COE206 – Principles of Artificial Intelligence

Mustafa MISIR

Istinye University, Department of Computer Engineering

mustafa.misir@istinye.edu.tr

http://mustafamisir.github.io
http://memoryrlab.github.io

ARTIFICIAL INTELLIGENCE IN MEDICINE RESEARCH CENTER
ISTINYE UNIVERSITY

MEMORY

L3-1: Problem Solving by Search

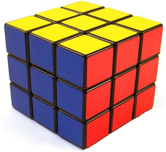
Uninformed (Blind) Search

Uninformed (Blind) Search[1]

The strategies have no additional information about states beyond that provided in the problem definition.

- All they can do is generate successors and distinguish a **goal state** from a **non-goal state**.


Start state


Goal state

Search strategies differ based on their node expansion schemes.

- Strategies that know whether one non-goal state is *more promising* than another are called **informed** or **heuristic** search strategies.

[1] image source: http://www.ccs.neu.edu/home/camato/5100-Fall16.html

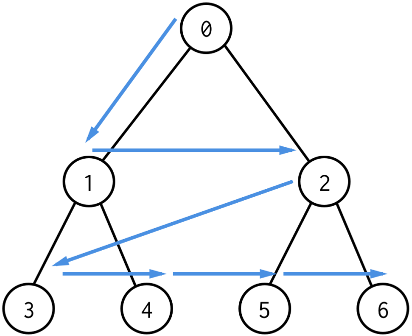
Outline

- Breadth-first Search (BFS)
- Uniform-cost Search (UCS)
- Depth-first Search (DFS)
- Iterative Deepening DFS (IDDFS)
- Bidirectional Search (BS)

Breadth-first Search (BFS)[2]

The root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.

- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.



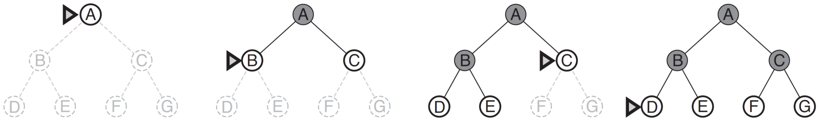
2 image source: http://mishadoff.com/blog/dfs-on-binary-tree-array/

BFS

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

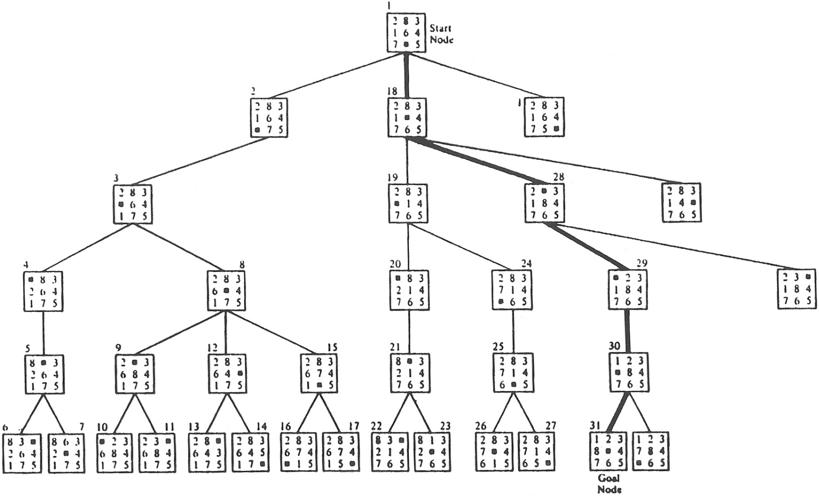
BFS

The memory requirements are a bigger problem than the execution time.



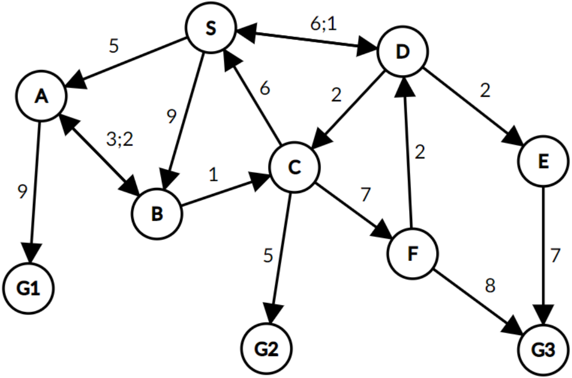
At each stage, the node to be expanded next is indicated by a marker.

BFS, e.g. 8-Puzzle[3]



[3] https://www.cs.drexel.edu/~jpopyack/Courses/AI/Sp15/notes/8-puzzle_comparison.html

BFS, e.g. Graph Search[4]



[4] BFS example by John Levine (U. Strathclyde): https://www.youtube.com/watch?v=1wu2sojwsyQ

BFS

Assume that 1 million nodes can be generated per second and that a node requires 1000 bytes of storage.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

BFS – Properties

For b as the branching factor[5] which is the number of children (successors / outgoing nodes) at each node and d as the depth (level) of the tree[6]

- **Completeness**[7]: Yes
- **Time Complexity**[8]: $\sum_{i=0}^{d} b^0 + b^1 + b^2 + \ldots + b^d = (1 - b^{d+1})/(1 - b) = O(b^{d+1})$
- **Space Complexity**[9]: $O(b^{d+1})$ as each node is kept in the memory
- **Optimality**[10]: Yes (if the cost per step is the same / uniform) – No (otherwise)[11]

---

5 https://en.wikipedia.org/wiki/Branching_factor

6 the root node is at level 0 – https://en.wikipedia.org/wiki/Tree-depth

7 Is the algorithm guaranteed to find a solution when there is one?

8 How long does it take to find a solution?

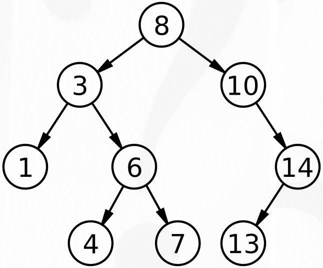
9 How much memory is needed to perform the search?

10 Does the strategy find the optimal solution?

11 **BFS** finds the shortest path in terms of number of actions, not the least-cost path = **Cost-sensitive search** – BFS example by John Levine (U. Strathclyde): https://www.youtube.com/watch?v=n3fPL9q_Nyc

Uninformed Search – Breadth-first Search

TASK: Implement Breadth-first Search (BFS) in order to explore a specific number-labeled node in a given tree



Submit your code to **Piazza** as a *private message*.

Uniform-cost Search (UCS)

Expand the least-cost (lowest path cost, $g(n)$) unexpanded node

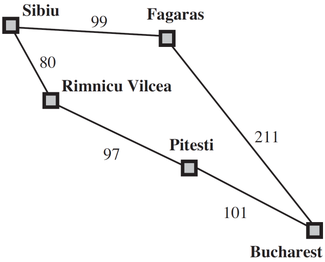
- This is done by storing the frontier as a priority queue (cumulative cost) ordered by g.

When all step costs are equal, **BFS** is optimal because it always expands the shallowest unexpanded node, except that **BFS** stops as soon as it generates a goal, whereas **UCS** examines all the nodes at the goal's depth to see if one has a lower cost.

UCS

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

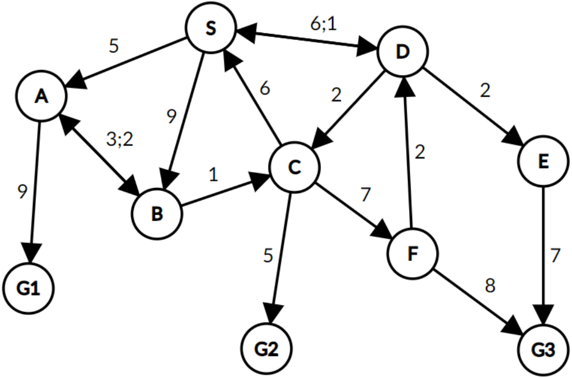
UCS, e.g.



Getting from Sibiu to Bucharest

- The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost $80 + 97 = 177$.
- The least-cost node is now Fagaras, so expanded, adding Bucharest with cost $99 + 211 = 310$.
- Choose Pitesti for expansion and adding a second path to Bucharest with cost $80 + 97 + 101 = 278$, which is the returned solution.

UCS, e.g. Graph Search[12]



12 UCS example by John Levine (U. Strathclyde): `https://www.youtube.com/watch?v=dRMvK76xQJI`

UCS – Properties

For C^* is the cost of the optimal solution and the minimum cost per action is $\in$

- **Completeness**[13]: Yes
- **Time Complexity**[14]: $O(b^{1+\lfloor C^*/\in \rfloor})$ (when all the step costs are the same, $b^{1+\lfloor C^*/\in \rfloor} = b^{d+1}$)
- **Space Complexity**[15]: $O(b^{1+\lfloor C^*/\in \rfloor})$
- **Optimality**[16]: Yes

13 Is the algorithm guaranteed to find a solution when there is one?

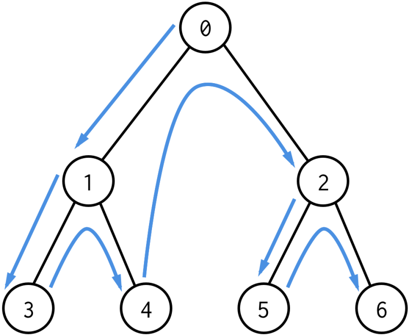
14 How long does it take to find a solution?

15 How much memory is needed to perform the search?

16 Does the strategy find the optimal solution?

Depth-first Search (DFS)[17]

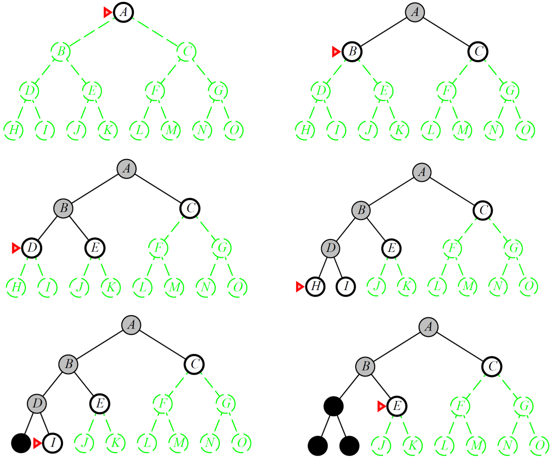
Expands the deepest node in the current frontier of the search tree.



[17] image source: http://mishadoff.com/blog/dfs-on-binary-tree-array/

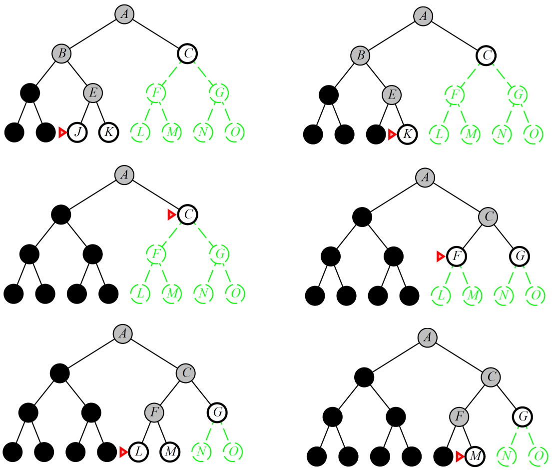
DFS, e.g.

When A is the starting node while M is the goal node

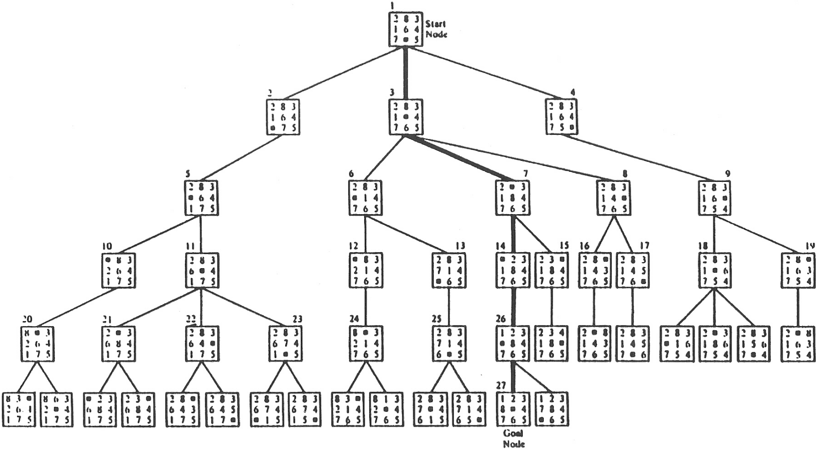


DFS, e.g.

When A is the starting node while M is the goal node

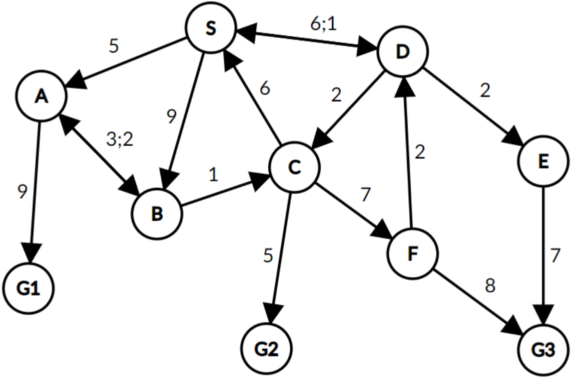


DFS, e.g. 8-Puzzle[18]



[18] https://www.cs.drexel.edu/~jpopyack/Courses/AI/Sp15/notes/8-puzzle_comparison.html

DFS, e.g. Graph Search[19]



[19] DFS example by John Levine (U. Strathclyde): https://www.youtube.com/watch?v=h1RYvCfuoN4

DFS – Properties

For m is the maximum tree depth

- **Completeness**[20]: No, fails in infinite-depth spaces, spaces with loops
- **Time Complexity**[21]: $O(b^m)$
- **Space Complexity**[22]: $O(bm)$ (Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored.)
- **Optimality**[23]: No

[20] Is the algorithm guaranteed to find a solution when there is one?

[21] How long does it take to find a solution?

[22] How much memory is needed to perform the search?

[23] Does the strategy find the optimal solution?

DFS – Depth-limited Search (DLS)

A failure of DFS in infinite state spaces can be alleviated by supplying DFS with a predetermined depth limit l.

- That is, nodes at depth l are treated as if they have no successors.

DFS – DLS (Recursive)

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

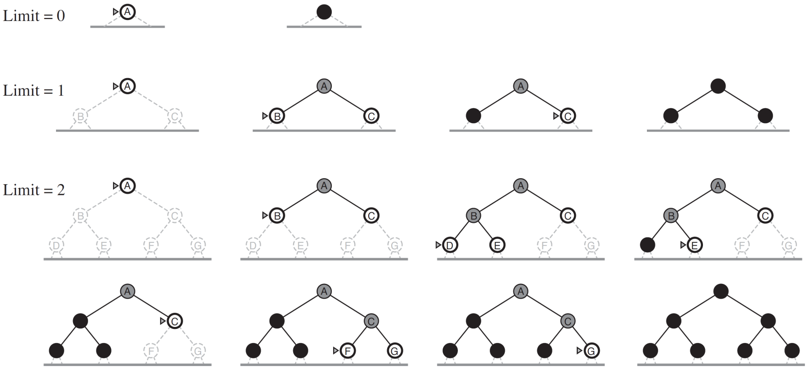
Iterative Deepening DFS (IDDFS / IDS)

A search strategy aiming at finding the best depth limit.

- It does this by gradually increasing the limit — first 0, then 1, then 2, and so on — until a goal is found.

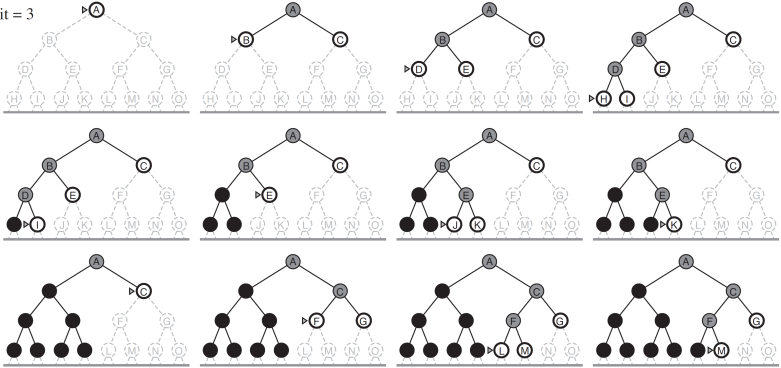
function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for $depth = 0$ **to** ∞ **do**
    $result \leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if $result \neq$ cutoff **then return** *result*

IDDFS, e.g.

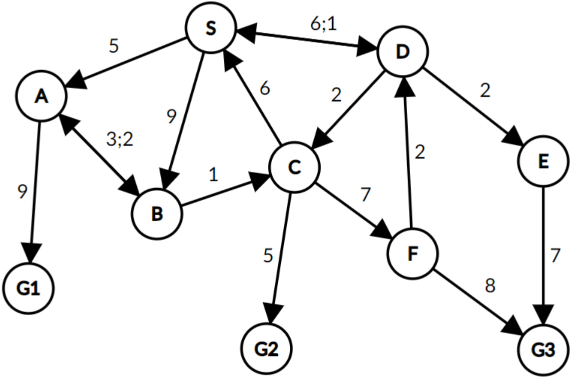


IDDFS, e.g.

Limit = 3



IDDFS, e.g. Graph Search[24]



[24] IDDFS example by John Levine (U. Strathclyde): https://www.youtube.com/watch?v=Y85ECk_H3h4

IDDFS – Properties

Combines the benefits of **DFS** (space advantage) and **BFS** (time / shallow-solution advantage)

- **Completeness**[25]: Yes
- **Time Complexity**[26]: $O(b^d)$
- **Space Complexity**[27]: $O(bd)$
- **Optimality**[28]: Yes

IDDFS vs. BFS for the total number of nodes to be generated – when $b = 10$, $d = 5$ and the goal node is located at the far right leaf

- N(BFS) $=$ 10 $+$ 100 $+$ 1,000 $+$ 10,000 $+$ 100,000 $=$ 111,110 nodes
- N(IDDFS) $=$ 50 $+$ 400 $+$ 3,000 $+$ 20,000 $+$ 100,000 $=$ 123,450 nodes

For IDDFS, there is some extra cost for generating the upper levels multiple times, but it tends to be negligible

---

25 Is the algorithm guaranteed to find a solution when there is one?

26 How long does it take to find a solution?

27 How much memory is needed to perform the search?

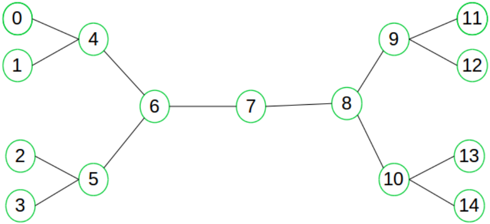
28 Does the strategy find the optimal solution?

Bidirectional Search (BS)[29]

Run two simultaneous searches

- one forward from the initial state
- the other backward from the goal

hoping that the two searches meet in the middle



The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d

Yet, requires a method for computing predecessors.

[29] image source: https://www.geeksforgeeks.org/bidirectional-search/

BS – Properties

Assuming that **BS** is using **BFS** for both searches

- **Completeness**[30]: Yes
- **Time Complexity**[31]: $O(b^{d/2})$
- **Space Complexity**[32]: $O(b^{d/2})$
- **Optimality**[33]: Yes

30 Is the algorithm guaranteed to find a solution when there is one?

31 How long does it take to find a solution?

32 How much memory is needed to perform the search?

33 Does the strategy find the optimal solution?

Summary – Comparison on Tree Search

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}

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 BFS.

Uninformed Search

TASK: For a densely connected, non-simple graph with 10 nodes, that you determined by yourself

- apply all the shown Uninformed Search algorithms while illustrating the search trees step-by-step
- compare the algorithms, explaining their advantages and disadvantages on your particular graph

Submit your report to **Piazza** as a *private message* by April 8, 23:59.