Uninformed Search

Artificial Intelligence

September 9, 2019

Plan

09/09/2019	Search	Uninformed Search	Search (Korf)	
09/11/2019	Search	Heuristic Search	Search (Korf)	
09/16/2019	Search	Adversarial Search	Search (Korf)	
09/18/2019	Search	Local Search	Local Search (Russell and Norvig)	
09/23/2019	**Search**	**Quiz**	*Search Worksheet*	Quiz 1 (Search)
09/25/2019	Multiagent Systems	Behavior-based AI		Discussion Question
09/30/2019	Multiagent Systems	Swarm Intelligence	Python Tutorial	**Assignment 1 (Search)**

Topic 2: Search

- **Uninformed search**
- Heuristic search
- Adversarial search
- Local search

This reading is at a higher conceptual level, so consult slides and online resources

We will cover the highlighted sections.
You do not need to read the rest.

22

Artificial Intelligence Search Algorithms

Agenda

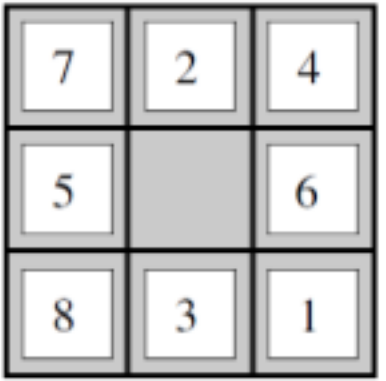
- Introduction to search & problem space model
- Basic brute force search strategies

- In-class activity

- Analysis of search algorithms
- Improving brute-force search algorithms

Introduction to Search & Problem Space Model

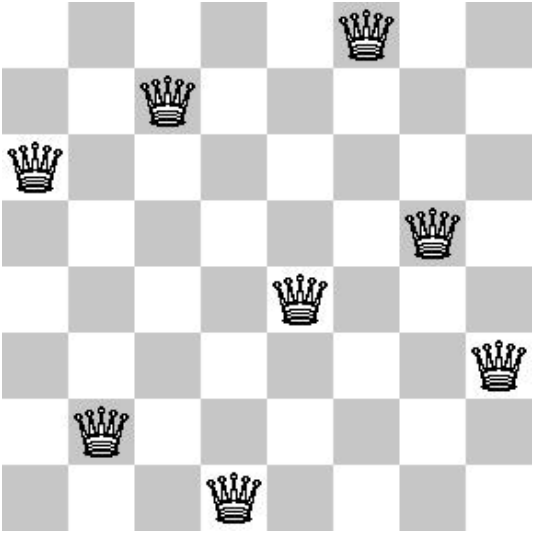
Single agent path finding: Eight Puzzle Problem



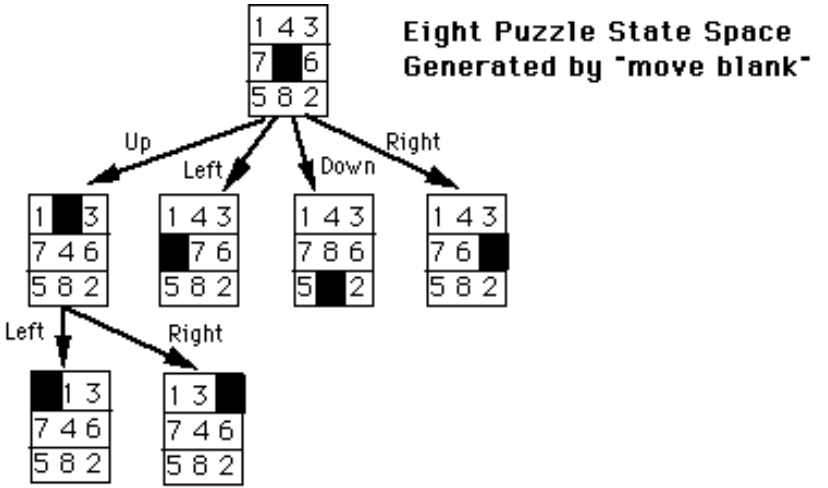
Start State

Goal State

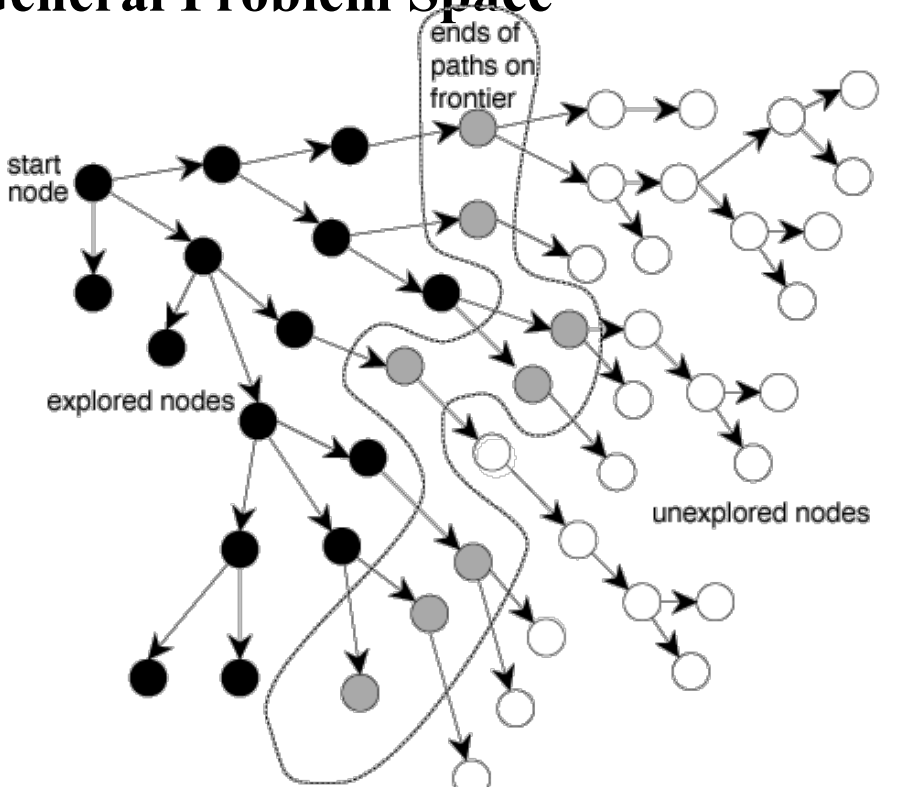
Constraint satisfaction problem - Eight-Queens Problem



Eight-Puzzle Problem Space (State Space)

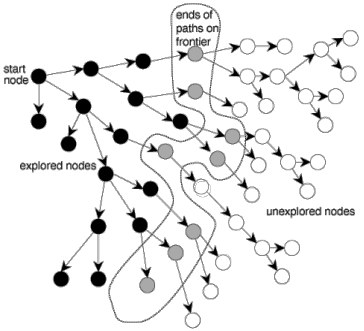


General Problem Space



Basic AI Search Algorithm

```
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
```

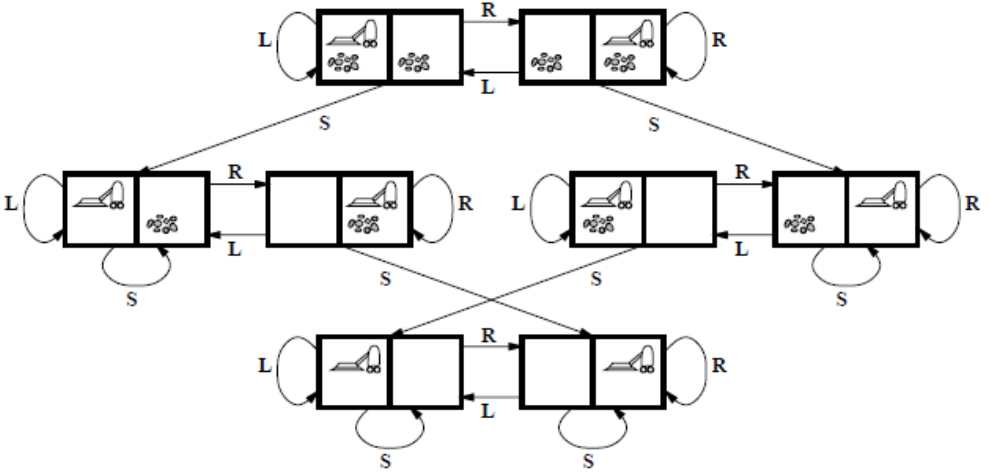


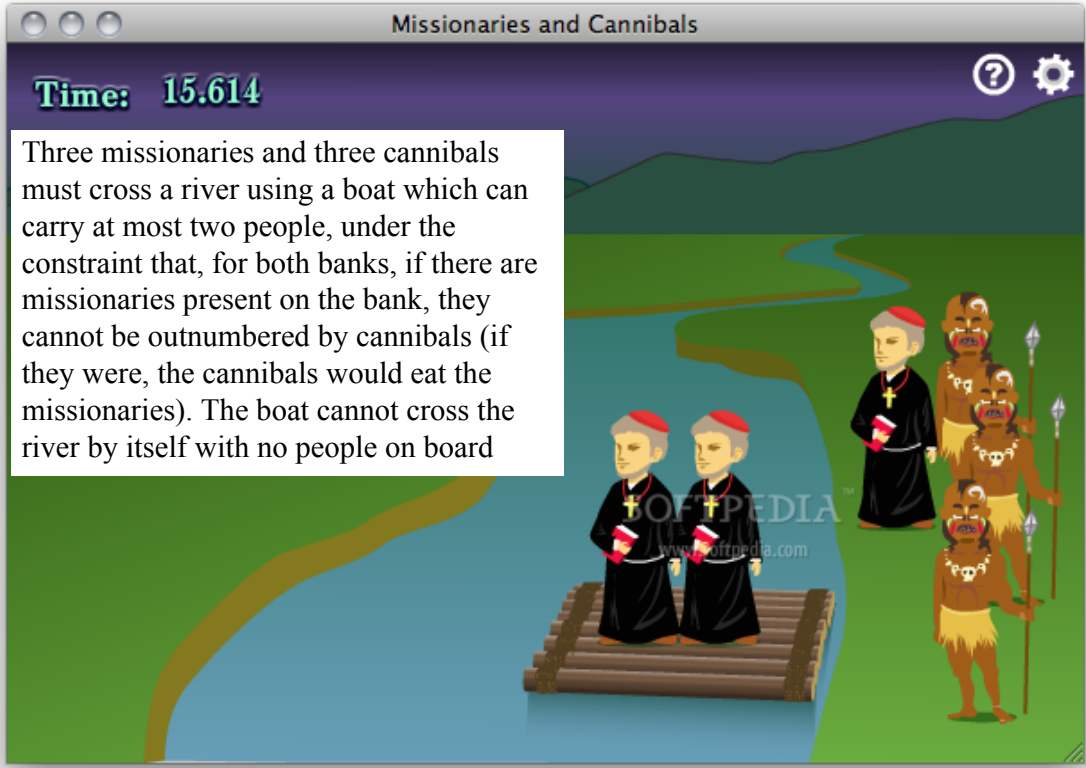
Tree search vs. graph search (differences in italics)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

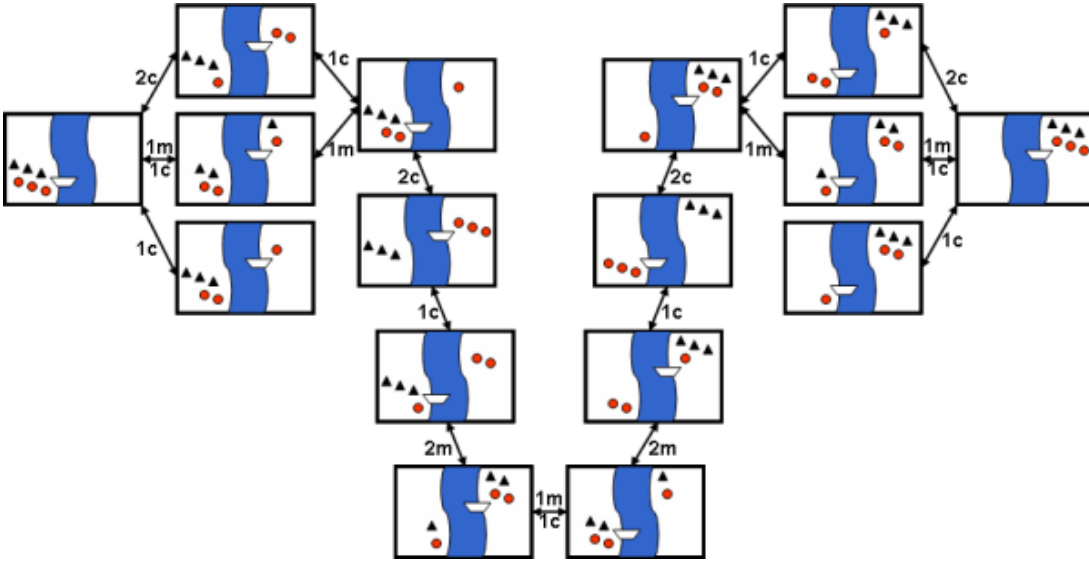
```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set
```

Robotic Vacuum Cleaner as a search problem



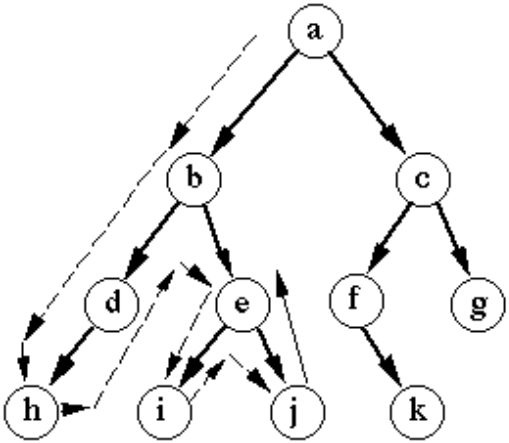

Missionaries and Cannibals
Time: 15.614
Three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board
SOFTPEDIA
www.softpedia.com

Search Tree for M & C

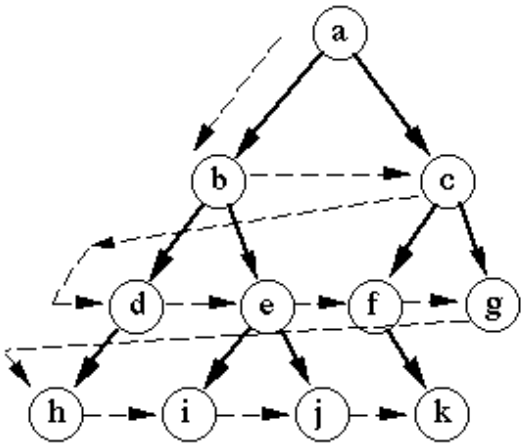


The Basic Brute force Search Strategies
Breadth-first [BFS] and Depth-first [DFS]

Depth first vs. Breadth first search

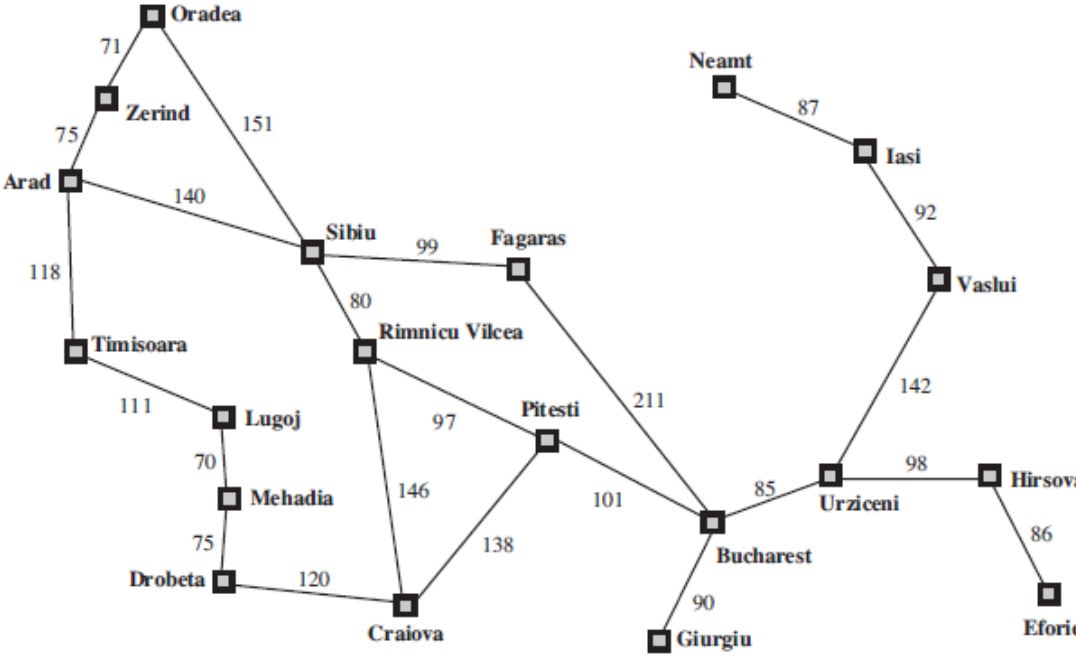


Depth-first search

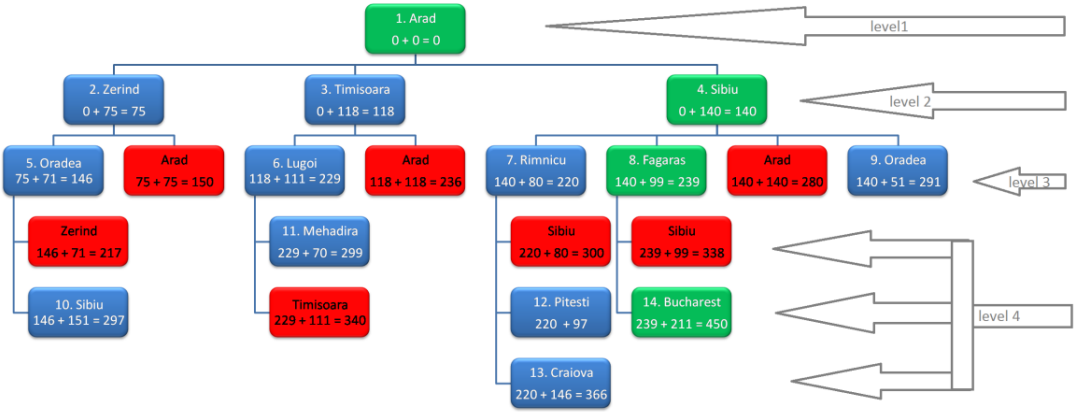


Breadth-first search

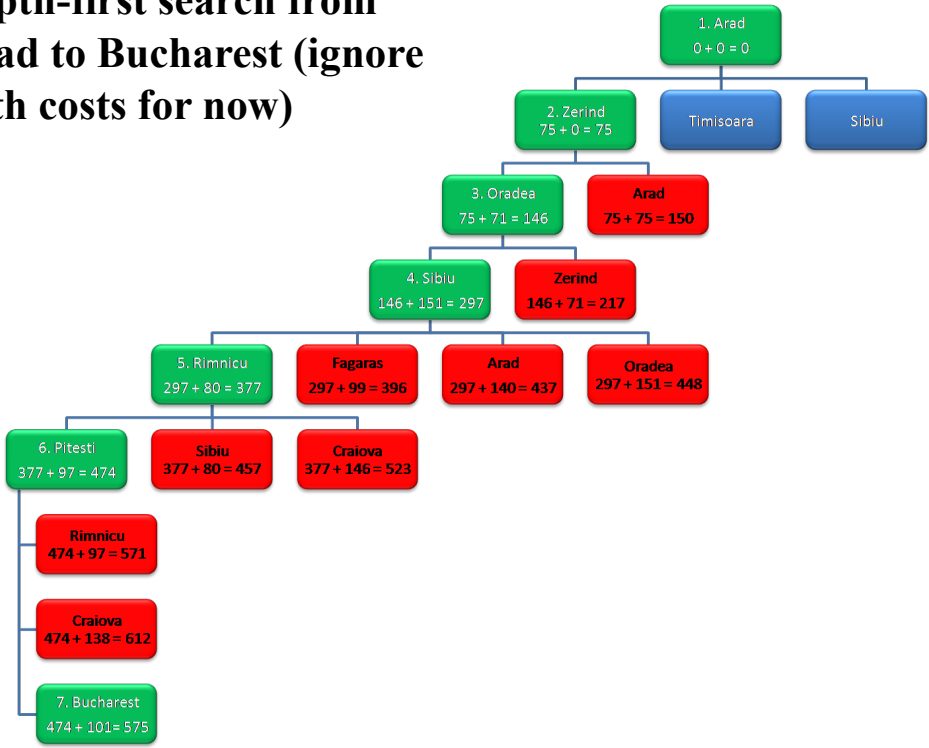
Path finding example (Arad to Bucharest)



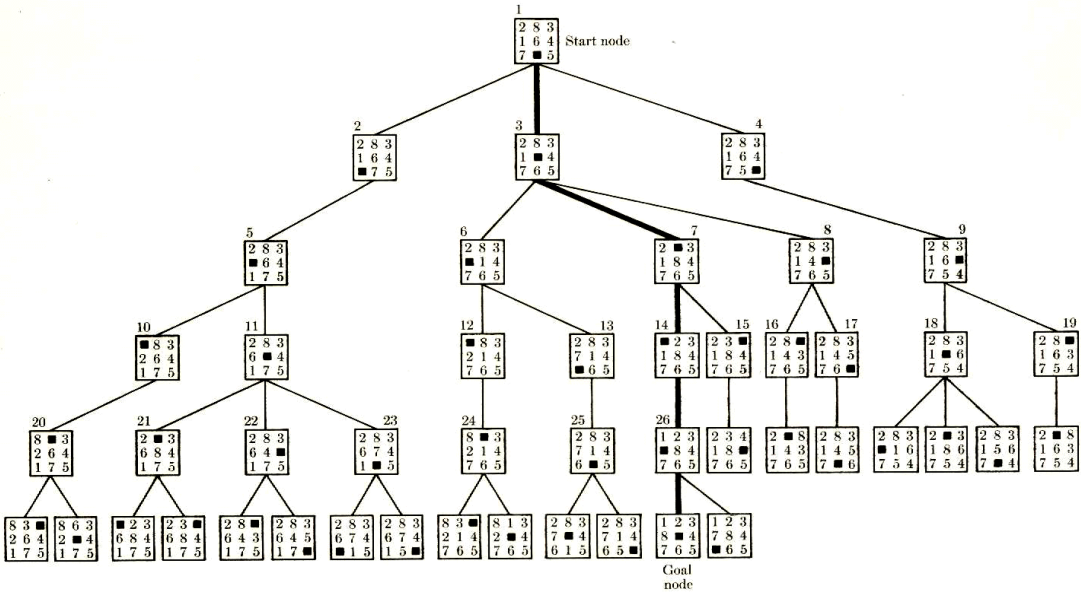
Breadth-first search from Arad to Bucharest (ignore path costs gives us Arad – Sibiu – Fagaras - Bucharest)



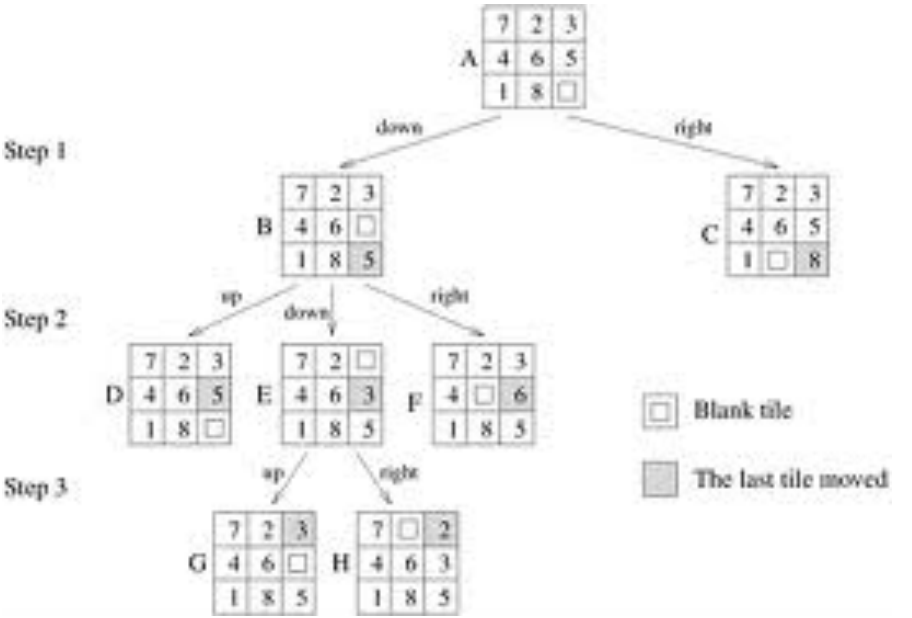
Depth-first search from Arad to Bucharest (ignore path costs for now)



Breadth-first search on eight puzzle



Depth-first search on eight-puzzle



In Class Activity

Breadth first search (Oradea – Giurgiu)

Assumption: Left most node is chosen for expansion first.

D_0: Oradea

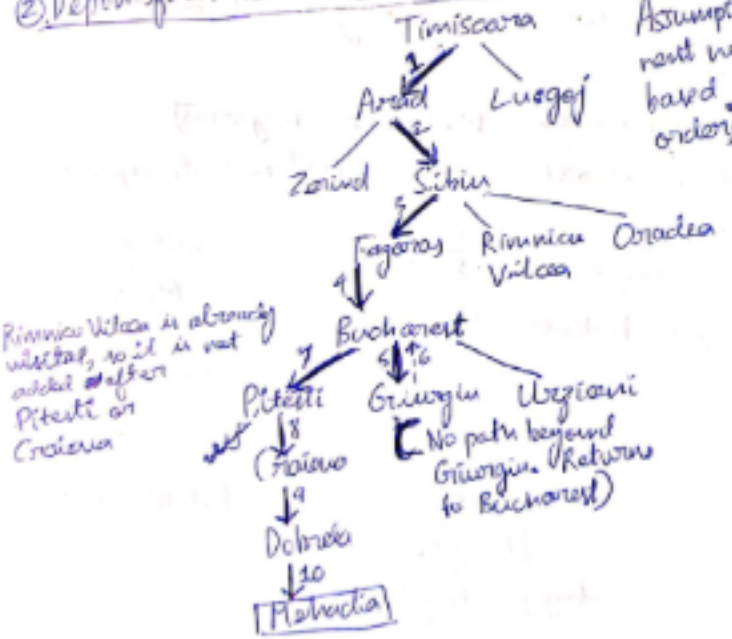
D_1: Oradea
- Zerind
- Sibiu

D_2: Oradea
- Zerind
  - Arad
- Sibiu
  - Rimnicu Vilcea
  - Fagaras

D_3: Oradea
- Zerind
  - Arad
    - Timisoara
- Sibiu
  - Rimnicu Vilcea
    - Craiova
    - Pitesti
  - Fagaras
    - Bucharest

D_4: Oradea
- Zerind
  - Arad
    - Timisoara
      - Lugoj
- Sibiu
  - Rimnicu Vilcea
    - Craiova
      - Dobreta
    - Pitesti
  - Fagaras
    - Bucharest
      - Giurgiu
      - Urziceni

② Depth first search (Timisoara, Mehadia)



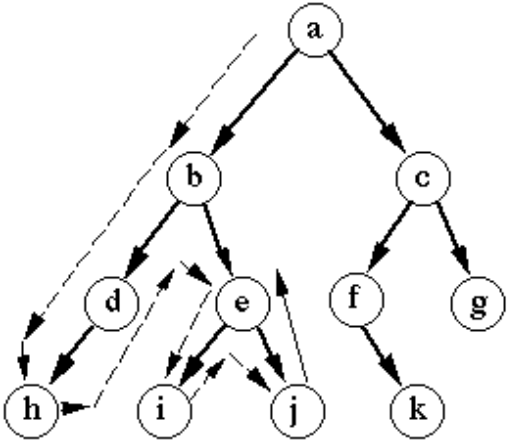
Assumption: Choose the next node for expansion based on alphabetical order.

Rimnicu Vilcea is already visited, so it is not added again in Pitesti or Craiova

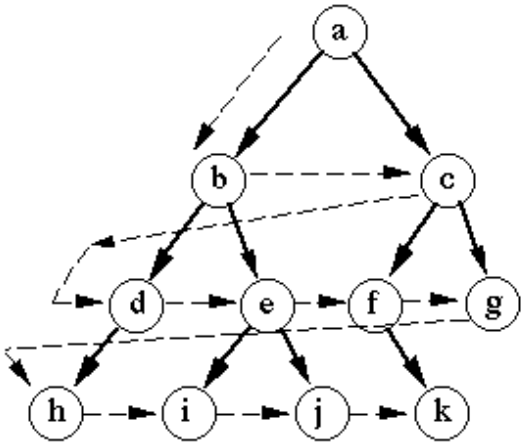
No path beyond Giurgiu. (Returns to Bucharest)

Evaluating Search Strategies

Depth first vs. Breadth first search



Depth-first search



Breadth-first search

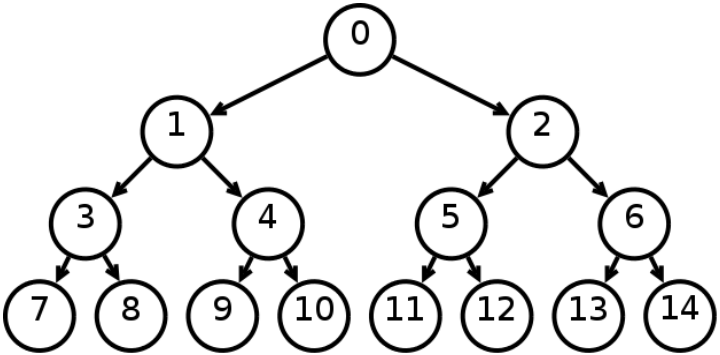
How many nodes in tree with branching factor(b) = 2?

d = 0; 1 node (20)

d = 1; 2 nodes (21)

d = 2; 4 nodes (22)

d = 3; 8 nodes (23)

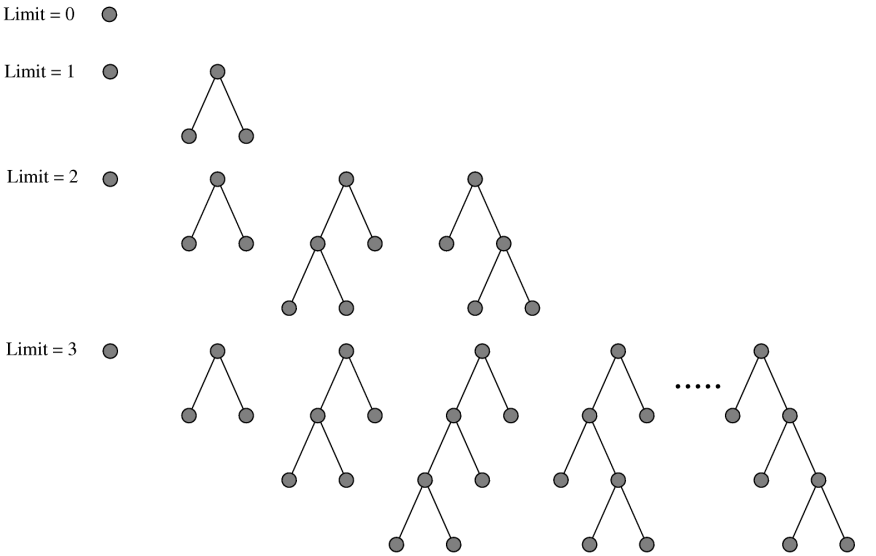


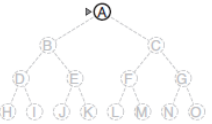
Total nodes – 20 + 21 + 22 +…. 2d = 2d where d = solution depth

If b = branching factor, then bd (the full tree)

Improved Search Strategies

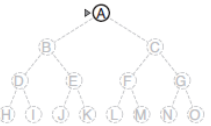
Depth Limited Search



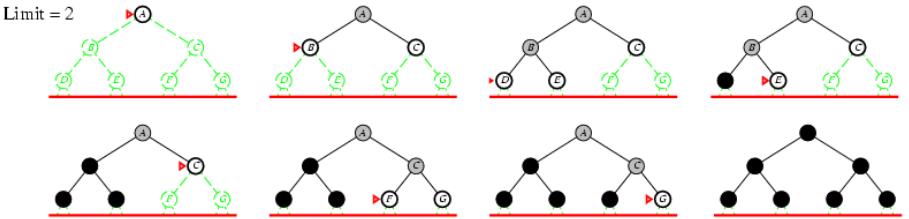


Iterative deepening search *l* =1

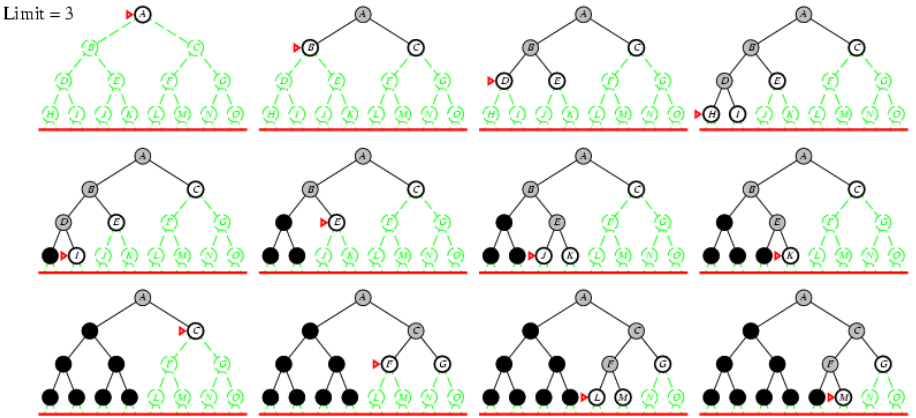
Limit = 1



Iterative deepening search *l* =2



Iterative deepening search *l* =3



Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,c]	No	No	Yes[a]	Yes[a,e]
Optimal?	Yes[b,d]	Yes[b]	No	No	Yes[b,d]	Yes[b,d,e]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

b	finite branching factor
d	depth of solution
m	maximum depth of the search tree
l	depth limit
C^*	optimal solution cost
$\epsilon > 0$	minimal action cost

Footnotes:

[a] if b is finite

[b] if action costs ≥ 0

[c] if action costs $\geq \epsilon > 0$

[d] if action costs are all identical

[e] if both directions use breadth-first search

Time and Memory Requirements for BFS

Depth	Nodes	Time	Memory
2	110	1.1 milliseconds	107 kilobytes
4	11,110	111 milliseconds	10.6 megabytes
6	10^6	11 seconds	1 gigabytes
8	10^8	19 minutes	103 gigabytes
10	10^{10}	31 hours	10 terabytes
12	10^{12}	129 days	1 petabytes
14	10^{14}	35 years	99 petabytes
16	10^{16}	3,500 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 100,000 nodes/second; 1000 bytes/node.

Animations

- https://qiao.github.io/PathFinding.js/visual/

Review