Introduction to

Artificial Intelligence (AI)

Computer Science cpsc502, Lecture 2

Sep, 13, 2011

R&Rsys we'll cover in this course

Problem

Environment

Problem		Deterministic	Stochastic
Static	Constraint Satisfaction	*Vars + Constraints* Arc Consistency Search	aka Bayesian Networks
	Query	*Logics* → Propositional → First Order → .... Search	*Belief Nets* Var. Elimination Approx. Inference Temporal. Inference
Sequential	Planning	*STRIPS* actions precs effects Search	aka influence diagrams *Decision Nets* Var. Elimination *Markov Processes* Value Iteration

Representation

Reasoning Technique

Today Sept 13

- Uninformed Search
- Informed Search
- ….
- ….

Simple Planning Agent

Deterministic, goal-driven agent

- Agent is given a goal (subset of possible states)
- Environment changes only when the agent acts
- Agent perfectly knows:
  - what actions can be applied in any given state
  - the state it is going to end up in when an action is applied in a given state

- The sequence of actions and their appropriate ordering is the solution

Three examples

1. Solving an 8-puzzle

2. Vacuum cleaner world

3. A delivery robot planning the route it will take in a bldg. to get from one room to another *(see textbook)*

Example 2: 8-Puzzle?

\# of states $= 9! \sim 360 * 10^3$

Possible start state

Goal state

the robot can be in two possible locations

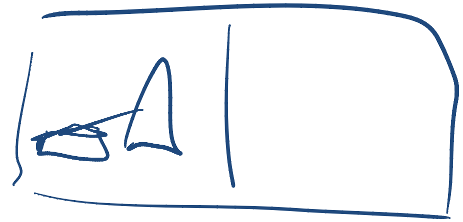
Example: vacuum world

3 Features

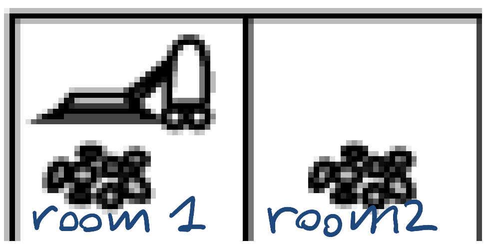
$\boxed{loc}$ has two values $\{r_1, r_2\}$

$\boxed{clean\text{-}r_1}$ " " $\{true, false\}$

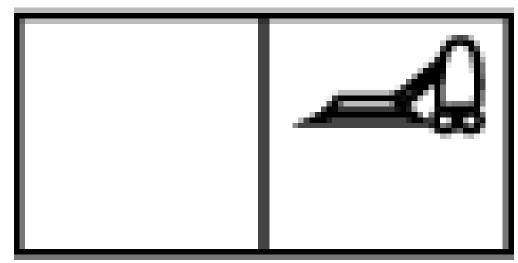
$\boxed{clean\text{-}r_2}$ " " $\{true, false\}$



r1 r2



Possible start state



Goal state(s)

# of states

$2 * 2^2$

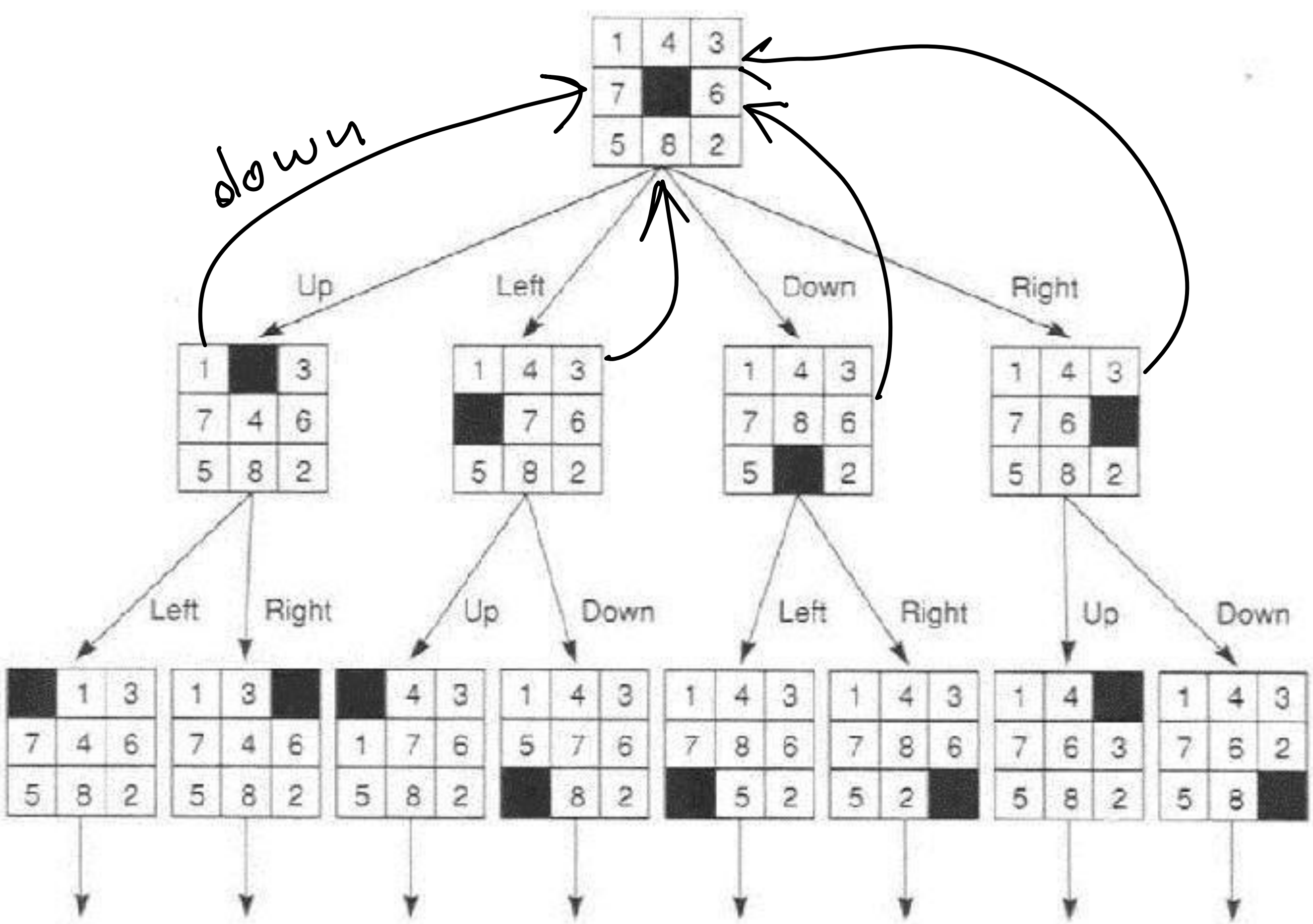
given K rooms

GENERALIZE

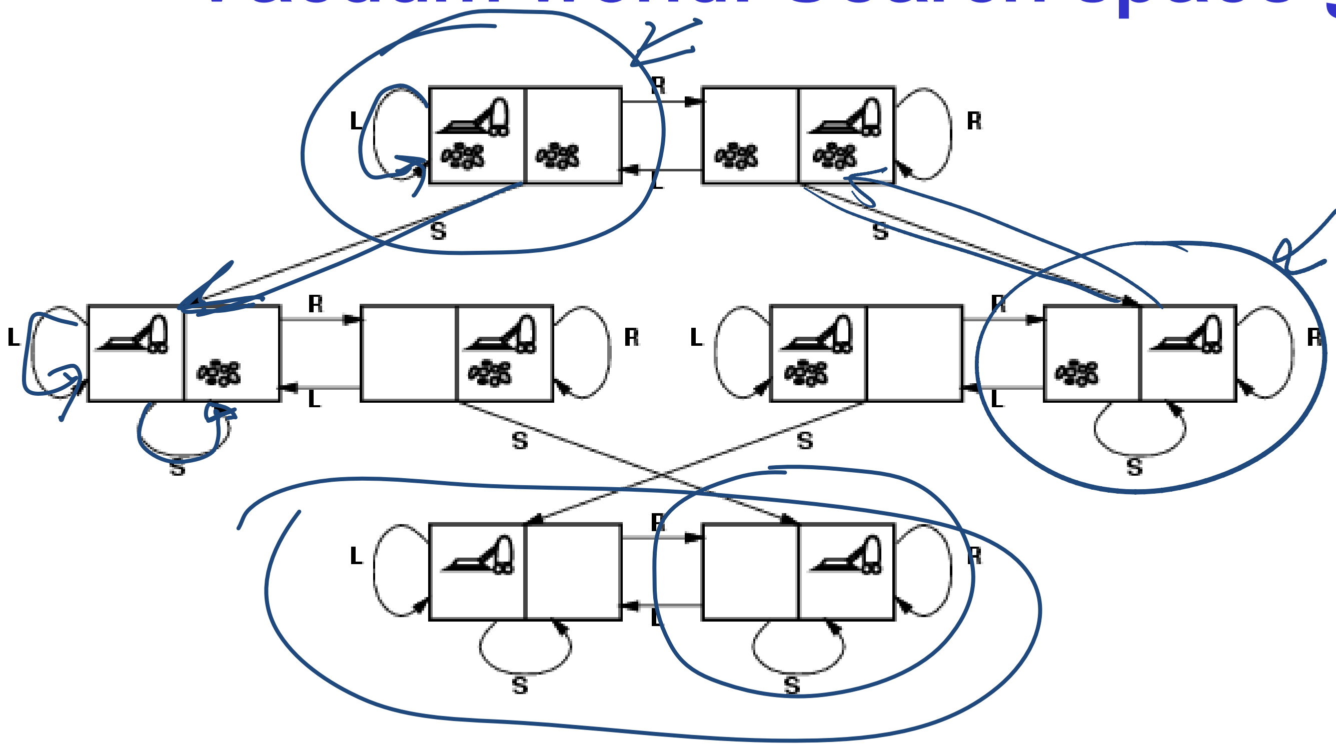
$K 2^K$ states

How can we find a solution?

- Define underlying **search space**. A graph where **nodes are states** and **edges are actions**.



Vacuum world: Search space graph



states? Where it is dirty and robot location

actions? *Left*, *Right*, *Suck*

Possible goal test? no dirt at all locations

Search: Abstract Definition

How to search

- Start at the start state
- Consider the effect of taking different actions starting from states that have been encountered in the search so far
- Stop when a goal state is encountered

To make this more formal, we'll need review the **formal definition of a graph**...

Search Graph

A *graph* consists of a set N of ***nodes*** and a set A of ordered pairs of nodes, called ***arcs***.

Node n_2 is a ***neighbor*** of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in$ A.

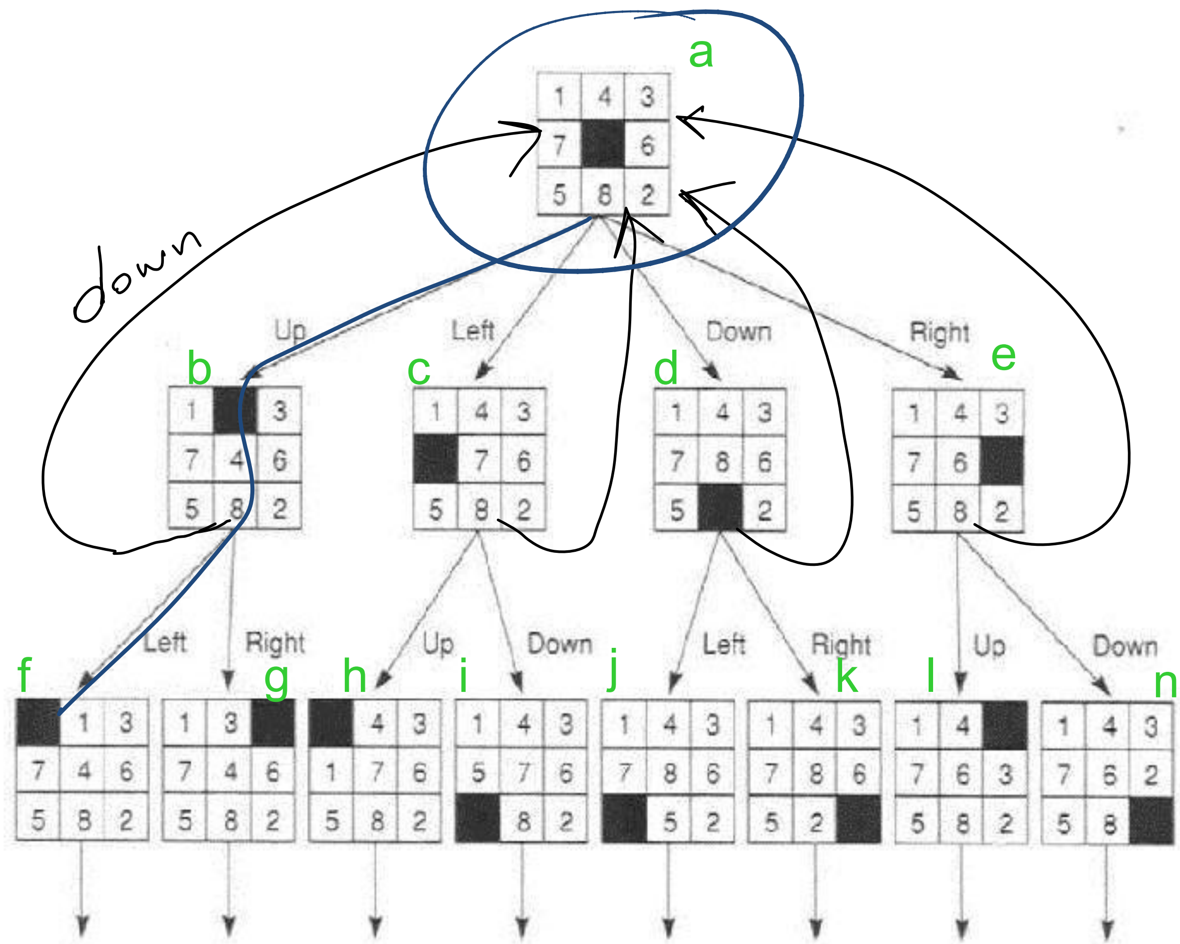
A ***path*** is a sequence of nodes $n_0, n_1, .., n_k$ such that $\langle n_{i-1}, n_i \rangle \in$ A.

A ***cycle*** is a non-empty path such that the start node is the same as the end node

A ***directed acyclic graph*** (DAG) is a graph with no cycles

Given a set of start nodes and goal nodes, a ***solution*** is a path from a start node to a goal node.

Examples for graph formal def.



Graph Searching

Generic search algorithm: given a graph, start node, and goal node(s), incrementally explore paths from the start node(s).

Maintain a **frontier of paths** from the start node that have been explored.

As search proceeds, the frontier expands into the unexplored nodes until (hopefully!) a goal node is encountered.

The way in which the frontier is expanded defines the search strategy.

For most problems, we can never actually build the whole graph

Generic Search Algorithm

Input: a graph, a start nodes, Boolean procedure *goal(n)* that tests if *n* is a goal node

frontier:= [$<s>$: *s* is a start node];

While *frontier* is not empty:

  select and **remove** path $<n_o, \ldots, n_k>$ from *frontier;*

  If *goal*(n_k)

    return $<n_o, \ldots, n_k>$;

  For every neighbor *n* of n_k

    add $<n_o, \ldots, n_k, n>$ to *frontier;*

end

Frontier

~~<A>~~

<A B>

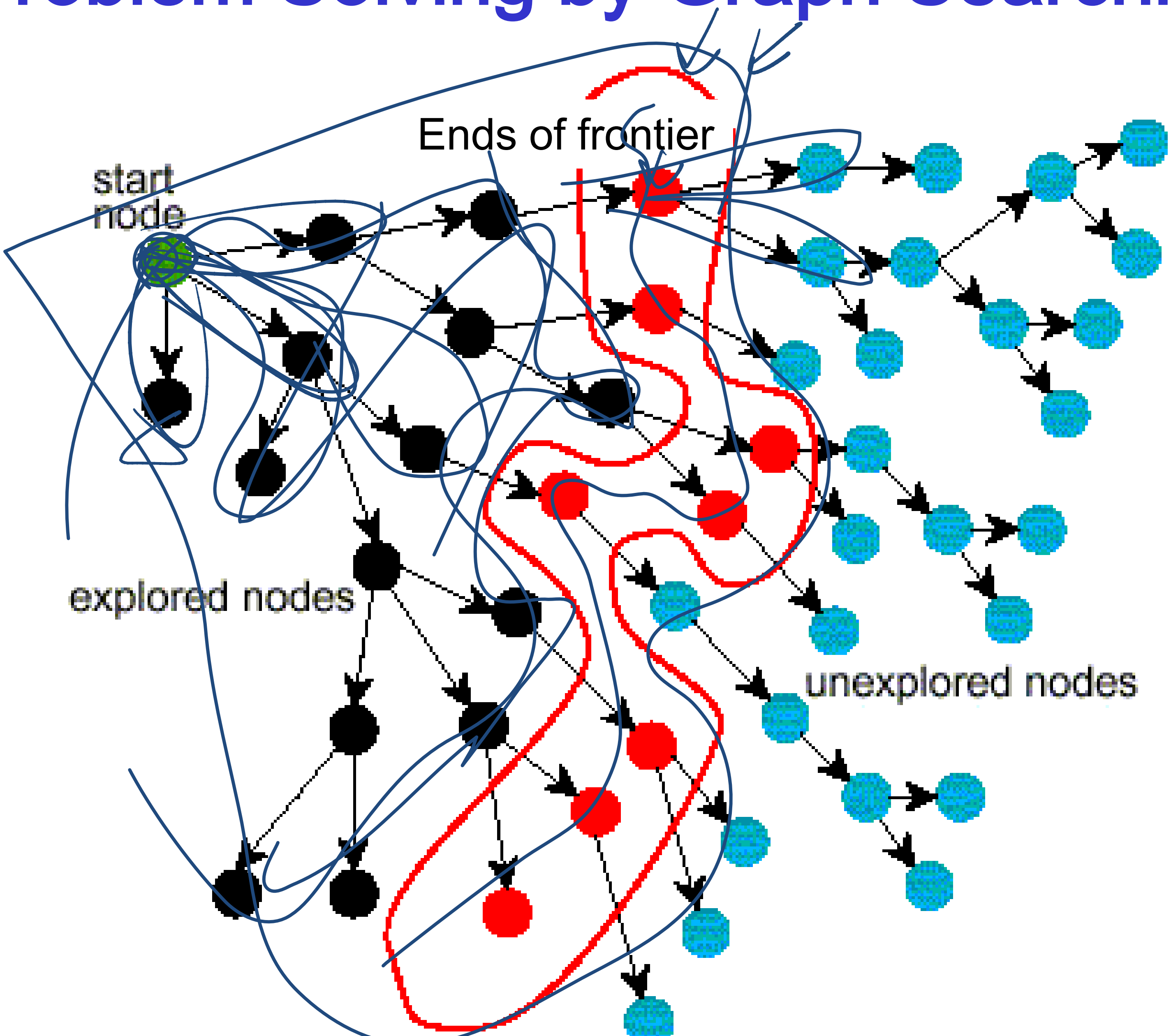
<A C>

~~<A D>~~

<A D E>

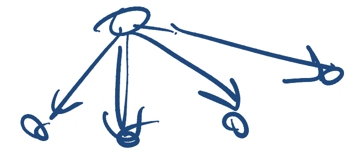
solution

Problem Solving by Graph Searching

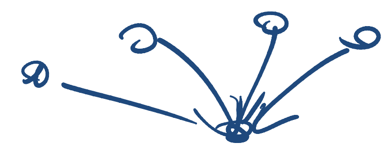


Branching Factor

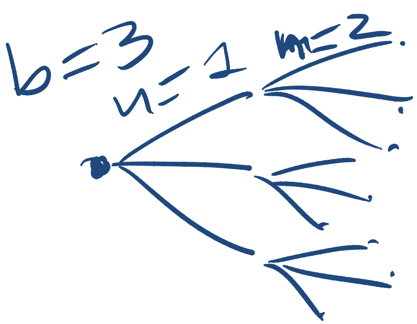
The ***forward branching factor*** of a node is the number of arcs going out of the node



The ***backward branching factor*** of a node is the number of arcs going into the node



If the forward branching factor of any node is *b* and the graph is a tree, How many nodes are *n* steps away from the root?



3^2

b^{30}

b^n

Comparing Searching Algorithms: will it find a solution? the best one?

Def. (complete): A search algorithm is complete if, whenever at least one solution exists, the algorithm **is guaranteed to find a solution** within a finite amount of time.

Def. (optimal): A search algorithm is optimal if, when it finds a solution , it is the best solution

Let’s look at two basic search strategies

Depth First and Breath First Search:

- To understand key properties of a search strategy
- They represent the basis for more sophisticated (heuristic / intelligent) search

Depth-first Search: DFS

- **Depth-first search** treats the frontier as a **stack**
- It always selects one of the last elements added to the frontier.

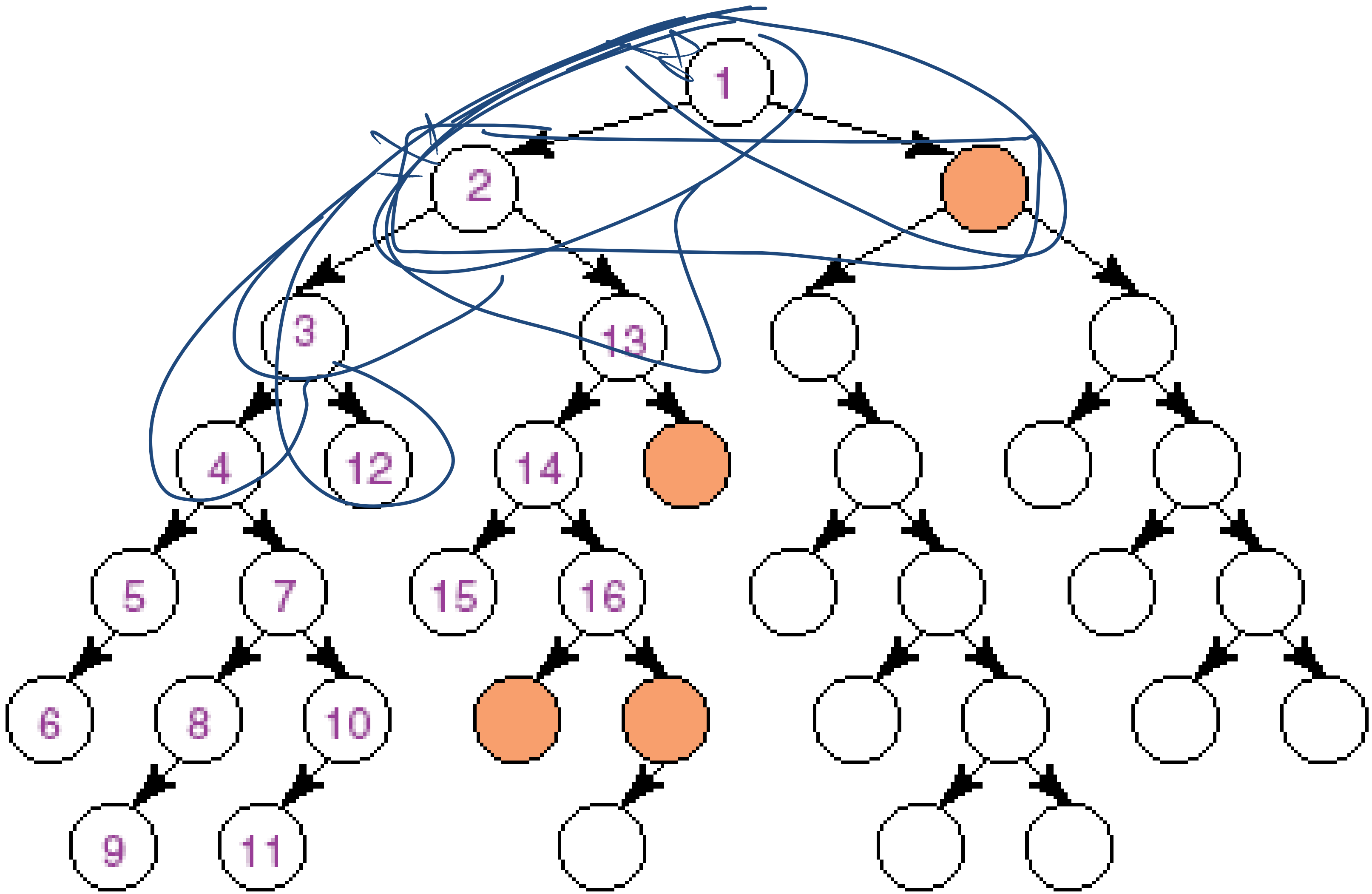
order in which these are added is not specified in pure DFS

Example: pop push

- the frontier is $[p_1, p_2, \ldots, p_r]$
- neighbors of last node of p_1 (its end) are $\{n_1, \ldots, n_k\}$

- What happens?
  - p_1 is selected, and its end is tested for being a goal. first
  - New paths are created attaching $\{n_1, \ldots, n_k\}$ to p_1 k new paths
  - These "replace" p_1 at the beginning of the frontier.
  - Thus, the frontier is now $[(p_1, n_1), \ldots, (p_1, n_k), p_2, \ldots, p_r]$.
  - p_2 is only selected when all paths extending p_1 have been explored.

Depth-first search: Illustrative Graph --- Depth-first Search Frontier



Depth-first Search: Analysis of DFS

- Is DFS complete?

- Is DFS optimal?

- What is its time complexity?

- What is its space complexity?

Breadth-first Search: BFS

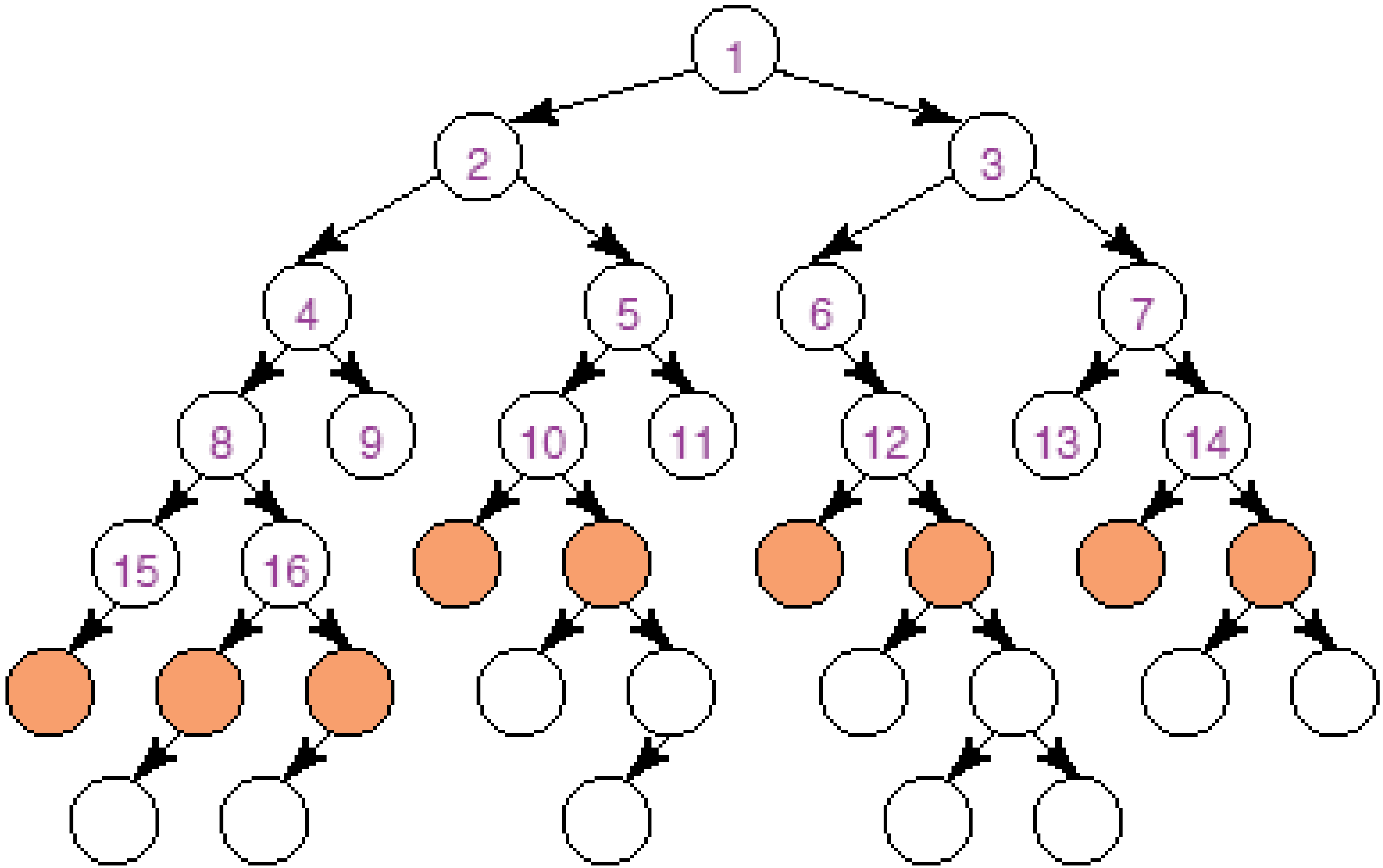
- Breadth-first search treats the frontier as a **queue**
  - it always selects one of the earliest elements added to the frontier.

Example: pop push

- the frontier is $[p_1, p_2, \ldots, p_r]$
- neighbors of the last node of p_1 are $\{n_1, \ldots, n_k\}$

- What happens?
  - p_1 is selected, and its end tested for being a path to the goal.
  - New paths are created attaching $\{n_1, \ldots, n_k\}$ to p_1
  - These follow p_r at the end of the frontier.
  - Thus, the frontier is now $[p_2, \ldots, p_r, (p_1, n_1), \ldots, (p_1, n_k)]$.
  - p_2 is selected next.

Illustrative Graph - Breadth-first Search



Breadth Search: Analysis of BFS

- Is BFS complete?
- Is BFS optimal?
- What is its time complexity?
- What is its space complexity?

Iterative Deepening (sec 3.6.3)

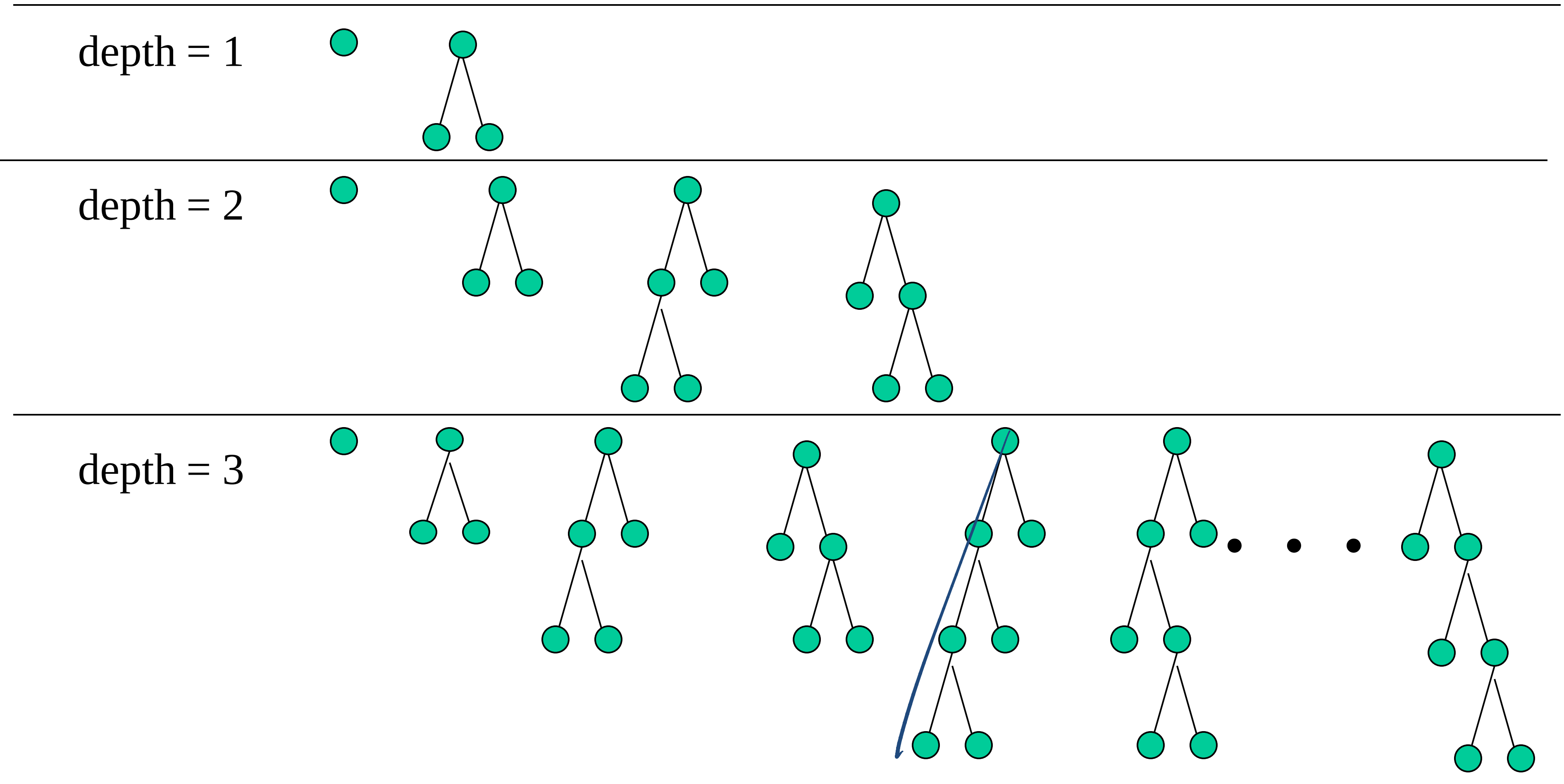
How can we achieve an acceptable (linear) space complexity maintaining completeness and optimality?

	Complete	Optimal	Time	Space
DFS	N	N	b^m	mb
BFS	Y	Y	b^m	b^m
:)	Y	Y	b^m	mb

Key Idea: let's re-compute elements of the frontier rather than saving them.

Iterative Deepening in Essence

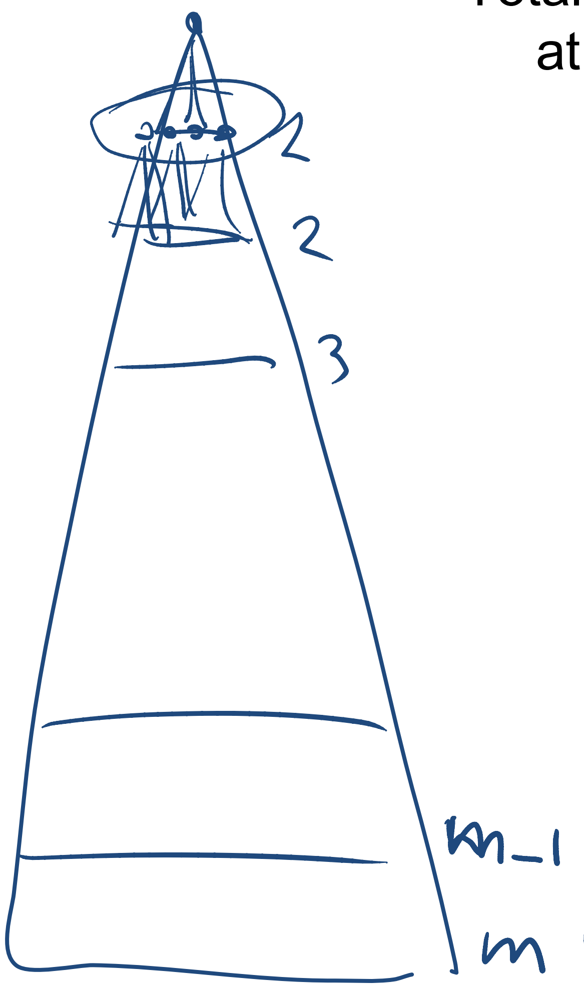
- Look with DFS for solutions at depth 1, then 2, then 3, etc.
- If a solution cannot be found at depth *D*, look for a solution at depth *D* + 1.
- You need a depth-bounded depth-first searcher.
- Given a bound B you simply assume that paths of length B cannot be expanded….


depth = 1
depth = 2
depth = 3

(Time) Complexity of Iterative Deepening

Complexity of solution at depth m with branching factor b

total # of paths created by IDS is sum of this column



Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	Σ
b	1	m	mb
b^2	1	$m-1$	$(m-1)b^2$
b^3	1		
	1		
b^{m-1}	1	2	$2b^{m-1}$
b^m	1	1	b^m

(Time) Complexity of Iterative Deepening

Complexity of solution at depth m with branching factor *b*

Total # of paths generated

$$\boxed{b^m + 2\,b^{m-1} + 3\,b^{m-2} + .. + mb} =$$

$$b^m\,(1 + 2\,b^{-1} + 3\,b^{-2} + .. + m\,b^{1-m}) \leq$$

$$b^m\boxed{\left(\sum_{i=1}^{\infty} i b^{1-i}\right)} = b^m\boxed{\left(\frac{b}{b-1}\right)^2} = O(b^m)$$

$b = 2$ $\quad 4$

$b = 3$ $\quad \frac{9}{4} = 2.25$

$b = 4$ $\quad \frac{16}{9} < 2$

Search with Costs

Sometimes there are costs associated with arcs.

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

Define optimality…..

Design an optimal search strategy……

Today Sept 13

- Uninformed Search
- **Informed Search**
- ….
- ….

Heuristic Search

Uninformed/Blind search algorithms do not take into account the goal until they are at a goal node.

Often there is extra knowledge that can be used to guide the search: an ***estimate*** of the distance from node *n* to a goal node.

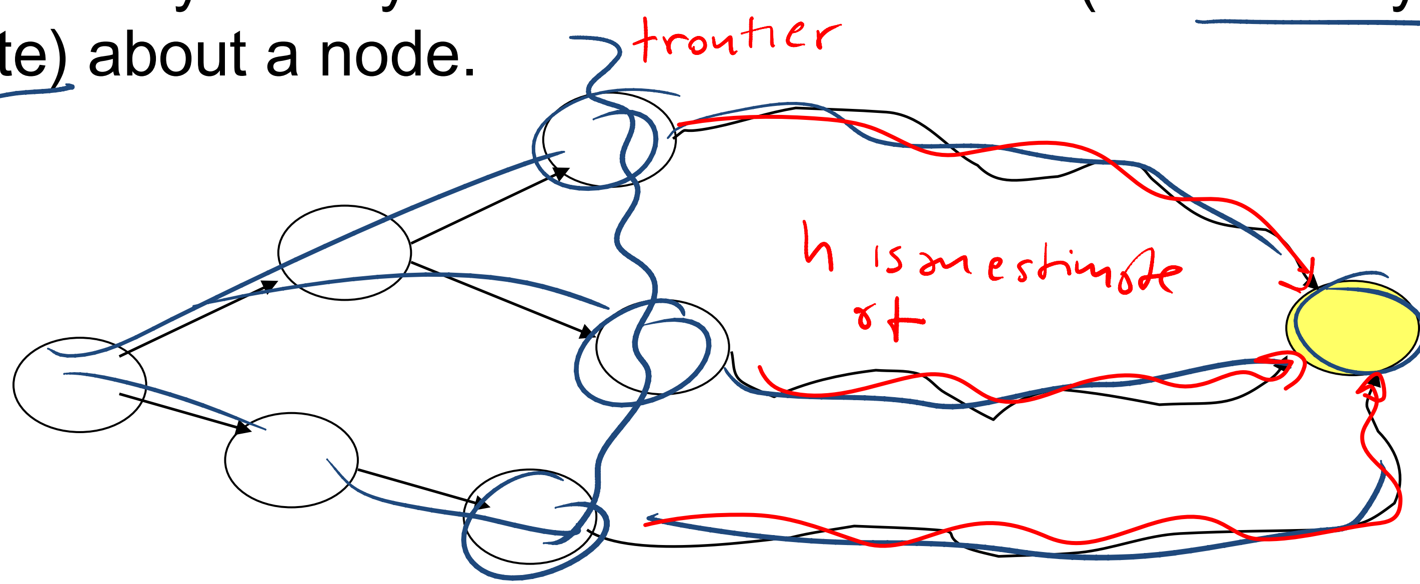
This is called a ***heuristic***

More formally

Definition (search heuristic)

A search heuristic *h(n)* is an estimate of the cost of the shortest path from node *n* to a goal node.

- *h* can be extended to paths: $h(\langle n_0,\ldots,n_k\rangle)=h(n_k)$
- *h(n)* uses only readily obtainable information (that is easy to compute) about a node.



More formally (cont.)

Definition (**admissible heuristic**)

A search heuristic *h(n)* is admissible if it is never an overestimate of the cost from *n* to a goal.

- There is never a path from *n* to a goal that has path length less than *h(n)*.
- another way of saying this: *h(n)* is a lower bound on the cost of getting from *n* to the nearest goal.

Example Admissible Heuristic Functions

Search problem: robot has to find a route from start location to goal location on a grid (discrete space with obstacles)

Final cost (quality of the solution) is the number of steps

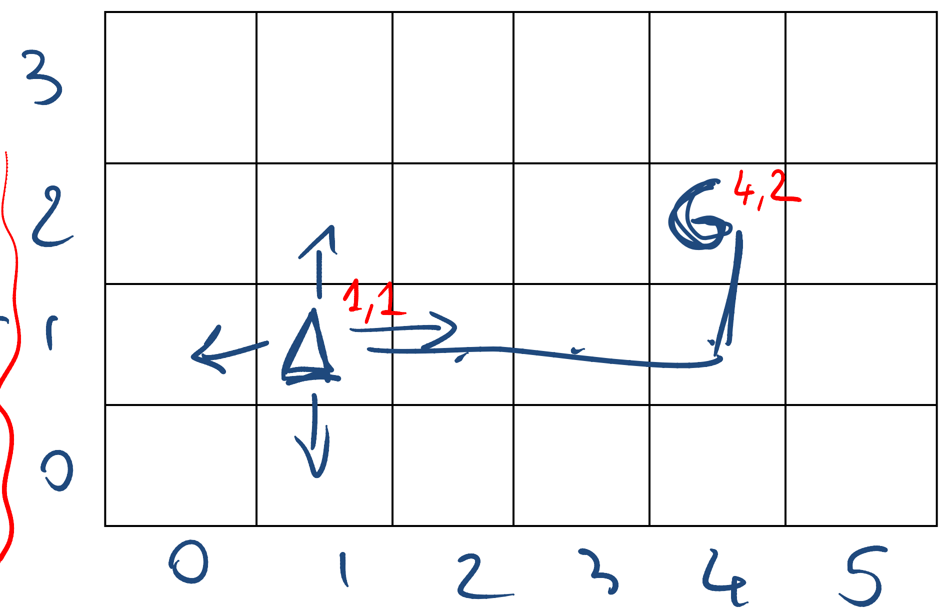
If no obstacles, cost of optimal solution is…

$X_G \quad Y_G$ Goal state

$X_C \quad Y_C$ Current state

In example

$$\underbrace{|X_G - X_C| + |Y_G + Y_C|}_{\text{Manhattan distance}} \qquad |4-1| + |2-1| = 4$$



Example Admissible Heuristic Functions

If there are obstacle, the optimal solution without obstacles is an admissible heuristic

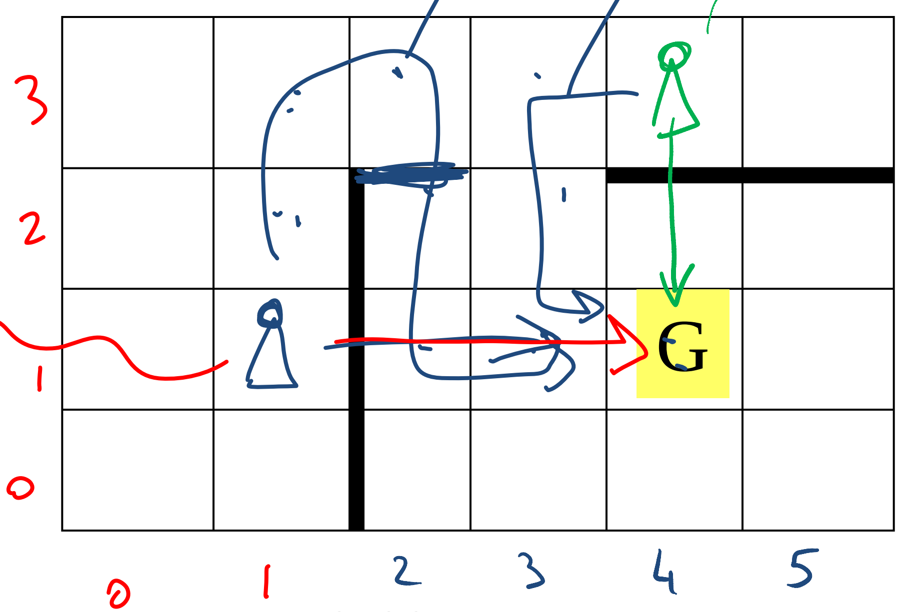
robot on grid
20 possible states

actual optimal solutions for states (1,1) and (4,3)

7

4

h = 2
for state (4,3)

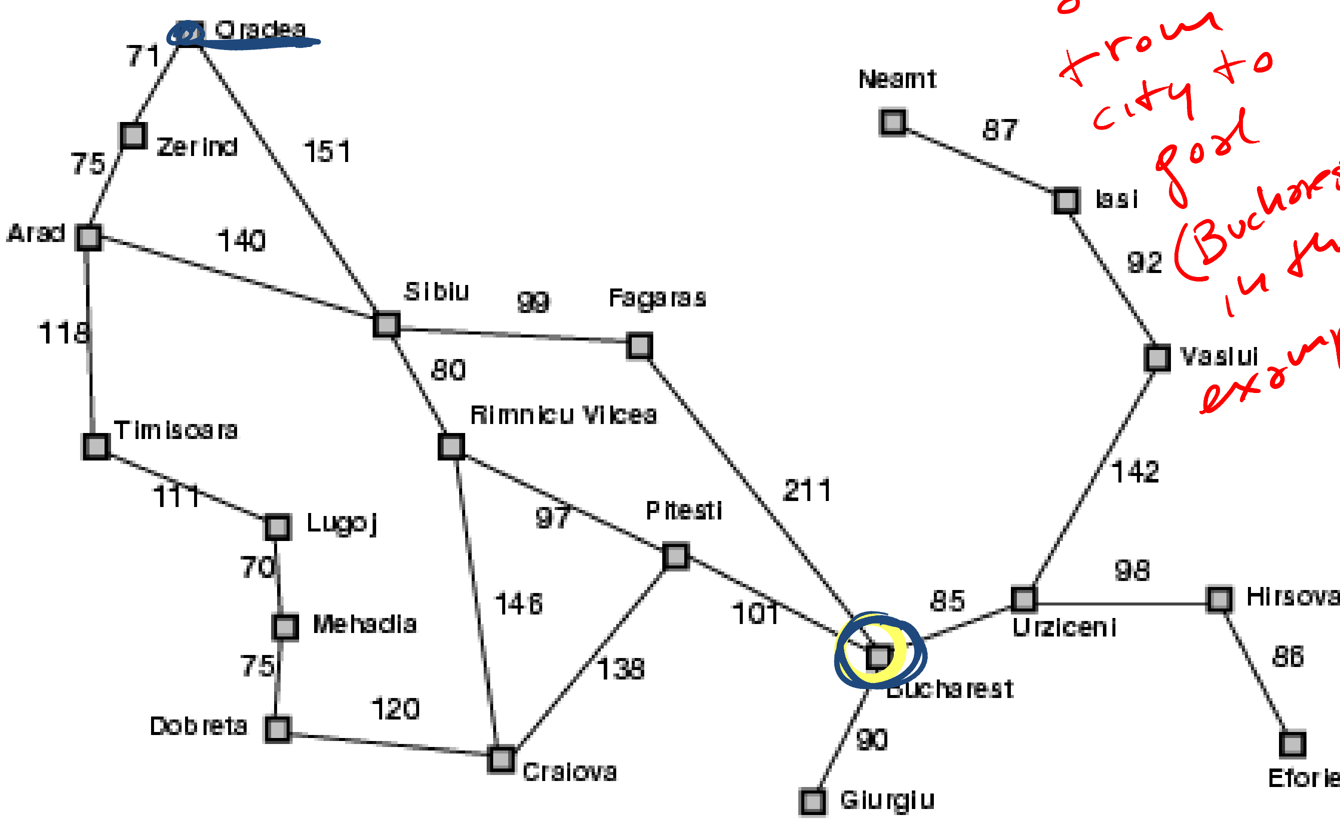


h = 3
for state (1,1)

Example Admissible Heuristic Functions

- Similarly, If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from *n* to the closest goal as the value of *h(n)*.

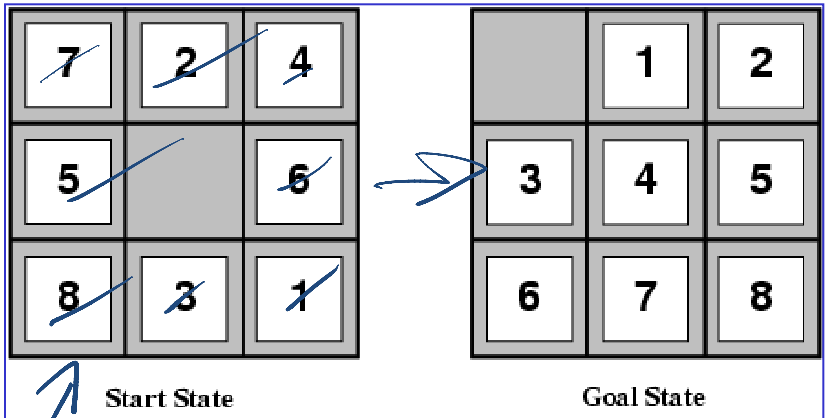
always <= than any road from city to goal (Bucharest in this example)

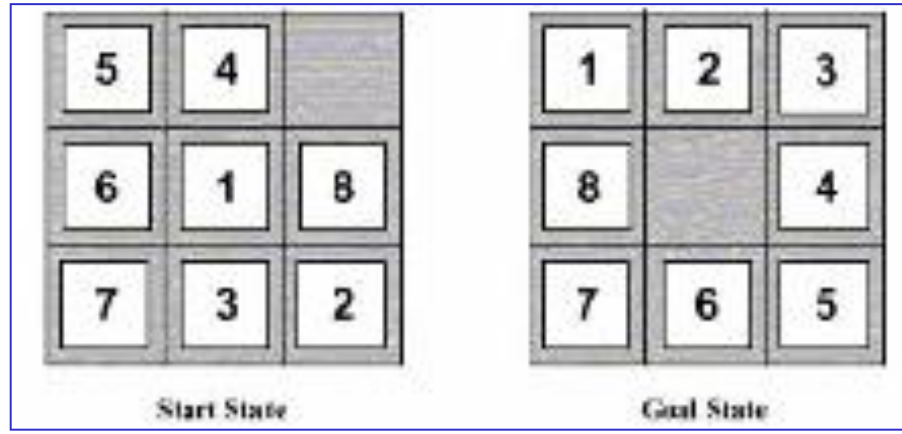


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example Heuristic Functions (1)

- In the 8-puzzle, we can use the number of misplaced tiles



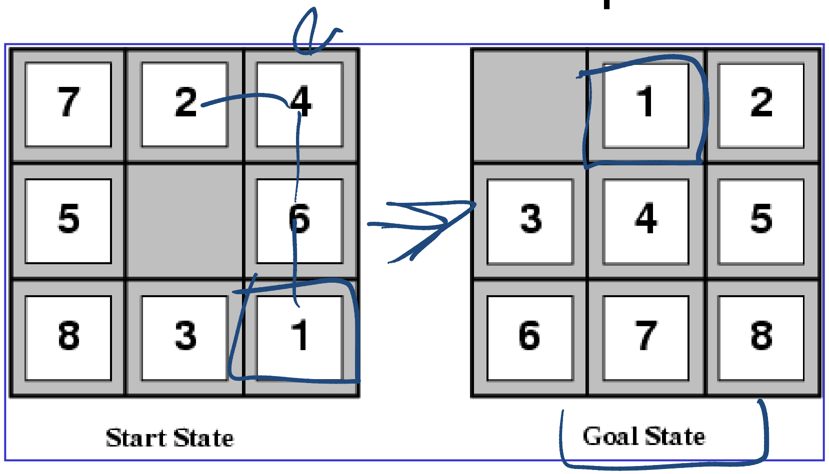


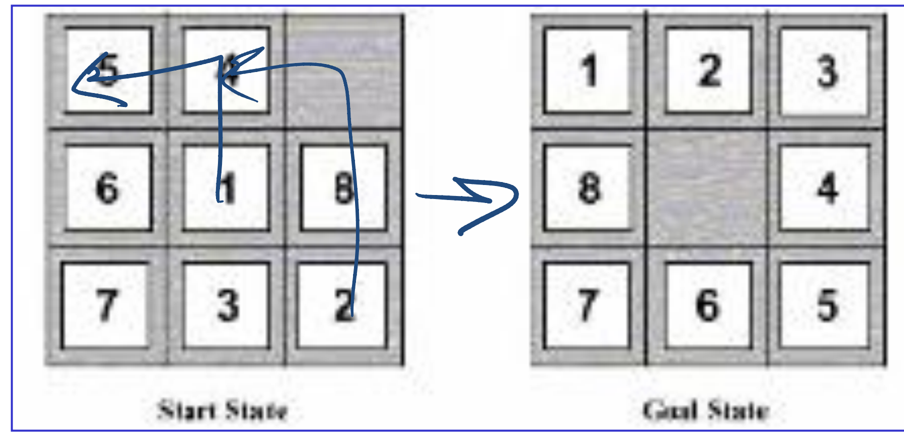
h 8

h 7

Example Heuristic Functions (2)

- Another one we can use the number of moves between each tile's current position and its position in the solution





tiles

1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8
Σ3	1	2	2	2	3	3	2	Σ	2	3						

= 18

How to Construct a Heuristic

You identify relaxed version of the problem:

- where one or more constraints have been dropped
- problem with fewer restrictions on the actions

Robot: the agent can move through walls

Driver: the agent can move straight

8puzzle: (1) tiles can move anywhere

(2) tiles can move to any adjacent square

Result: The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem (because it is always weakly less costly to solve a less constrained problem!)

How to Construct a Heuristic (cont.)

You should identify constraints which, when dropped, make the problem extremely easy to solve

- this is important because heuristics are not useful if they're as hard to solve as the original problem!

This was the case in our examples

Robot: ***allowing*** the agent to move through walls. Optimal solution to this relaxed problem is Manhattan distance

Driver: ***allowing*** the agent to move straight. Optimal solution to this relaxed problem is straight-line distance

8puzzle: (1) tiles **can move anywhere** Optimal solution to this relaxed problem is number of misplaced tiles

(2) tiles can move to **any adjacent square….**

Another approach to construct heuristics

Solution cost for a subproblem

1 2 3 4

Original Problem

	1	3
8	2	5
7	6	4

Current node

simpler!

SubProblem

	1	3
@	2	@
@	@	4

1	2	3
8		4
7	6	5

Goal node

1	2	3
@		4
@	@	@

Goal

Heuristics: Dominance

state

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

then h_2 dominates h_1

h_2 is better for search (why?)

(2) >= (1)

8puzzle: (1) tiles can move anywhere

(2) tiles can move to any adjacent square

(Original problem: tiles can move to an adjacent square if it is empty)

why

Iterative deepening (not using any heuristic)

search costs for the 8-puzzle (average number of paths expanded):

-> depth of solution

d=12 IDS = 3,644,035 paths
$A^*(h_1)$ = 227 paths
$A^*(h_2)$ = 73 paths

d=24 IDS = too many paths
$A^*(h_1)$ = 39,135 paths
$A^*(h_2)$ = 1,641 paths

	h_1	h_2
If tile in correct position	0	0
If tile 1 move from correct position	1	1
otherwise	1	>1

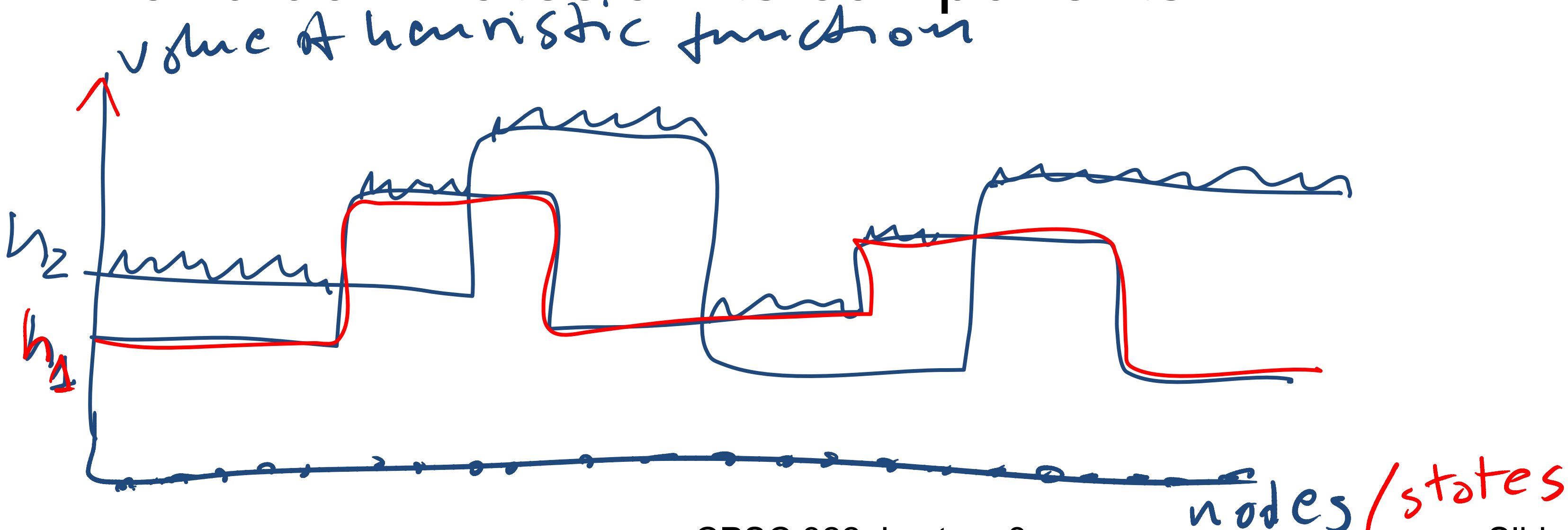
Combining Heuristics

How to combine heuristics when there is no dominance?

If $h_1(n)$ is admissible and $h_2(n)$ is also admissible then

$h(n)=$ max(h_1, h_2) is also admissible

… and dominates all its components



Combining Heuristics: Example

In 8-puzzle, solution cost for the 1,2,3,4 subproblem is substantially more accurate than Manhattan distance **in some cases**

sum of

of each tile from its position in goal

So…..

max

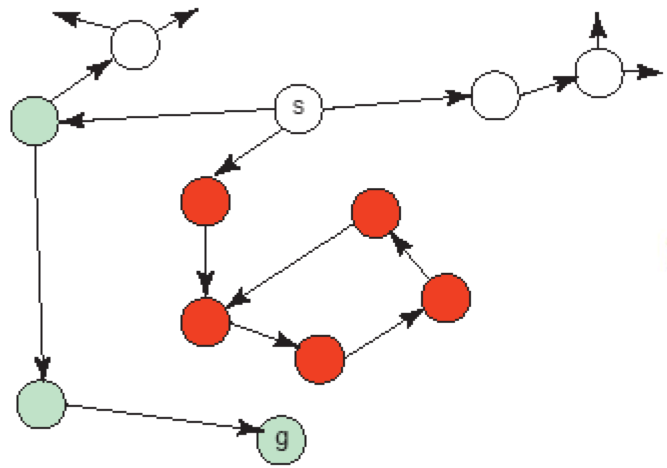
better heuristic

Best-First Search

- **Idea:** select the path whose end is closest to a goal according to the heuristic function.
- **Best-First search** selects a path on the frontier with minimal *h*-value (for the end node).
- It treats the frontier as a priority queue ordered by *h*. (similar to ?) LCFS → Cost
- This is a greedy approach: it always takes the path which appears locally best

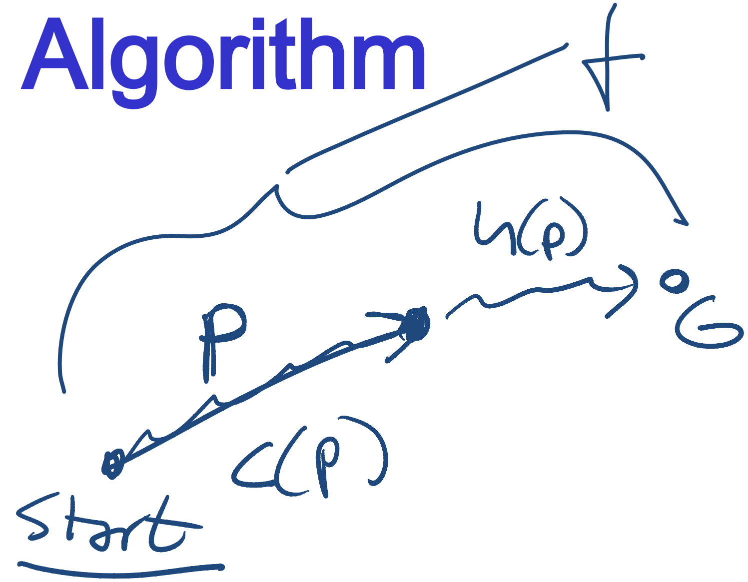
Analysis of Best-First Search

- Complete no: a low heuristic value can mean that a cycle gets followed forever.



- Optimal: no (why not?)
- Time complexity is $O(b^m)$
- Space complexity is $O(b^m)$

A^* Search Algorithm



- A^* is a mix of:
  - **lowest-cost-first** and
  - **best-first search**

- A^* treats the frontier as a priority queue ordered by $f(p)= c(p)+h(p)$

- It always selects the node on the frontier with the ......lowest........ estimated ....total.......distance.

Analysis of A^*

Let's assume that arc costs are strictly positive.

- Time complexity is $O(b^m)$
  - the heuristic could be completely uninformative and the edge costs could all be the same, meaning that A^* does the same thing as BFS
- Space complexity is $O(b^m)$ like BFS, A^* maintains a frontier which grows with the size of the tree
- Completeness: yes.
- Optimality: yes.

Optimality of A^*

If A^* returns a solution, that solution is guaranteed to be optimal, as long as

When

- the branching factor is finite
- arc costs are strictly positive
- *h(n)* is an underestimate of the length of the shortest path from *n* to a goal node, and is non-negative

admissible

Theorem

If A^* selects a path p, p is the shortest (i.e., lowest-cost) path.

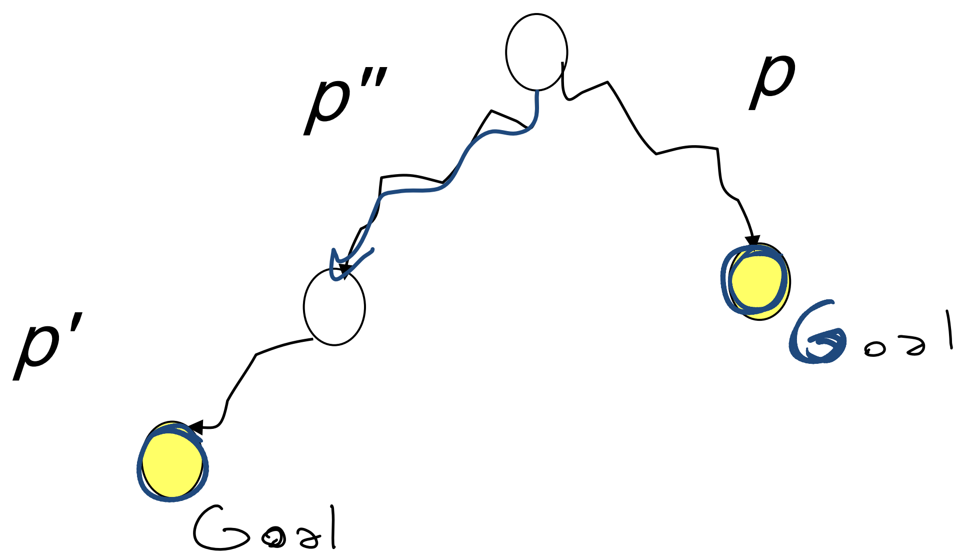
Why is A^* optimal?

$f = c + h$

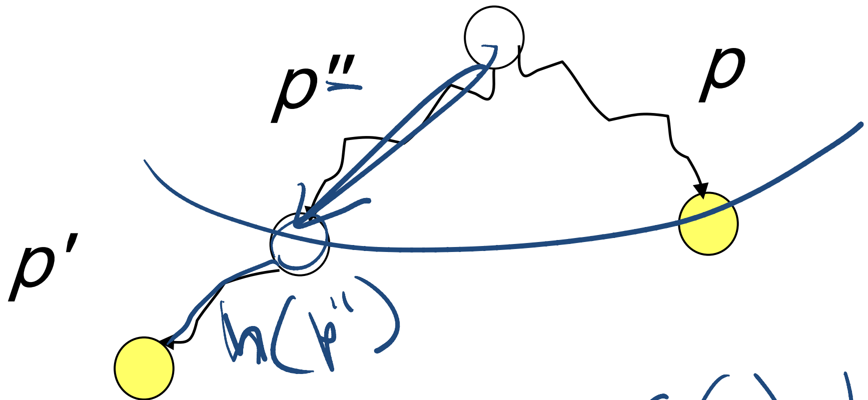
- Assume for contradiction that some other path p' is actually the shortest path to a goal

  $\text{cost}(p') < \text{cost}(p)$

- Consider the moment when p is chosen from the frontier. Some part of path p' will also be on the frontier; let's call this partial path p''.



Why is A^* optimal? (cont')



$\underbrace{c(p)+h(p)}_{f(p)} \leq \underbrace{c(p'')+h(p'')}_{f(p'')}$

- Because p was expanded before p'', $f(p) \leq f(p'')$
- Because p is a goal, $h(p)=0$ Thus $\boxed{c(p) \leq c(p'')+h(p'')}$
- Because h is admissible, $\boxed{cost(p'') + h(p'') \leq cost(p')}$ for any path p' to a goal that extends p''
- Thus $cost(p) \leq cost(p')$ for any other path p' to a goal.

This contradicts our assumption that p' is the shortest path.

that $cost(p') < cost(p)$

Branch-and-Bound Search

- What is the biggest advantage of A*?

$f = c + h$

- What is the biggest problem with A*?

b^m

- Possible Solution:

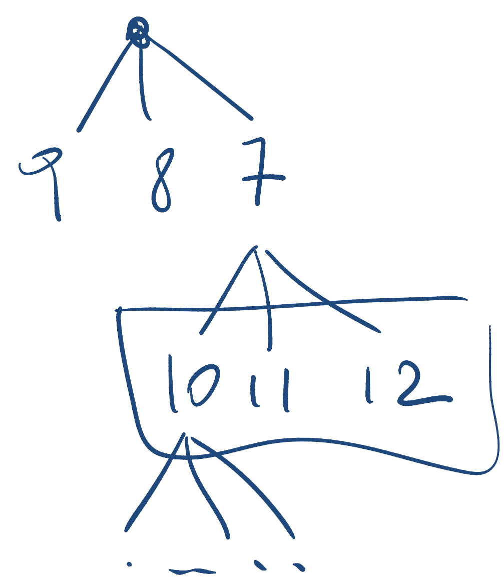
use f

save space

Branch-and-Bound Search Algorithm

- Follow exactly the same search path as **depth-first search**
  - treat the frontier as a stack: expand the most-recently added path first
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic

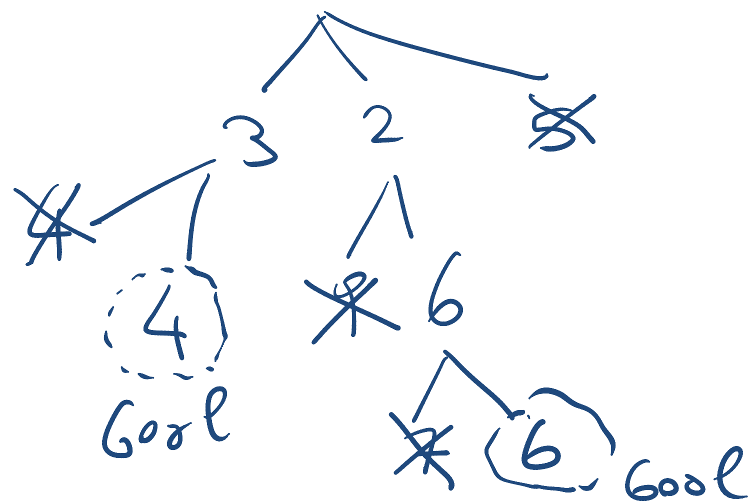
$f = c + h$



Branch-and-Bound Search Algorithm

- Keep track of a lower bound and upper bound on solution cost at each path
  - lower bound: $LB(p) = f(p) = cost(p) + h(p)$
  - upper bound: UB = cost of the best solution found so far.
    - ✓ if no solution has been found yet, set the upper bound to ∞.
- When a path p is selected for expansion:
  - if $LB(p) \geq UB$, remove p from frontier without expanding it (pruning)
  - else expand p, adding all of its neighbors to the frontier

$UB = \infty, 6, 4$



Other A^* Enhancements

The main problem with A^* is that it uses exponential space. Branch and bound was one way around this problem. Are there others?

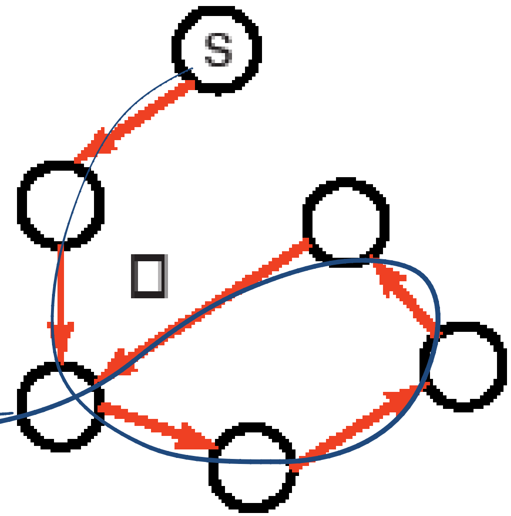
- Iterative deepening A*
- Memory-bounded A^*

Other A^* Enhancements

The main problem with A^* is that it uses exponential space. Branch and bound was one way around this problem. Are there others?

- Iterative deepening A*
- Memory-bounded A^*

Cycle Checking



Goal

You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

- The time is ...linear........ in path length.

$\langle n_0, n_1, n_2 \ldots n_k \rangle$

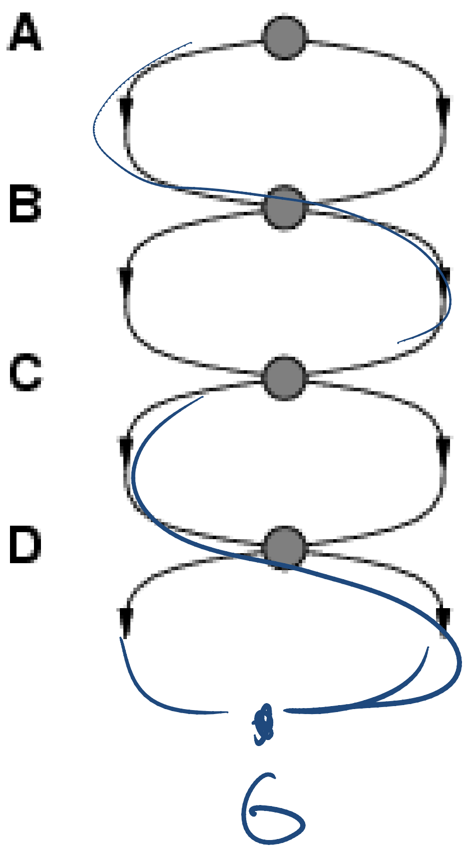
n_2

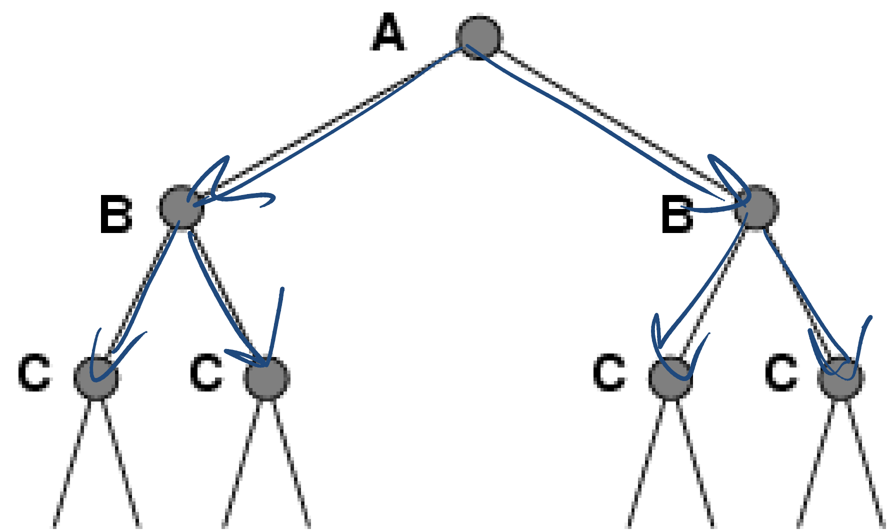
n_i

n_j

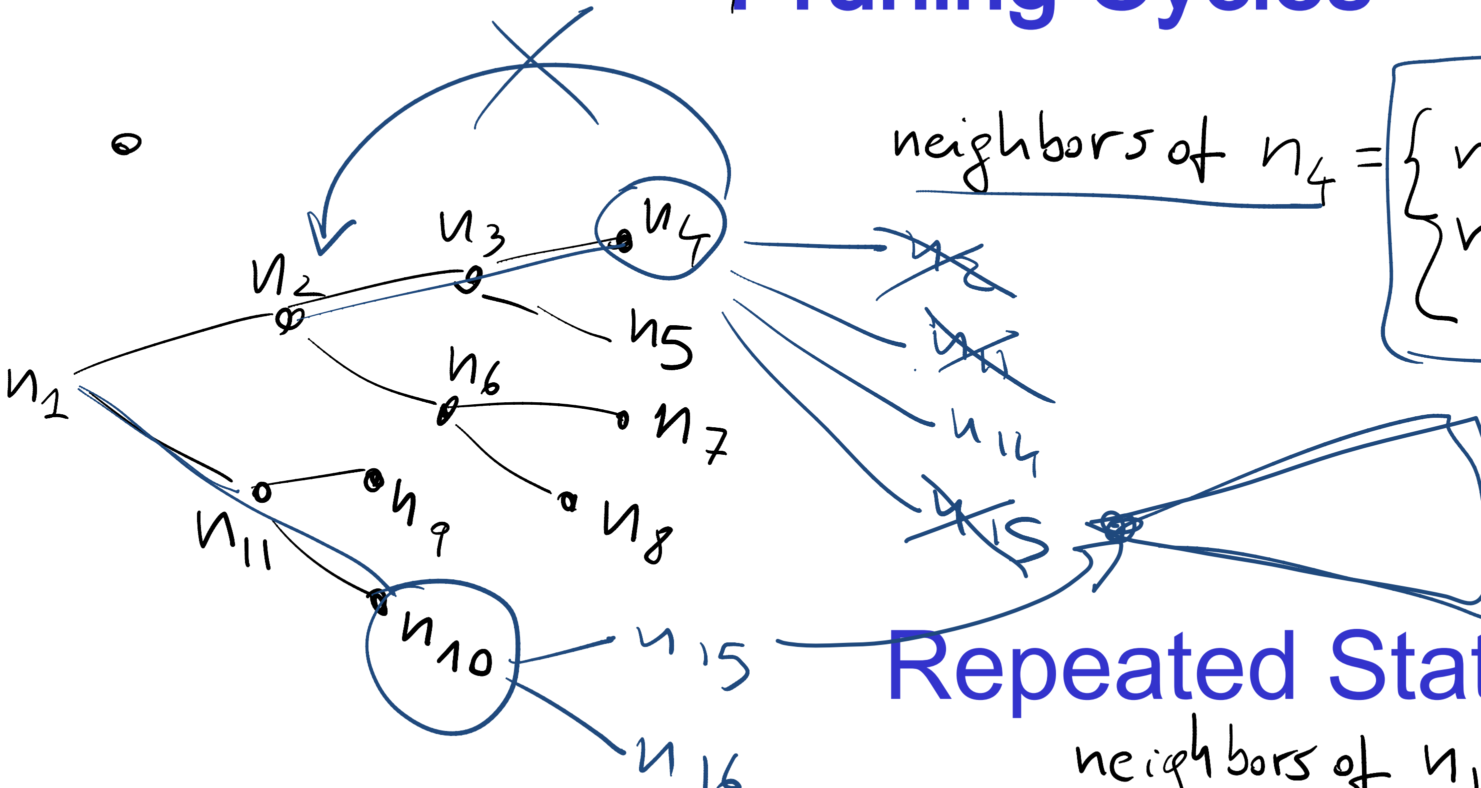
Repeated States / Multiple Paths

Failure to detect repeated states can turn a linear problem into an exponential one!





Pruning Cycles



neighbors of n_4 = $\boxed{\{n_2, n_{11}, n_{14}, n_{15}\}}$

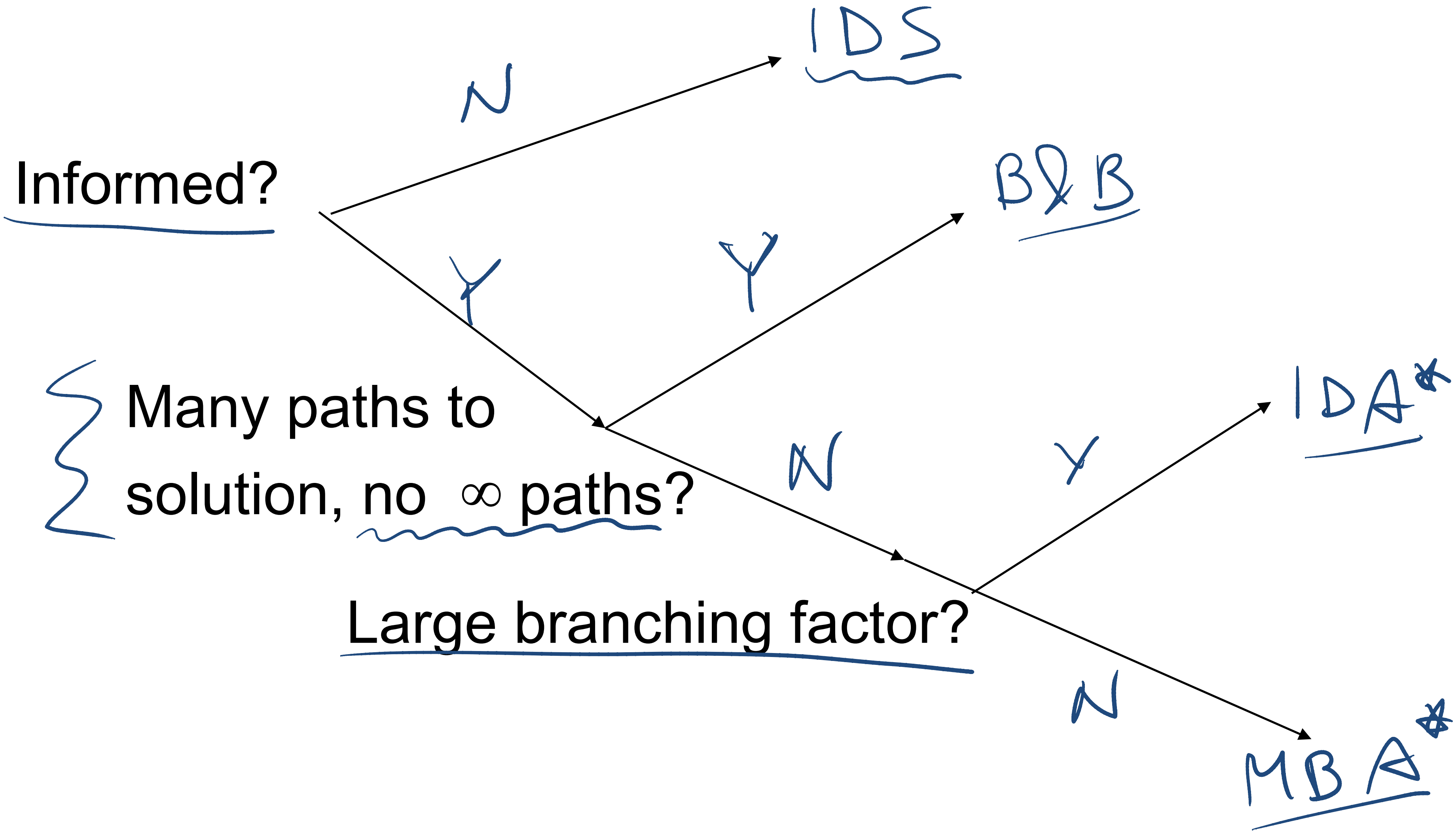
Repeated States

neighbors of n_{10} = $\{n_{15}, n_{16}\}$

Search in Practice

	Complete	Optimal	Time	Space
DFS	N	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
IDS(C)	Y	Y	$O(b^m)$	$O(mb)$
LCFS	Y	Y	$O(b^m)$	$O(b^m)$
BFS	N	N	$O(b^m)$	$O(b^m)$
A*	Y	Y	$O(b^m)$	$O(b^m)$
B&B	N	Y	$O(b^m)$	$O(mb)$
IDA*	Y	Y	$O(b^m)$	$O(mb)$
MBA*	N	N	$O(b^m)$	$O(b^m)$
BDS	Y	Y	$O(b^{m/2})$	$O(b^{m/2})$

Search in Practice (cont')



Sample A* applications

- **An Efficient A* Search Algorithm For Statistical** Machine Translation. 2001
- **The Generalized A* Architecture**. Journal of Artificial Intelligence Research (2007)
  - Machine Vision … Here we consider a new compositional model for finding salient curves.
- **Factored A*search for models over sequences and trees** International Conference on AI. 2003…. It starts saying… *The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate…* applied to NLP and BioInformatics

(Natural Language Processing)

Class Forum: Piazza

Join the class **asap** via the signup link below.

http://www.piazza.com/ubc.ca/fall2011/cpsc502

You need a **ubc.ca or cs.ubc.ca** email address to sign up. If you do not have one, please send an email to rjoty@cs.ubc.ca

TODO for this Thurs

Read Chp 4 of textbook

Do all the "Graph Searching exercises" available at http://www.aispace.org/exercises.shtml
Please, look at solutions only after you have tried hard to solve them!

- Join piazza (the class discussion forum)

Lecture Summary

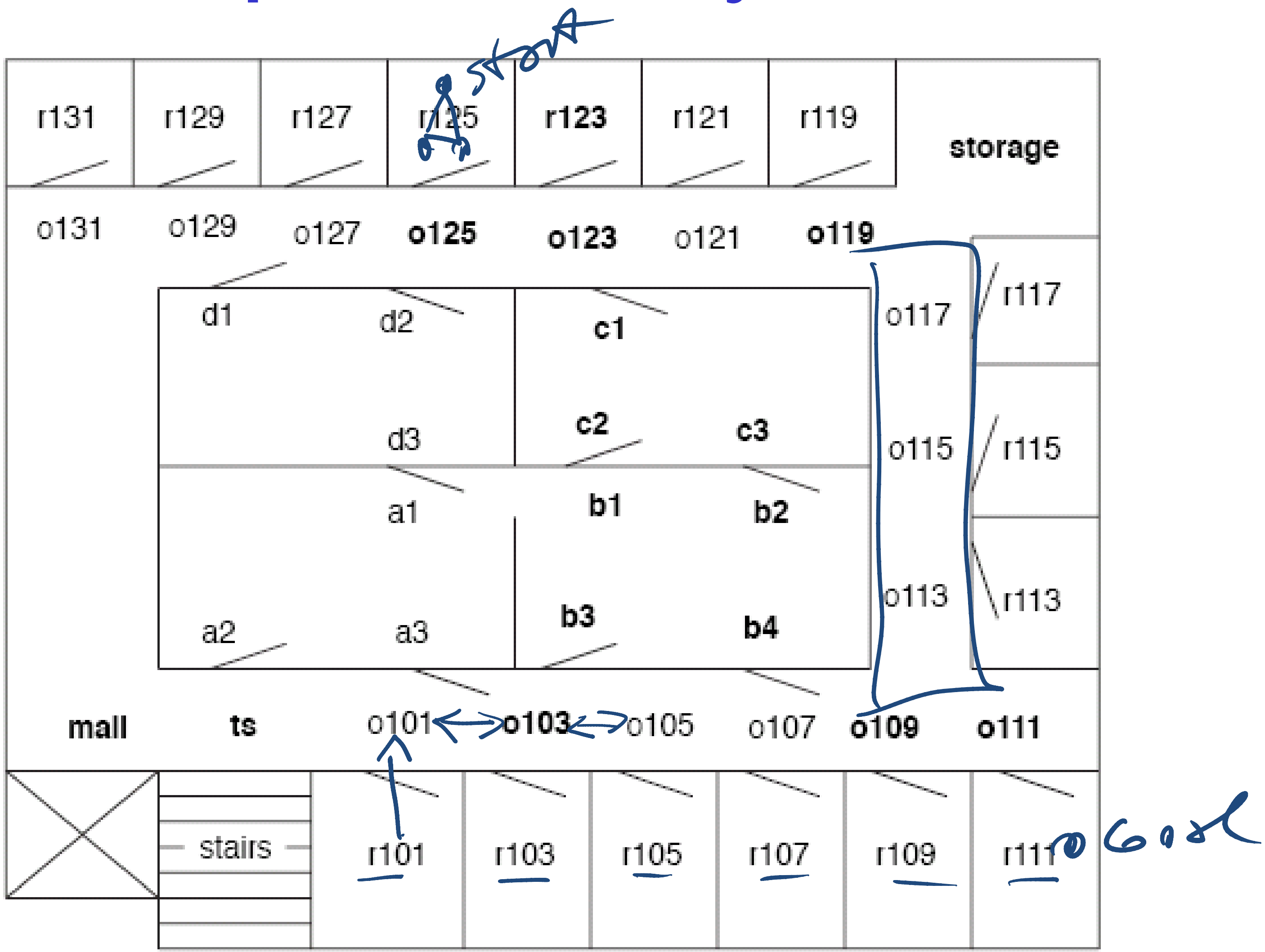
- Search is a key computational mechanism in many AI agents
- We will study the basic principles of search on the simple **deterministic planning agent model**

Generic search approach:

- define a search space graph,
- start from current state,
- incrementally explore paths from current state until goal state is reached.

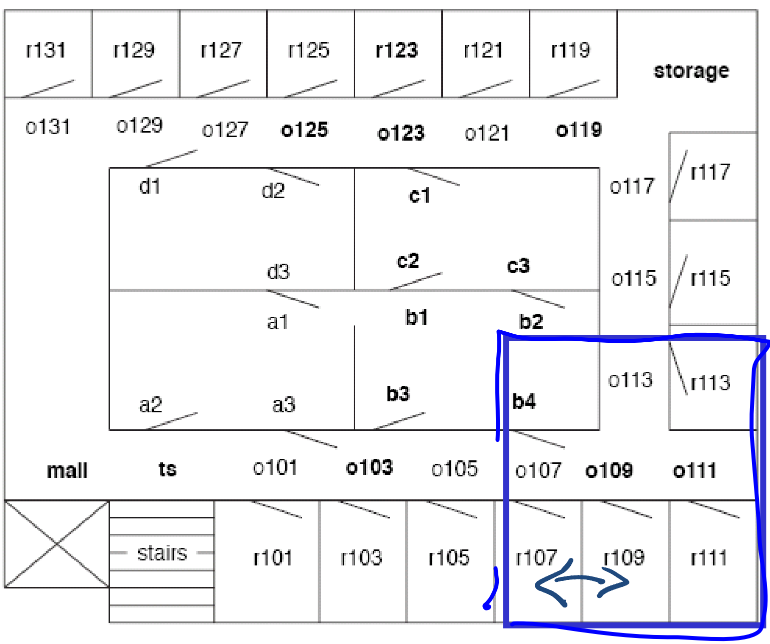
The way in which the frontier is expanded defines the search strategy.

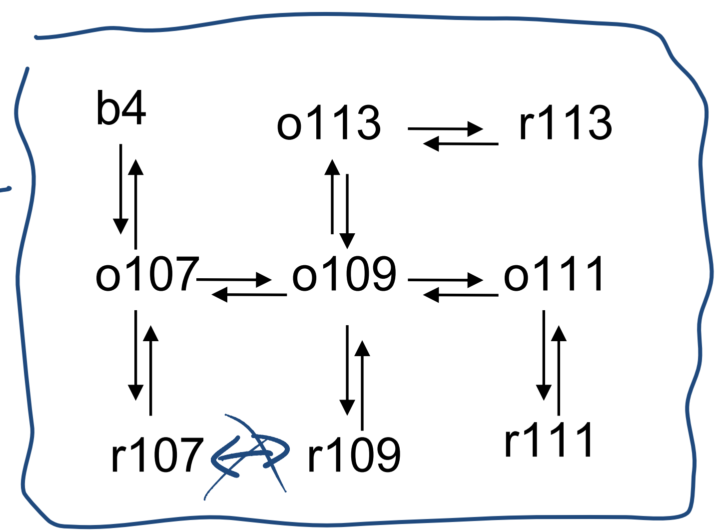
Example1: Delivery Robot



How can we find a solution?

- How can we find a sequence of actions and their appropriate ordering that lead to the goal?
- Define underlying **search space**. A graph where **nodes are states** and **edges are actions**.





Examples of solution

- Start state b4, goal r113
- Solution <b4, o107, o109, o113, r113>
-