Search

COMP9414 24t2

version 1.1

Dr Armin Chitizadeh

UNSW SYDNEY

By the end of this lecture you will learn

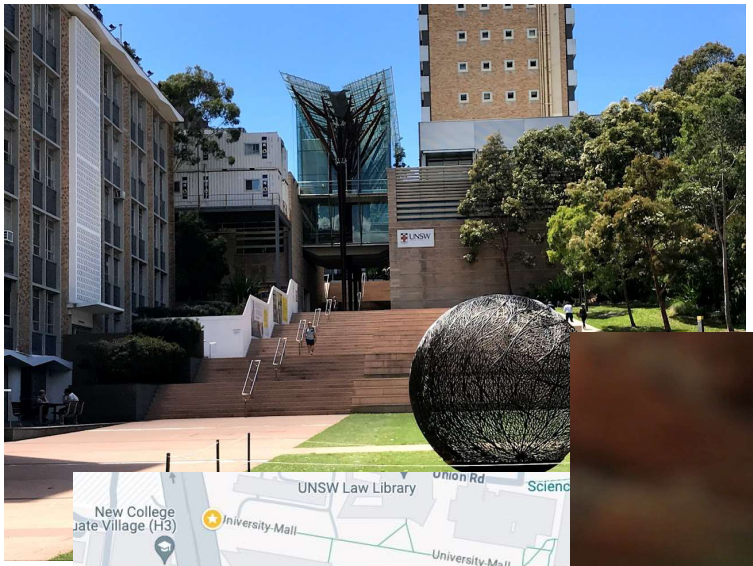
- The theory behind search in AI
- To develop a smart agent with search capability to solve complicated problem

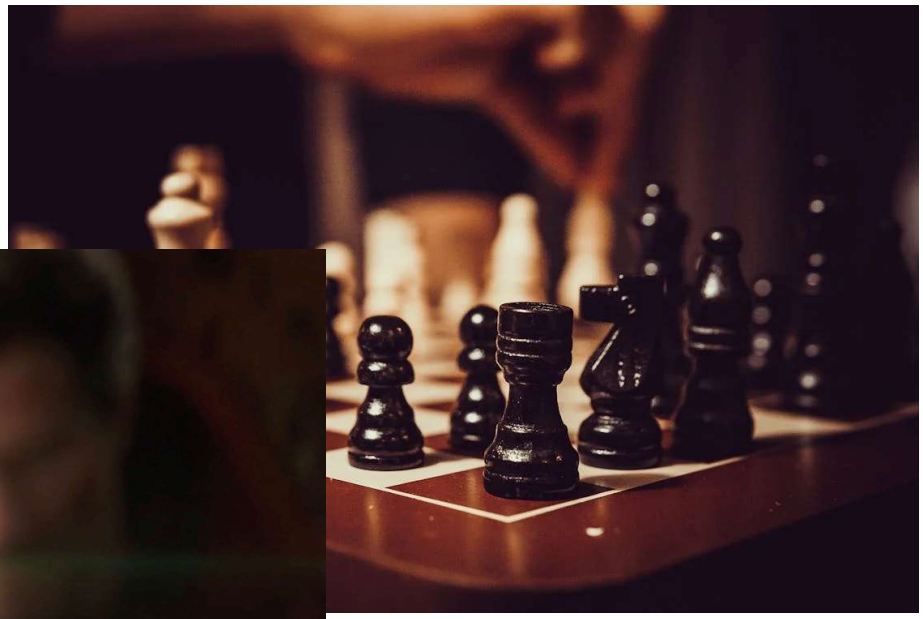
Content

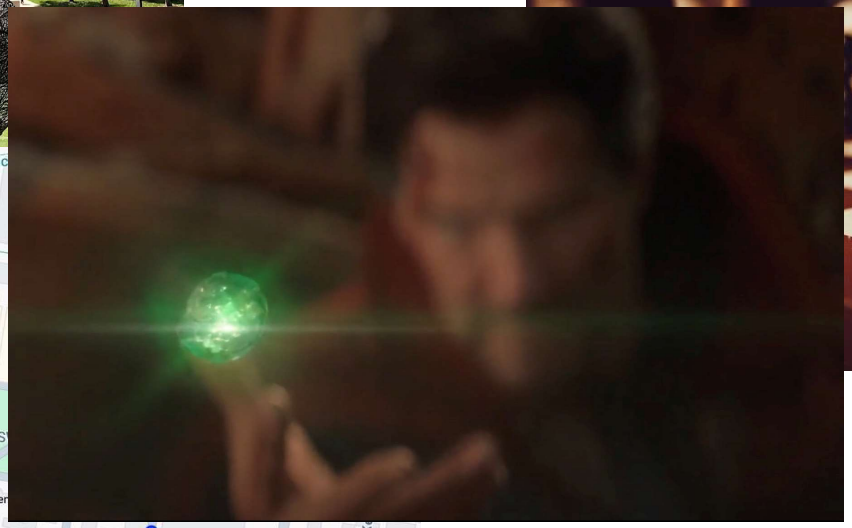
- What is Search and why is it needed?
- Human Search vs Computer Search
- Converting real-world problem into machine understandable problem
- Choosing the best search algorithm

What is Search and Why is it needed

> *In which we see how an agent can find a sequence of actions that achieves its goals when no single action will do.*
>
> - Peter Norvig and Stuart J. Russel
> Artificial Intelligence: A Modern Approach







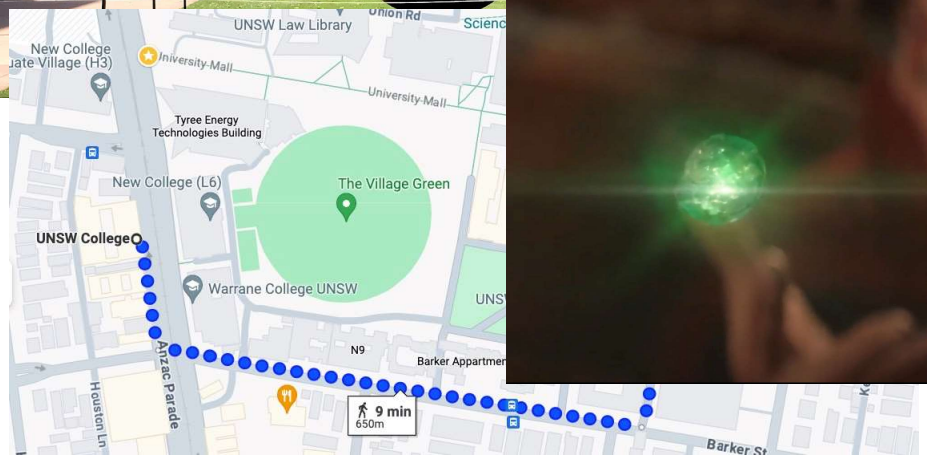


Photo by Charlie Solorzano
from pexels.com

When is Search Needed?

Motion Planning

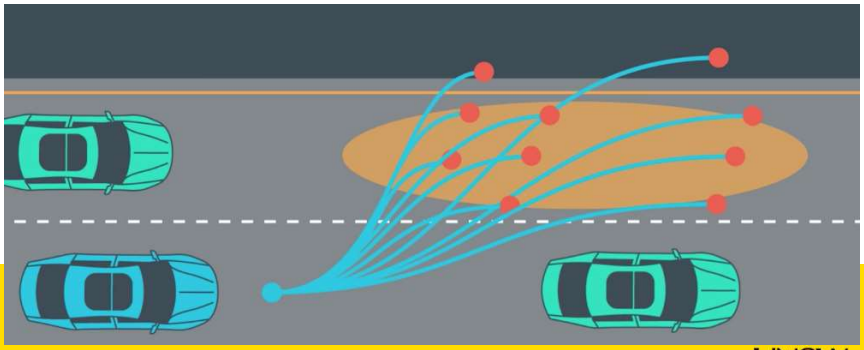
Navigation

Speech and Natural Language

Task Planning

Machine Learning

Game Playing



Human Search vs Computer Search

Human Search vs Computer Search

Photo generated by Microsoft Copilot

Human Search vs Computer Search

Photo generated by Microsoft Copilot

Converting real-world problem into machine understandable problem

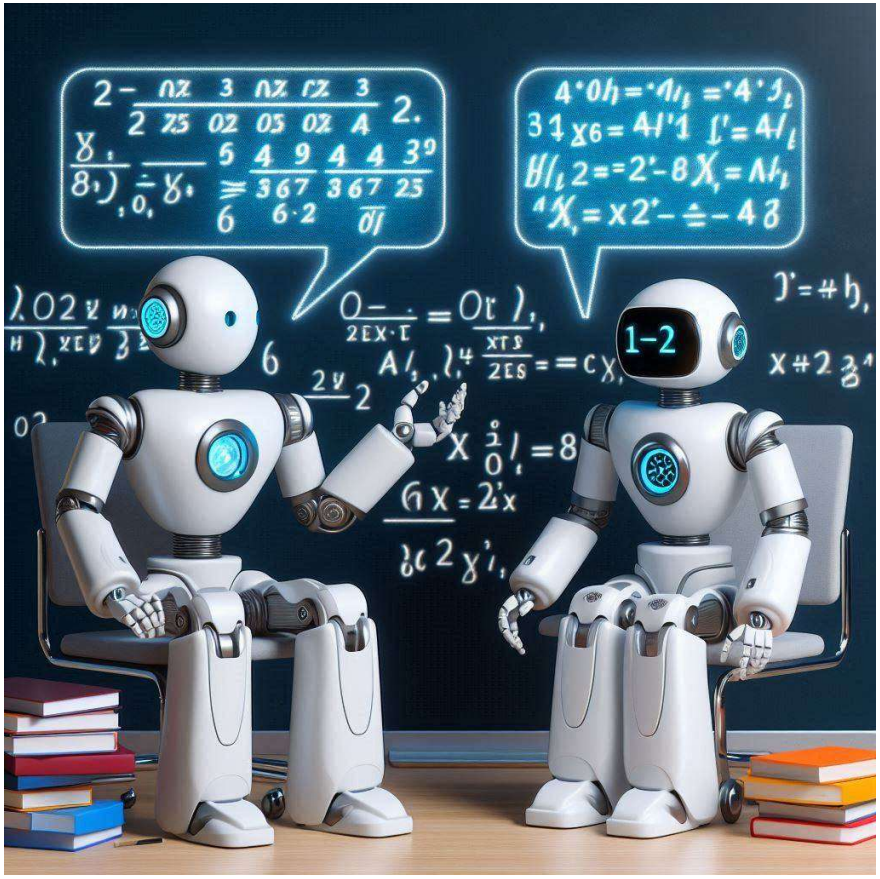
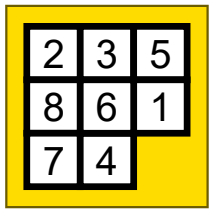


Photo generated by Microsoft Copilot

Let's formalize it!

A problem can be described by

- state




```
[[2,3,5],
 [8,6,1],
 [7,4,0]]
```

```
[[rB,0 ,bB,qB,KB,bB,0 , 0],
 [pB,0 ,pB,pB, 0, 0, 0, 0],
 [0 ,0 ,kB, 0, 0,pW, 0,rB],
...
```

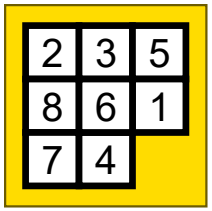
- action {"up", "left"} { 

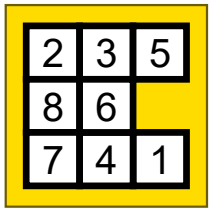
, …}

- Transition Function do: S × A → S

Let’s formalize it

Transition Function

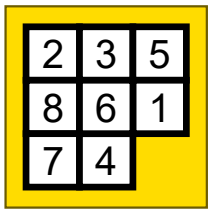
do (

, “up”) → 

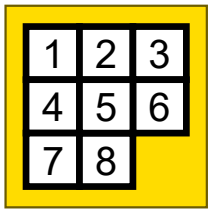
Let's formalize it

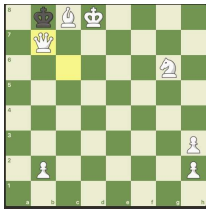
Special states:

- Initial state: s_0

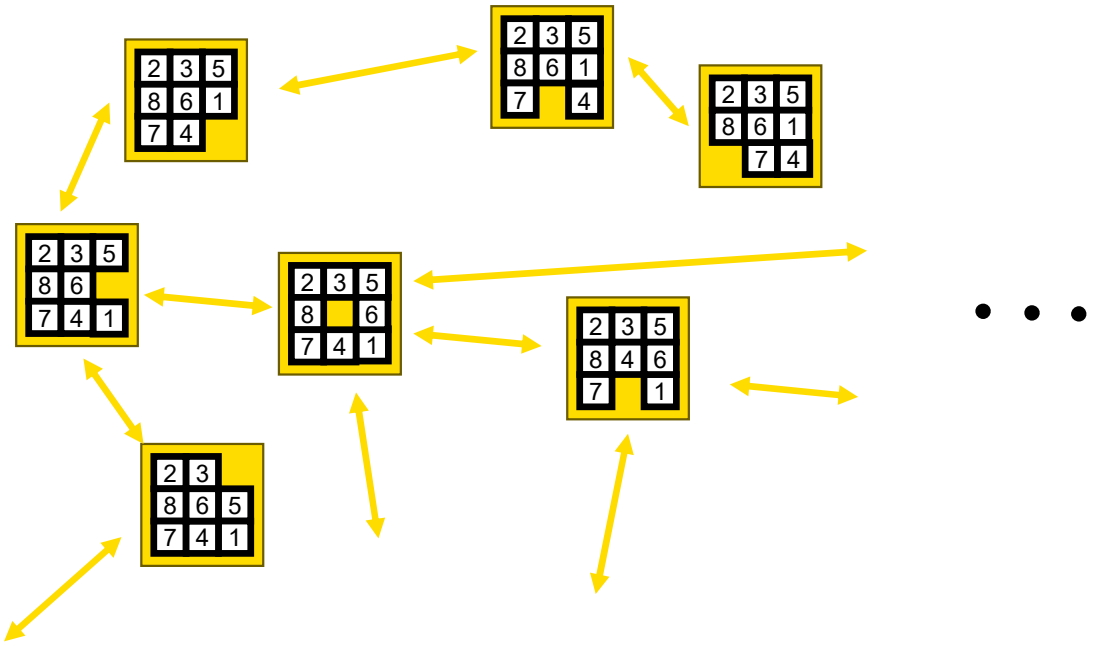


- Terminal states: $Z \subset S$



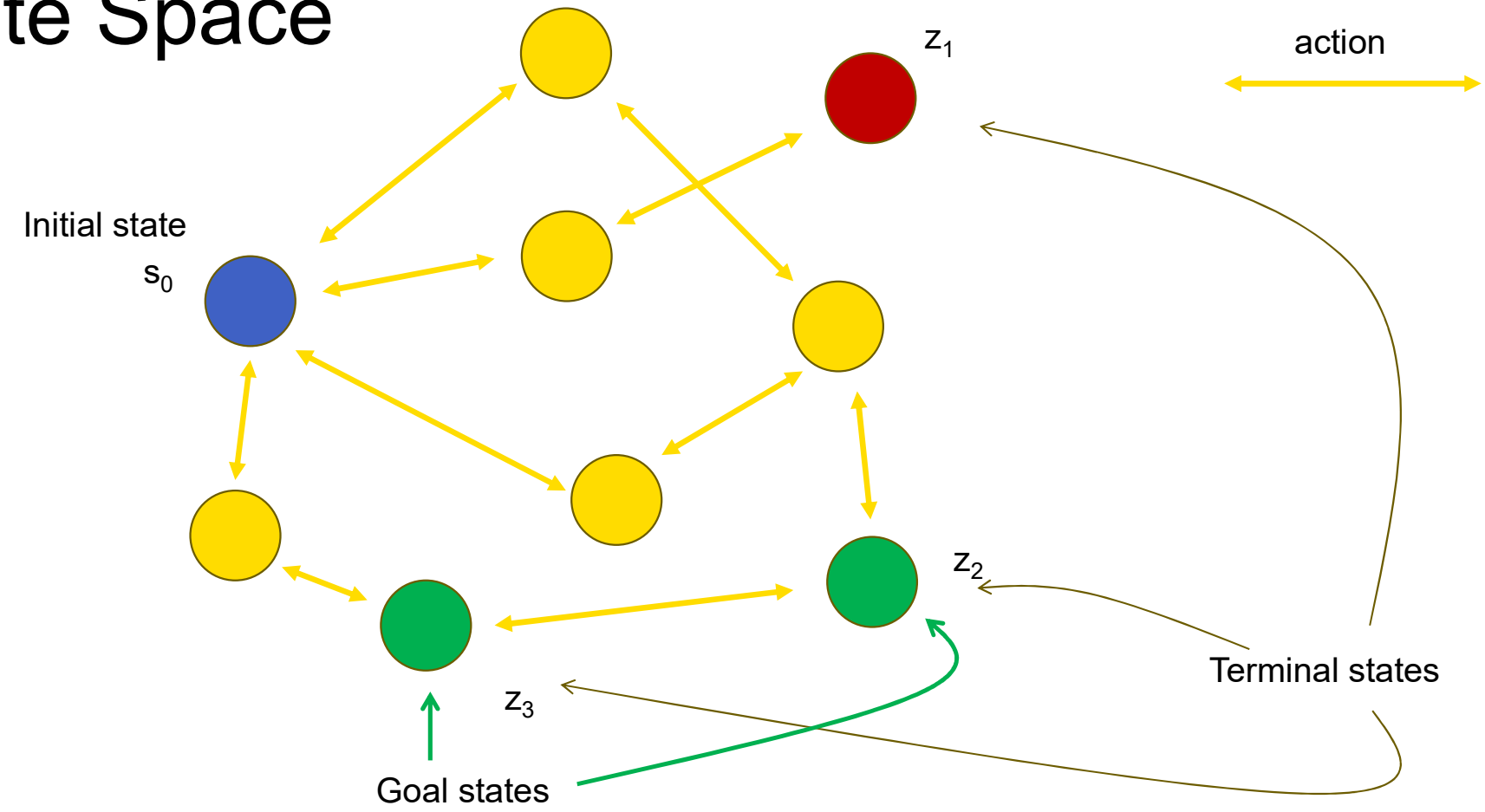


State Space


2 3 5
8 6 1
7 4
2 3 5
8 6 1
7 4
2 3 5
8 6 1
7 4
2 3 5
8 6
7 4 1
2 3 5
8 6
7 4 1
2 3 5
8 4 6
7 1
2 3
8 6 5
7 4 1

• • •

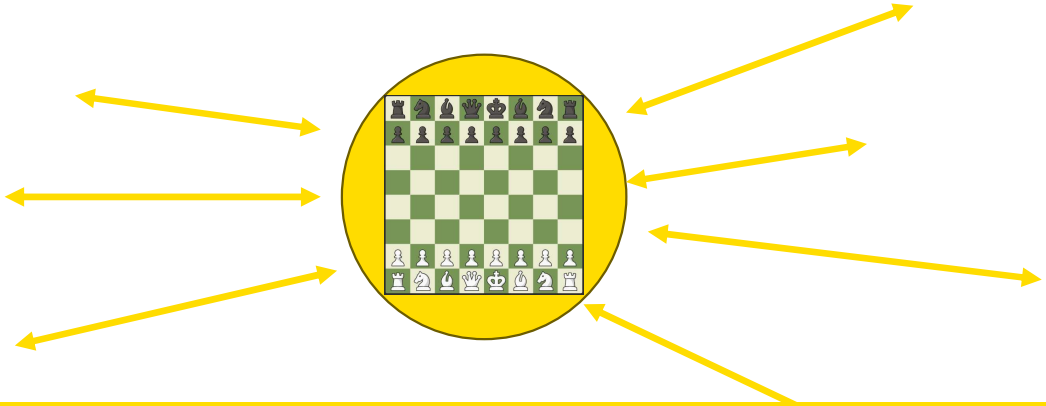
State Space


z_1
action
Initial state
s_0
z_2
Terminal states
z_3
Goal states

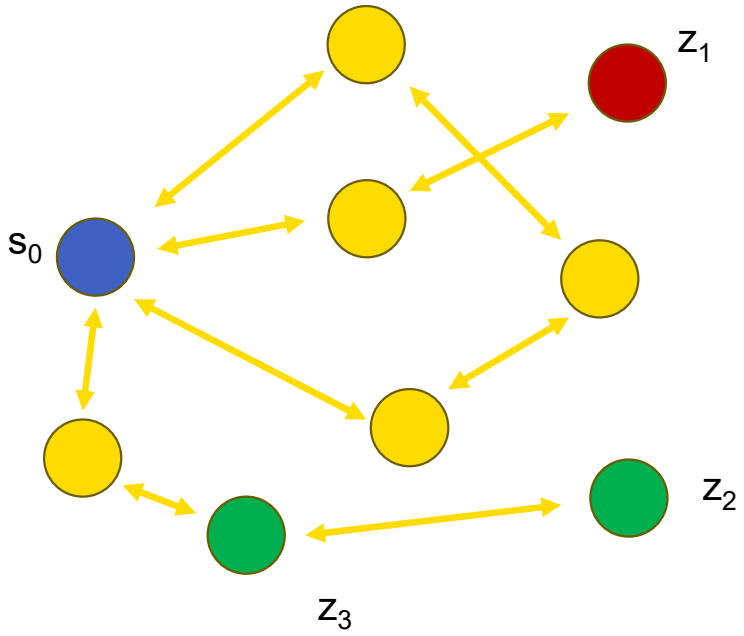
State Space in Graph

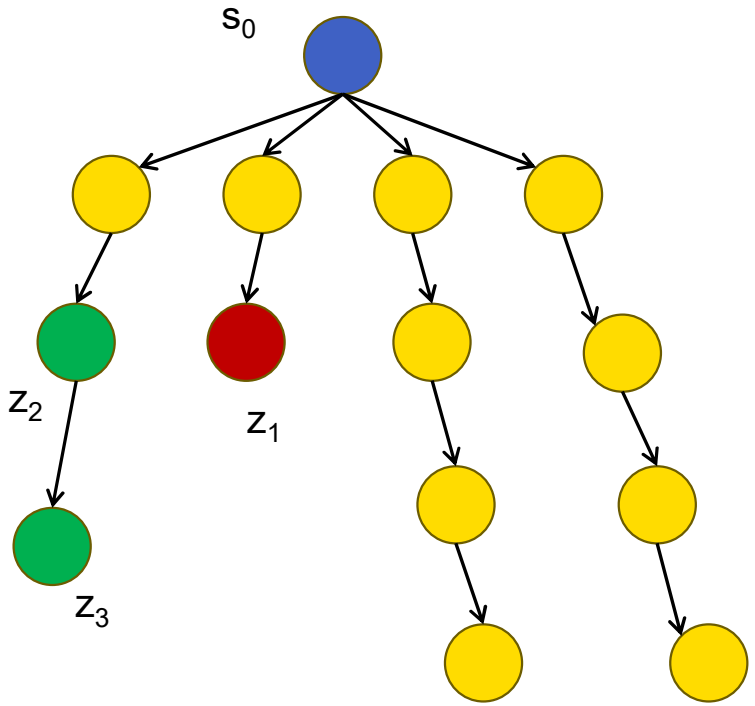
Node vs **State**

- Node in Graph is a data structure which contains a state and other related information such as legal actions from the state



Graph State Space vs Tree State Space

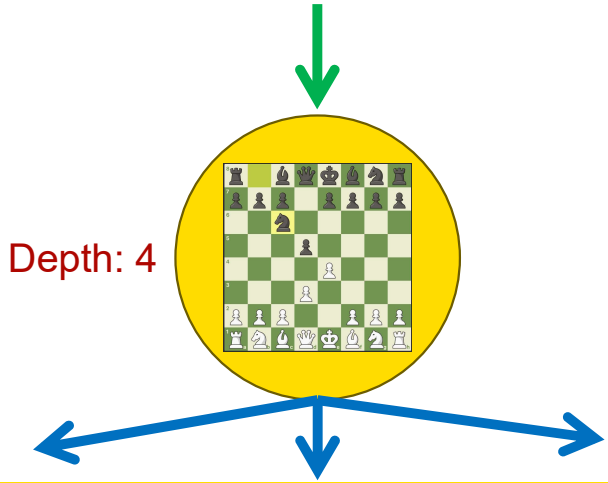




State Space in Tree

Node vs **State**

- Node in Tree is a data structure which contains a state and other related information such as children nodes, parent node and depth



Now we have converted problems in their tree structure, can we solve all problems?

Approximately 10^{24} possible states

Photo by Charlie Solorzano
from pexels.com

- For many problems you can't generate the whole tree
- So you can not search all possible path
- When you have a choice, which path will you choose?

https://forms.office.com/r/bBb6bt2sv7

Choosing the best Search algorithm

	Search Algorithms Covered at this Lecture
Uninformed Search	Breath First Search (BFS)
	Depth First Search (DFS)
	Depth Limited Search
	Bidirectional Search
	Iterative Deepening Search (IDDFS)
	Uniform Cost Search (UCS)
Informed Searches	~~Heuristic~~
	Greedy Best-First Search
	A* Search

Overview Uninformed Search

Uninformed Search	Background & Terminology
	Breath First Search (BFS)
	Depth First Search (DFS)
	Uniform Cost Search (UCS)
	Depth Limited Search
	Iterative Deepening Search (IDDFS)
	Bidirectional Search
Informed Searches	Heuristic
	Greedy Best-First Search
	A* Search

Search Methods

Uninformed search

- use no problem-specific information
- Uninformed (or “blind”) search strategies use only the information available in the problem definition (can only distinguish a goal from a non-goal state)

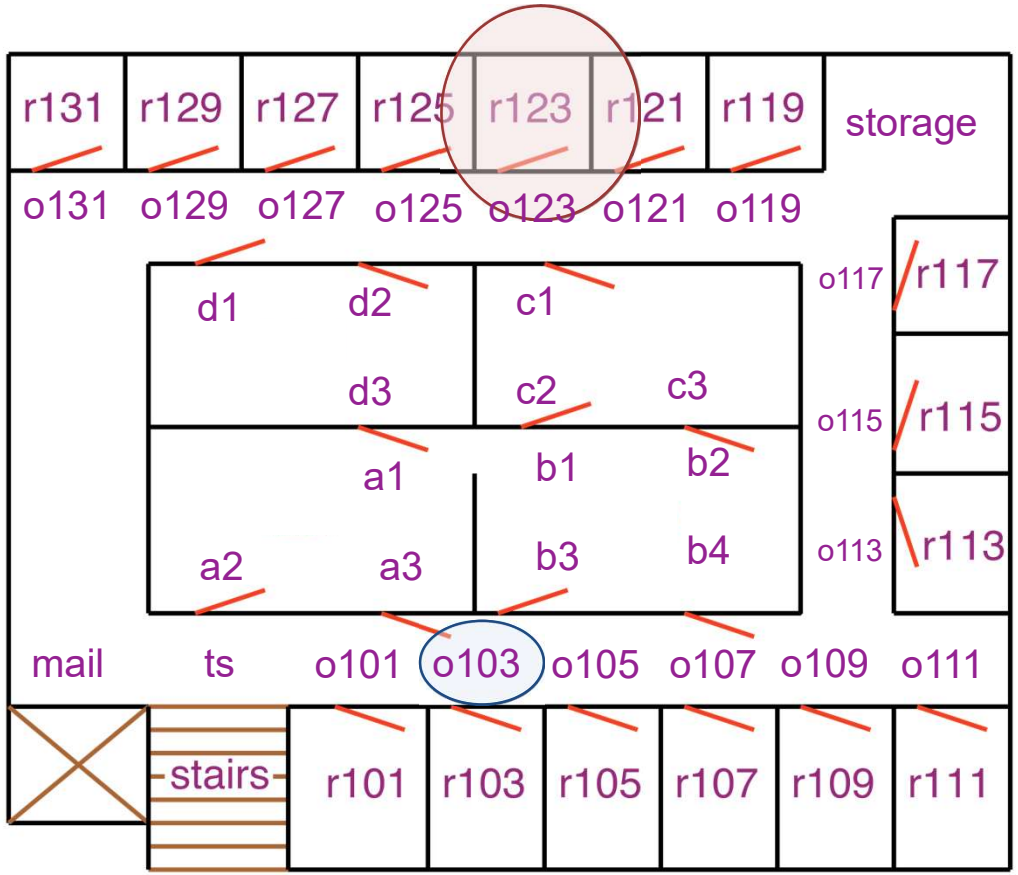
Informed search

Delivery Robot

The robot wants to get from outside room 103 to the inside of room 123.

- The only way a robot can get through a doorway is to push the door open in the direction shown.

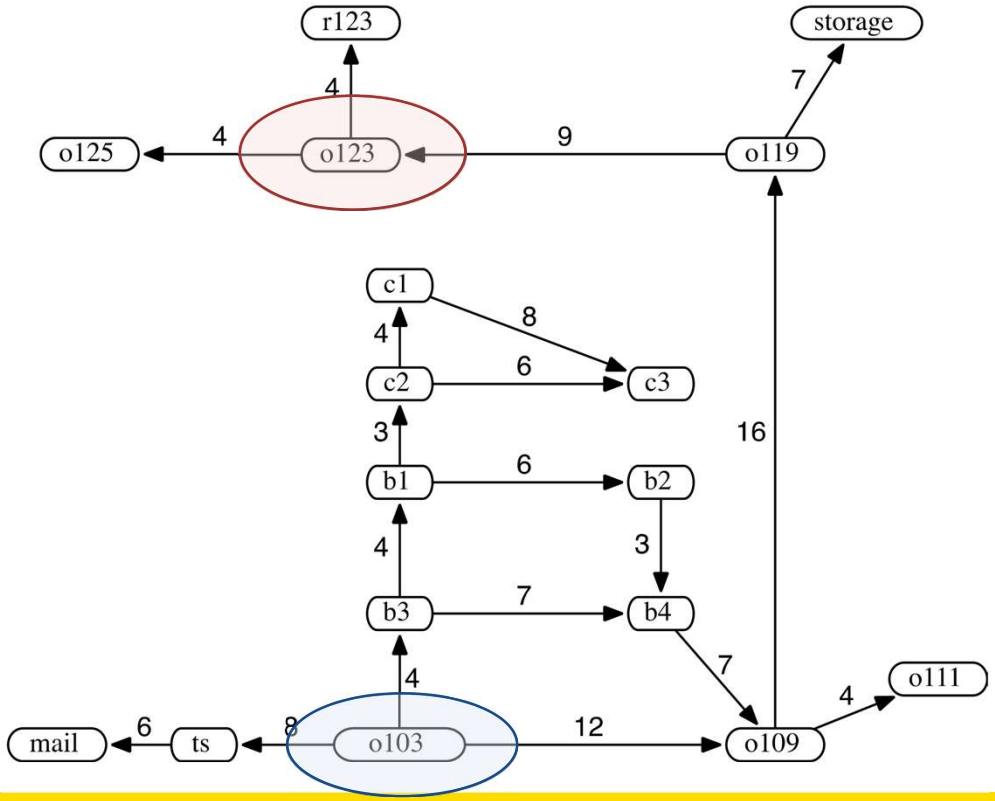
The task is to find a path from o103 to to r123



State-Space Graph for Delivery Robot

Modelled as a state-space search problem

States are locations.



State Space Search Problems

State space — set of all states reachable from initial state(s) by any action sequence

Initial state(s) — element(s) of the state space

Transitions

- **Operators** — set of possible actions at agent's disposal; describe state reached after performing action in current state, or
- **Successor function** — s(x)= set of states reachable from state x by performing a single action

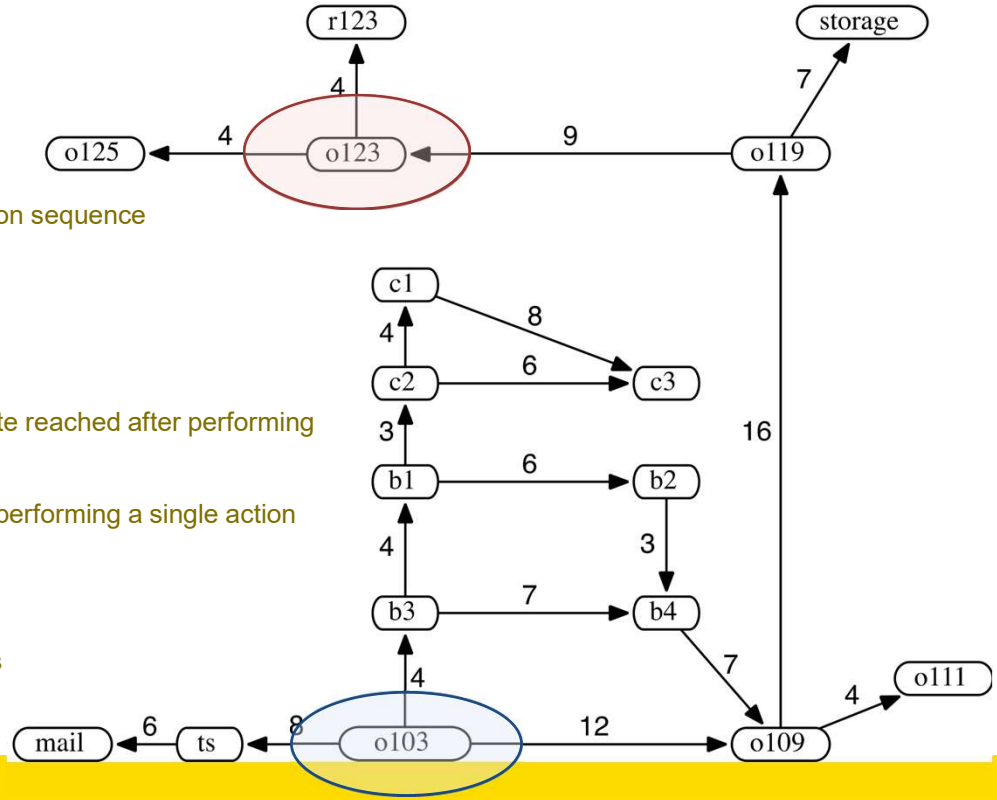
Goal state(s) — element(s) of the state space

Path cost — cost of a sequence of transitions used to evaluate solutions (applies to optimisation problems)

State-Space Graph for Delivery Robot

- Modelled as a state-space search problem
- States are locations.

- **State space** — set of all states reachable from initial state(s) by any action sequence
- **Initial state(s)** — element(s) of the state space
- Transitions
  - **Operators** — set of possible actions at agent's disposal; describe state reached after performing action in current state, or
  - **Successor function** — s(x)= set of states reachable from state x by performing a single action
- **Goal state(s)** — element(s) of the state space
- **Path cost** — cost of a sequence of transitions used to evaluate solutions (applies to optimisation problems)



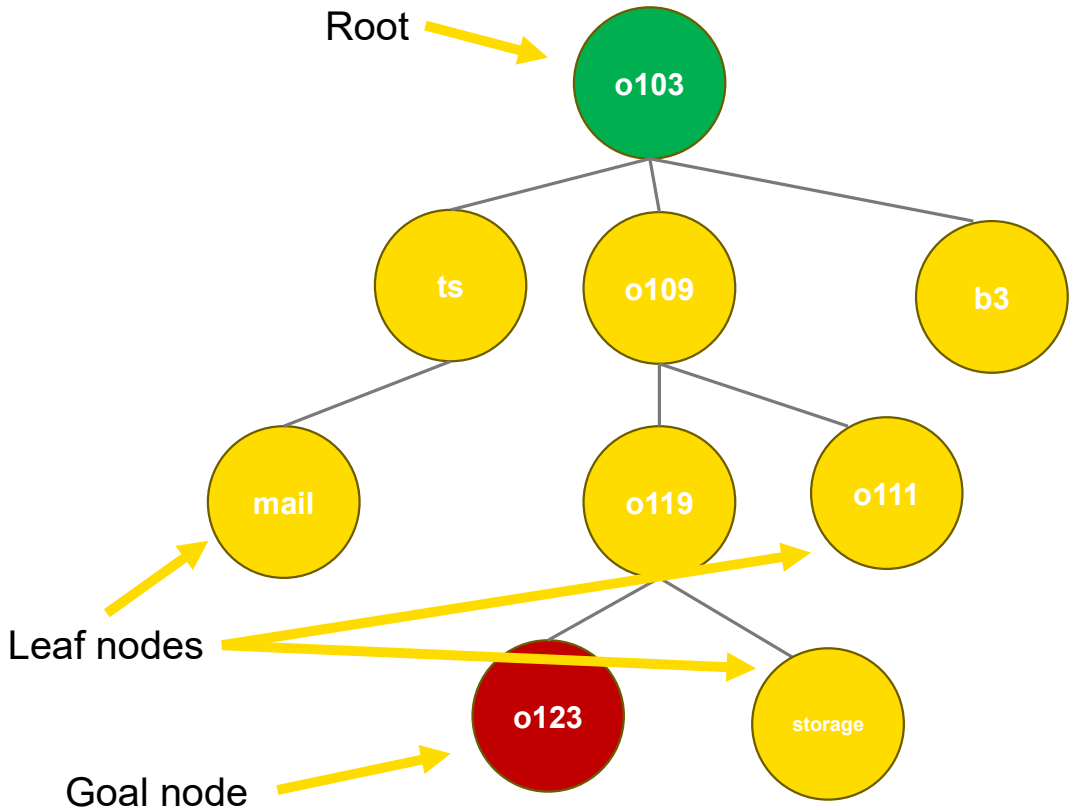
UNSW SYDNEY

Search Tree

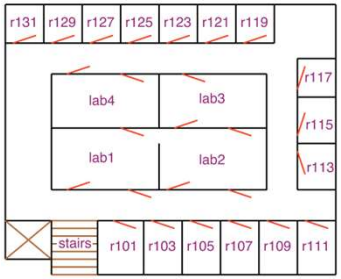
Search tree: superimposed over the state space.

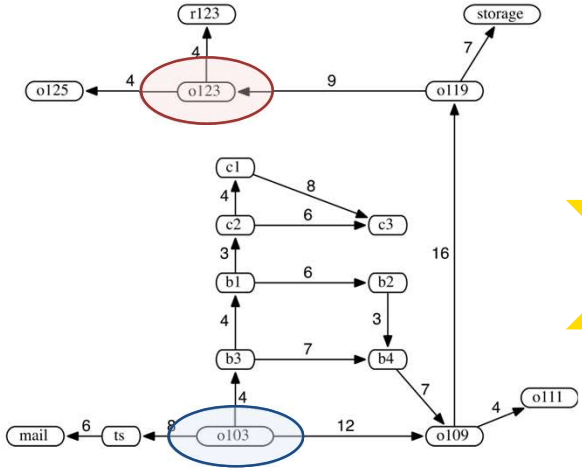
Root: search node corresponding to the initial state.

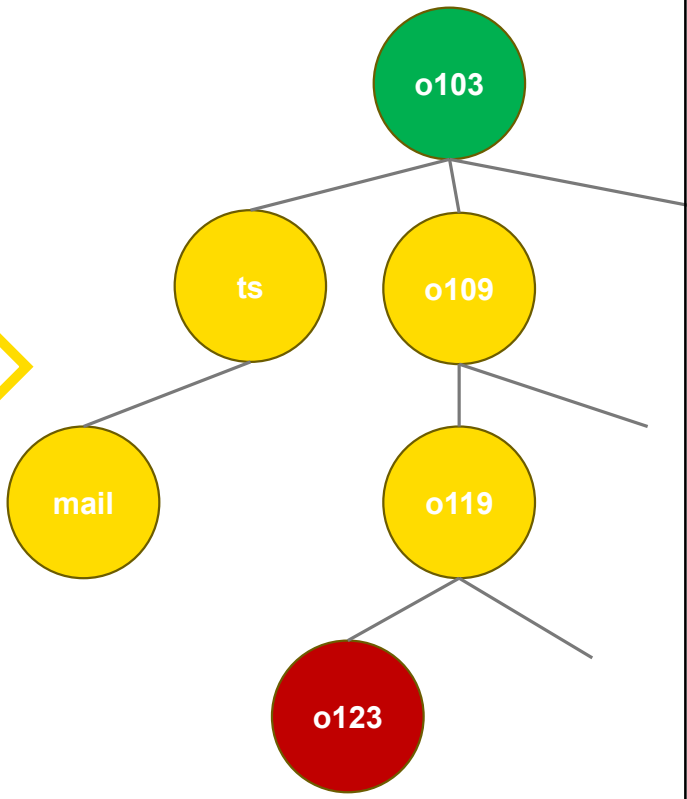
Leaf nodes: correspond to nodes that have no successors in the tree because do no have any successor or are not expanded



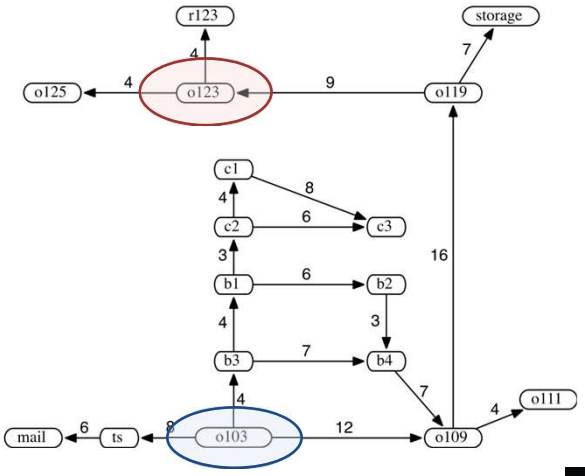
Problem to State-Space to Search Tree

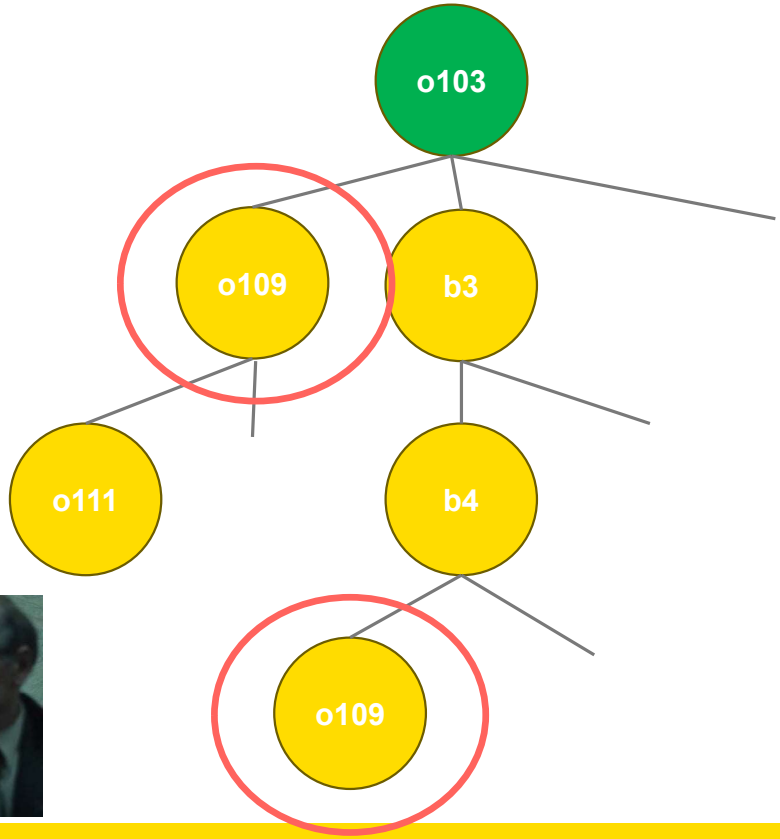






Problem to State-Space to Search Tree



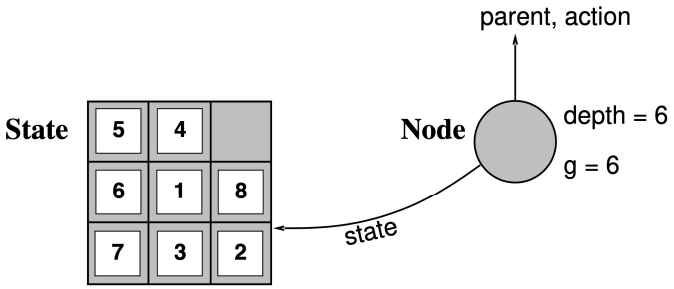


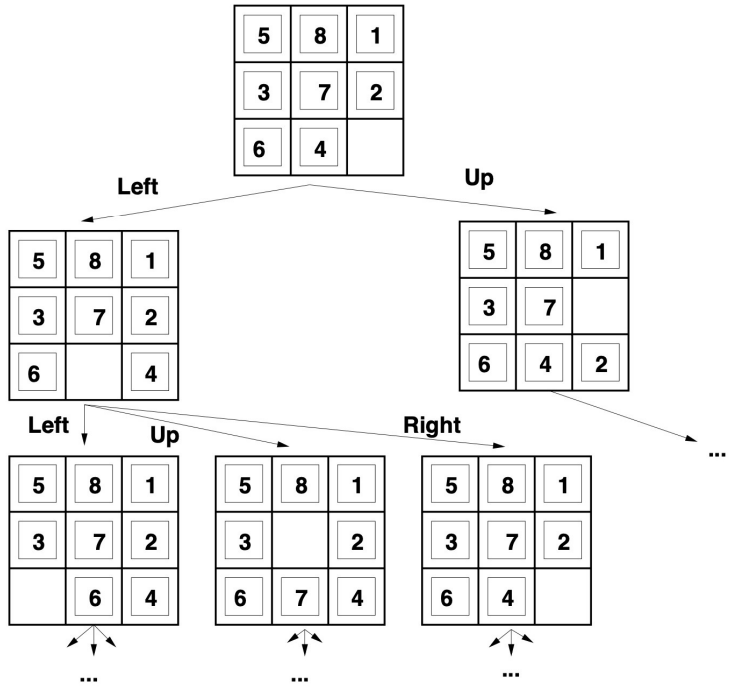
What about Chess?

State-Space Graph

- **State** is a representation of problem at a given time.

- In **State-Space Graph or Tree Search** each node contains a states and arcs are actions

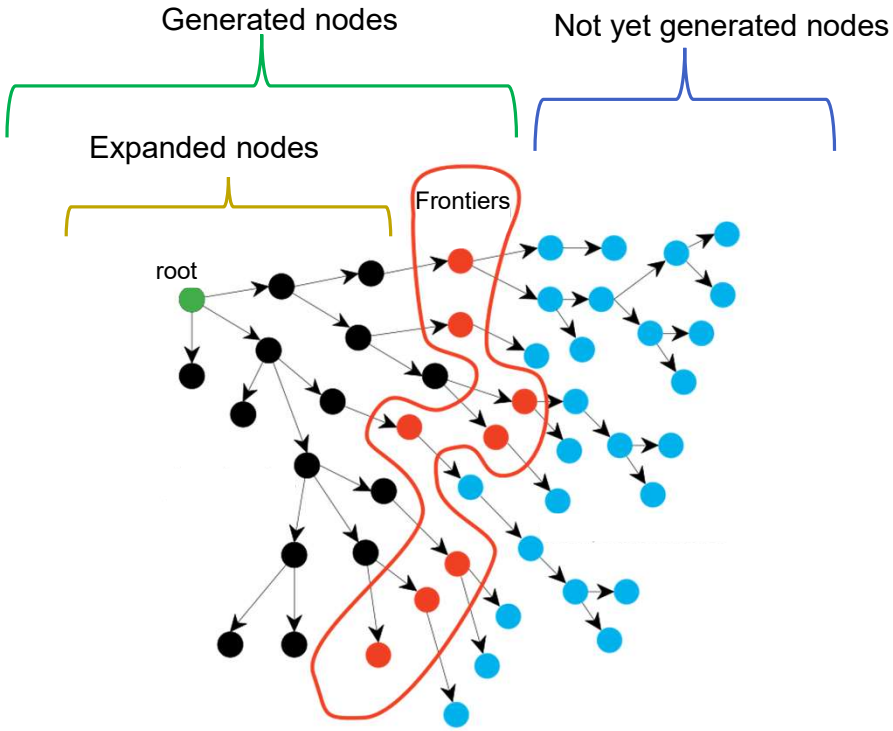




States do not have parents, children, depth or path cost!

Problem Solving by Graph Searching

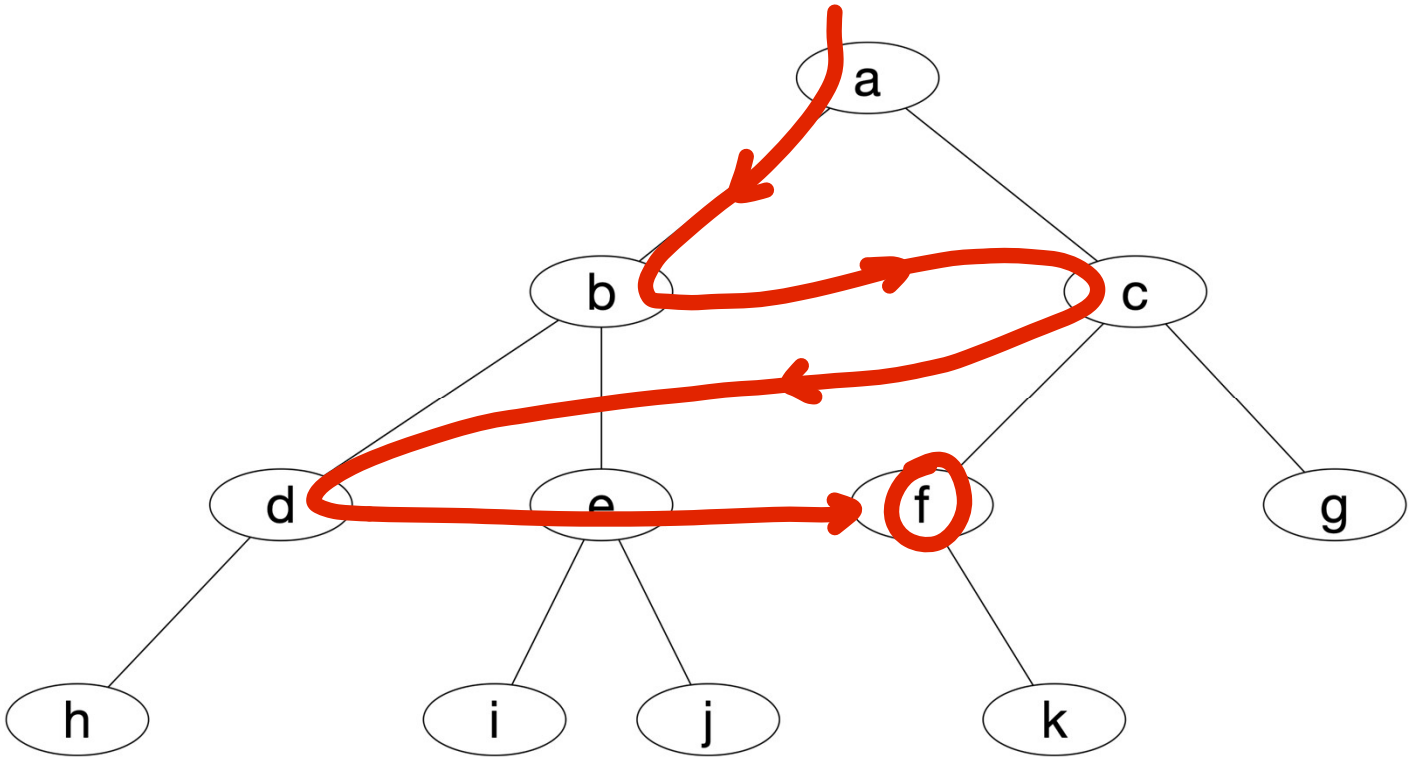
- **Expanded nodes**: green and black
- **Frontiers**: red
- Generated: green, black and red
- Not yet generated nodes: blue



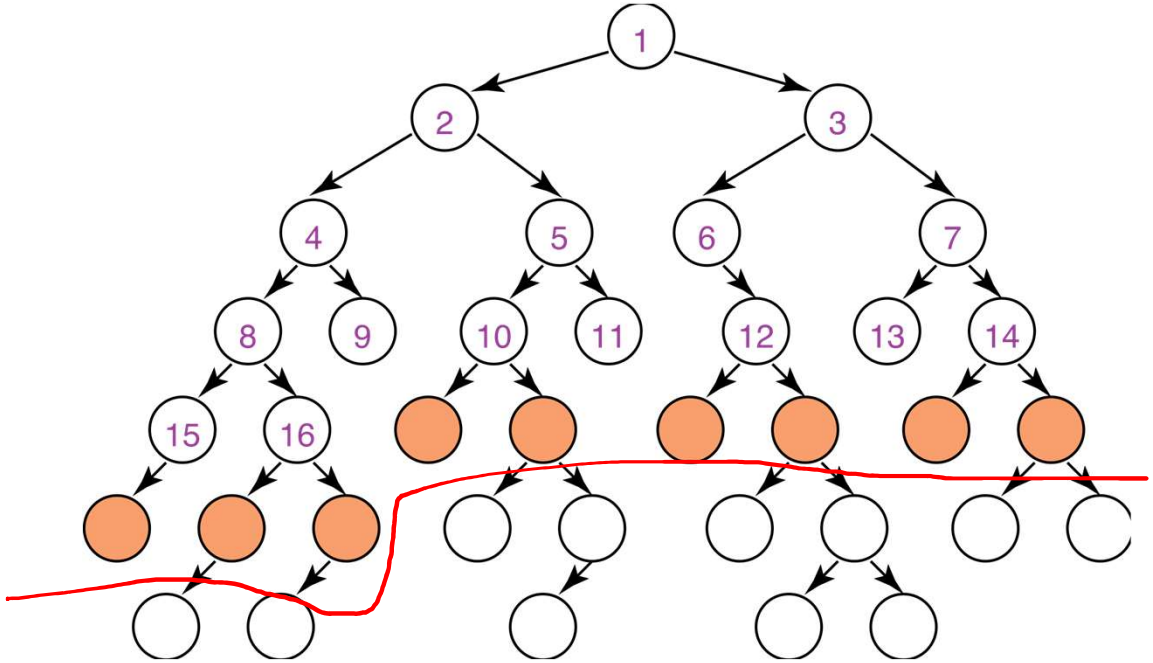
Search strategy differ in the way they expand the frontier

Breath First Search (BFS)

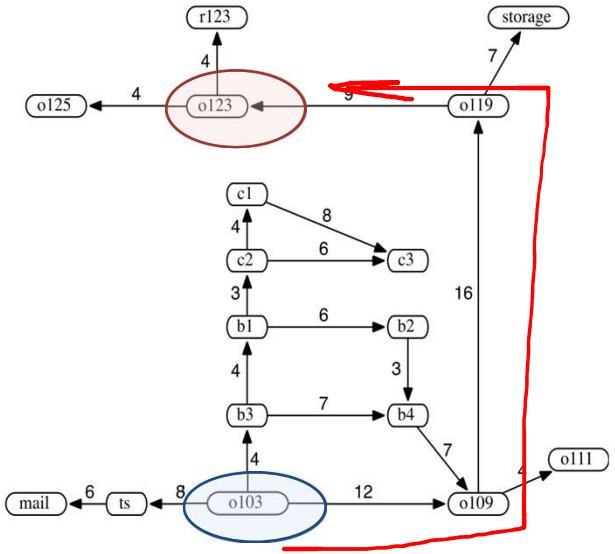
Breadth-First Search

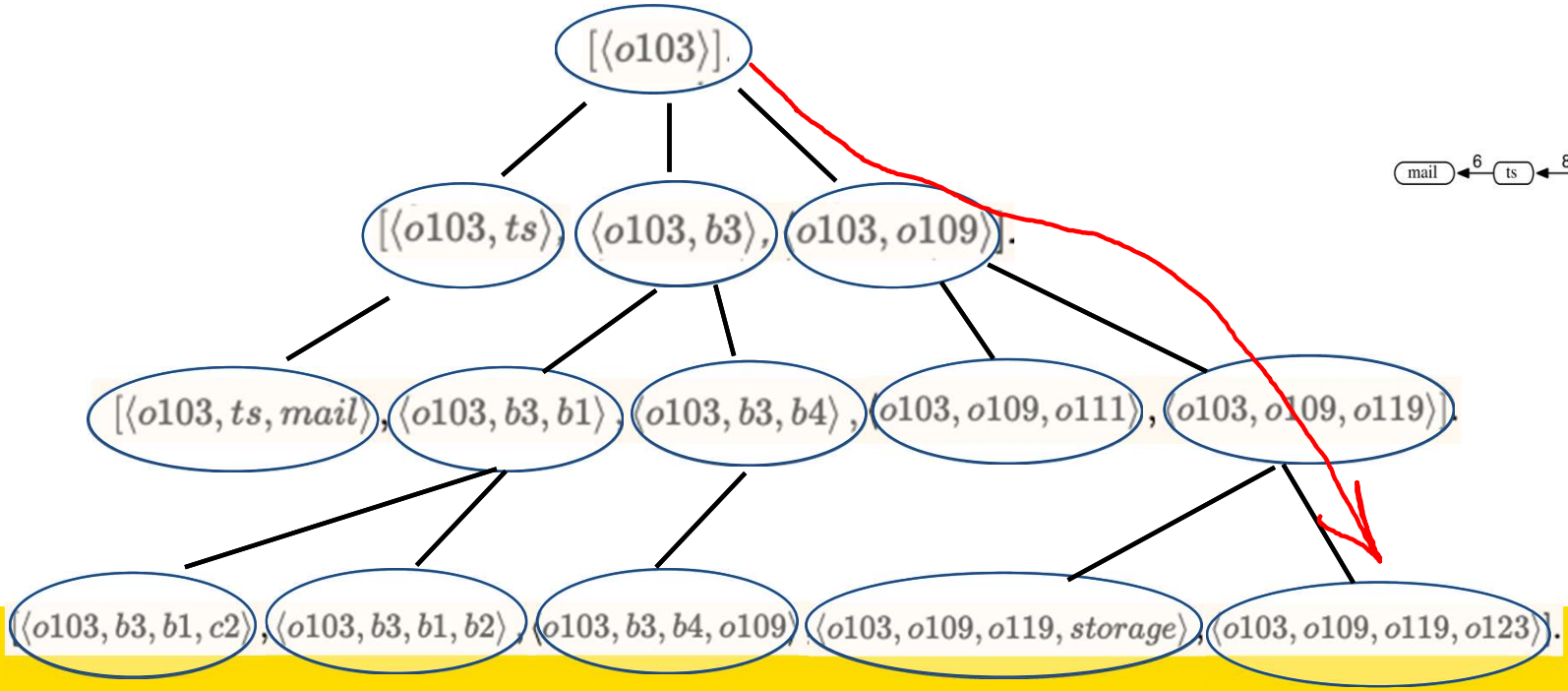


Breadth-first Search Frontier



Breadth-First Search





Breadth-first Search

Breadth-first search treats the frontier as a queue

It selects the first element in the queue to explore next

If the list of paths on the frontier is $[p_1,p_2,...,p_r]$:

- p_1 is selected. Its children are added to the end of the queue, after p_r .
- p_2 is selected next.

Breadth-First Search

All nodes are expanded at a same depth in the tree before any nodes at the next level are expanded

Can be implemented by using a queue to store frontier nodes

- put newly generated successors at end of queue

Include check that state has not already been explored

- Needs a new data structure for set of explored states

Finds the shallowest goal first

Complexity of Breadth-first Search

- Does breadth-first search guarantee finding the shortest path?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the length of the path selected?
- What is the space complexity as a function of the length of the path selected?
- How does the goal affect the search?

Properties of breadth-first search

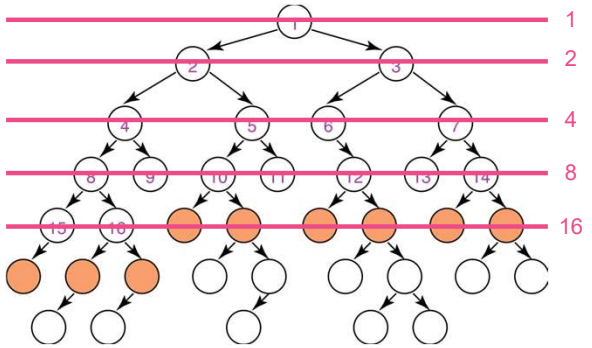
Complete? Yes (if breadth, b, is finite, the shallowest goal is at a fixed depth, d, and will be found before any deeper nodes are generated)

Time? $1 + b^1 + b^2 + b^3 + \ldots + b^d = \frac{b^{d+1}-1}{b-1} = O(b^d)$

Space? $O(b^d)$) keeps every node in memory; generate all nodes up to level d)

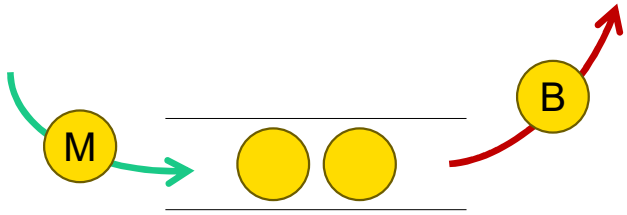
Optimal? Yes, but only if all actions have the same cost

Space is the big problem for BFS. It grows exponentially with depth



Expanded vs Generated

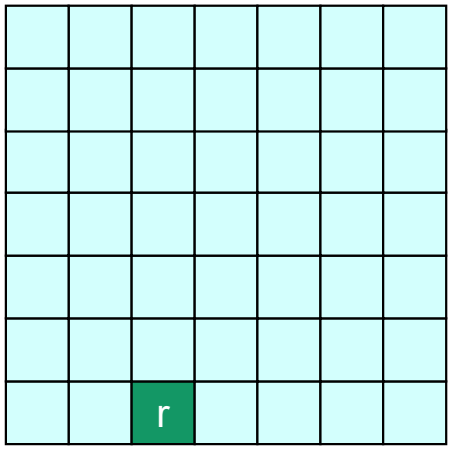
- When a node is inserted in the list, it is **Generated**

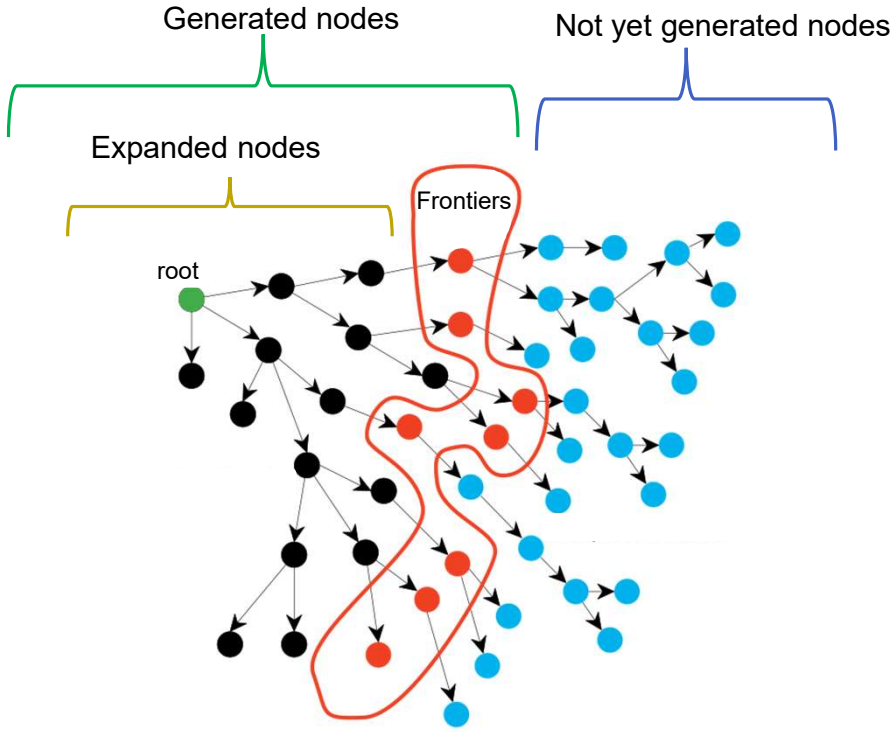


Node M is generated
and
Node B is in the process of being expanded

- When a node is removed from the list, it is in the process of being **Expanded**

BFS in searching a grid





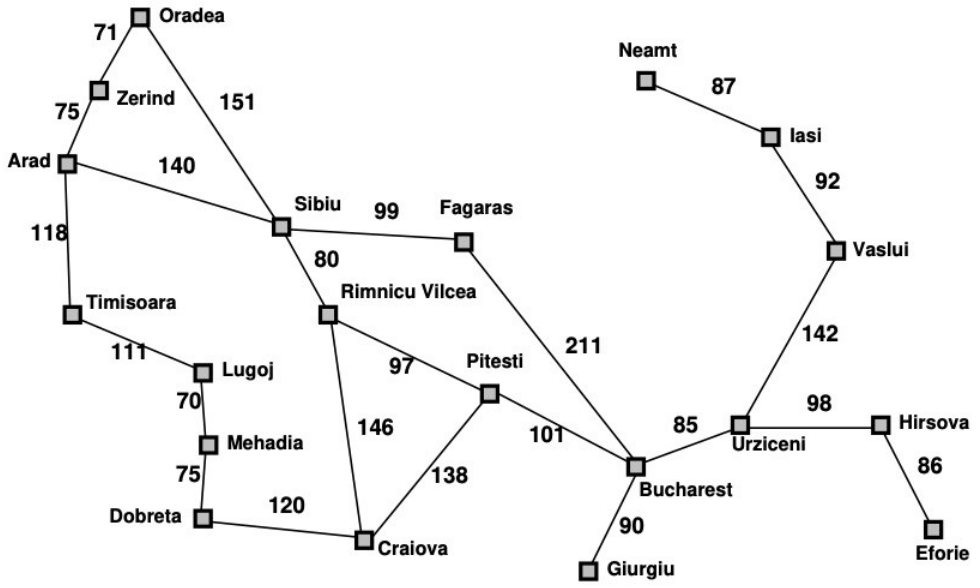
Exercise

This exercise uses the route-finding example with the Romanian map from Russell & Norvig

(Artificial Intelligence: A Modern Approach) as an example.

For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes take the first one by alphabetical ordering. For breath-first search, stop the search when the goal state is generated and use a check to ensure that nodes with the same state as a previously expanded node are not added to the frontier.

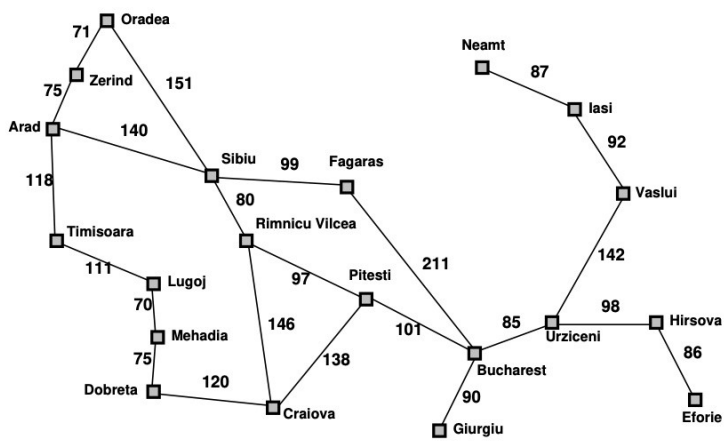
* Breadth-First Search



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exercise
BFS

B O L R O F Z T S A



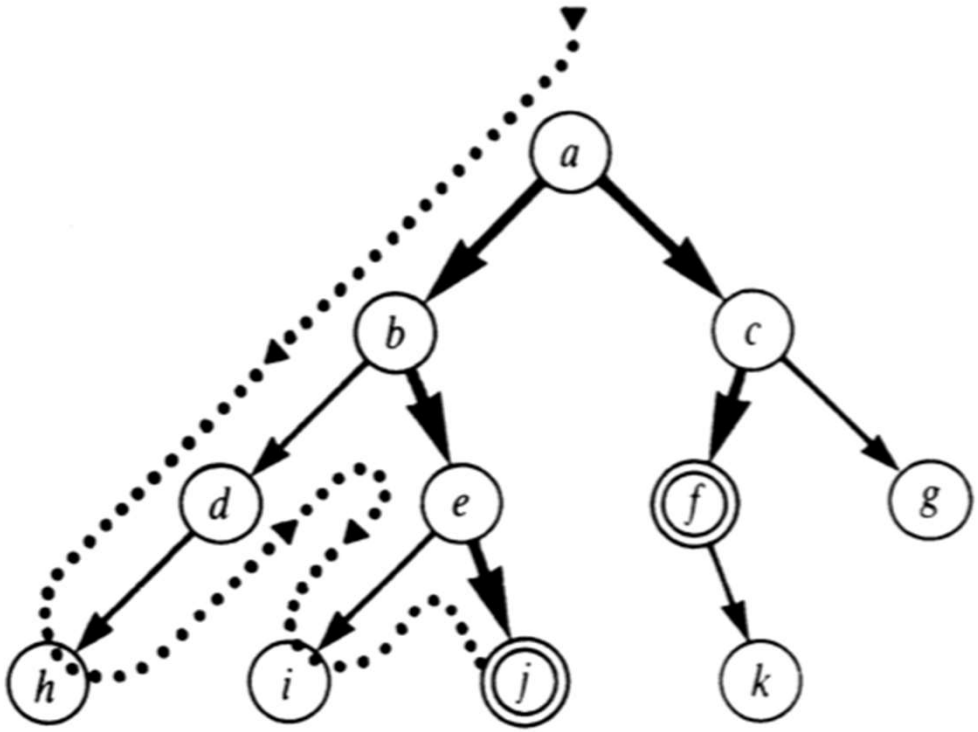
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

expanded

A S T Z F

Depth First Search (DFS)

Depth-first Search - DFS



Depth First Search

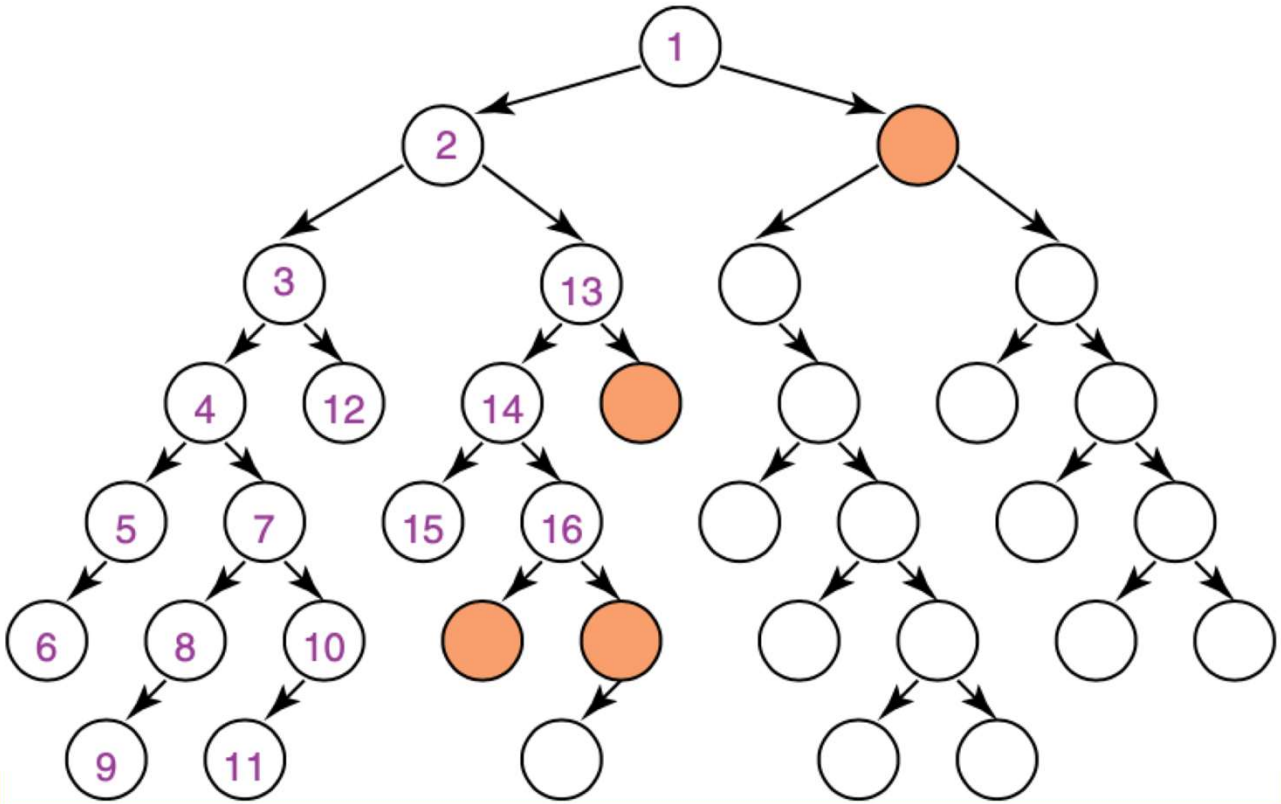
Expand one node at the deepest level reached so far

Implementation:

- Implement the frontier as a stack, i.e. insert newly generated states at the front of the open list (frontier)
- Can be implemented by recursive function calls

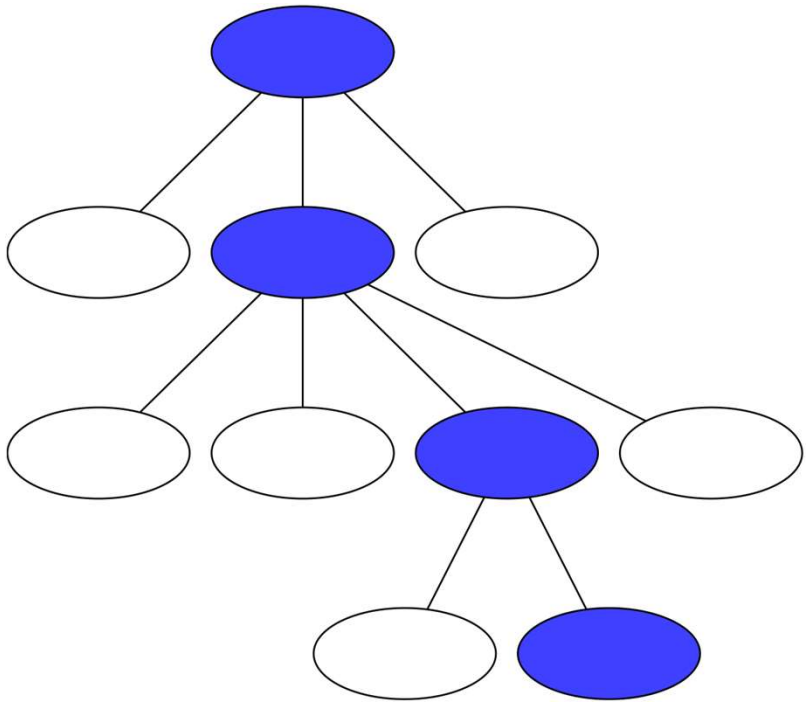
In depth-first search, like breadth-first, the order in which the paths are expanded does not depend on the goal.

Depth-first Search Example



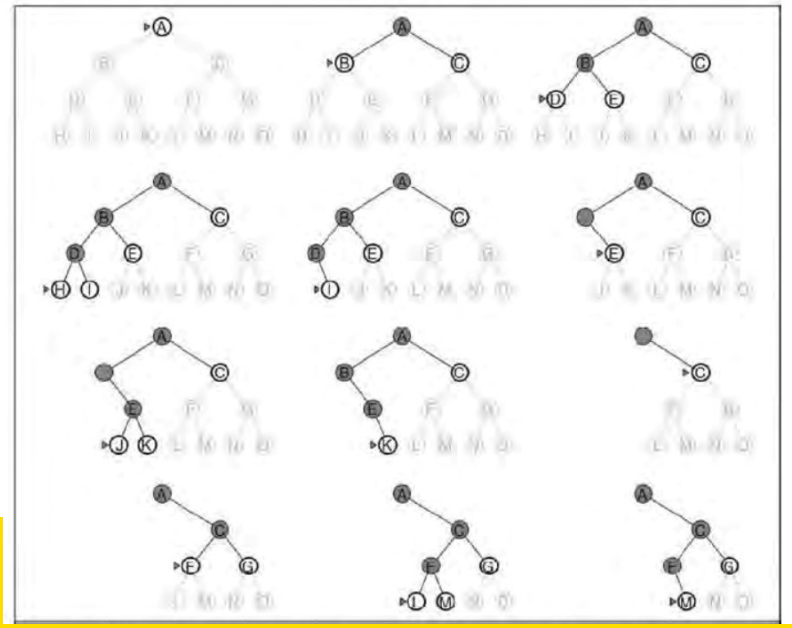
Depth First Search

- At any point depth-first search stores single path from root to leaf, together with any remaining unexpanded siblings of nodes along path
- Stop when node with goal state is expanded
- Include check that state has not already been explored **along a path** – **cycle checking**



Why DFS is memory efficient?

- Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored. Thus, giving us a time complexity of O(bm)



Tree-Search DFS vs. Graph-Search DFS

Tree-Search DFS

- We covered Tree-Search DFS in this course
- Can **be used** for iterative deepening dfs
- Checks new states against those on the **path** from the **root** to the **current node** to avoid infinite loops in finite state space.
- Does not avoid the proliferation of redundant paths.
- Both are non-optimal
- The space complexity can be O(bm)

Graph-Search DFS

- You probably covered Graph-Search in COMP2521
- Can **not** be used for iterative deepening dfs!
- Keeps a record of every visited, expanded node
- Both are non-optimal
- The space complexity is as bad as BFS: O(b^m)

Depth-Limited Search

Depth-Limited Search

- A depth-first search with a predetermined depth limit l.
- Nodes at depth l, are treated as if they have no successors

Depth-Limited Search

```
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 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
```

- The algorithm above does not check the current node against those on the path from the root to the current node.

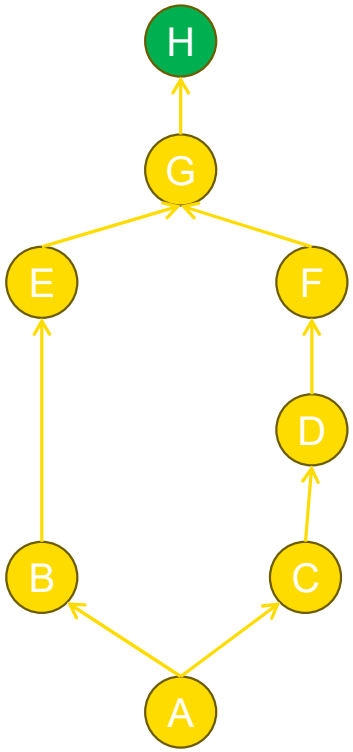
Depth-Limited Search

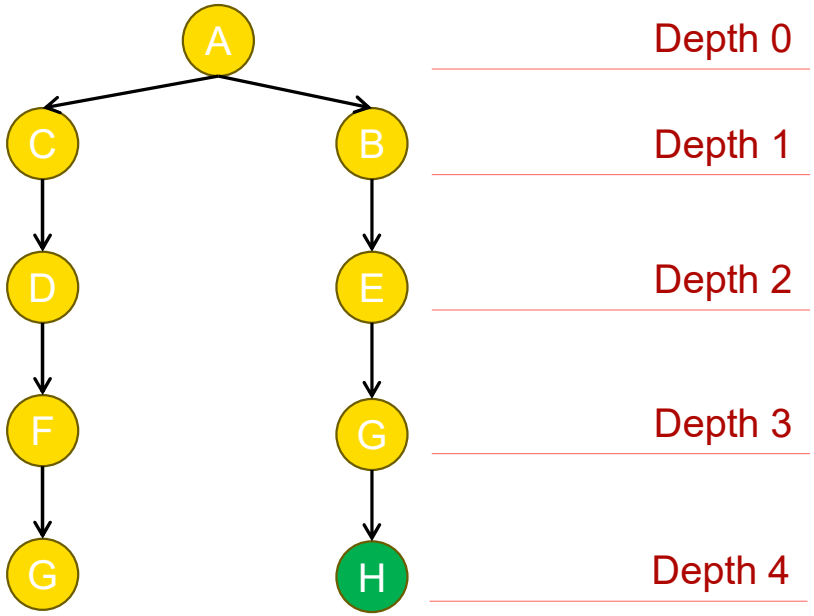
- Will it cause any issue if we do not check for a loop in depth limited search?

Depth-Limited Search

- Depth limited search can be implemented as a simple recursive algorithm.
- Notice that depth-limited search can be terminate with two kinds of failure: the standard failure value indicates no solution; the cutoff value indicates no solution *within* the depth limit
- The algorithm on previous slide does not check the current node against those on the path from the root to the current node.

How Depth Limited DFS finds a path to to goal with depth limit of 4





https://forms.office.com/r/bBb6bt2sv7

Iterative Deepening Depth-First Search (IDDFS)

Iterative Deepening Search

Depth-bounded search: hard to decide on a depth bound

Iterative deepening: Try all possible depth bounds in turn

Combines benefits of depth-first and breadth-first search

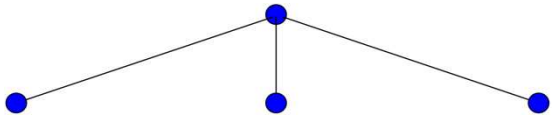
Iterative Deepening Search

Tries to combine the benefits of depth-first (low memory) and breadth-first (optimal and complete)

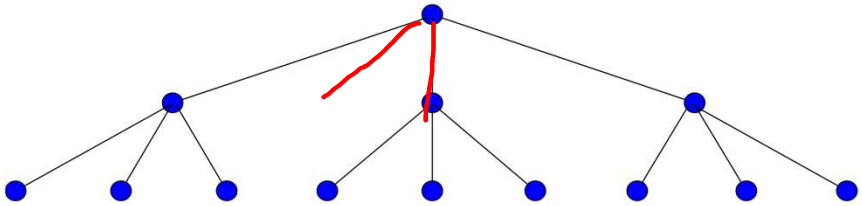
Does a series of depth-limited depth-first searches to depth 1, 2, 3, etc.

Early states will be expanded multiple times, but that might not matter too much because most of the nodes are near the leaves.

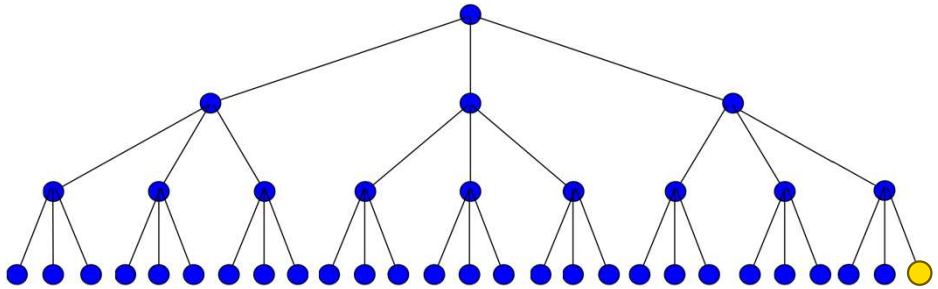
Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search



Properties of Iterative Deepening Search

Complete? Yes.

Time: $O(b^d)$

Space? $O(bd)$

Optimal? Yes, if step costs are identical.

In general, iterative deepening is the preferred search strategy for a large search space where depth of solution is not known

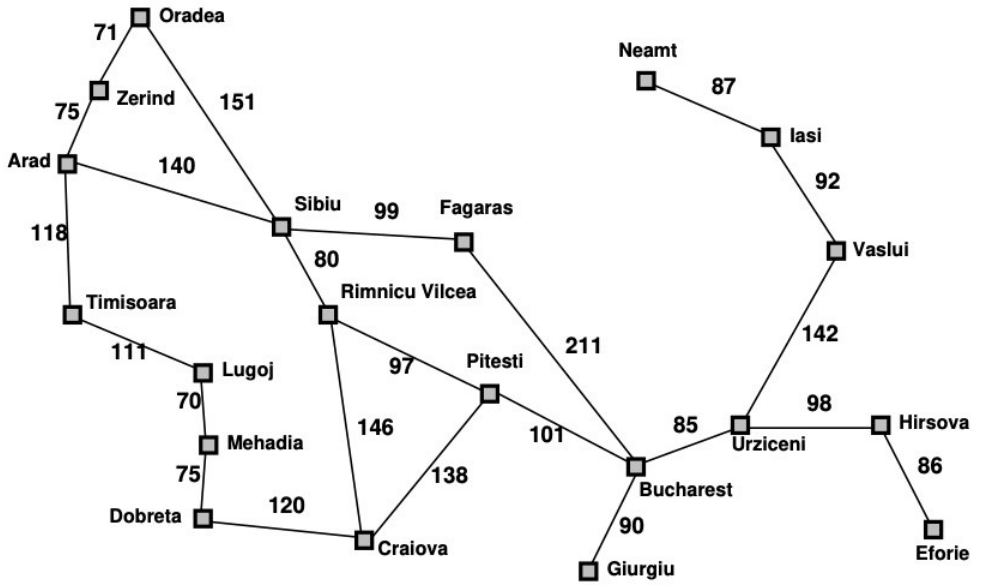
Exercise

This exercise uses the route-finding example with the Romanian map from Russell & Norvig

(Artificial Intelligence: A Modern Approach) as an example.

For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes take the first one by alphabetical ordering. For depth-first search use cycle checking along a path to avoid repeated states that may lead to infinite branches. Stop the search when the goal state is expanded.

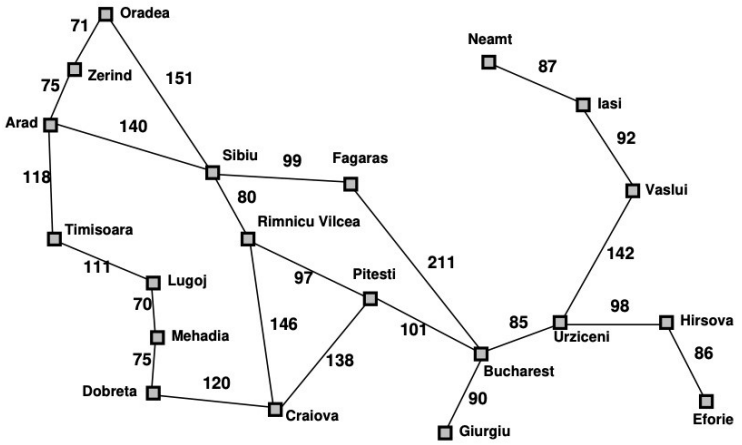
* Iterative Deepening
Depth First Search



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

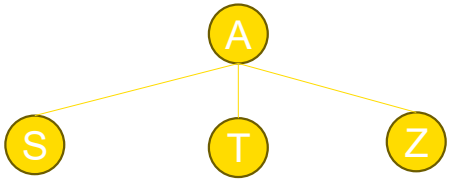
Exercise
IDDFS d=1

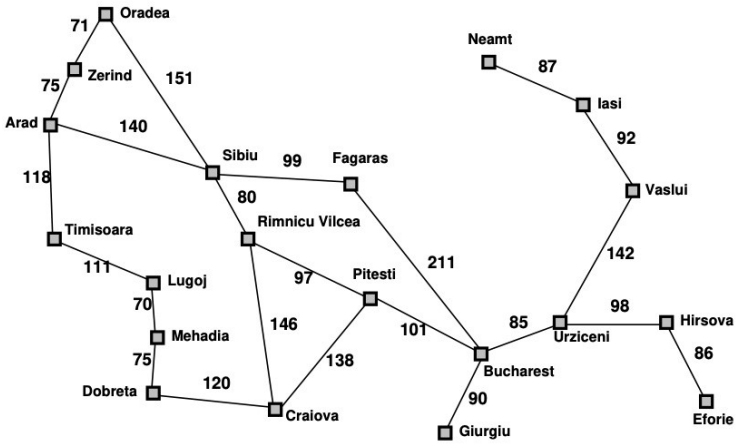
A



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

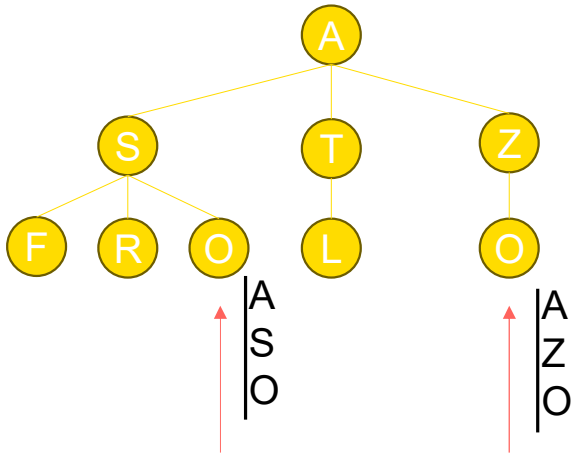
Exercise
IDDFS d=2

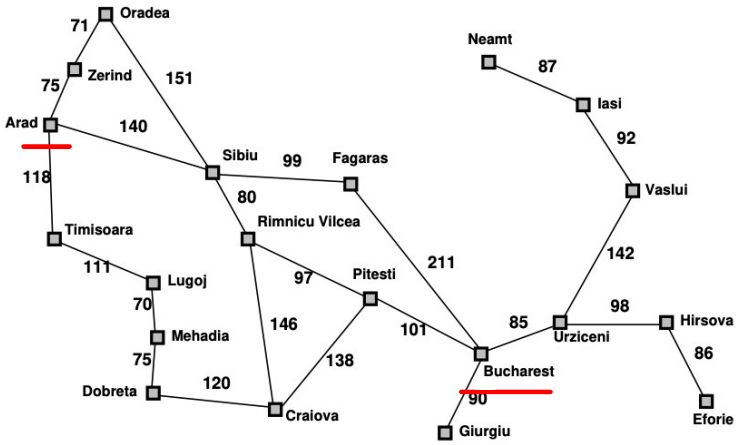




Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

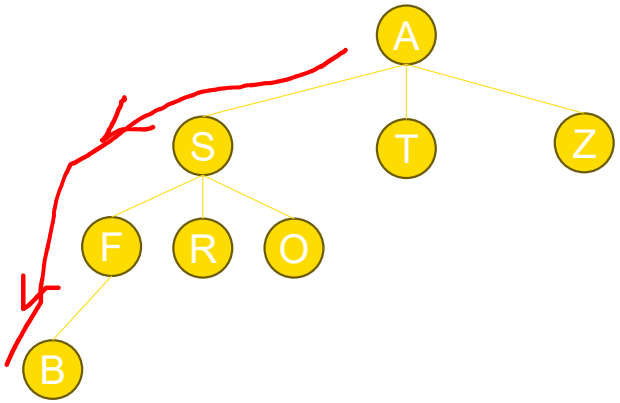
Exercise
IDDFS d=3

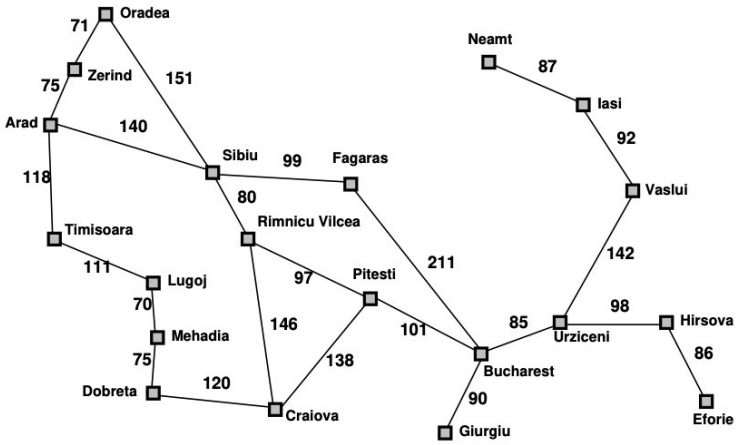




Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Exercise
IDDFS d=4



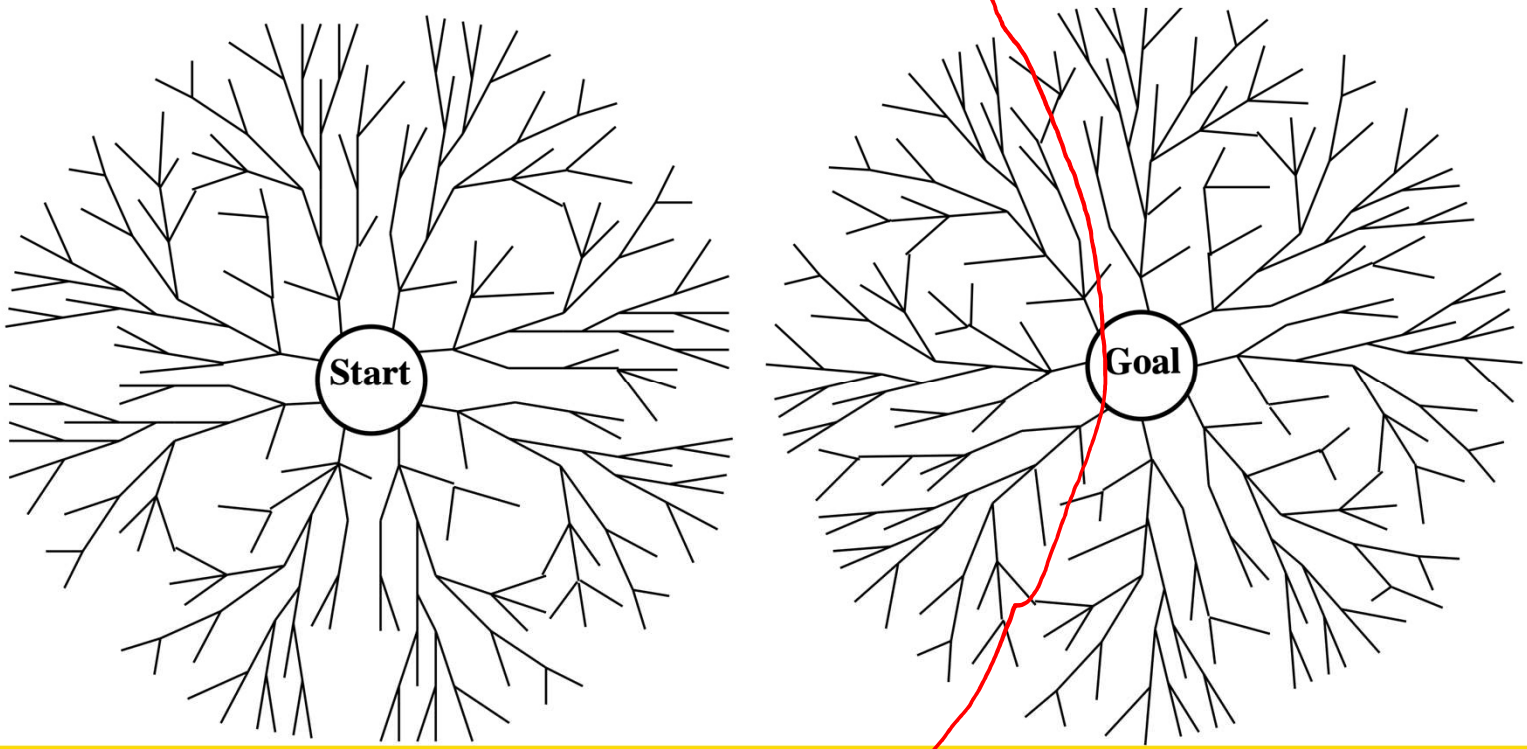


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Did you understand IDDFS and DFS?

Bidirectional Search

Bidirectional Search



Bidirectional Search

- Search both forward from the initial state and backward from the goal
  - stop when the two searches meet in the middle.
- Need efficient way to check if a new node appears in the other half of the search.
  - Complexity analysis assumes this can be done in constant time, using a hash table.
- Assume branching factor = b in both directions and that there is a solution at depth = d:
  - Then bidirectional search finds a solution in $O(2b^{d/2}) = O(b^{d/2})$ time steps.

Bidirectional Search Analysis

If solution exists at depth d then bidirectional search requires time

$$O(2b^{\frac{d}{2}}) = O(b^{\frac{d}{2}})$$

(assuming constant time checking of intersection)

To check for intersection must have all states from one of the searches in memory, therefore space complexity is $O(b^{\frac{d}{2}})$

Uniform Cost Search (UCS)

Uniform-Cost Search

Lowest-cost-first Search

Sometimes transitions have a cost

Cost of a **path** is the sum of the costs of its arcs:

$$cost(\langle n_0, \cdots, n_k \rangle) = \sum_{i=1}^{k} cost(\langle n_{i-1}, n_i \rangle)$$

An optimal solution has minimum cost

Delivery robot example:

- cost of arc may be resources (e.g., time, energy) required to execute action represented by the arc
- aim is to reach goal using least resources

Uniform-Cost Search

The simplest search method that is guaranteed to find a minimum cost path is **lowest-cost-first** search or **uniform-cost search**

- similar to breadth-first search, but instead of expanding path with least number of arcs, select path with lowest cost
- implemented by treating the frontier as a priority queue ordered by the cost function

$$cost(\langle n_0, \cdots, n_k \rangle) = \sum_{i=1}^{k} cost(\langle n_{i-1}, n_i \rangle)$$

Uniform-Cost Search for Delivery Robot

Edges are labelled with cost

- e.g. distance to travel

Sort queue by increasing cost of path to the node

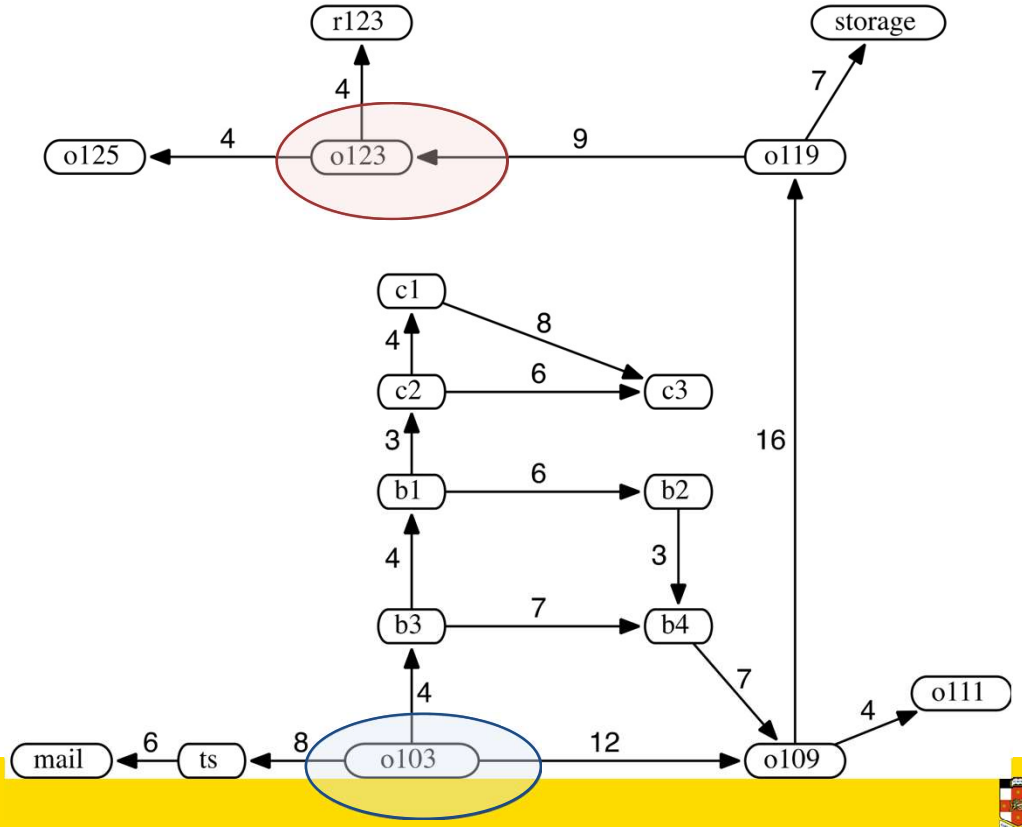
$[o103_0]$

$[b3_4, ts_8, o109_{12}]$

$[b1_8, ts_8, b4_{11}, o109_{12}]$

$[ts_8, c2_{11}, b4_{11}, o109_{12}, b2_{14}]$

$[c2_{11}, b4_{11}, o109_{12}, mail_{14}, b2_{14}]$



Uniform-Cost Search

- Expand root first, then expand least-cost unexpanded node
- Implementation with priority queue
  - insert nodes in order of increasing path cost - lowest path cost is $g(n)$.
- Reduces to breadth-first search when all actions have same cost
- Finds the cheapest goal provided path cost is monotonically increasing along each path (i.e. no negative-cost steps)

Properties of Uniform-Cost Search

Complete? Yes, if b is finite and if transition $cost \geq \epsilon$ with $\epsilon > 0$

Time? Worst case, $O(b^{[C^*/\epsilon]})$ where C^* = cost of the optimal solution
every transition costs at least ϵ
$\therefore$ cost per step is $\frac{C^*}{\epsilon}$

Space? $O(b^{[C^*/\epsilon]})$, $b^{[C^*/\epsilon]} = b^d$ if all step costs are equal

Optimal? Yes – nodes expanded in increasing order of lower path cost, $g(n)$

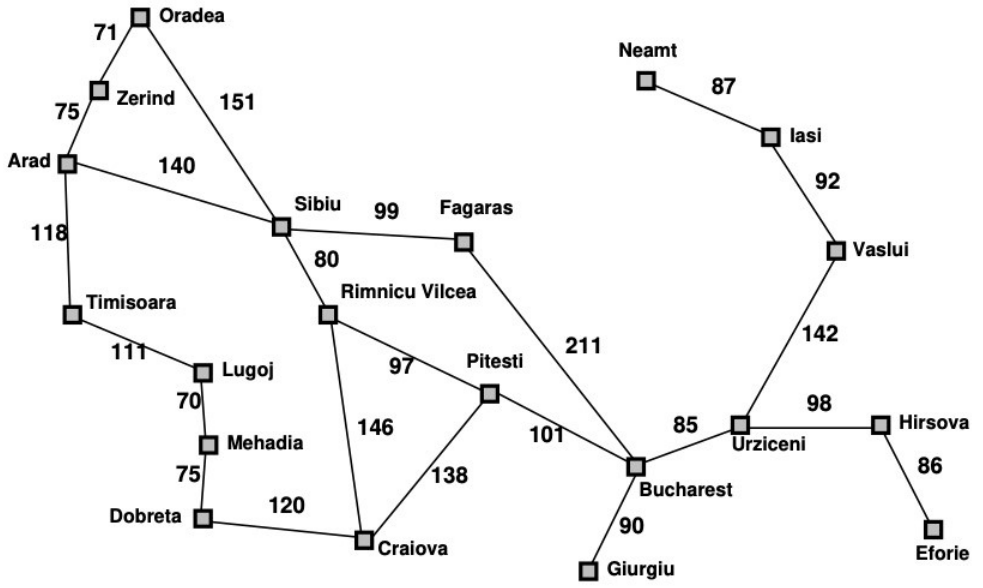
Exercise

This exercise uses the route-finding example with the Romanian map from Russell & Norvig

(Artificial Intelligence: A Modern Approach) as an example.

For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes take the first one by alphabetical ordering. For uniform0cost search include a check that nodes with the same state as previously expanded nodes are not added to the frontier. Stop the search when the goal state is expanded.

* Uniform Cost Search



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Complexity Results for Uninformed Search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$	$O(b^k)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bk)$	$O(bd)$
Complete?	Yes[1]	Yes[2]	No	No[4]	Yes[1]
Optimal ?	Yes[3]	Yes	No	No	Yes[3]

b = branching factor, d = depth of the shallowest solution,
m = maximum depth of the search tree, k = depth limit.

1 = complete if b is finite.

2 = complete if b is finite and step costs $\geq \varepsilon$ with $\varepsilon > 0$.

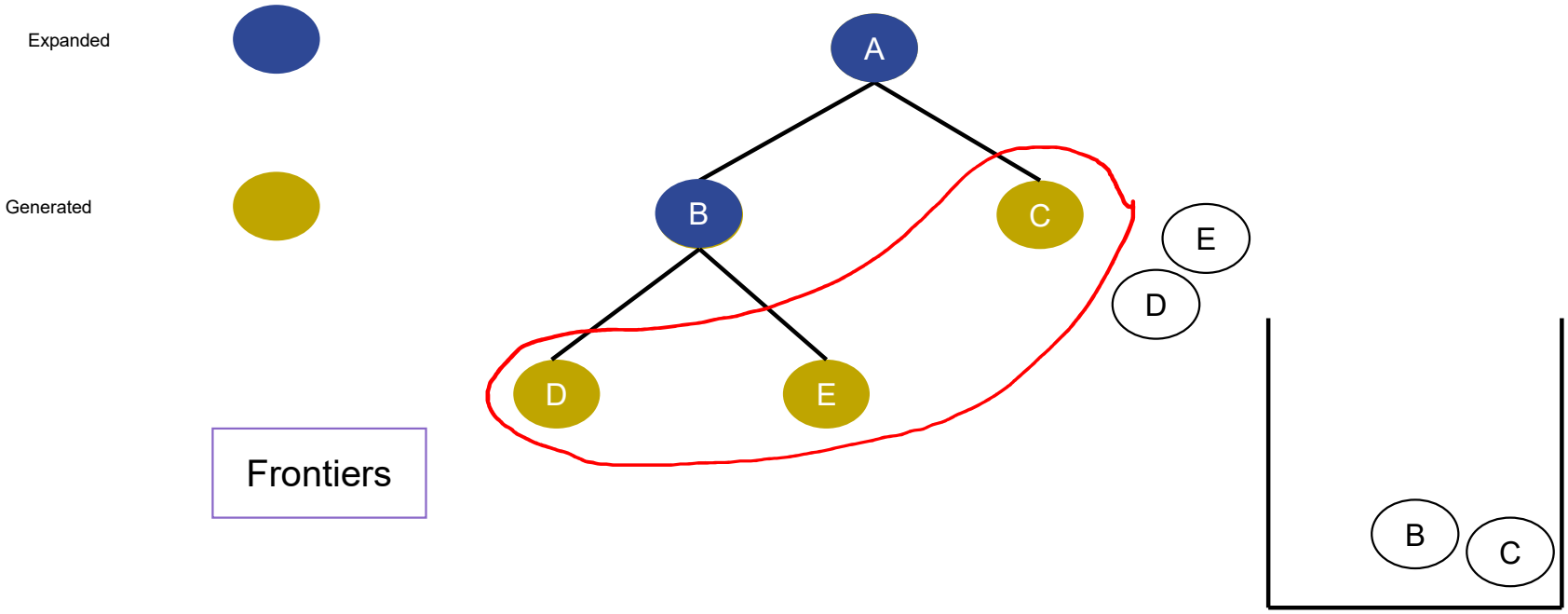
3 = optimal if actions all have the same cost.

4 = incomplete if the goal is not within the depth bound

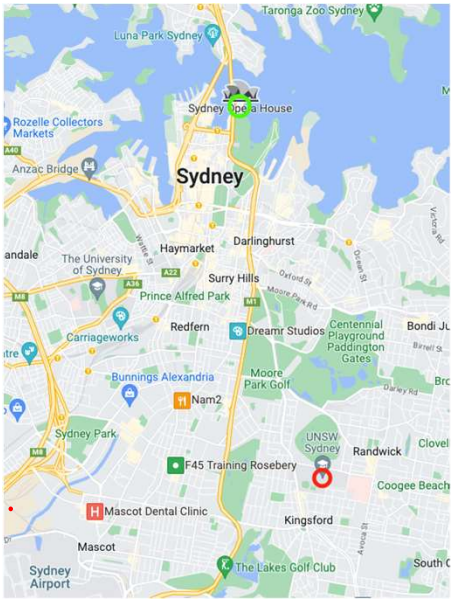
https://forms.office.com/r/bBb6bt2sv7

Solving Problems by Searching
Informed Search

Search strategies



Focus the search on the path that is most likely to lead to the goal state



Overview Informed Search

Uninformed Search	Background & Terminology
	Breath First Search (BFS)
	Depth First Search (DFS)
	Uniform Cost Search (UCS)
	Depth Limited Search
	Iterative Deepening Search (IDDFS)
	Bidirectional Search
Informed Searches	Heuristic
	Greedy Best-First Search
	A* Search

Search Strategies

General Search algorithm:

- `add initial state to the list`
- `repeat:`
  - `take node from the list`
  - `test if it is a goal state; if so, terminate`
  - `“expand” it, i.e. generate successor nodes and add them to the list`

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

Uniformed vs Informed Search

- Uninformed - keeps searching until it stumbles on goal
  - No domain knowledge
- Informed - searches in direction of best guess to goal
  - Uses domain knowledge

Informed (Heuristic) Search

- Informed search strategy
  - use problem-specific knowledge more efficiently than uninformed search
- Uninformed search algorithms have no information about problem other than its definition.
  - some can solve any solvable problem, none of them can do it efficiently
- Informed search algorithms can do well given guidance on where to look for solutions.
- Implemented using a priority queue to store frontier nodes

Search Strategies

BFS and DFS treat all new nodes the same way:

- BFS add all new nodes to the back of the list
- DFS add all new nodes to the front of the list

Best First Search uses an evaluation function $f()$ to order the nodes in the list

- Similar to uniform cost search

Informed or Heuristic:

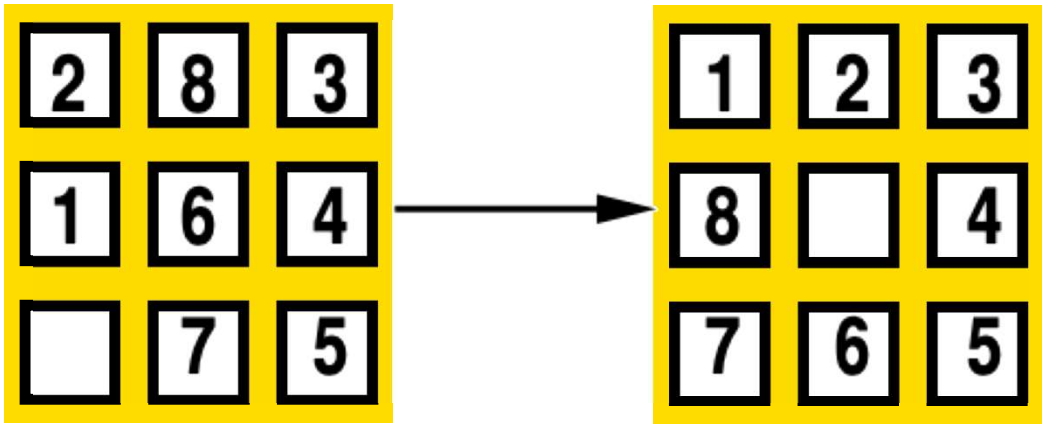
- Greedy Search $f(n) = h(n)$ (estimates cost from node n to goal)
- A* Search $f(n) = g(n) + h(n)$ (cost from start to n plus estimated cost to goal)

Heuristics

- Heuristics are “rules of thumb” for deciding which alternative is best
- Heuristic must ***underestimate*** actual cost to get from current node to goal
  - Called an admissible heuristic
- Denoted $h(n)$
  - $h(n) = 0$ when ever n is a goal node

Heuristics — Example

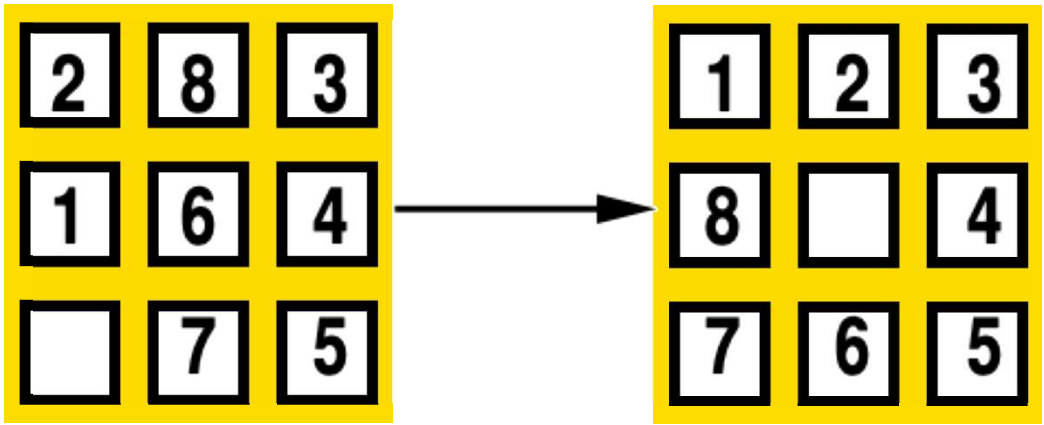
8-Puzzle — number of tiles out of place



$h(n) = 5$

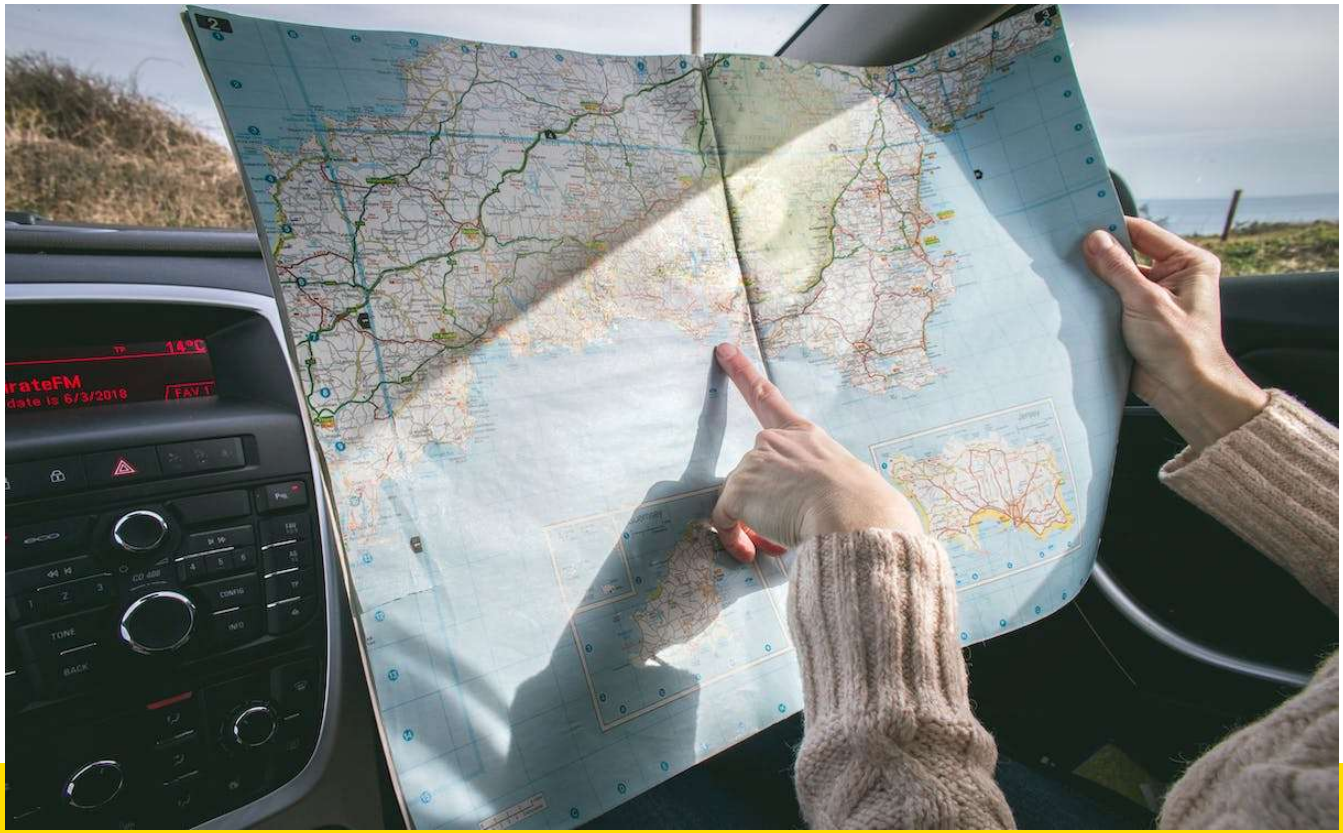
Heuristics — Example

8-Puzzle — Manhattan distance (distance tile is out of place)



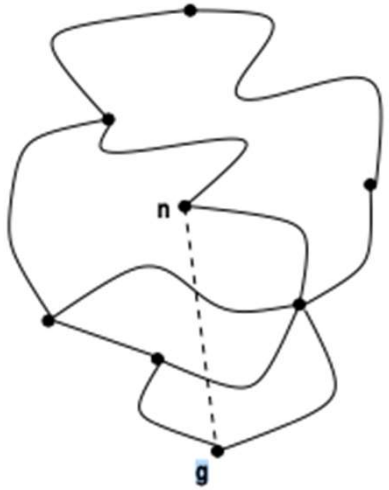
$h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6$

Heuristics — Example



Heuristics — Example

Another common heuristic is the straight-line distance ("as the crow flies") from node to goal



Therefore $h(n) = \textit{distance from } n \textit{ to } g$

Example Heuristic Functions

If nodes are points on a Euclidean plane and cost is distance, $h(n)$ can be straight-line distance from n to closest goal.

If nodes are locations and cost is time,
can use distance to goal divided by maximum speed.

- If goal is to collect a bunch of coins and not run out of fuel,
  cost is an estimate of how many steps to collect rest of the coins,
  refuel when necessary, and return to goal.

Heuristic function can be found by simplifying calculation of true cost

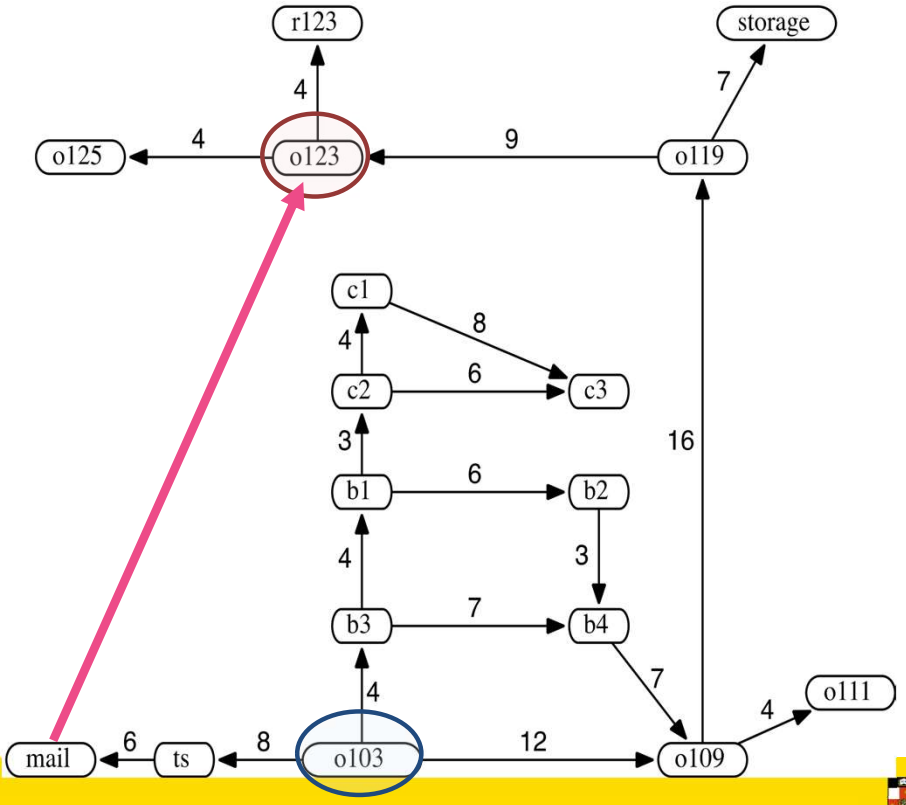
Delivery Robot Heuristic Function

Use straight-line distance as heuristic, and assume these values:

$$
\begin{array}{lll}
h(mail) = 26 & h(ts) = 23 & h(o103) = 21 \\
h(o109) = 24 & h(o111) = 27 & h(o119) = 11 \\
h(o123) = 4 & h(o125) = 6 & h(r123) = 0 \\
h(b1) = 13 & h(b2) = 15 & h(b3) = 17 \\
h(b4) = 18 & h(c1) = 6 & h(c2) = 10 \\
h(c3) = 12 & h(storage) = 12 &
\end{array}
$$

$h(loc)$ = distance from loc to goal

Heuristic function can be extended to paths by making heuristic value of path equal to heuristic value of node at the end of the path: $h(\langle n_0, \cdots, n_k \rangle) = h(n_k)$.



Exercise

What are some other heuristics that you can think of?

Greedy Best-First Search

Greedy Best-First Search

Always select node closest to goal according to heuristic function

$h(n)$ is estimated cost to goal

- $h(n) = 0$ if n is a goal state

Frontier is a **priority queue** ordered by h.

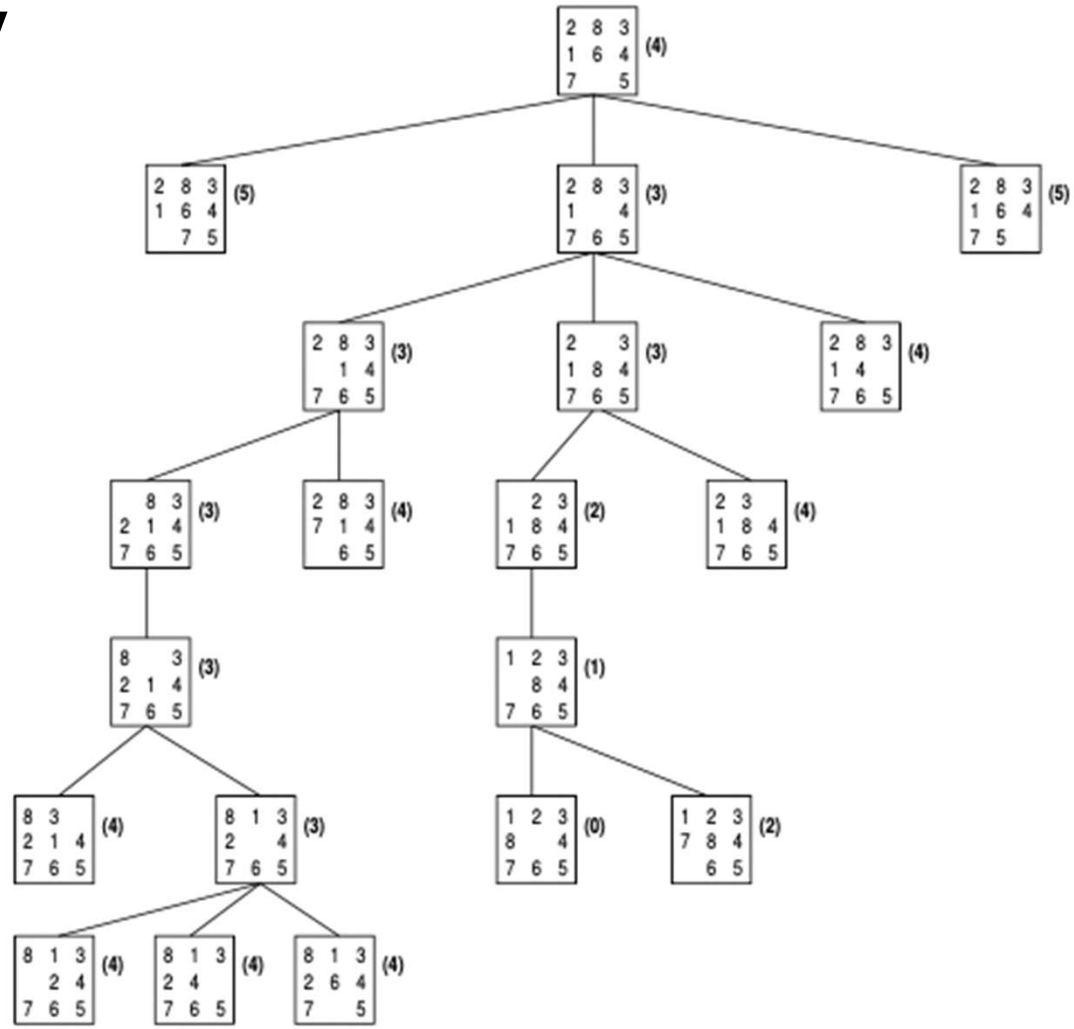
"Greedy" algorithm takes "best" node first.

$f(n) = h(n)$

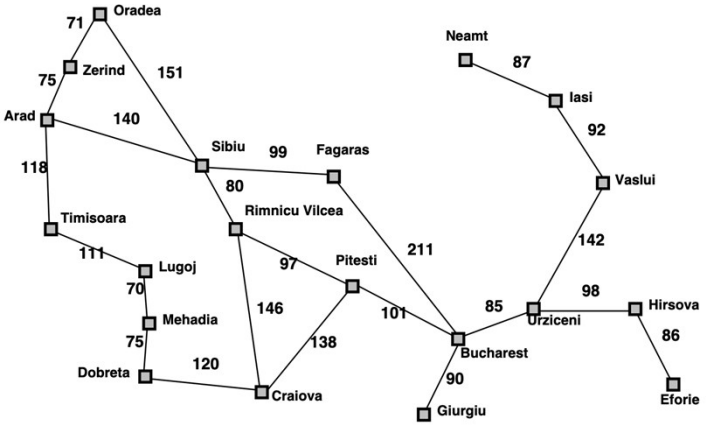
Like depth-first search, except pick next node by $h(n)$

Examples of Greedy Best-First Search

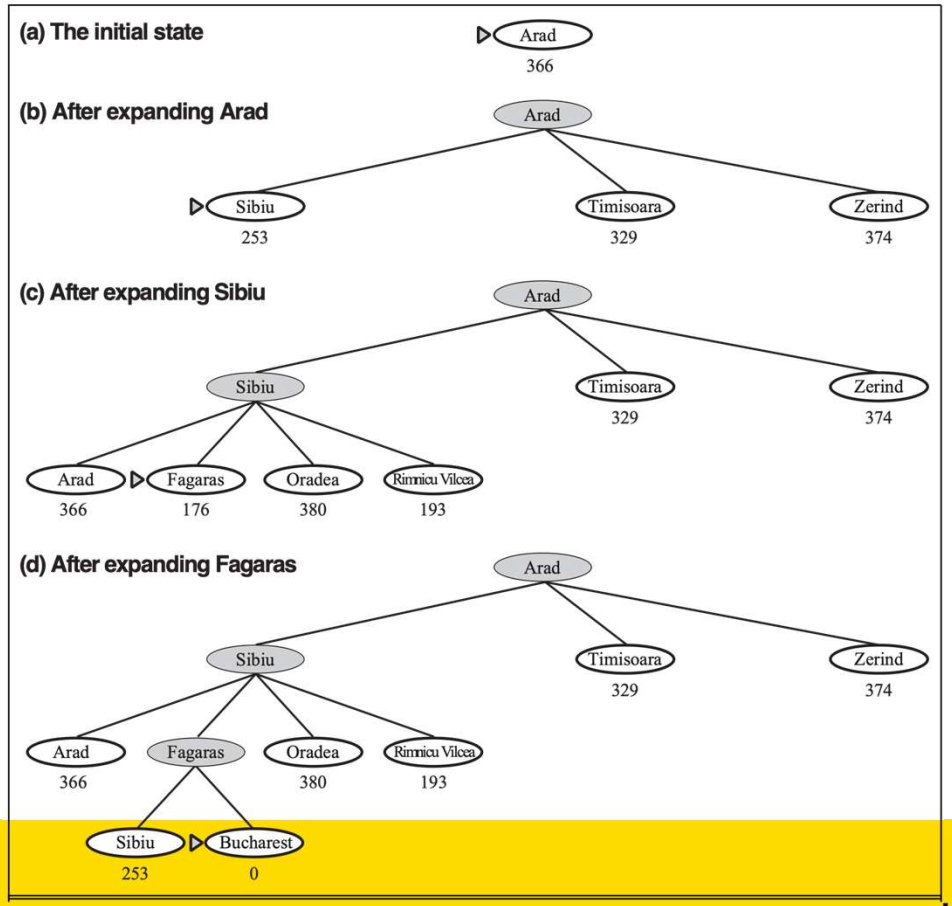
h is number of tiles out of place



Examples of Greedy Best-First Search



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



- Stages in a greedy best-first tree search for route from Arad to Bucharest with the straight-line distance heuristic.
- Note that **straight-line distances are less than actual distances** in map.

UNSW SYDNEY

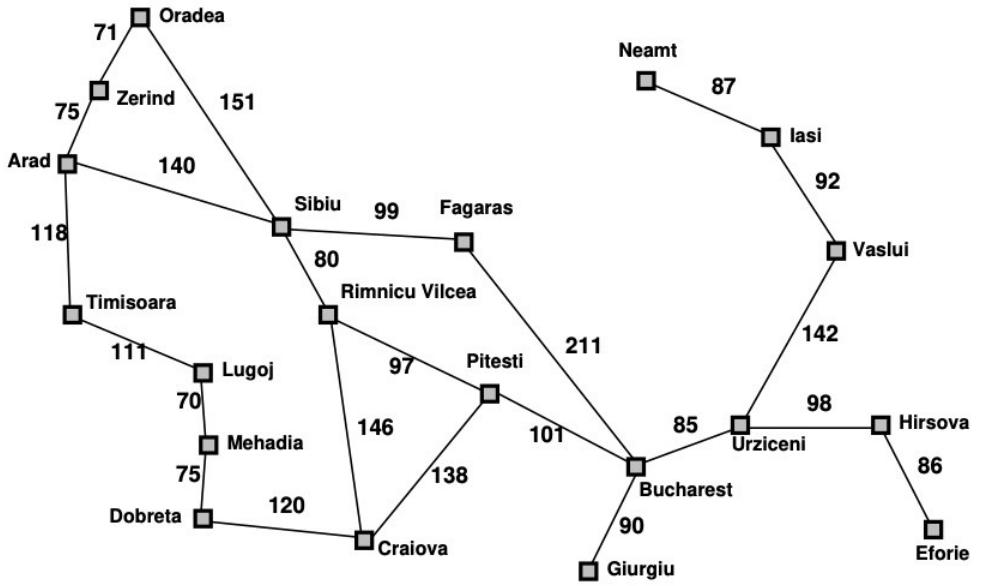
Exercise

This exercise uses the route-finding example with the Romanian map from Russell & Norvig

(Artificial Intelligence: A Modern Approach) as an example.

For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes take the first one by alphabetical ordering. Stop the search when the goal state is expanded.

* Greedy best-first search



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of Greedy Best-First Search

Complete: No. Can get stuck in loops.
(Complete in finite space with repeated-state checking)

Time: $O(b^m)$, where m is the maximum depth in search space.

Space: $O(b^m)$ (retains all nodes in memory)

Optimal: No.

Greedy Search is not efficient

However, a good heuristic can reduce time and memory costs substantially.

A* Search

Uniform-Cost Search

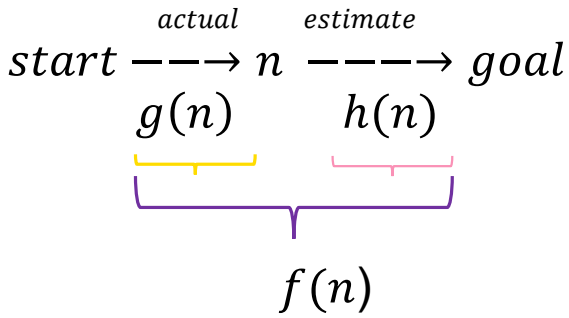
Greedy Search

A* Search

A* Search

Use both cost of path generated and estimate to goal to order nodes on the frontier

- $g(n)$ = cost of path from start to n
- $h(n)$ = estimate from n to goal

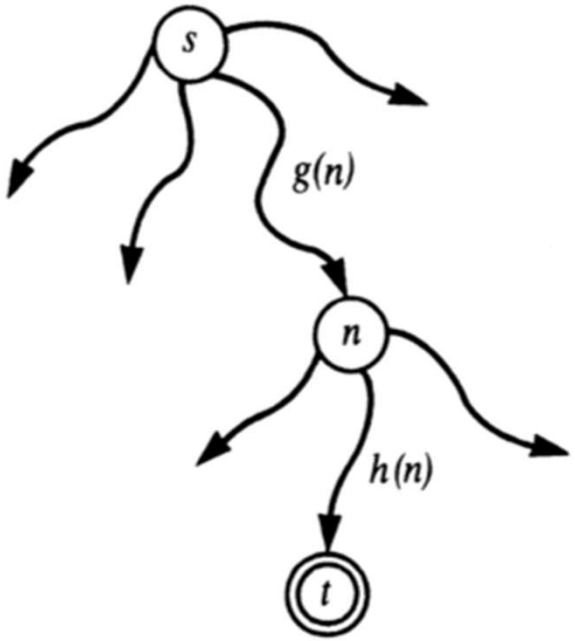


- Order priority queue using function $f(n) = g(n) + h(n)$
- $f(n)$ is the estimated cost of the cheapest solution extending this path

A* Search

- Combines uniform-cost search and greedy search
- Greedy Search minimises $h(n)$
  - efficient but not optimal or complete
- Uniform Cost Search minimises $g(n)$
  - optimal and complete but not efficient

Heuristic Function

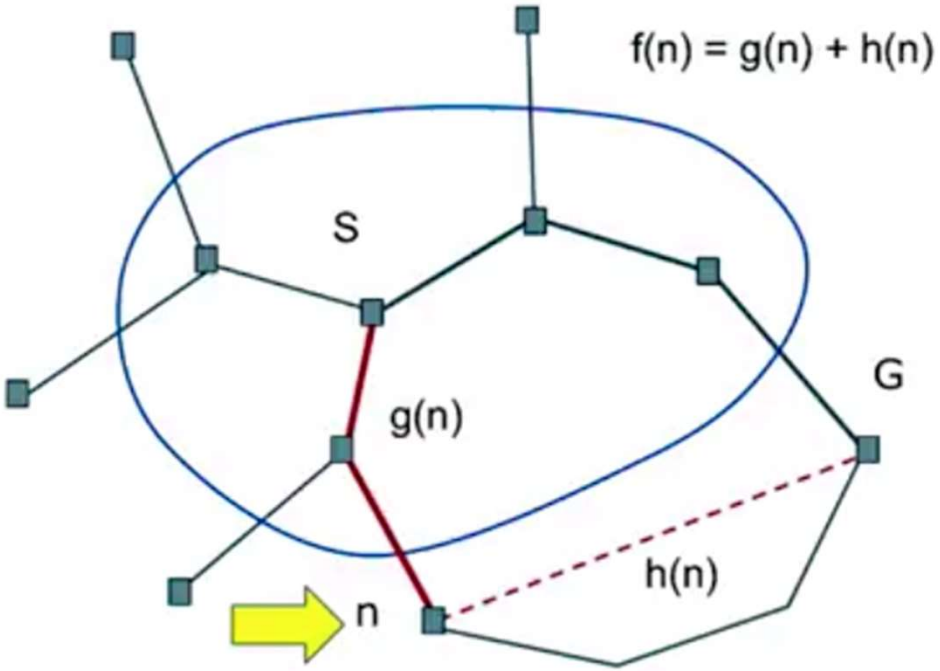


Heuristic estimate $f(n)$ = cost of the cheapest path from s to t via n: $f(n) = g(n) + h(n)$

g(n) is the cost the path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.

A* Search



S = start

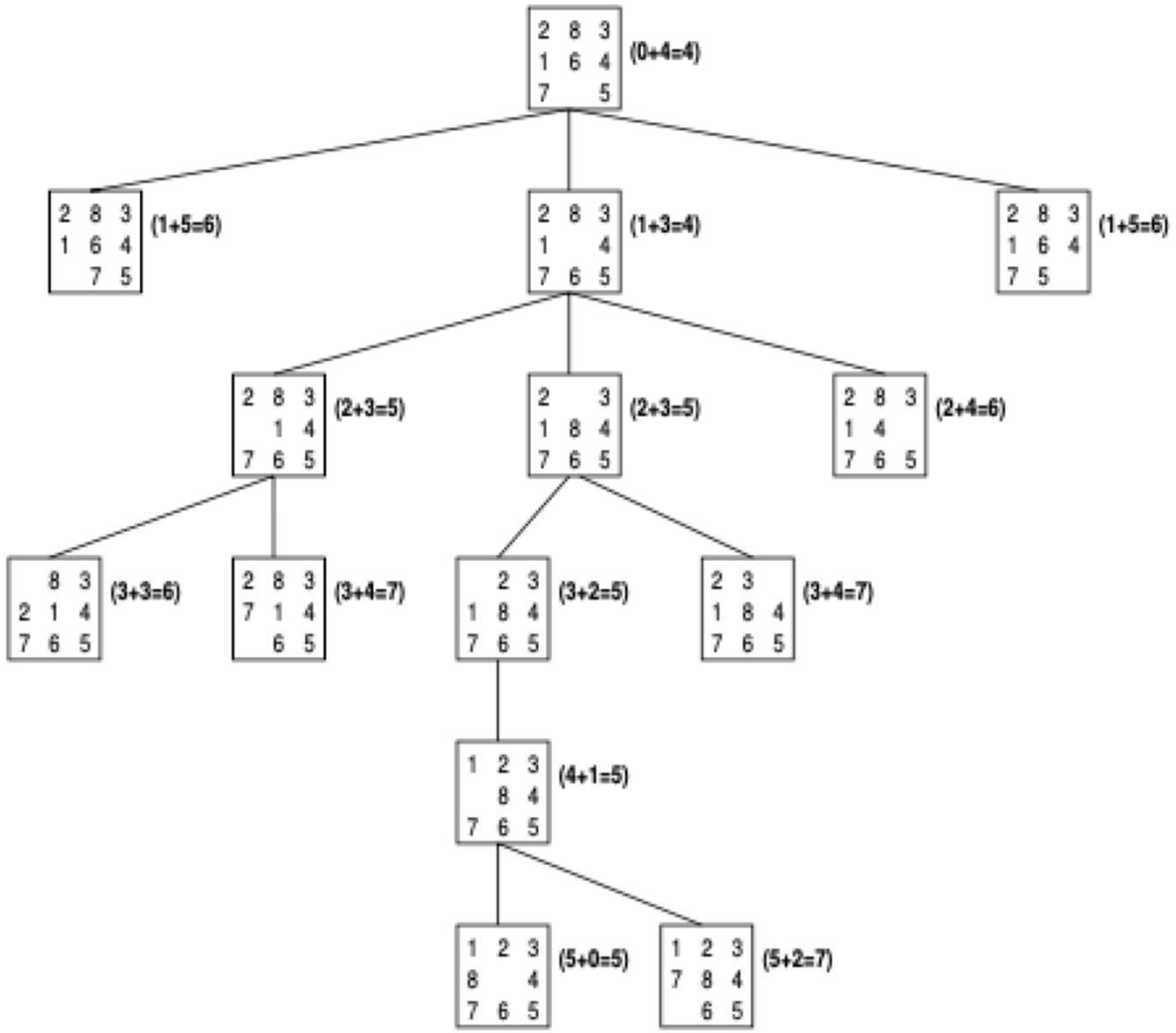
G = Goal

n = current node

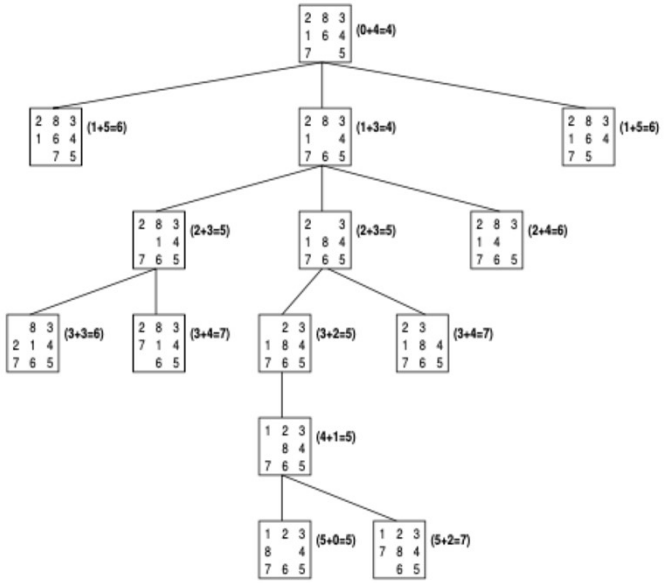
g(n) = actual cost from **S** to **n**

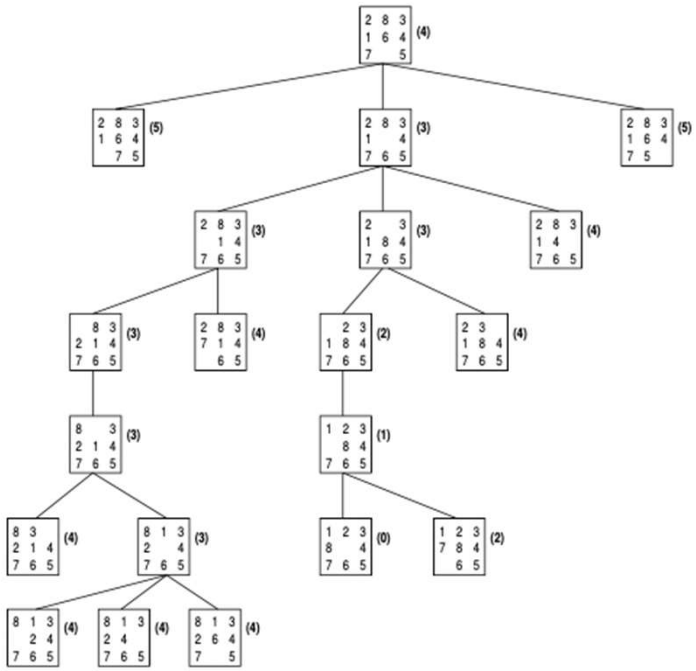
h(n) = estimated distance from **n** to **G**

A* Search



A* vs Greedy

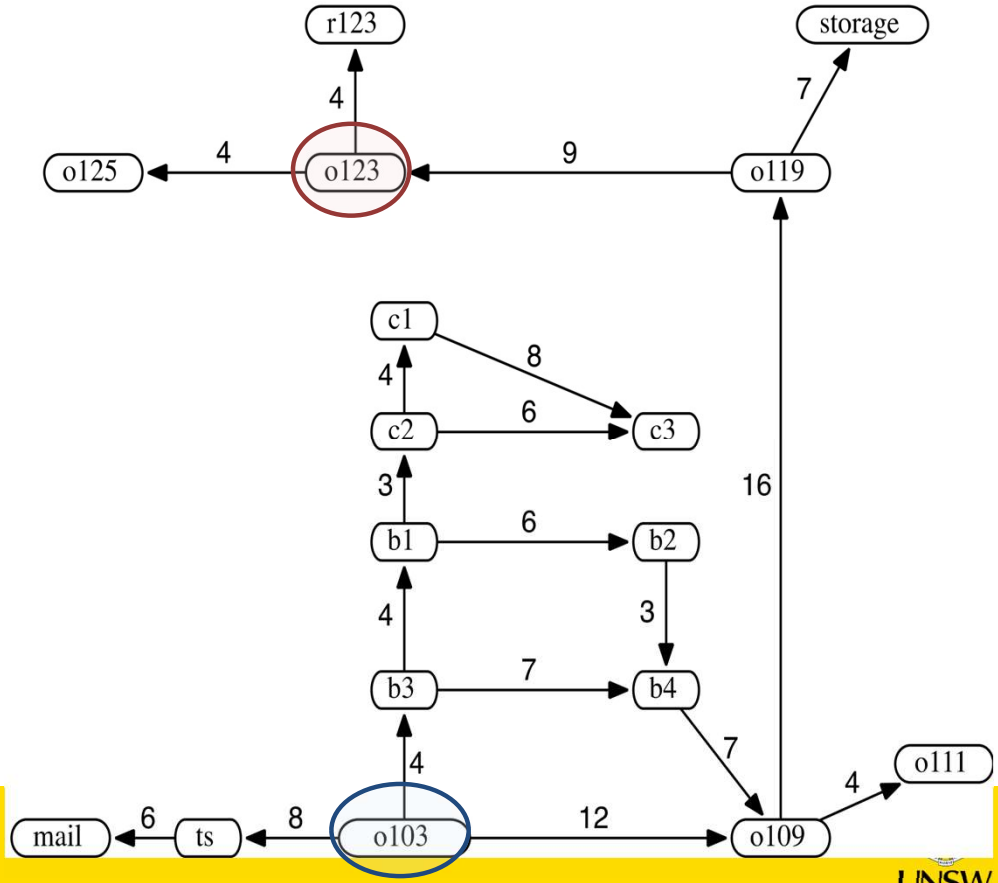

(0+4=4)
(1+5=6)
(1+3=4)
(1+5=6)
(2+3=5)
(2+3=5)
(2+4=6)
(3+3=6)
(3+4=7)
(3+2=5)
(3+4=7)
(4+1=5)
(5+0=5)
(5+2=7)


(4)
(5)
(3)
(5)
(3)
(3)
(4)
(3)
(4)
(2)
(4)
(3)
(1)
(4)
(3)
(0)
(2)
(4)
(4)
(4)

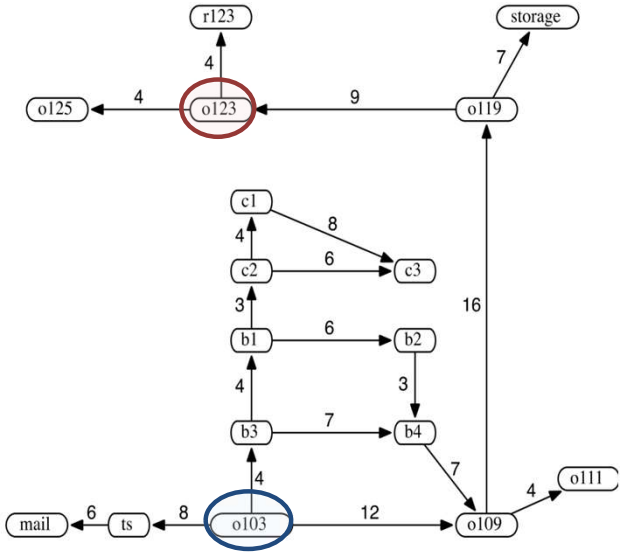
Delivery Robot Heuristic Function

Use straight-line distance as heuristic, and assume these values:

$$
\begin{array}{lll}
h(mail) = 26 & h(ts) = 23 & h(o103) = 21 \\
h(o109) = 24 & h(o111) = 27 & h(o119) = 11 \\
h(o123) = 4 & h(o125) = 6 & h(r123) = 0 \\
h(b1) = 13 & h(b2) = 15 & h(b3) = 17 \\
h(b4) = 18 & h(c1) = 6 & h(c2) = 10 \\
h(c3) = 12 & h(storage) = 12 &
\end{array}
$$



A* Search - The Delivery Robot



$h(mail) = 26$ $h(ts) = 23$ $h(o103) = 21$
$h(o109) = 24$ $h(o111) = 27$ $h(o119) = 11$
$h(o123) = 4$ $h(o125) = 6$ $h(r123) = 0$
$h(b1) = 13$ $h(b2) = 15$ $h(b3) = 17$
$h(b4) = 18$ $h(c1) = 6$ $h(c2) = 10$
$h(c3) = 12$ $h(storage) = 12$

1. [$o103_{21}$]
2. [$b3_{21}$, ts_{31}, $o109_{36}$]
3. [$b1_{21}$, $b4_{29}$, ts_{31}, $o109_{36}$]
4. [$c2_{21}$, $b2_{29}$, $b4_{29}$, ts_{31}, $o109_{36}$]
5. [$c1_{21}$, $b2_{29}$, $b4_{29}$, $c3_{29}$, ts_{31}, $o109_{36}$]
6. [$b2_{29}$, $b4_{29}$, $c3_{29}$, ts_{31}, $c3_{35}$, $o109_{36}$]
7. [$b4_{29}$, ts_{31}, $c3_{35}$, $o109_{36}$]
8. [ts_{31}, $c3_{35}$, $b4_{35}$, $o109_{36}$, $o109_{42}$]

…..

$$h(o103) = 21$$

$$f(\langle o103, b3\rangle) = g(\langle o103, b3\rangle) + h(b3) = 4 + 17 = 21$$

- Lowest-cost path is eventually found.
- Forced to try many different paths, because some temporarily seem to have the lowest cost.
- Still does better than lowest-cost-first search and greedy best-first search.

UNSW SYDNEY

Optimality of A*

- Heuristic h is said to be admissible if

  $\forall n h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n to goal

- If h is admissible then $f(n)$ never overestimates the actual cost of the best solution through n.

  $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost to n and $h(n)$ is an underestimate

- Example: h = *straight line distance* is admissible because the shortest path between any two points is a line.

- A^* is optimal if h is admissible.

- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

Optimality of A* Search

Complete: Yes, unless infinitely many nodes with $f \leq$ *cost of solution*

Time: Exponential in *relative error in* $h \times$ *length of solution*

Space: Keeps all expanded nodes is memory

Optimal: Yes (assuming h is admissible).

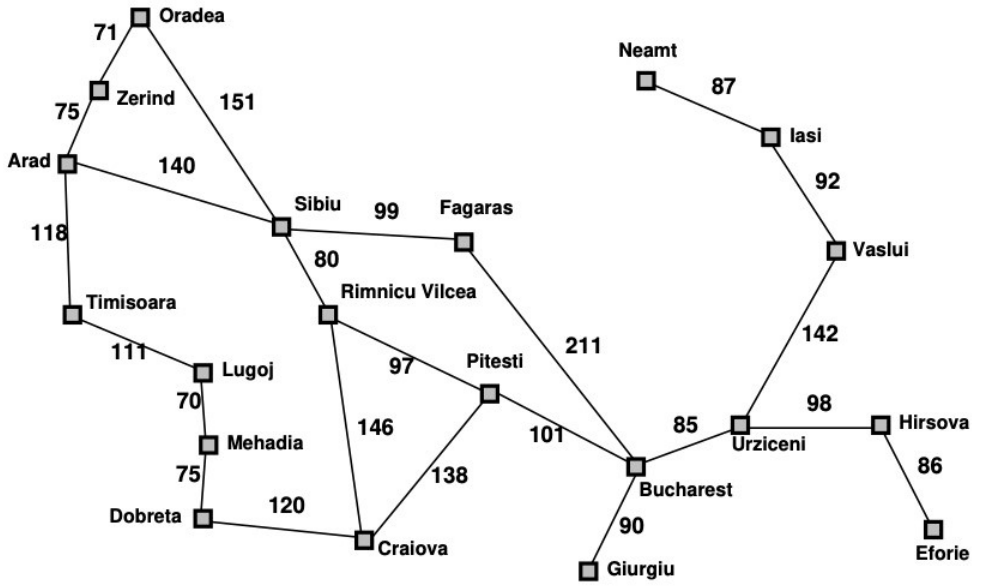
Exercise

This exercise uses the route-finding example with the Romanian map from Russell & Norvig

(Artificial Intelligence: A Modern Approach) as an example.

For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes take the first one by alphabetical ordering. For uniform0cost search include a check that nodes with the same state as previously expanded nodes are not added to the frontier. Stop the search when the goal state is expanded.

- A*



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Did you understand A*?

homework

What sort of search will greedy search emulate if we run it with:

- $h(n) = -g(n)$?
- $h(n) = g(n)$
- $h(n) = number\ of\ steps\ from\ intial\ state\ to\ node\ n$?

Iterative Deepening A* Search

Iterative Deepening A* Search

Iterative Deepening A* is a low-memory variant of A* that performs a series of depth-first searches but cuts off each search when the $_f$ exceeds current threshold, initially $f(start)$.

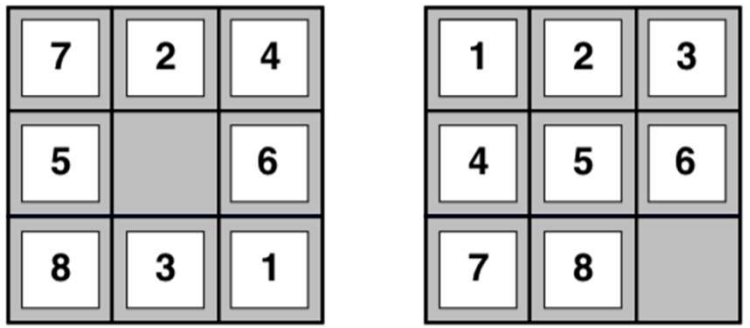
The threshold is increased with each successive search.

Homework

If $h(n)$ is an underestimate of the actual cost, then it is called admissible heuristic. An A* search with an admissible heuristic always finds an optimal path. Can you show why this is always the case?

Examples of Admissible Heuristics

$h_1(n)$ = total number of misplaced tiles
$h_2(n)$ = total Manhattan distance = ∑distance from goal position



$h_1(Start) = 6$
$h_2(Start) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$

More on Heuristic

How to Find Heuristic Functions ?

Admissible heuristics can often be derived from the exact solution cost of a simplified or “relaxed” version of the problem. (i.e. with some of the constraints weakened or removed)

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Dominance

if $h_2(n) \geq h_1(n)$ for all n (both admissible)

- h_2 dominates h_1 and is better for search.
- Try make the heuristic h as large as possible, without exceeding h^*.

typical search costs:

14-puzzle	IDS	$= 3{,}473{,}941$ nodes
	$A^*(h_1)$	$= 539$ nodes
	$A^*(h_2)$	$= 113$ nodes
24-puzzle	IDS	$\approx 54 \times 10^9$ nodes
	$A^*(h_1)$	$= 39{,}135$ nodes
	$A^*(h_2)$	$= 1{,}641$ nodes

Summary of Informed Search

Informed search makes use of problem-specific knowledge to guide progress of search

This can lead to a significant improvement in performance

Heuristics can be applied to reduce search cost.

Greedy Search tries to minimise cost from current node *n* to the goal.

Summary of Informed Search

A* combines the advantages of Uniform-Cost Search and Greedy Search

A* is complete, optimal and optimally efficient among all optimal search algorithms.

Memory usage is still a concern for A*. IDA* is a low-memory variant.

Much research has gone into admissible heuristics

- Even on the automatic generation of admissible heuristics

How to solve any problem using search?

How to solve any problem using search?

1. Turn the states, actions and transition function into a machine-readable format

2. Choose a search strategy according to the problem

Updates

Update	Page	Thanks to
Children of Sibiu are Fagaras, Oradea and Rimincu Vlicea. The way asked in the question was to put them in alphabetical order. It means the order should have been **R, O, F**. However, on the original slide, they were placed in a wrong order which is not fixed.	Page 27	Amal Ramadan Alshammari

Anonymous feedback for Armin’s Lecture

https://forms.office.com/r/eFhvP4dz0y