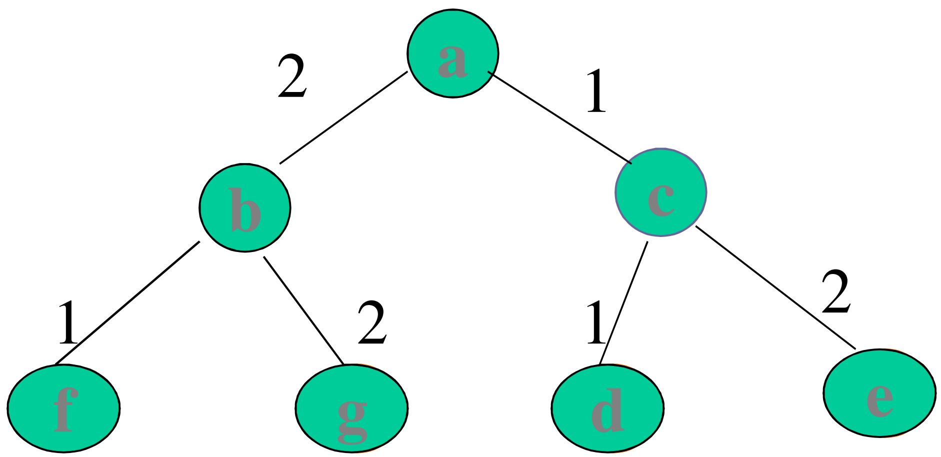
- General minimum cost path-search problem:
  - Find shortest path form start state to one of the goal states in a weighted graph.
  - Path cost function is *g(n)*: sum of weights from start state to goal.

Uniform Cost Search

- Also known as Dijkstra's algorithm.
- Expand the node with the minimum path cost first.
- Implementation: priority queue.

Example of Uniform Cost Search

- Assume an example tree with different edge costs, represented by numbers next to the edges.

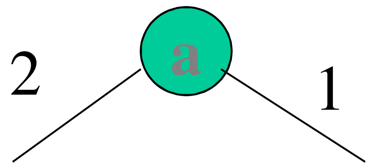


Notations for this example:

generated node

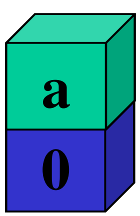
expanded node

Example of Uniform Cost Search

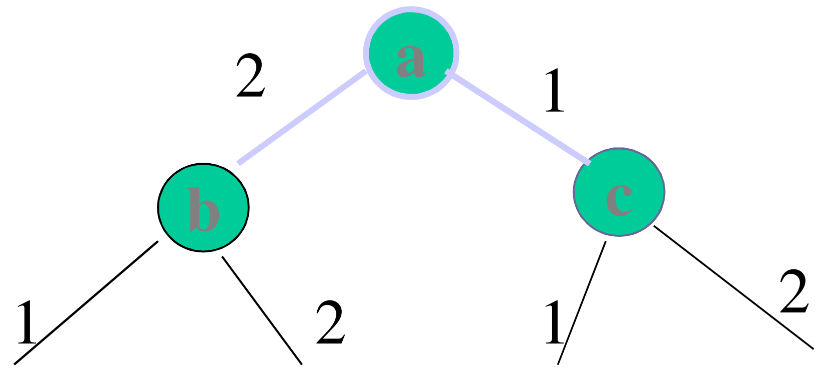


Closed list:

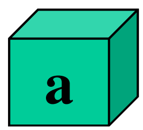
Open list:



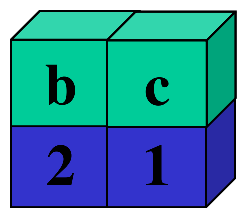
Example of Uniform Cost Search



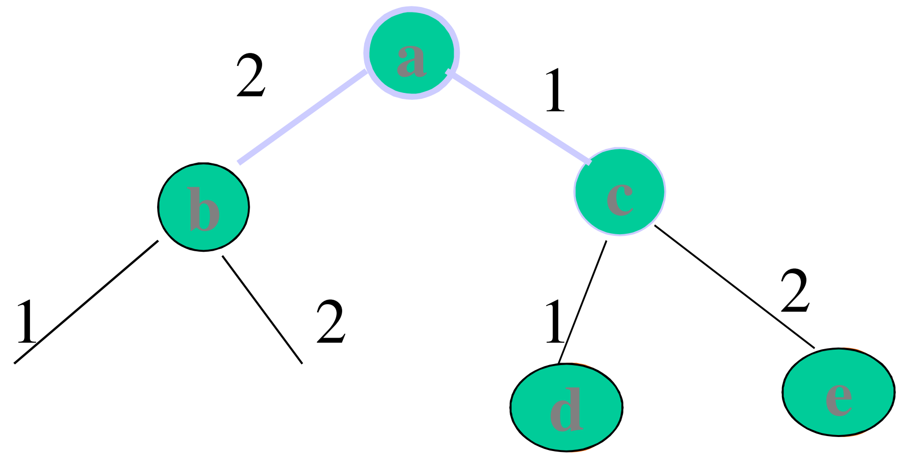
Closed list:

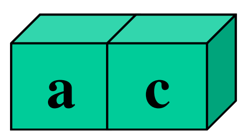


Open list:

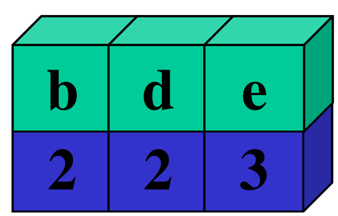


Example of Uniform Cost Search



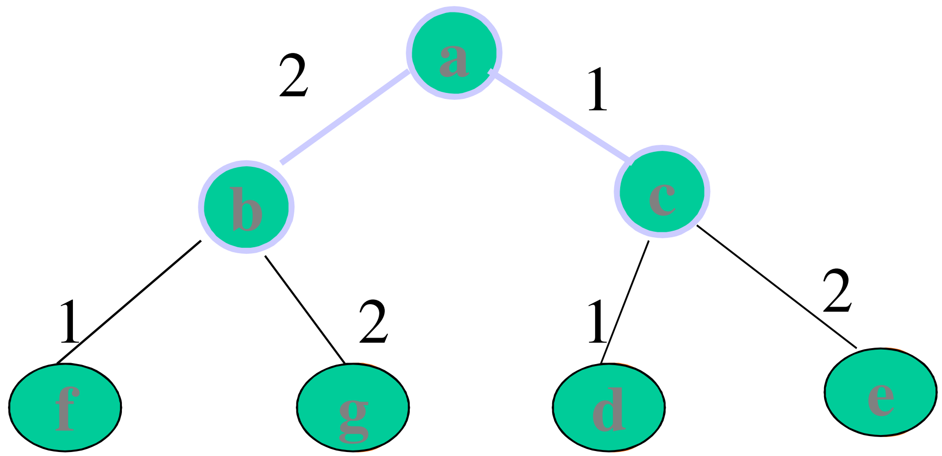


Closed list:

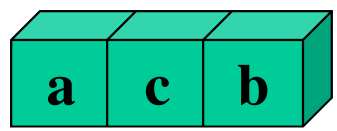


Open list:

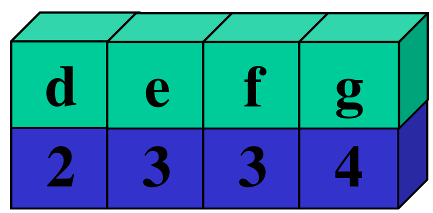
Example of Uniform Cost Search



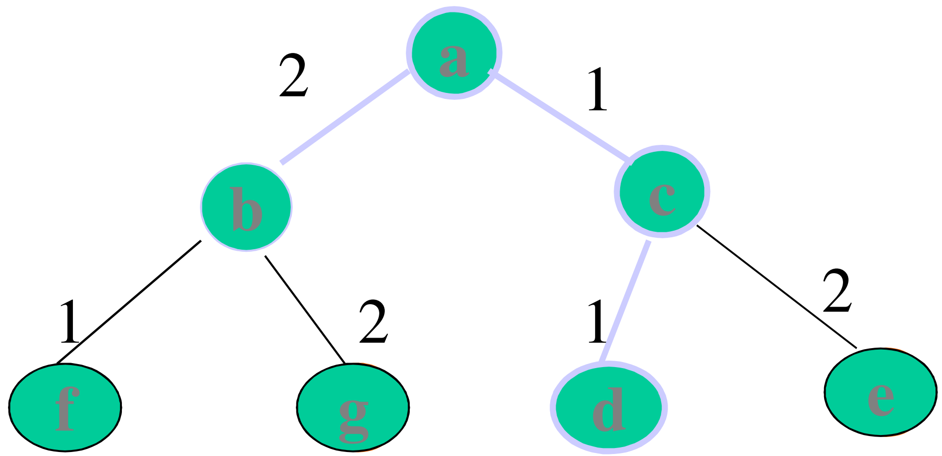
Closed list:

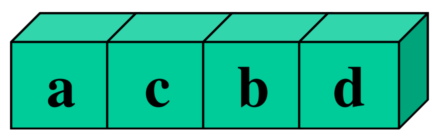


Open list:

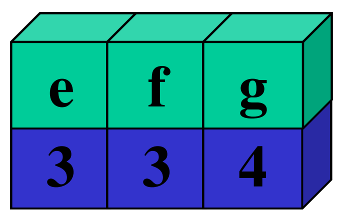


Example of Uniform Cost Search



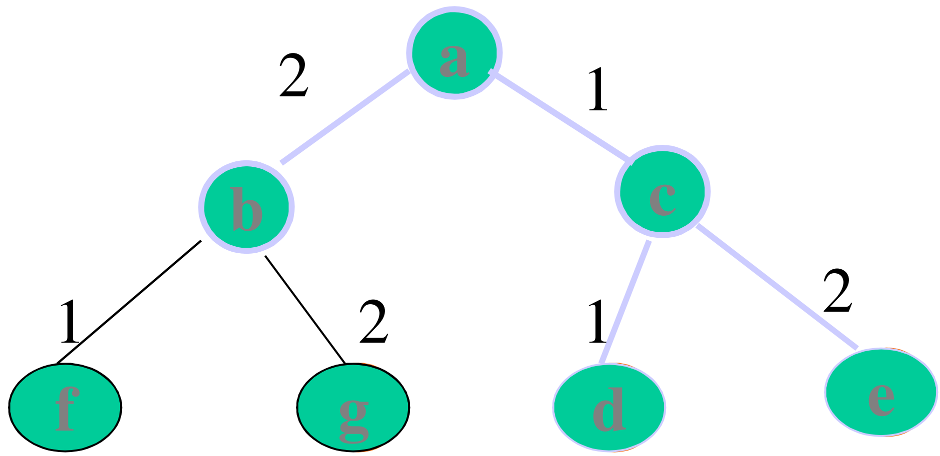


Closed list:

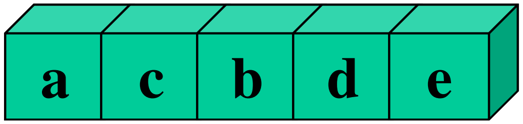


Open list:

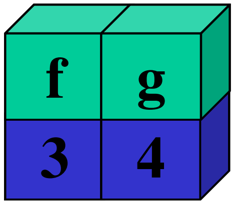
Example of Uniform Cost Search



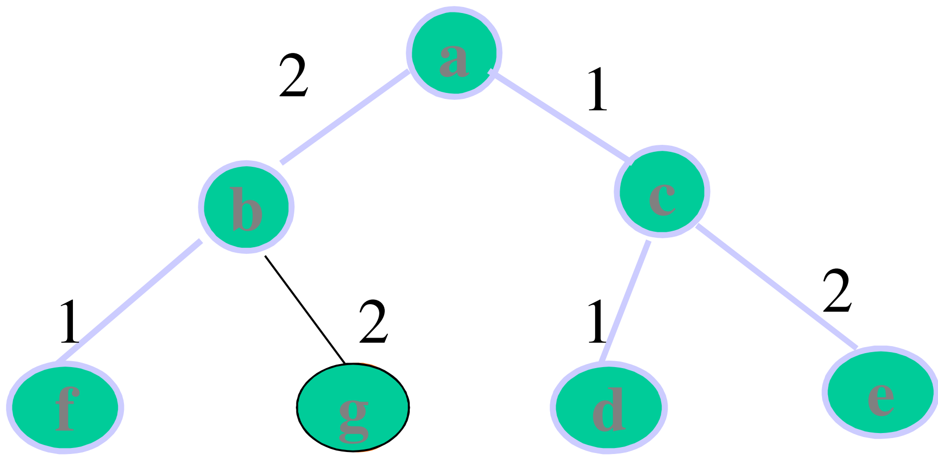
Closed list:



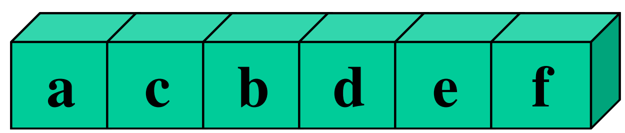
Open list:



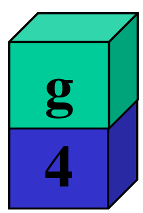
Example of Uniform Cost Search



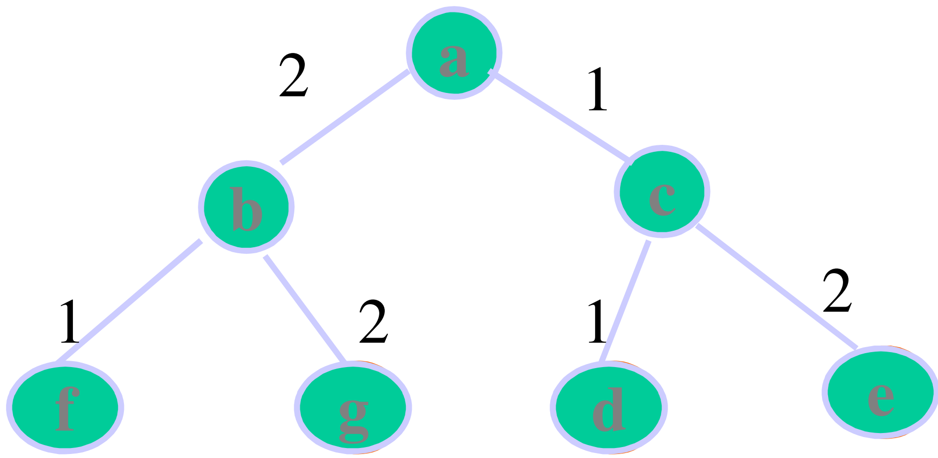
Closed list:

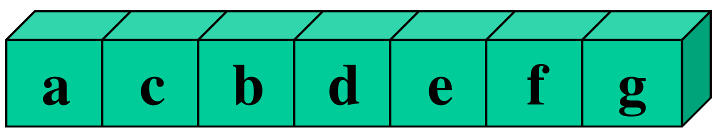


Open list:



Example of Uniform Cost Search





Closed list:

Open list:

Uniform Cost Search Attributes

- Completeness: yes, for positive weights
- Optimality: yes
- Time & Memory complexity: $\boxed{O(b^{\lfloor c/e \rfloor})}$
  - Where *b* is branching factor, *c* is the optimal solution cost and *e* is the minimum edge cost (each operator cost at least *e*).
  - UCS uses cost path and not length path, therefore, the '*d*' term is not used here.

Informed Search

- Incorporate additional measure of a potential of a specific state to reach the goal.
- A potential of a state to reach a goal is measured through a heuristic function $h(n)$.
- An evaluation function is denoted $f(n)$.

Best First Search Algorithms

- Principle: Expand node *n* with the best evaluation function value *f(n)*.
- Implement via a priority queue
- Algorithms differ with definition of *f* :
  - Greedy Search: $f(n) = h(n)$
  - A*: $f(n) = g(n) + h(n)$
  - IDA*: iterative deepening version of A*
  - Etc'

Exercise

- Q: Does a Uniform-Cost search be considered as a Best-First algorithm?
- A: Yes. It can be considered as a Best-First algorithm with evaluation function $f(n)=g(n)$.
- Q: In what scenarios IDS outperforms DFS?, BFS?
- A:
  - IDS outperforms DFS when the search tree is a lot deeper than the solution depth.
  - IDS outperforms BFS when BFS run out of memory.

Exercise – Cont.

- Q: Why do we need a closed list?
- A: Generally a closed list has two main functionalities:
  - Prevent re-exploring of nodes.
  - Hold solution path from start to goal (DFS based algorithms have it anyway).
- Q: Does Breadth-FS find optimal path length in general?
- A: No, unless the search graph is un-weighted.
- Q: Will IDS always find the same solution as BFS given that the nodes expansion order is deterministic?
- A: Yes. Each iteration of IDS explores new nodes the same order a BFS does.