Search

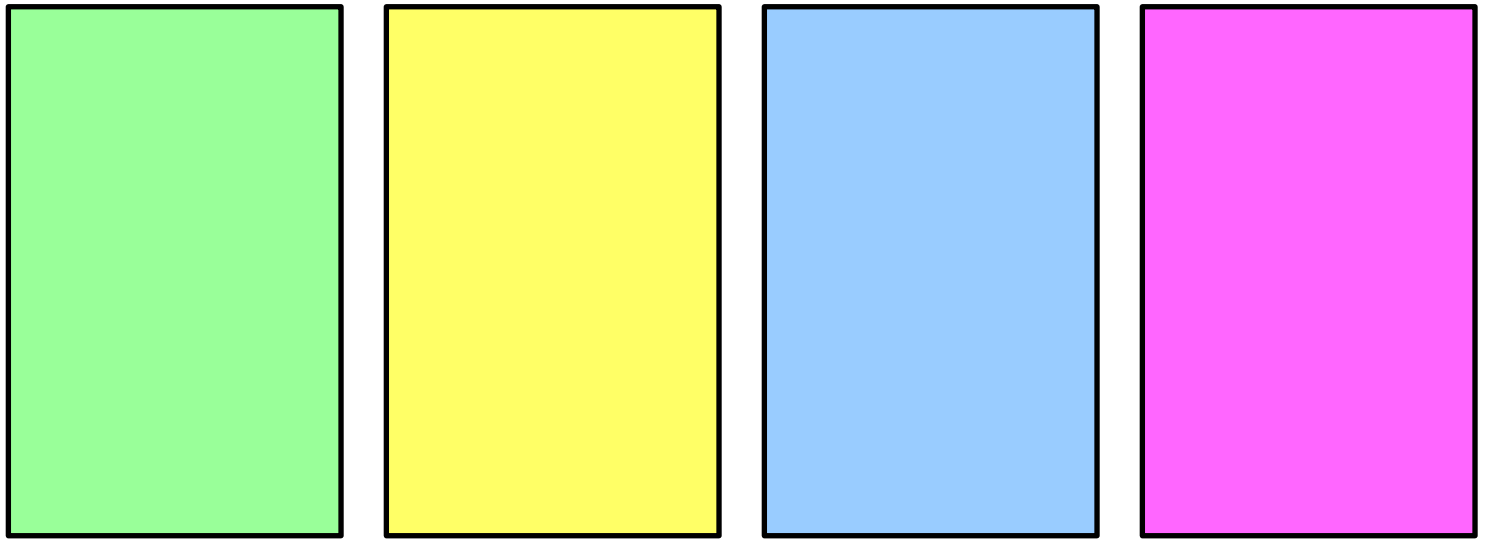
Computer Science cpsc322, Lecture 2

(Textbook Chpt 3.0-3.4)

May, 10, 2012

Colored Cards

- You need to have 4 colored index cards
  - Come and get them from me if you still don't have them



- You will use these as voting cards
  - Cheap low tech variant of clickers

Please bring them to class every time

"Deterministic agent" means an agent that

Has perfect knowledge of its environment

Has perfect knowledge of the effect that its actions can have on the environment

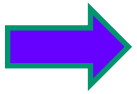
Both of the above

None of the above

"Deterministic agent" means an agent that

Has perfect knowledge of its environment

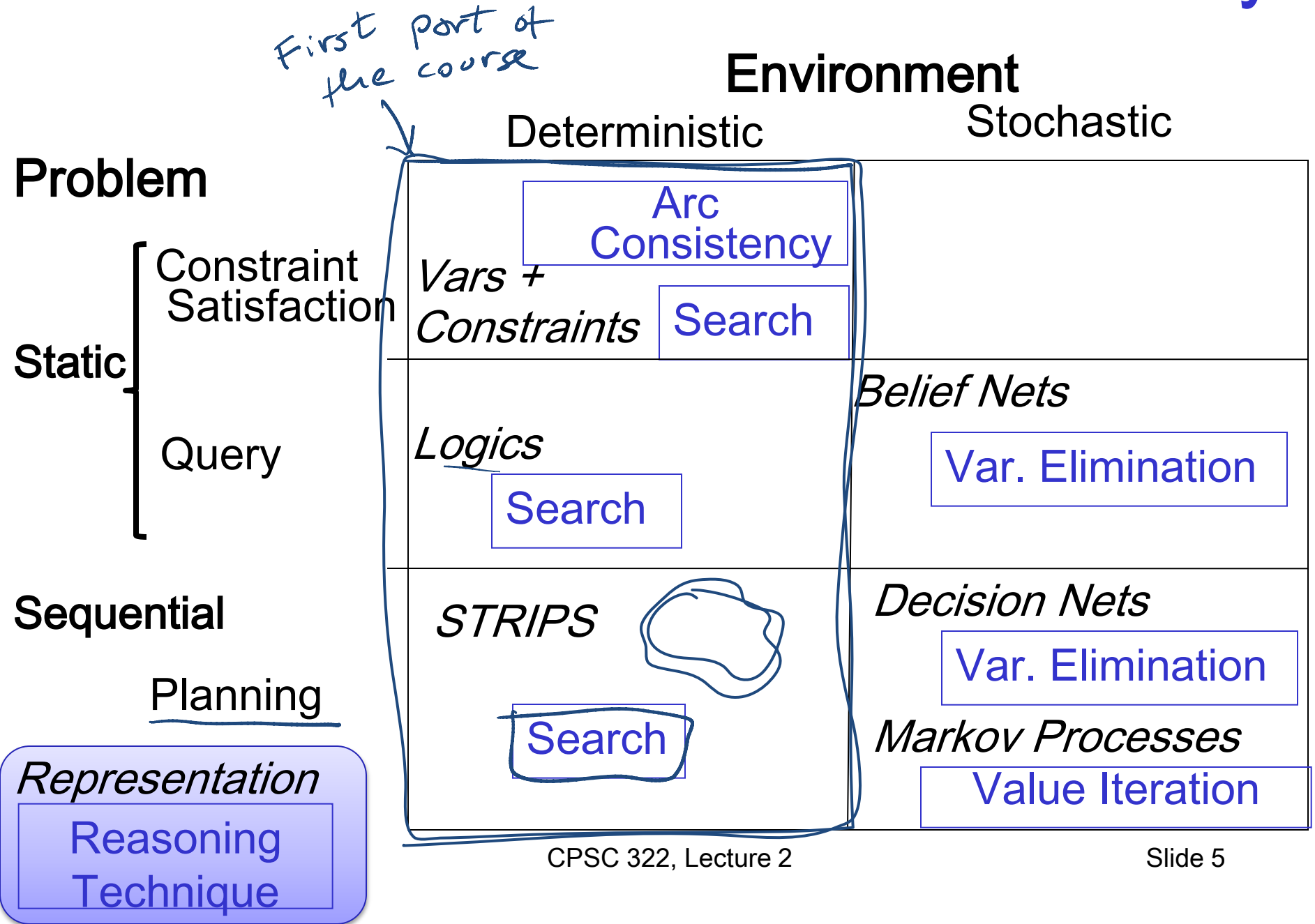
Has perfect knowledge of the effect that its actions can have on the environment



Both of the above

None of the above

Modules we'll cover in this course: R&Rsys



Lecture Overview

- **Simple Agent and Examples**
- Search Space Graph
- Search Procedure
- Criteria to compare Search Strategies

- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- Search with Costs

Simple Planning Agent

Deterministic, goal-driven agent

- Agent is in a **start state**
- 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**

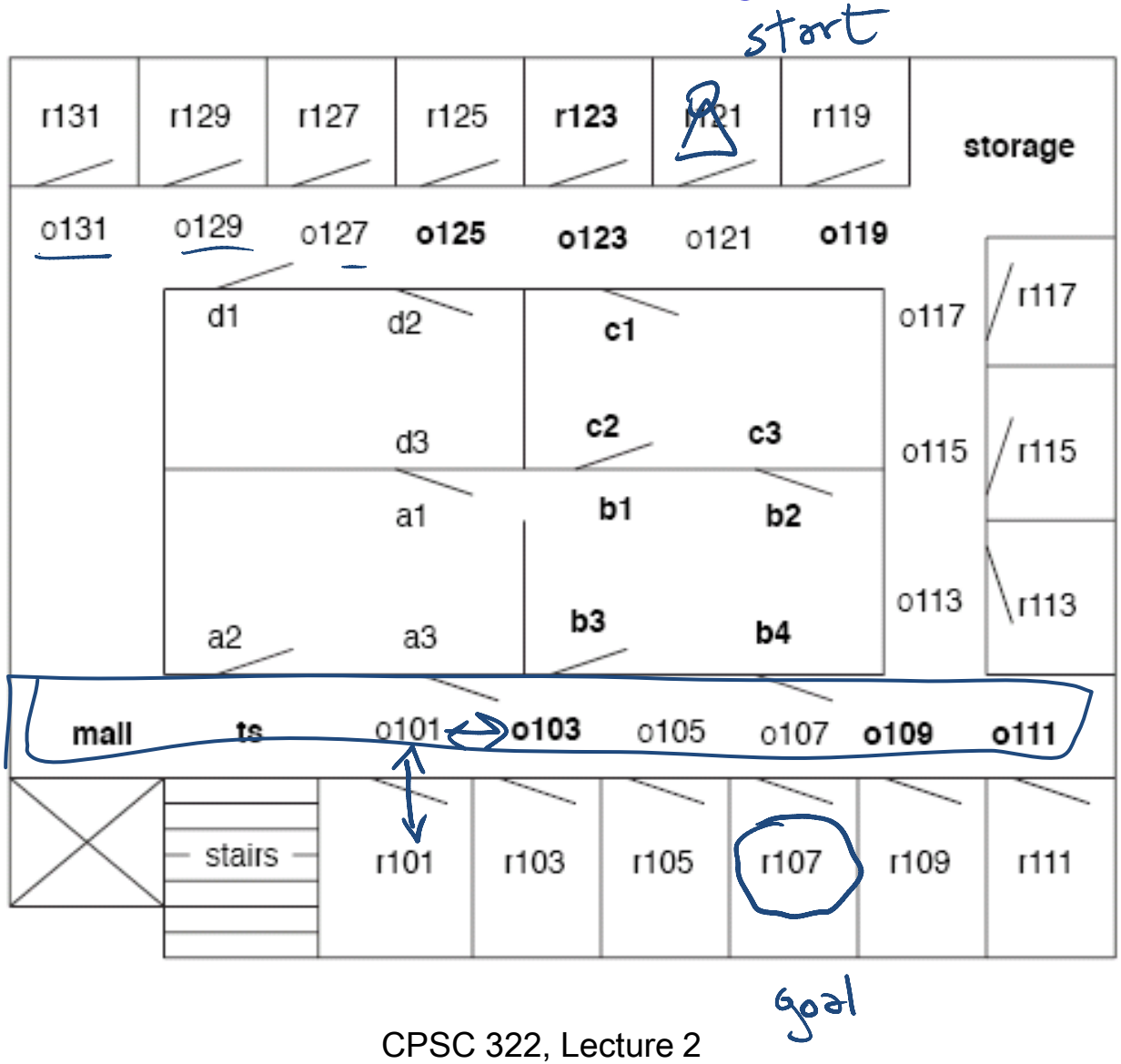
Three examples

1. A delivery robot planning the route it will take in a bldg. to get from one room to another

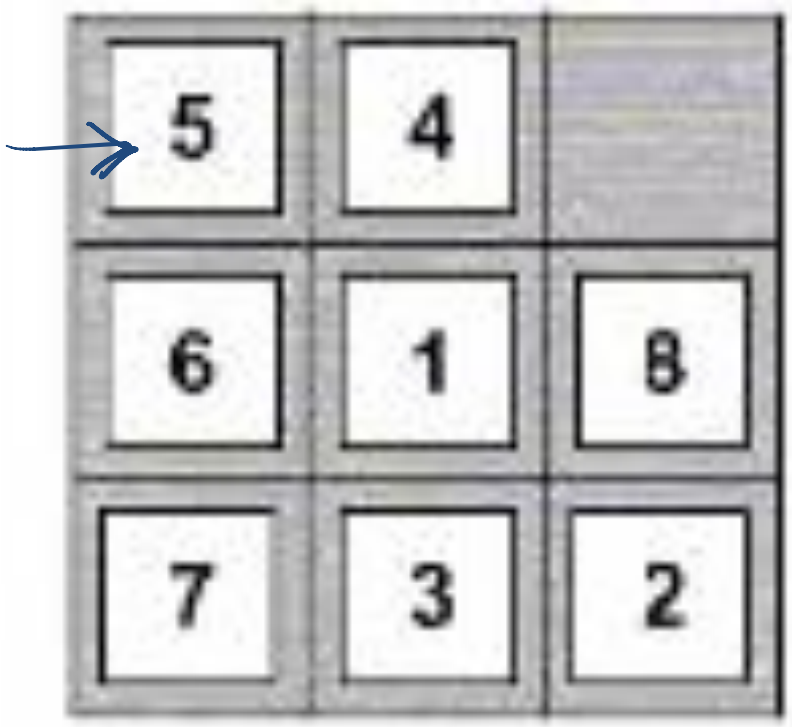
2. Solving an 8-puzzle

3. Vacuum cleaner world

Example1: Delivery Robot



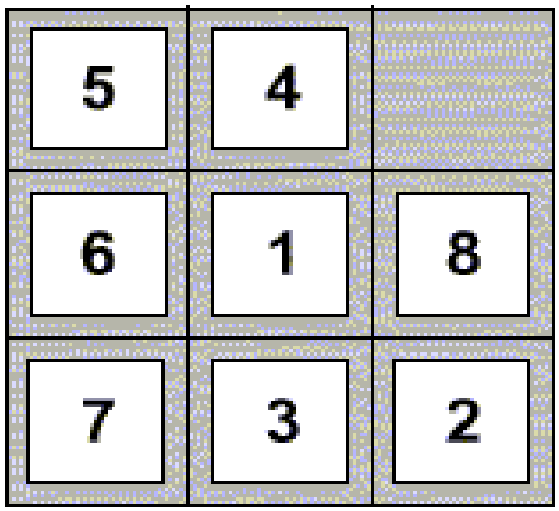
Example 2: 8-Puzzle?



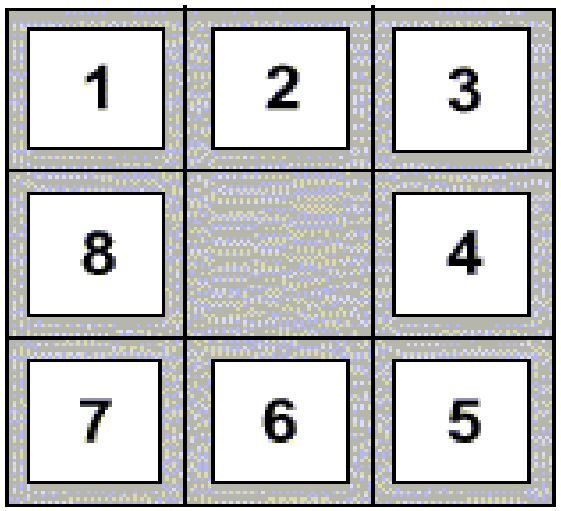
Possible start state

Goal state

Eight Puzzle



Start State



Goal State

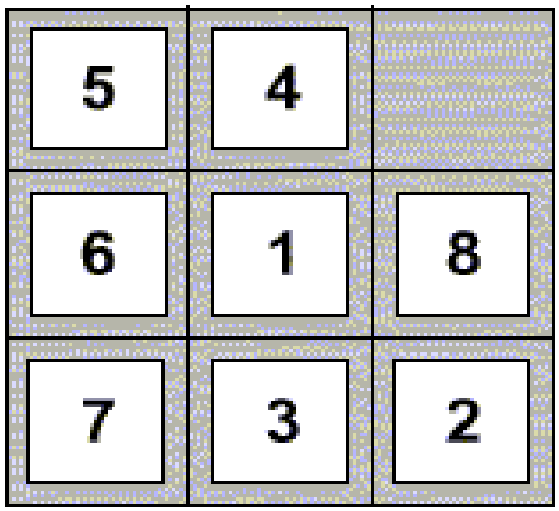
States: each state specifies which number/blank occupies each of the 9 tiles

HOW MANY STATES ? 8^9 2^9 9^9 $9!$

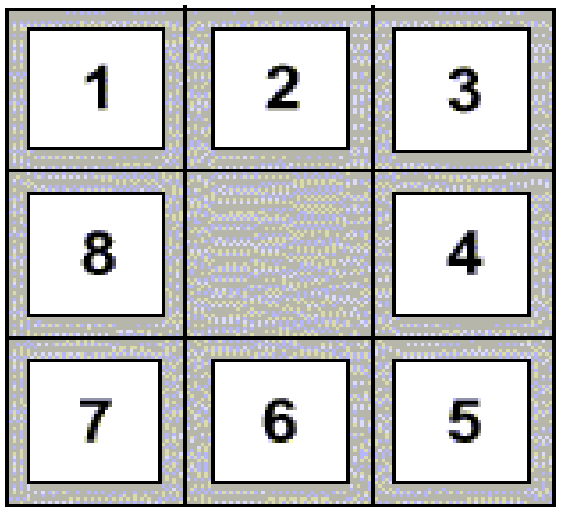
Operators:

Goal:

Eight Puzzle



Start State



Goal State

States: each state specifies which number/blank occupies each of the 9 tiles

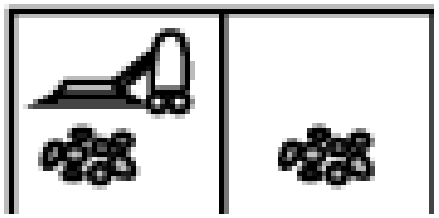
HOW MANY STATES ? *9!*

Operators: blank moves left, right, up down

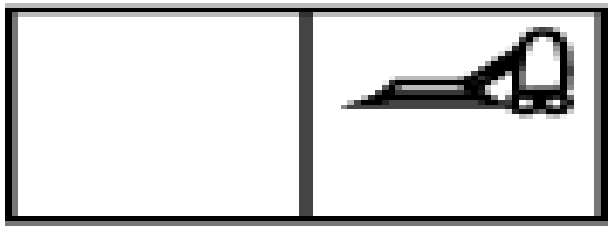
Goal: configuration with numbers in right sequence

Example: vacuum world

- States
  - Two rooms: r1, r2
  - Each room can be either dirty or not
  - Vacuuming agent can be in either in r1 or r2



Possible start state



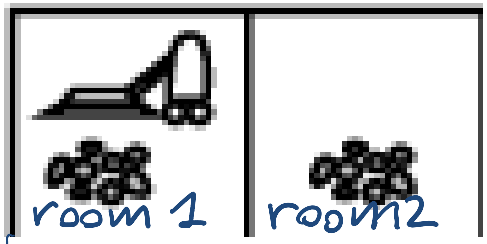
Possible goal state

Example: vacuum world

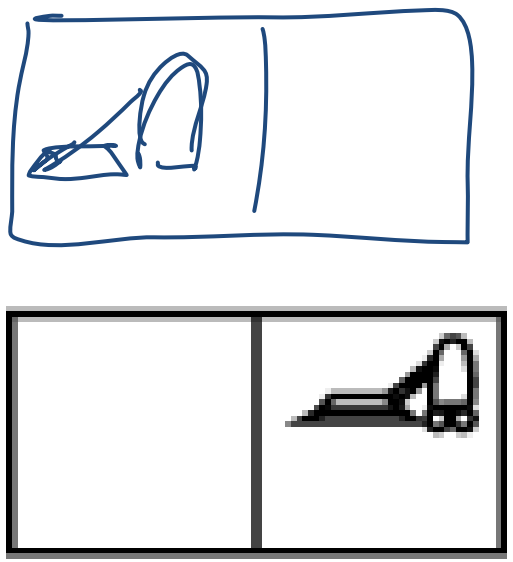
loc 2 values $\{r_1, r_2\}$

r_1-clean T/F

r_2-clean



Possible start state



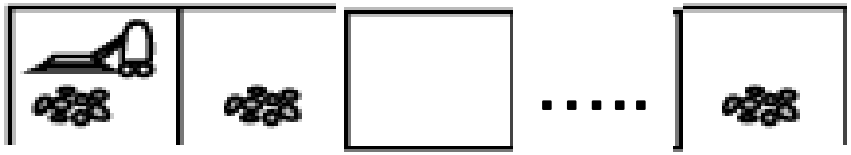
Goal state

# of states

2^2

2^2

can be subset of states



Suppose we have the same problem with k rooms.

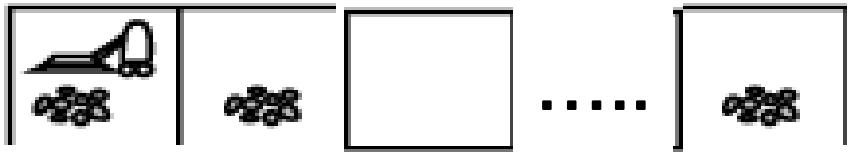
The number of states is….

k^3

$k * 2k$

$k * 2^k$

$2 * k^k$



Suppose we have the same problem with k rooms.

The number of states is….

$k * 2^k$

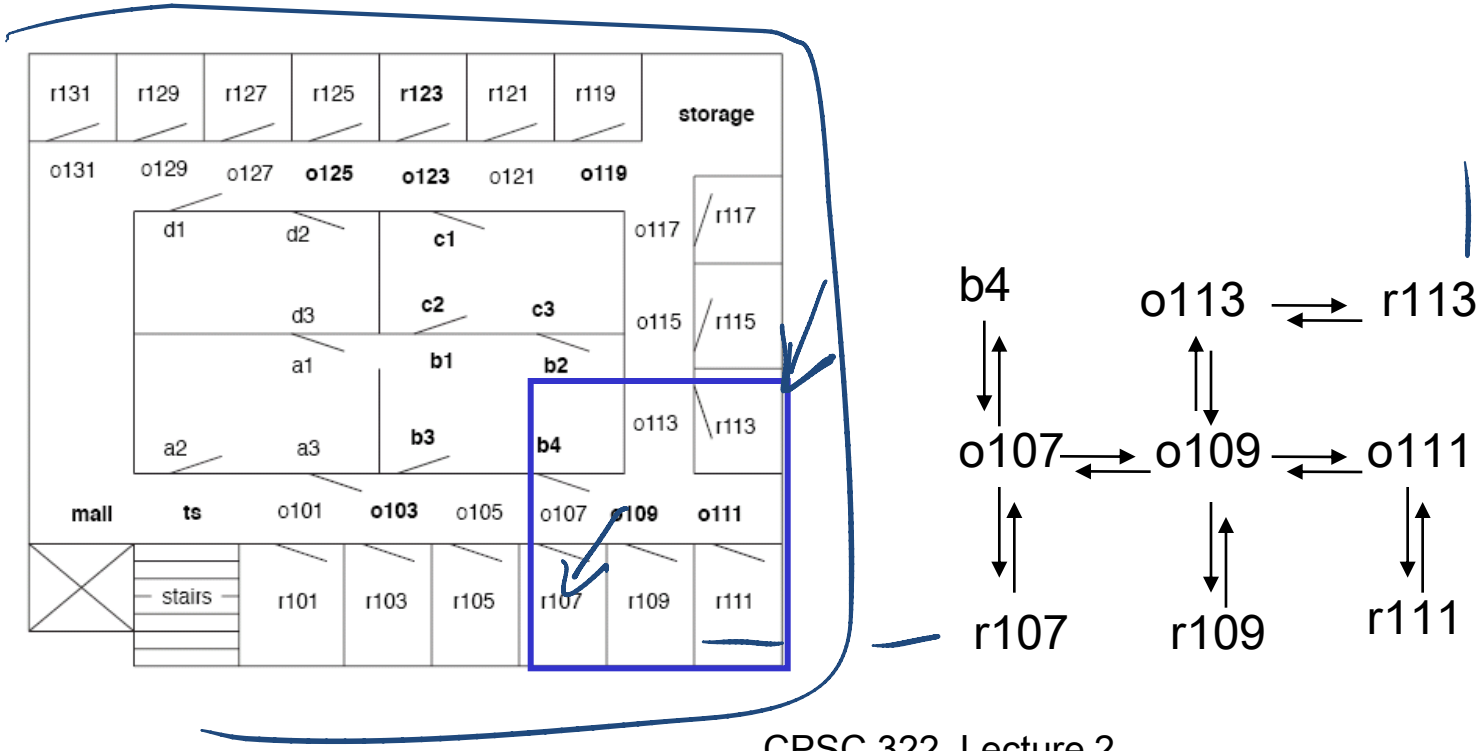
Lecture Overview

- Simple Agent and Examples
- **Search Space Graph**
- Search Procedure
- Criteria to compare Search Strategies

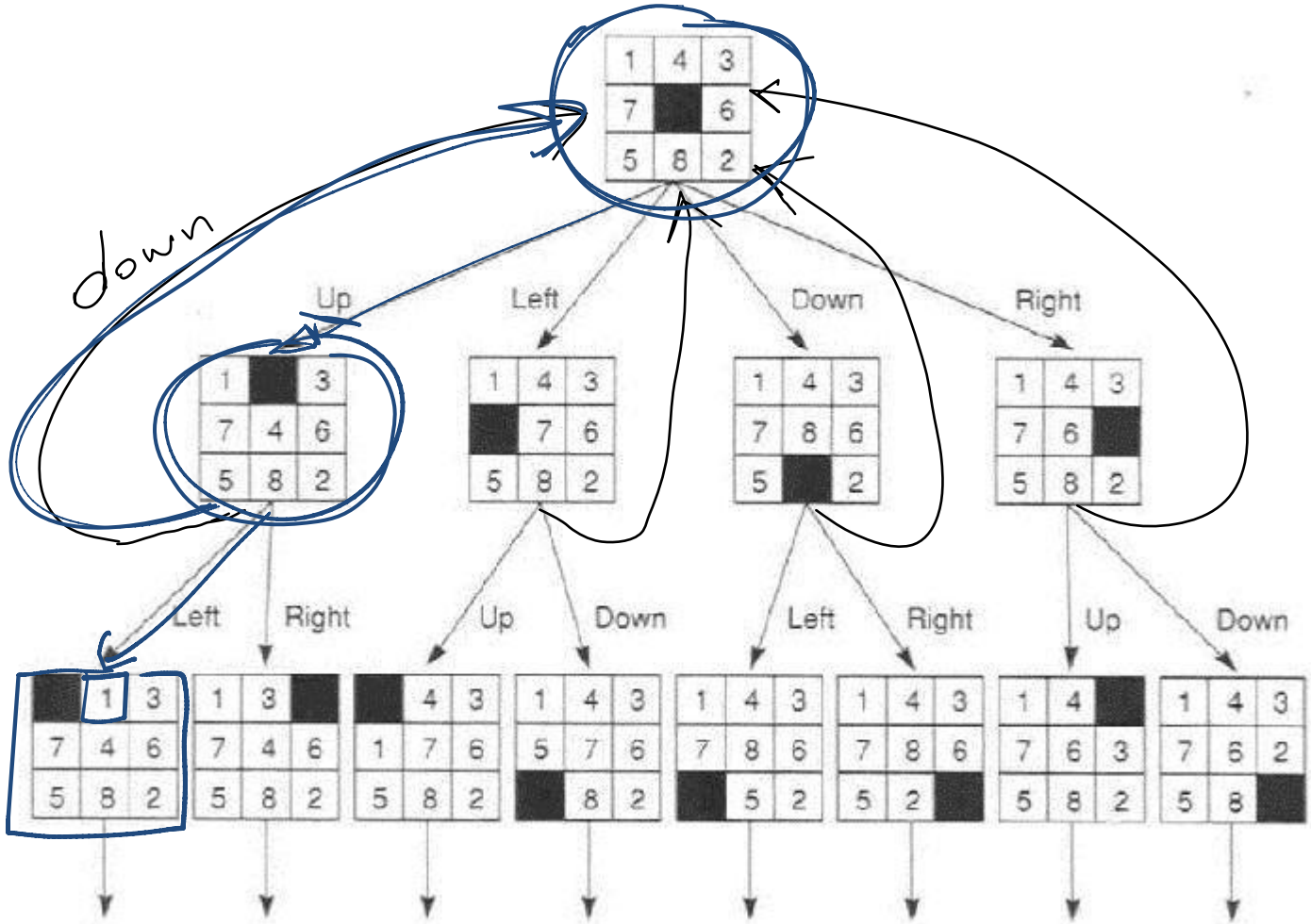
- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- Search with Costs

How can we find a solution?

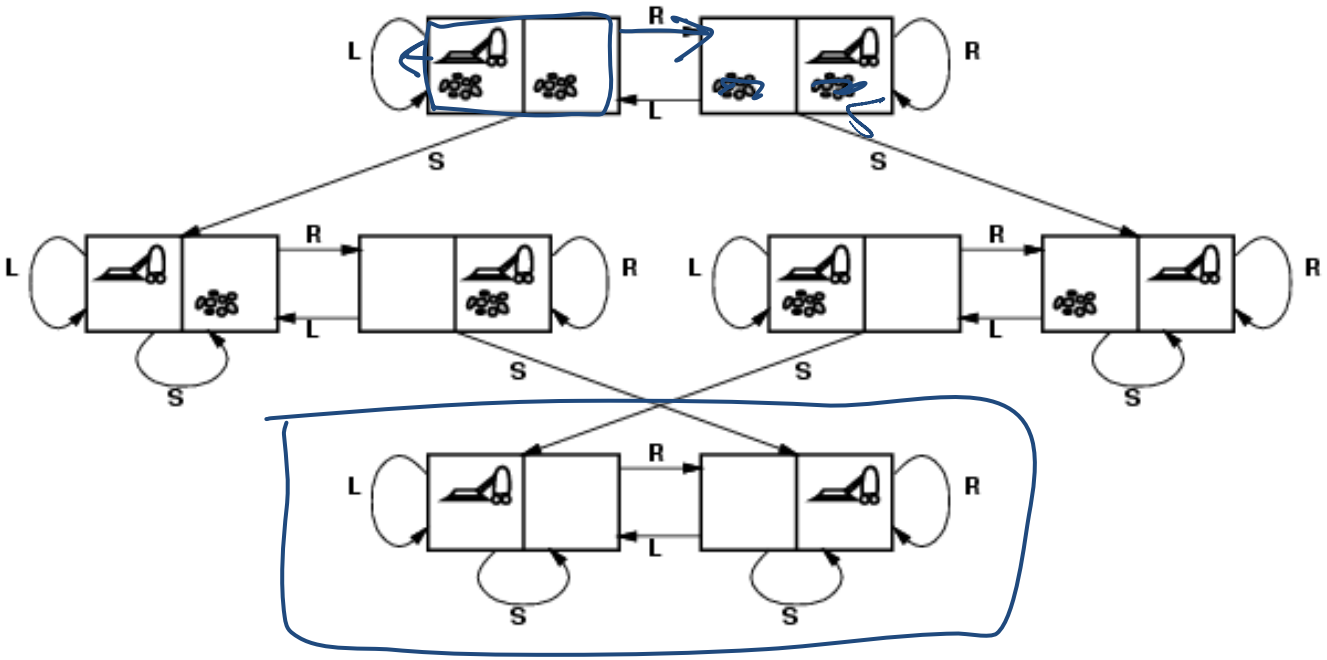
- How can we find a sequence of actions and their appropriate ordering that lead to the goal?
- Define underlying **search space graph** where **nodes are states** and **edges are actions**.



Search space for 8puzzle



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

Lecture Overview

- Simple Agent and Examples
- Search Space Graph
- **Search Procedure**
- Criteria to compare Search Strategies

- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- Search with Costs

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

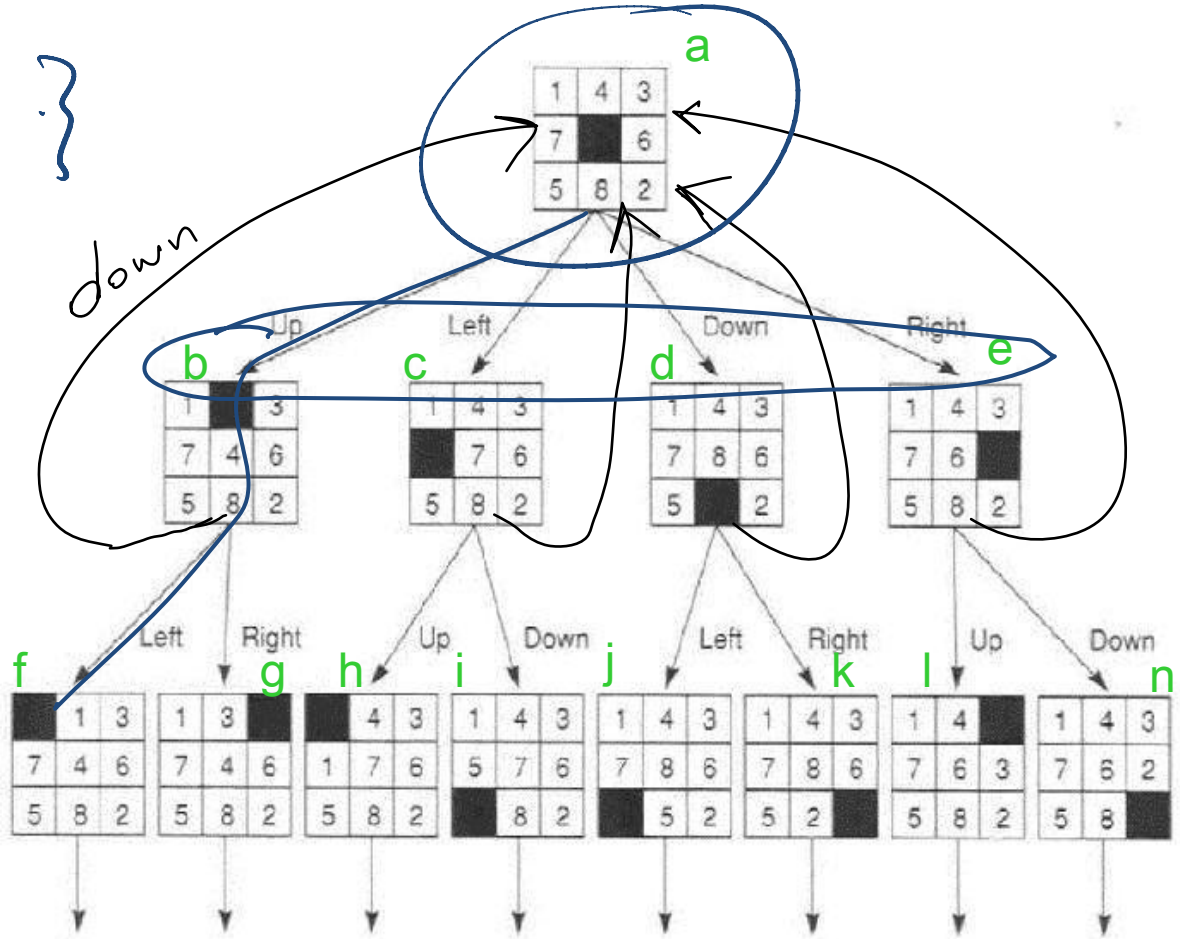
Given a start node and goal nodes, a ***solution*** is a path from a start node to a goal node.

Examples for graph formal def.

N {a b _ _ }

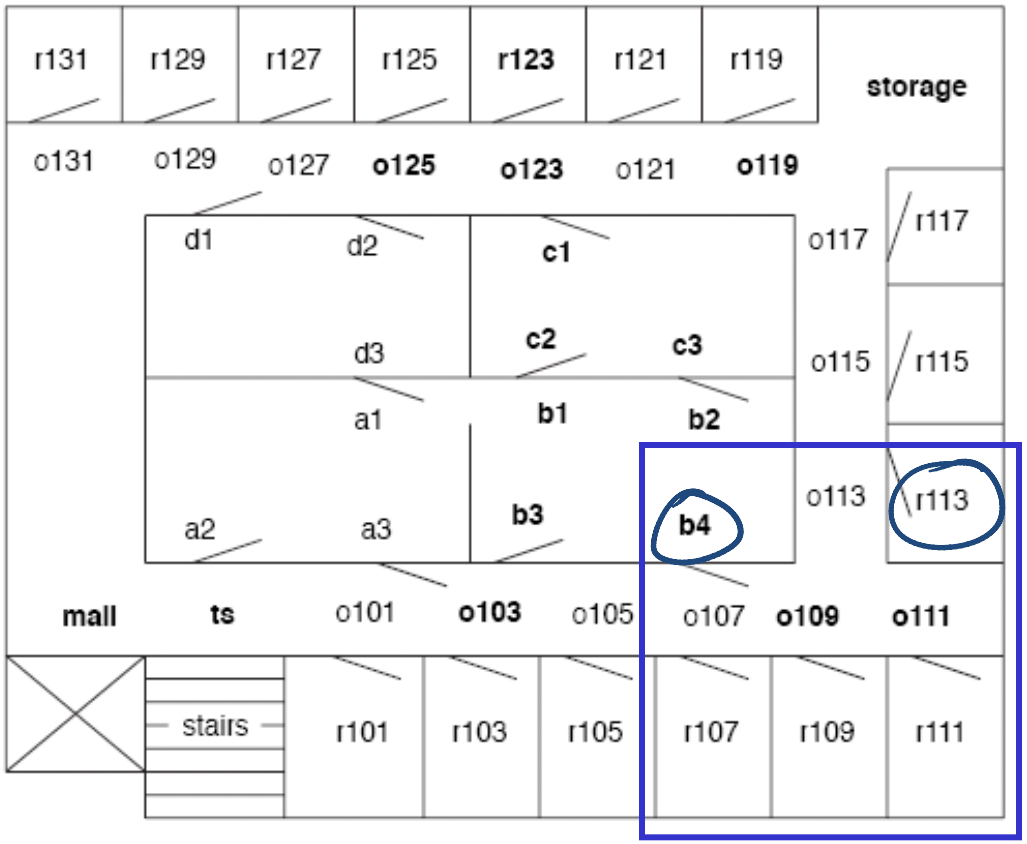
A {<a,b><a,c>}

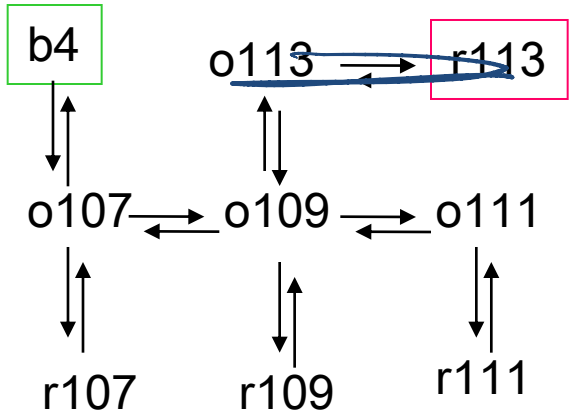
. . . .



Examples of solution

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





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.

Generic Search Algorithm

Input: a graph, a start node, 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,….,n_k> from *frontier;*

  If *goal*(n_k)

    return <n_o,….,n_k>;

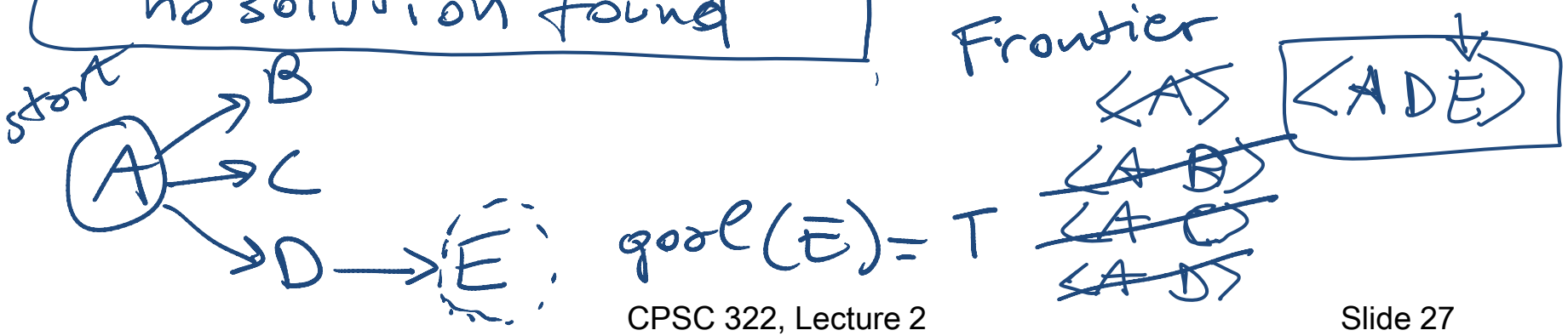
  For every neighbor *n* of n_k

    add <n_o,….,n_k, *n*> to *frontier;*

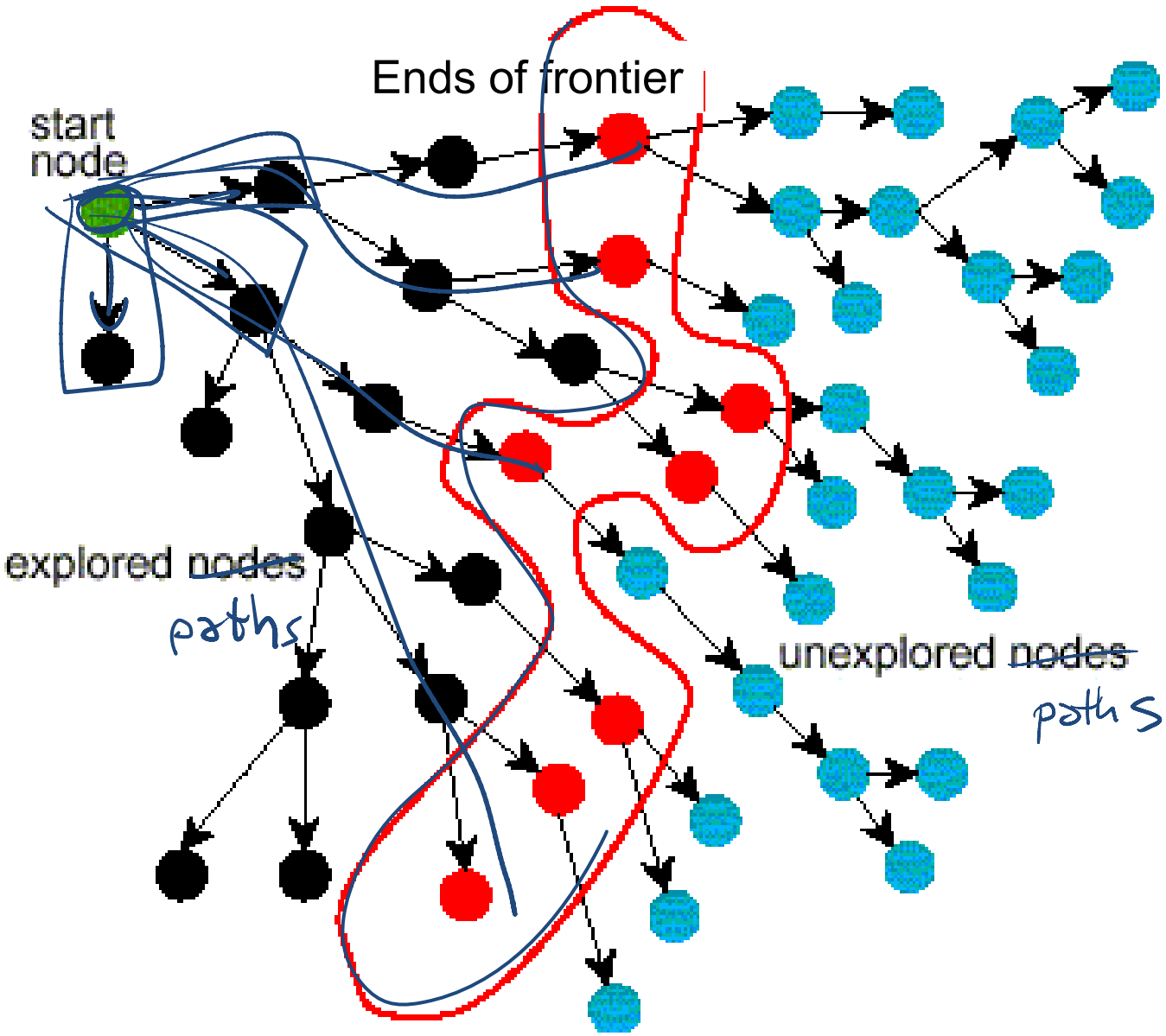
end

with cycles may get into infinite loop

no solution found



Problem Solving by Graph Searching

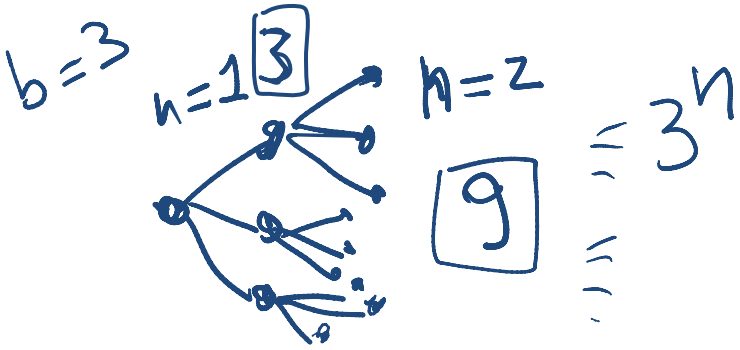


Branching Factor

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 a node?



nb b^n n^b n/b

Summary Generic Search Approach

- 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.

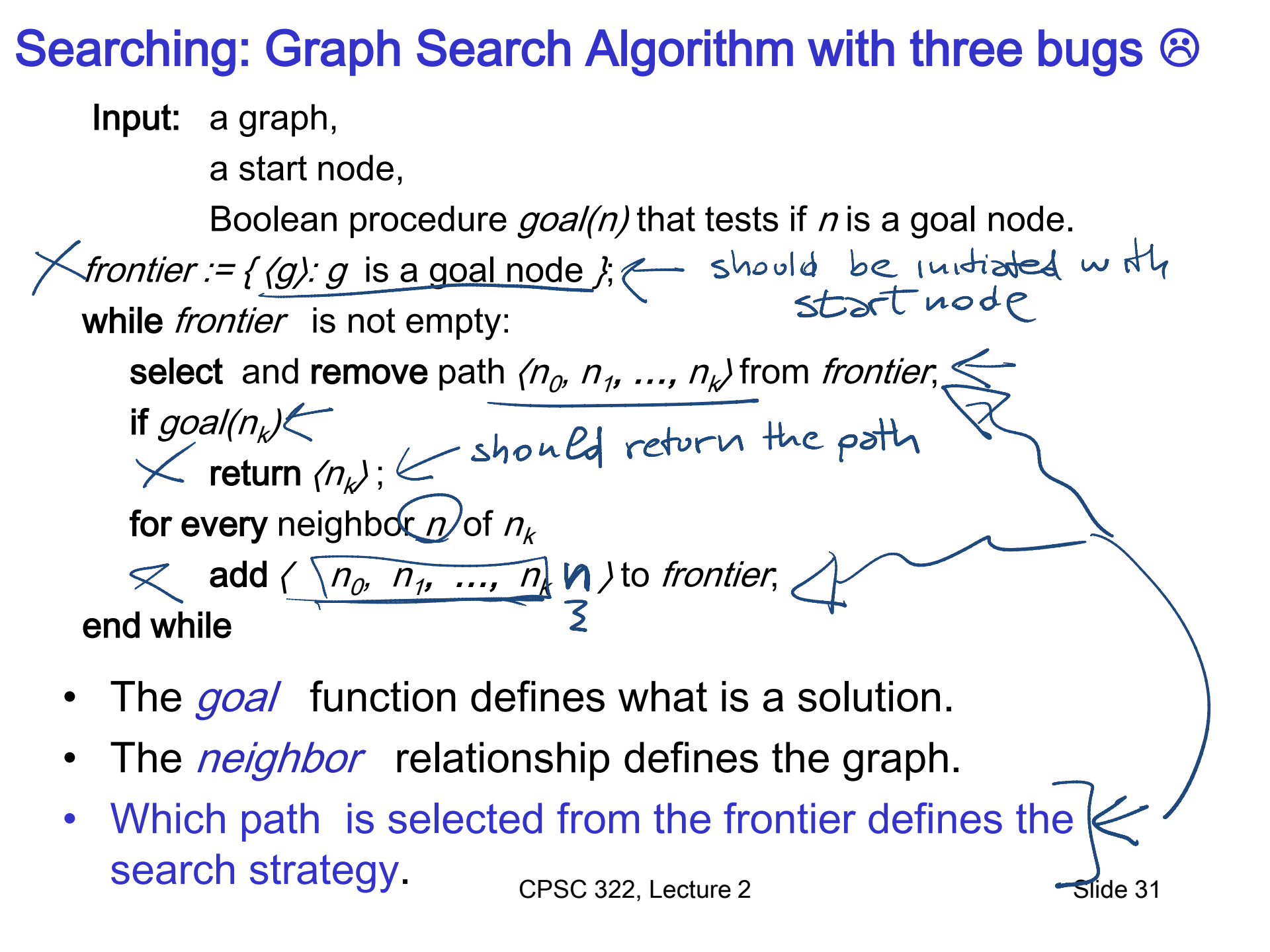
Searching: Graph Search Algorithm with three bugs ☹

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

frontier := { ⟨*g*⟩: *g* is a goal node };
while *frontier* is not empty:
  select and **remove** path ⟨n_0, n_1, …, n_k⟩ from *frontier*;
  if *goal*(n_k)
    return ⟨n_k⟩;
  for every neighbor *n* of n_k
    add ⟨ n_0, n_1, …, n_k *n* ⟩ to *frontier*;
end while

should be initiated with start node

should return the path



- The *goal* function defines what is a solution.
- The *neighbor* relationship defines the graph.
- Which path is selected from the frontier defines the search strategy.

Lecture Overview

- Simple Agent and Examples
- Search Space Graph
- Search Procedure
- **Criteria to compare Search Strategies**

- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- Search with Costs

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

Comparing Searching Algorithms: Complexity

Def. (time complexity)

The time complexity of a search algorithm is an expression for the **worst-case** amount of time it will take to run,

- expressed in terms of the **maximum path length *m*** and the **maximum branching factor *b***.

Def. (space complexity) : The space complexity of a search algorithm is an expression for the **worst-case** amount of memory that the algorithm will use (*number of nodes*),

- Also expressed in terms of ***m* and *b***.

Learning Goals for today's Part1

- **Identify** real world examples that make use of deterministic, goal-driven planning agents
- **Assess** the size of the search space of a given search problem.
- **Implement** the generic solution to a search problem.

How many possible states

see also Mars Explorer Lecture 1

Lecture Overview

- Simple Agent and Examples
- Search Space Graph
- Search Procedure
- Criteria to compare Search Strategies

30 mins BREAK

- **Simple (Uninformed) Search Strategies**
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- Search with Costs

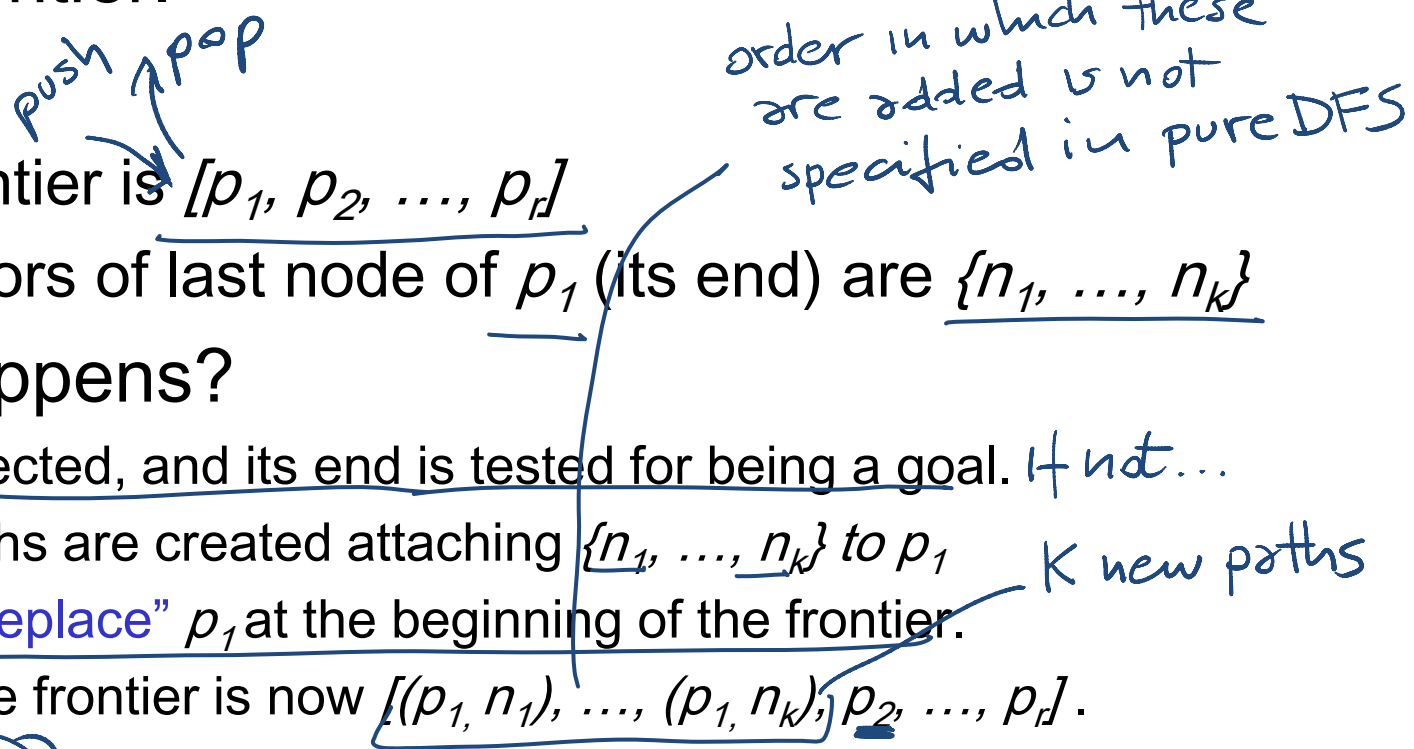
Depth-first Search: DFS

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

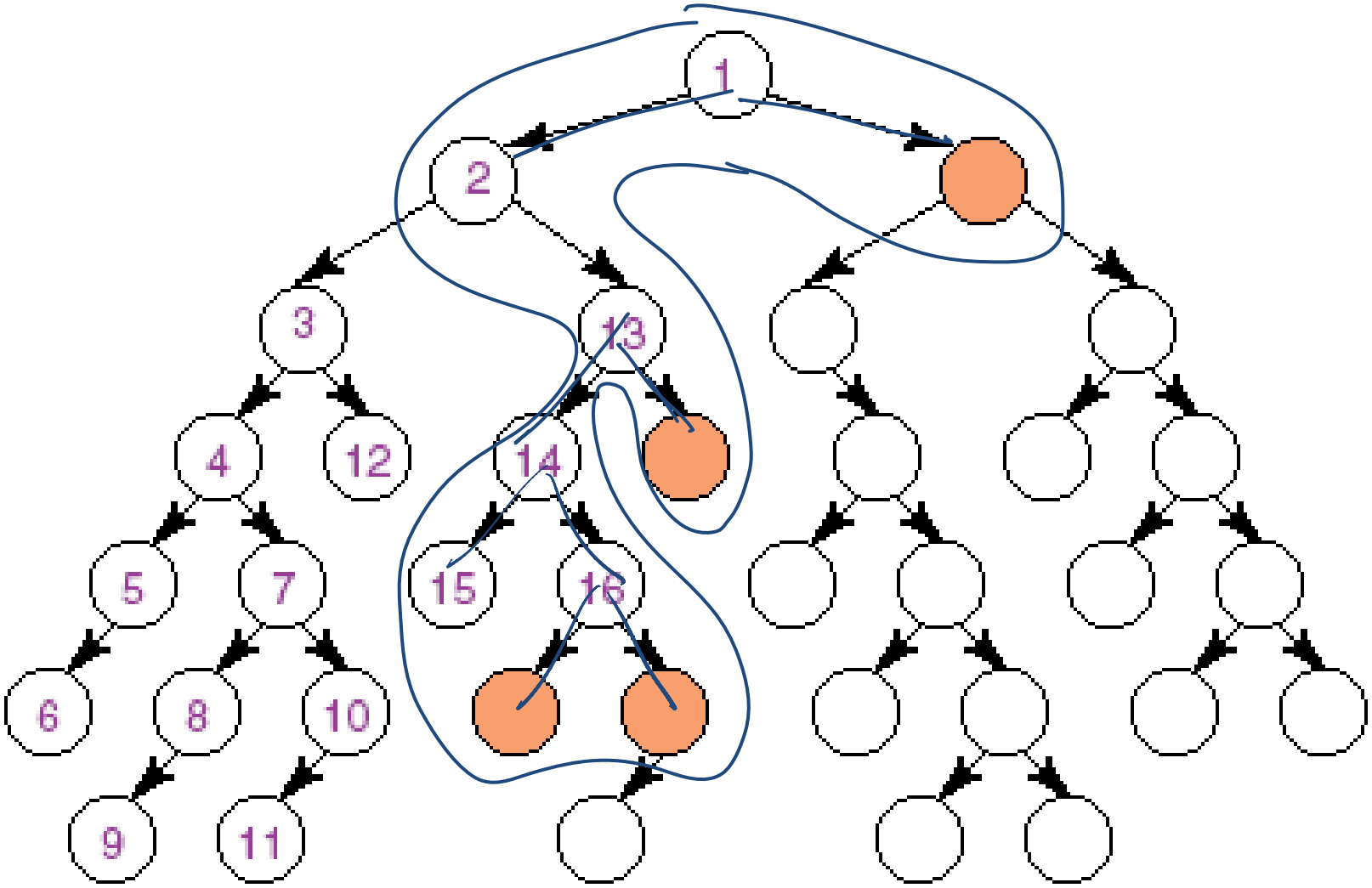
Example:

- 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.
  - New paths are created attaching $\{n_1, \ldots, n_k\}$ to p_1
  - These "replace" p_1 at the beginning of the frontier.
  - Thus, the frontier is now $[(p_1, n_1), \ldots, (p_1, n_k), p_2, \ldots, p_r]$.
  - *NOTE:* p_2 is only selected when all paths extending p_1 have been explored.



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

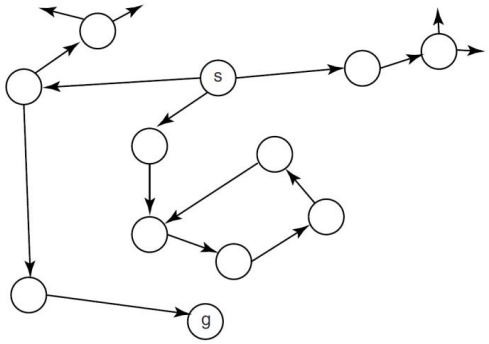


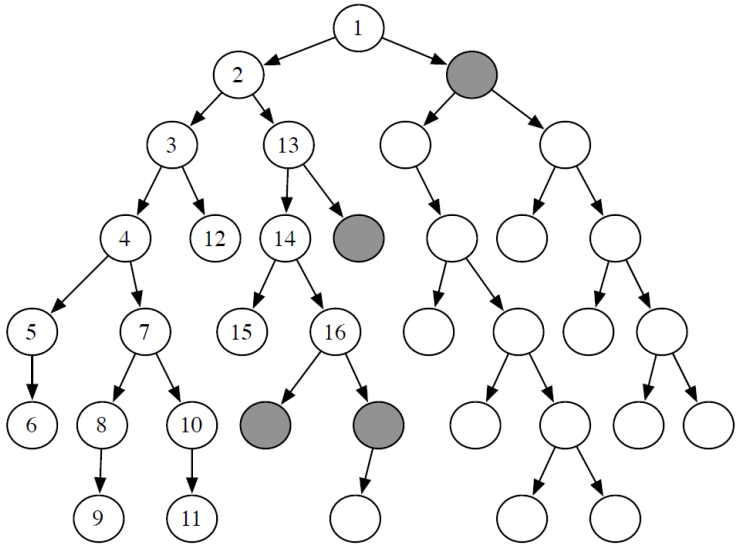
Analysis of DFS

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

Is DFS complete?

No



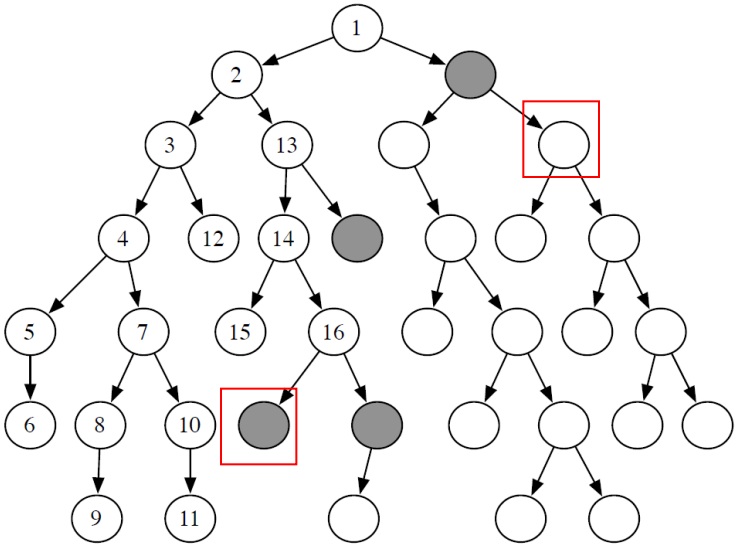


- If there are cycles in the graph, DFS may get "stuck" in one of them
- see this in AISpace by loading "Cyclic Graph Examples" or by adding a cycle to "Simple Tree"
  - e.g., click on "Create" tab, create a new edge from N7 to N1, go back to "Solve" and see what happens

Analysis of DFS

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

Is DFS optimal? Yes No

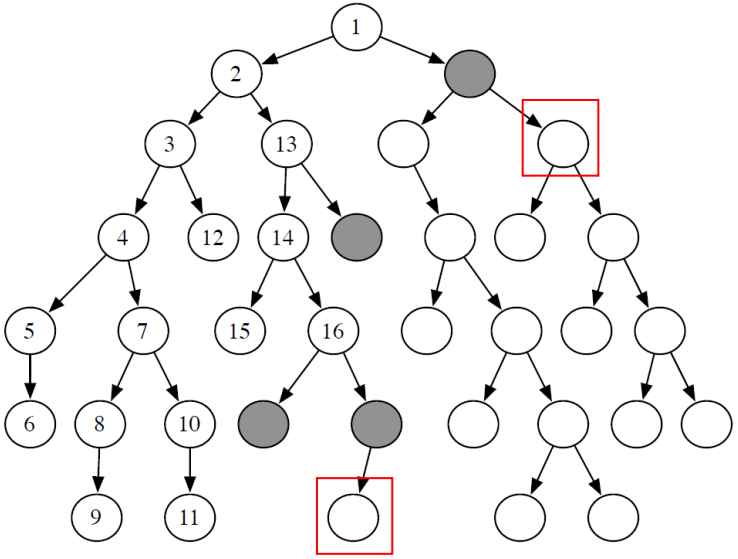


- E.g., goal nodes: red boxes

Analysis of DFS

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

Is DFS optimal? No



- It can "stumble" on longer solution paths before it gets to shorter ones.
  - E.g., goal nodes: red boxes

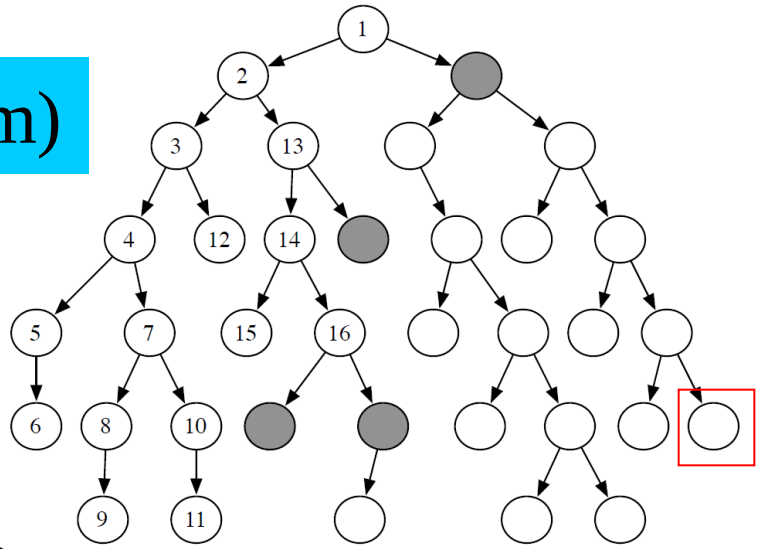
- see this in AISpace by loading "Extended Tree Graph" and set N6 as a goal
  - e.g., click on "Create" tab, right-click on N6 and select "set as a goal node"

Analysis of DFS

Def.: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of
- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$



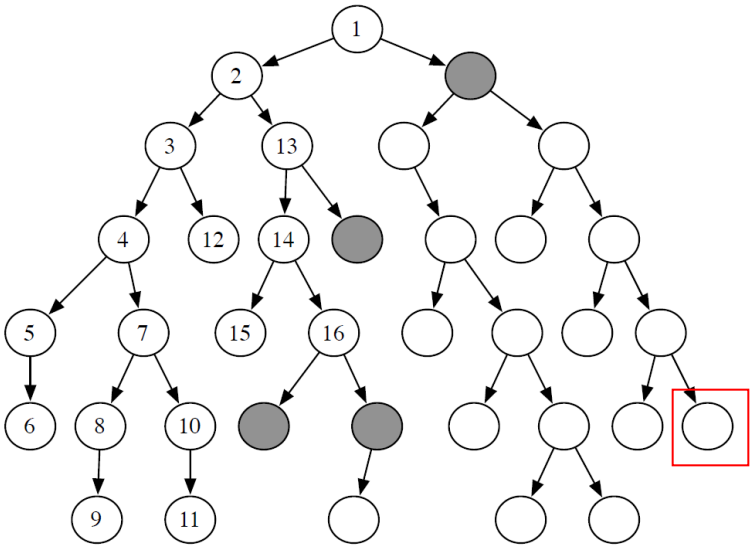
- E.g., single goal node -> red box

Analysis of DFS

Def.: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of
- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$



- In the worst case, must examine every node in the tree
  - E.g., single goal node -> red box

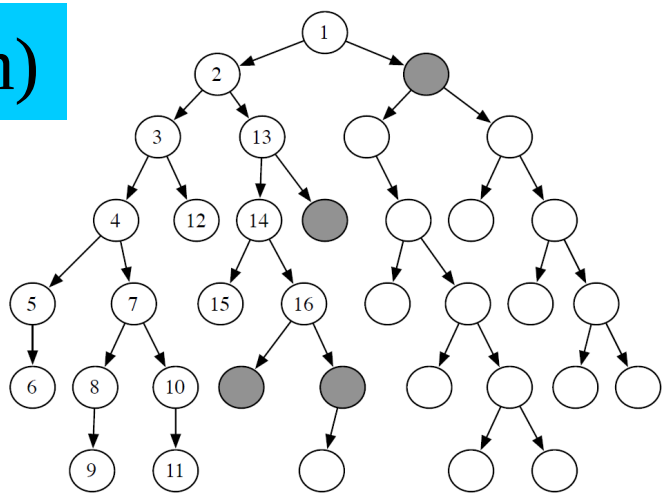
Analysis of DFS

Def.: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length m
- maximum forward branching factor b.

- What is DFS's space complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$



See how this works in AIspace

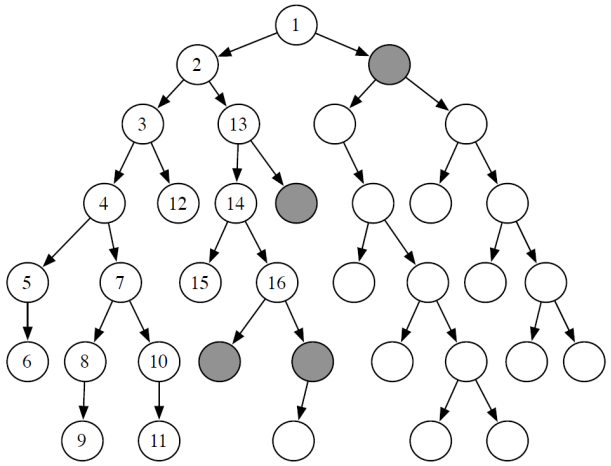
Analysis of DFS

Def.: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximum number of nodes on the frontier), expressed in terms of

- maximum path length m
- maximum forward branching factor b.

- What is DFS's space complexity, in terms of m and b ?

O(bm)



- for every node in the path currently explored, DFS maintains a path to its unexplored siblings in the search tree
  - Alternative paths that DFS needs to explore
- The longest possible path is m, with a maximum of b-1 alterative paths per node

See how this works in AIspace

Analysis of DFS: Summary

- Is DFS complete? NO
  - Depth-first search isn't guaranteed to halt on graphs with cycles.
  - However, DFS ***is*** complete for finite acyclic graphs.
- Is DFS optimal? NO
  - It can "stumble" on longer solution paths before it gets to shorter ones.
- What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b* ?
  - $O(b^m)$: must examine every node in the tree.
  - Search is unconstrained by the goal until it happens to stumble on the goal.
- What is the *space complexity*?
  - $O(bm)$
  - the longest possible path is *m*, and for every node in that path must maintain a fringe of size *b*.

Depth-first Search: When it is appropriate?

Appropriate

- Space is restricted (complex state representation e.g., robotics)
- There are many solutions, perhaps with long path lengths, particularly for the case in which all paths lead to a solution

Inappropriate

- Cycles
- There are shallow solutions
- if you care about optimality!

Why DFS need to be studied and understood?

- It is simple enough to allow you to learn the basic aspects of searching (When compared with breadth first)

- It is the basis for a number of more sophisticated / useful search algorithms

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.

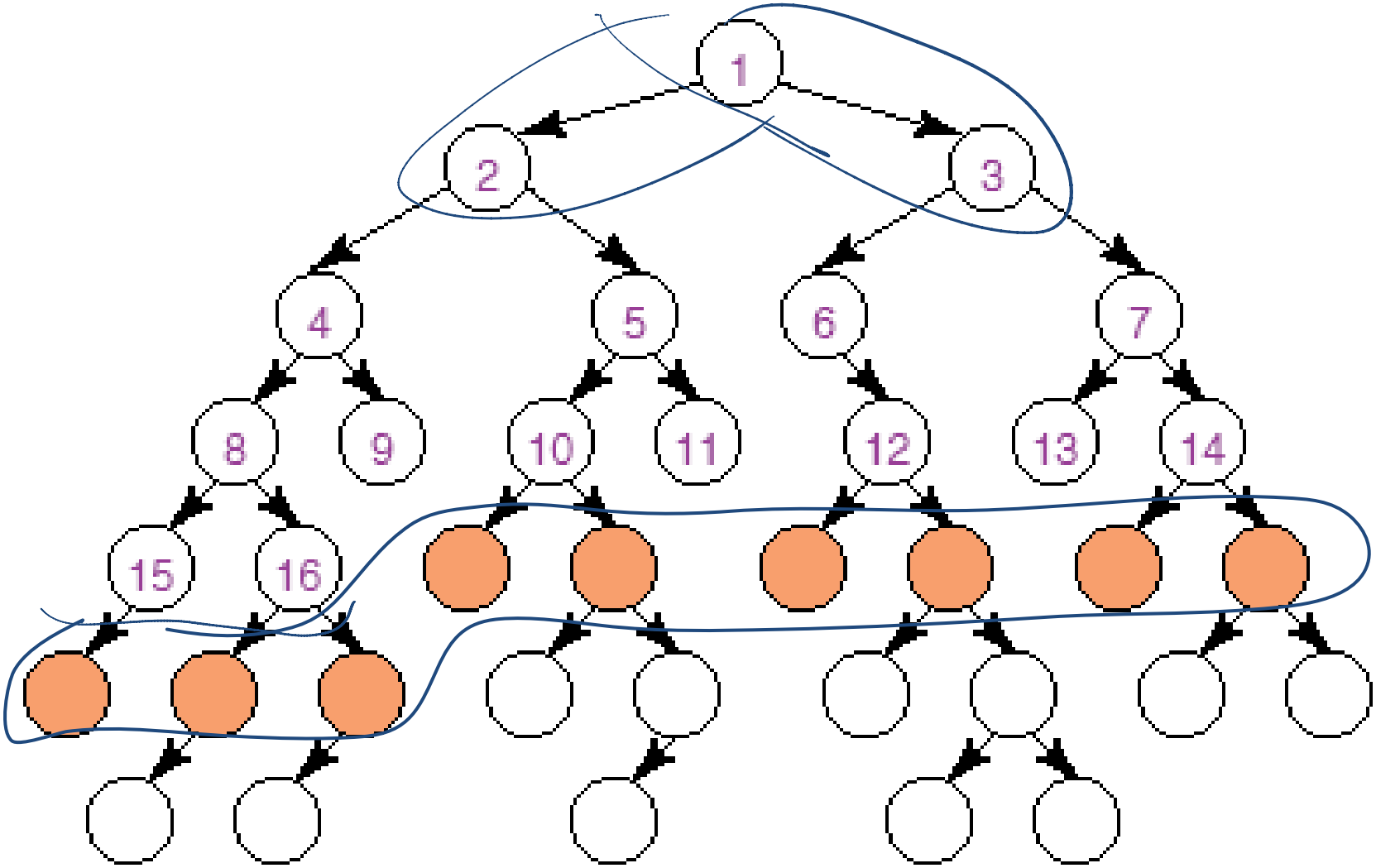
Example:

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.

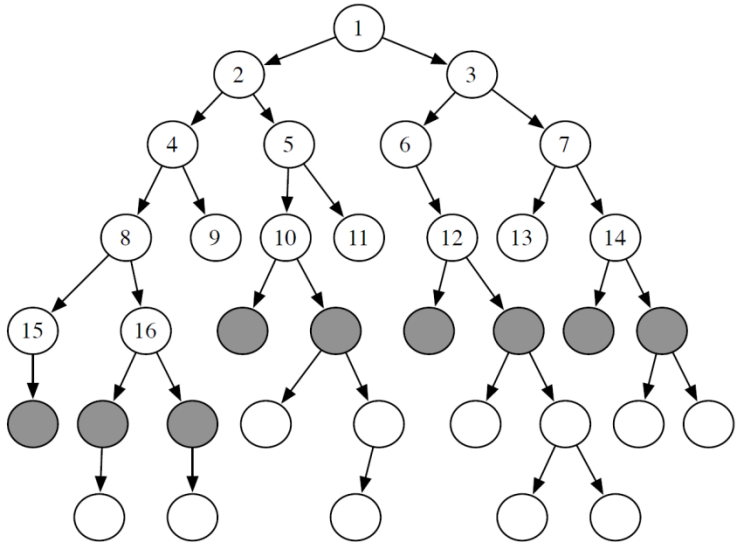
Illustrative Graph - Breadth-first Search



Analysis of BFS

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

Is BFS complete? Yes No

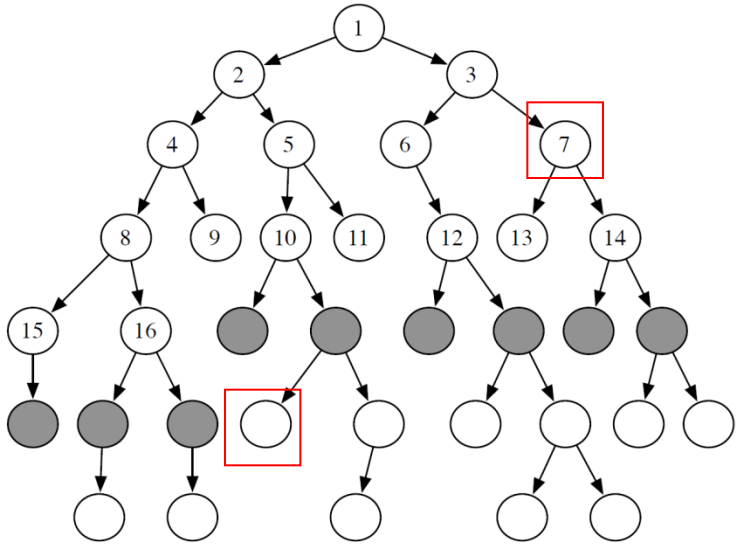


Analysis of BFS

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

Is BFS optimal? Yes No

- E.g., two goal nodes: red boxes



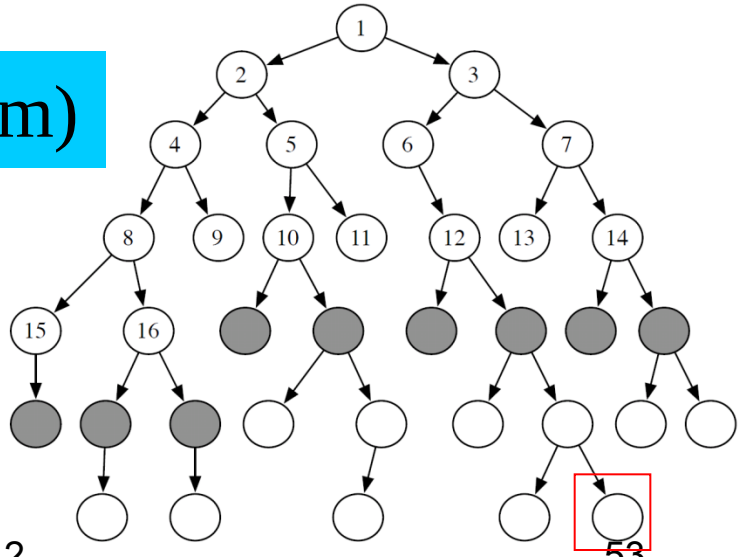
Analysis of BFS

Def.: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of
- maximum path length *m*
- maximum forward branching factor *b*.

- What is BFS's time complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

- E.g., single goal node: red box



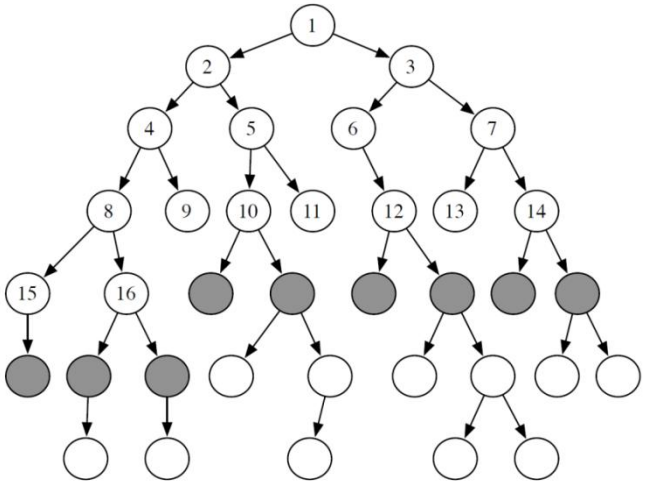
Analysis of BFS

Def.: The space complexity of a search algorithm is the worst case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length *m*
- maximum forward branching factor *b*.

- What is BFS's space complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$



- How many nodes at depth m?

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes

  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)

- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is ? $O(b^m)$? must examine every node in the tree.
  - The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.
- What is the space complexity?
  - Space complexity is ? $O(b^m)$?

Using Breadth-first Search

- When is BFS appropriate?
  - space is not a problem
  - it's necessary to find the solution with the fewest arcs
  - although all solutions may not be shallow, at least some are

optimality

- When is BFS inappropriate?
  - space is limited
  - all solutions tend to be located deep in the tree
  - the branching factor is very large

What have we done so far?

GOAL: study search, a set of basic methods underlying many intelligent agents

AI agents can be very complex and sophisticated

Let's start from a very simple one, **the deterministic, goal-driven agent** for which: the sequence of actions and their appropriate ordering is the solution

We have looked at two search strategies DFS and BFS:

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

Recap: Comparison of DFS and BFS

	Complete	Optimal	Time	Space
DFS	N (Y if no cycles)	N	$O(b^m)$	$O(bm)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$

Lecture Overview

- Simple Agent and Examples
- Search Space Graph
- Search Procedure
- Criteria to compare Search Strategies

- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- **Uninformed Iterative Deepening (IDS)**
- Search with Costs

Iterative Deepening (sec 3.6.3)

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

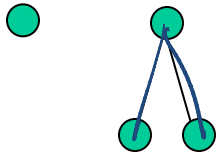
	Complete	Optimal	Time	Space
DFS	N	N	b^m	mb
BFS	Y	Y	b^m	b^m
IDS	Y	Y	b^m	mb

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

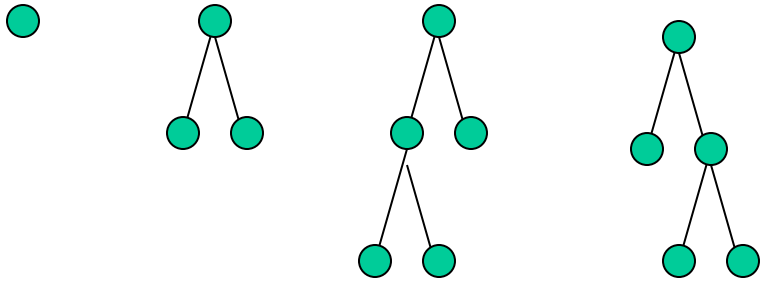
Iterative Deepening in Essence

- **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….

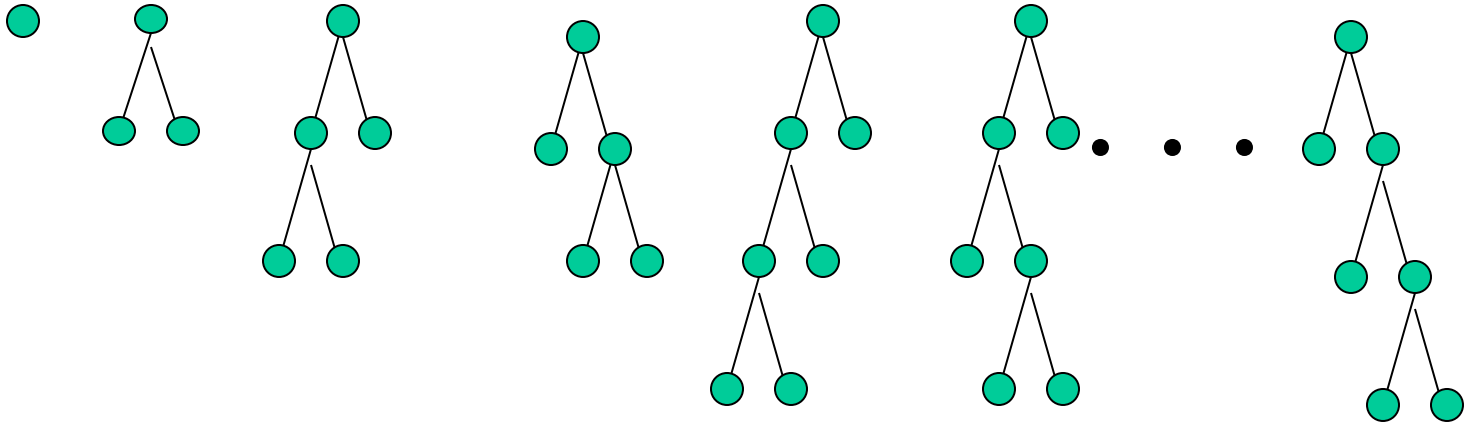
depth = 1



depth = 2

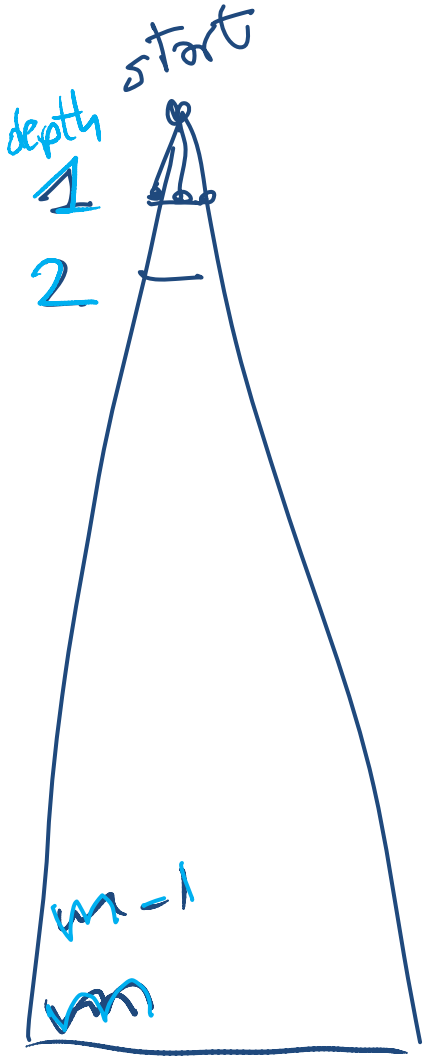


depth = 3



(Time) Complexity of Iterative Deepening

Complexity of solution at depth m with branching factor b



Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS
b	1	m
b^2	1	$m-1$
...	...	...
b^m	1	1

total # of paths created by IDS

$\rightarrow m\,b$

$m-1+b^2$

$\vdots$

$2b^{m-1}$

b^m

(Time) Complexity of Iterative Deepening

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

Total # of paths generated

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

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

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

B

$A < B$

$= \left(\frac{b}{b-1} \right)^2$

$b = 2$ → 4

$b = 3$ → $\frac{9}{4} = 2.25$

$b = 4$ → $\frac{16}{9} < 2$

Further Analysis of Iterative Deepening DFS (IDS)

- Space complexity

  O(bm)

  - DFS scheme, only explore one branch at a time

- Complete?

  Yes

  - Only paths up to depth m, doesn't explore longer paths – cannot get trapped in infinite cycles, gets to a solution first

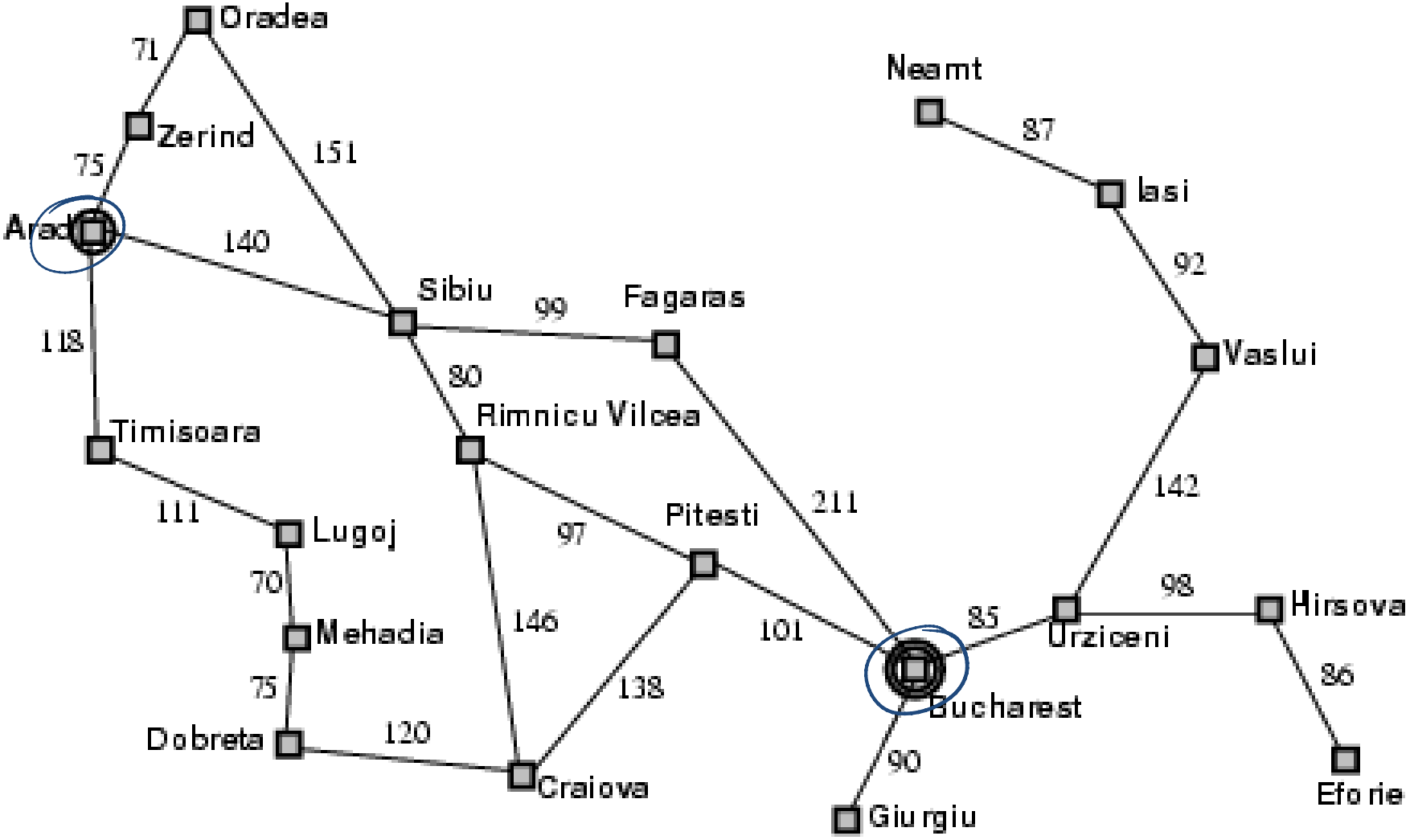
- Optimal? Yes

Lecture Overview

- Simple Agent and Examples
- Search Space Graph
- Search Procedure
- Criteria to compare Search Strategies

- Simple (Uninformed) Search Strategies
  - Depth First and Breadth First
- Uninformed Iterative Deepening (IDS)
- **Search with Costs**

Example: Romania



Search with Costs

Sometimes there are costs associated with arcs.

Definition (cost of a path)

The cost of a path is the sum of the costs of its arcs:

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

In this setting we often don't just want to find just any solution

- **we usually want to find the solution that minimizes cost**

Definition (optimal algorithm)

A search algorithm is optimal if it is complete, and only returns cost-minimizing solutions.

Lowest-Cost-First Search

- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
  - The frontier is a priority queue ordered by path cost
  - We say ``a path'' because there may be ties

- Example of one step for LCFS:
  - the frontier is $[\langle p_2, 5\rangle, \langle p_3, 7\rangle, \langle p_1, 11\rangle,]$
  - p_2 is the lowest-cost node in the frontier
  - "neighbors" of p_2 are $\{\langle p_9, 10\rangle, \langle p_{10}, 15\rangle\}$
- What happens?
  - p_2 is selected, and tested for being a goal.
  - Neighbors of p_2 are inserted into the frontier
  - Thus, the frontier is now $[\langle p_3, 7\rangle, \langle p_9, 10\rangle, \langle p_1, 11\rangle, \langle p_{10}, 15\rangle]$.
  - ? p_3 ? is selected next.
  - Etc. etc.

- When arc costs are equal LCFS is equivalent to..

DFS

BFS

IDS

None of the above

Analysis of Lowest-Cost Search (1)

- Is LCFS complete?
  - not in general: for instance, a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive $\geq \varepsilon > 0$

see how this works in AIspace:
- e.g, add arc with cost -20 to the simple search graph from N4 to S

- Is LCFS optimal?

YES **NO** **IT DEPENDS**

Analysis of Lowest-Cost Search (1)

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive

- Is LCFS optimal?
  - Not in general. Why not?
  - Arc costs could be negative: a path that initially looks high-cost could end up getting a ``refund".
  - However, LCFS ***is*** optimal if arc costs are guaranteed to be non-negative.

Analysis of Lowest-Cost Search

- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.

- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory.

Learning Goals for Search (up to today)

- Apply basic properties of search algorithms: completeness, optimality, time and space complexity of search algorithms.

	Complete	Optimal	Time	Space
DFS	N	N	b^m	bm
BFS	Y	Y	n	b^m
IDS	Y	Y	n	bm
LCFS	N Y if $c>0$	N Y if $c \geq 0$	b^m	b^m

Beyond uninformed search….

- So far the selection of the next path to examine (and possibly expand) is based on ….

- Assignment 1 out today
- Start working on the practice exercises

Next Class

- Heuristic Search
- Other Search Strategies

textbook.:

3.6 intro
3.6.1
3.7.1, 3.7.4

Learning Goals for Search (cont')
(up to today)

- Select the most appropriate search algorithms for specific problems.
  - BFS vs DFS vs IDS vs BidirS-
  - LCFS vs. BFS –
  - A* vs. B&B vs IDA* vs MBA*

uninformed

informed

- Define/read/write/trace/debug different search algorithms
  - With / Without cost
  - Informed / Uninformed

Lecture Overview

- Recap
- Criteria to compare Search Strategies
- Simple (Uninformed) Search Strategies
  - Depth First
  - Breadth First

start playing with

Learning Goals for today's class

- Apply basic properties of search algorithms: completeness, optimality, time and space complexity of search algorithms.

	Comp	Opt	time	Space
DFS	False	False	b^m	mb
BFS	True	True	b^m	b^m

- Select the most appropriate search algorithms for specific problems.
  - BFS vs DFS vs IDS vs ~~BidirS-~~
  - LCFS vs. BFS –
  - A* vs. B&B vs IDA* vs MBA*

next 4 lectures

informed

Lecture Overview

- Recap DFS vs BFS
- Uninformed Iterative Deepening (IDS)
- Search with Costs

Recap: Graph Search Algorithm

Input: a graph, a start node, 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,….,n_k> from *frontier;*

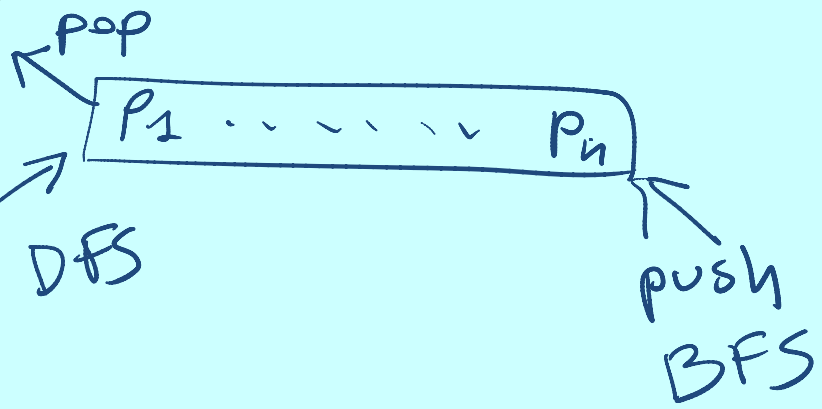
If *goal*(n_k)

return <n_o,….,n_k>;

For every neighbor *n* of n_k

add <n_o,….,n_k, *n*> to *frontier;*

end



In what aspects DFS and BFS differ when we look at the generic graph search algorithm?