COMP111: Artificial Intelligence

Introduction to Tree Search Problems

Frank Wolter

Content

- Introduce problem solving through tree search;

Content

- Introduce problem solving through tree search;
- Show how such problems can be stated as search graphs;

Content

- Introduce problem solving through tree search;
- Show how such problems can be stated as search graphs;
- Show how the search tree is generated from the search graph;

Content

- Introduce problem solving through tree search;
- Show how such problems can be stated as search graphs;
- Show how the search tree is generated from the search graph;
- Introduce main blind (also called uninformed) tree search algorithms:
  - breadth first-search (BFS);
  - depth first-search (DFS).

Content

- Introduce problem solving through tree search;
- Show how such problems can be stated as search graphs;
- Show how the search tree is generated from the search graph;
- Introduce main blind (also called uninformed) tree search algorithms:
  - breadth first-search (BFS);
  - depth first-search (DFS).
- Introduce main performance measures for tree search algorithms and analyse BFS and DFS.

Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).

Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).
- The goal is the state of affairs the agent wants to bring about.

Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).
- The goal is the state of affairs the agent wants to bring about.
- The difficulty is deciding the order in which the actions are carried out.

Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).
- The goal is the state of affairs the agent wants to bring about.
- The difficulty is deciding the order in which the actions are carried out.
- The agent thus wants to compute a sequence of actions which leads to the goal and maybe satisfies additional optimality conditions (such as being as short as possible).

Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).
- The goal is the state of affairs the agent wants to bring about.
- The difficulty is deciding the order in which the actions are carried out.
- The agent thus wants to compute a sequence of actions which leads to the goal and maybe satisfies additional optimality conditions (such as being as short as possible).
- The process of describing the goal, the relevant states of the world, the possible actions, and the optimality criterion is called problem formulation.

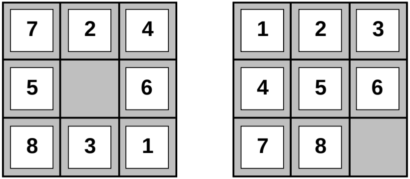
Solving Problems by Searching

- Consider an agent who has a goal and a set of actions he can perform to reach the goal. (In particular, the agent is goal-based agent).
- The goal is the state of affairs the agent wants to bring about.
- The difficulty is deciding the order in which the actions are carried out.
- The agent thus wants to compute a sequence of actions which leads to the goal and maybe satisfies additional optimality conditions (such as being as short as possible).
- The process of describing the goal, the relevant states of the world, the possible actions, and the optimality criterion is called problem formulation.
- Once the problem has been formulated, looking for a sequence of actions that lead to a goal state and is optimal is called search.

Four Examples

- 8-Puzzle
- Holiday in Romania
- Vacuum cleaner world
- The 8 Queens Problem

Example 1: 8-Puzzle

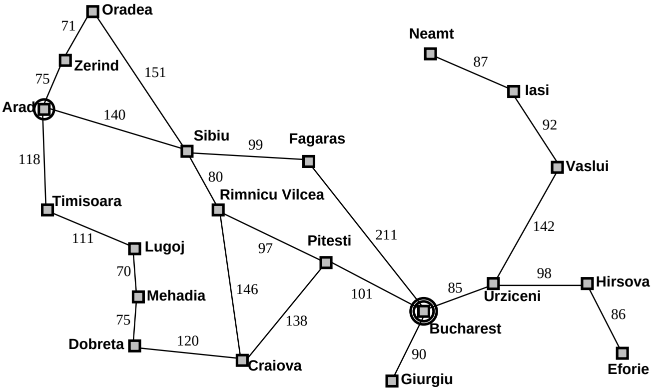


Start State   Goal State

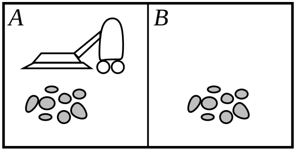
The 8-puzzle is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

Example 2: Holiday in Romania

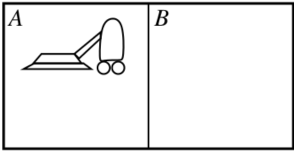
On holiday in Romania; currently in Arad.
Drive as quickly as possible to Bucharest.



Example 3: Vacuum cleaner world



Possible Start

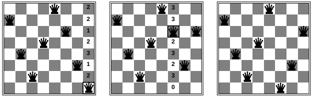


Possible Goal

The vacuum cleaner world is composed of

- two rooms
- each room might contain dirt
- a vacuum cleaner located in one of the rooms
  - vacuum actions: move left/right, suck up the dirt, do nothing
- the goal is to clean all rooms

Example 4: The 8 Queens Problem



- This is a problem from chess.
- Place 8 queens on chess board such that no queen attacks any other.

(A queen attacks any piece in the same row, column or diagonal.)

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;
  - a set S_{goal} of states from S, the goal states;

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;
  - a set S_{goal} of states from S, the goal states;
  - a set A of actions that can be performed and that take the agent from one state in S to another state in S; we often do not give a name to the actions but just say which states are reachable from a state by a single action.

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;
  - a set S_{goal} of states from S, the goal states;
  - a set A of actions that can be performed and that take the agent from one state in S to another state in S; we often do not give a name to the actions but just say which states are reachable from a state by a single action.
  - a cost function that states how expensive an action is. If all actions are equally expensive, we set $\text{cost}(a) = 1$ for all actions a.

Problem Formulation: the Search Graph

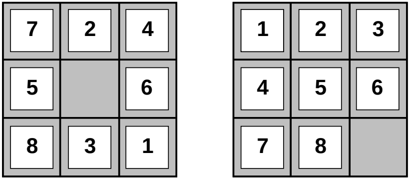
- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;
  - a set S_{goal} of states from S, the goal states;
  - a set A of actions that can be performed and that take the agent from one state in S to another state in S; we often do not give a name to the actions but just say which states are reachable from a state by a single action.
  - a cost function that states how expensive an action is. If all actions are equally expensive, we set $\text{cost}(a) = 1$ for all actions a.
- A solution to the search problem is a sequence of actions such that when performed in the start state s_{start}, the agent reaches a goal state in S_{goal}. A solution is optimal if there is no less expensive solution.

Problem Formulation: the Search Graph

- First determine the set S of possible states of the problem.
- Then the search graph for our problem is given by:
  - the set S of states;
  - a start state s_{start} from S, the starting point;
  - a set S_{goal} of states from S, the goal states;
  - a set A of actions that can be performed and that take the agent from one state in S to another state in S; we often do not give a name to the actions but just say which states are reachable from a state by a single action.
  - a cost function that states how expensive an action is. If all actions are equally expensive, we set $\text{cost}(a) = 1$ for all actions a.
- A solution to the search problem is a sequence of actions such that when performed in the start state s_{start}, the agent reaches a goal state in S_{goal}. A solution is optimal if there is no less expensive solution.

We determine the search graph for our four problems.

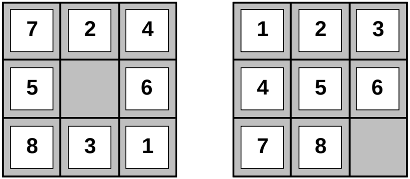
Example 1: 8-Puzzle



Start State **Goal State**

- States S: every possibility of having the 3×3 grid filled with numbers 1–8 and a blank.

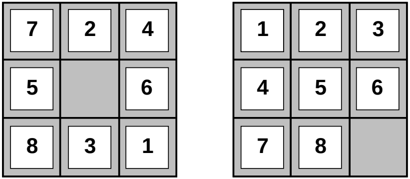
Example 1: 8-Puzzle



Start State   Goal State

- States S: every possibility of having the 3×3 grid filled with numbers 1–8 and a blank.
- Start state s_{start}: a single state. For example, the one shown on the left.

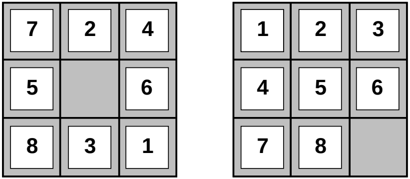
Example 1: 8-Puzzle



Start State Goal State

- States S: every possibility of having the 3×3 grid filled with numbers 1–8 and a blank.
- Start state s_{start}: a single state. For example, the one shown on the left.
- Set S_{goal} of goal states: the state shown on the right.

Example 1: 8-Puzzle

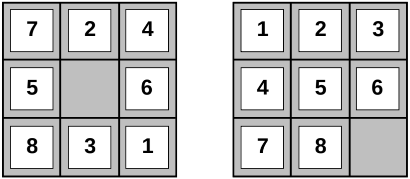


Start State

Goal State

- States S: every possibility of having the 3×3 grid filled with numbers 1–8 and a blank.
- Start state s_{start}: a single state. For example, the one shown on the left.
- Set S_{goal} of goal states: the state shown on the right.
- Four actions: move the tile to left of empty square to right; etc..

Example 1: 8-Puzzle



Start State **Goal State**

- States S: every possibility of having the 3×3 grid filled with numbers 1–8 and a blank.
- Start state s_{start}: a single state. For example, the one shown on the left.
- Set S_{goal} of goal states: the state shown on the right.
- Four actions: move the tile to left of empty square to right; etc..
- Cost function: all actions have the same cost. (We set cost$=1$ for each action.) We want a shortest sequence of actions that leads from s_{start} to the goal.

Exercise

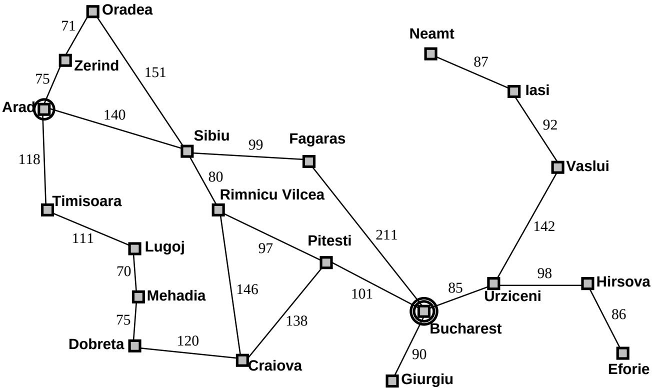
- When drawing a search graph, we often abstract from the actions. Instead, we just draw arrows from any state s to the states that can be reached by a single action. They are called the successor states of s.

Exercise

- When drawing a search graph, we often abstract from the actions. Instead, we just draw arrows from any state s to the states that can be reached by a single action. They are called the successor states of s.
- Draw a partial picture of the search graph of the 8-puzzle by listing the successor states of the start state from the previous slide and the predecessor states of the goal state. (We call a state s' is a predecessor of a state s if s is a sucessor of s').

Holiday in Romania

On holiday in Romania; currently in Arad.
Drive as quickly as possible to Bucharest.



Search Graph for holiday in Romania

- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc

Search Graph for holiday in Romania

- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc
- Start state s_{start}: in Arad;

Search Graph for holiday in Romania

- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc
- Start state s_{start}: in Arad;
- Set S_{goal} of goal states: in Bucharest;

Search Graph for holiday in Romania

- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc
- Start state s_{start}: in Arad;
- Set S_{goal} of goal states: in Bucharest;
- Actions: drive from a to b for neighbouring cities a and b, for example drive from Arad to Zerind. Note that there is no single action that takes you from a city to a non-neighbouring city (for example, from Arad to Oradea);

Search Graph for holiday in Romania

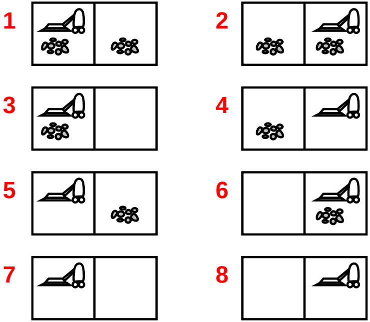
- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc
- Start state s_{start}: in Arad;
- Set S_{goal} of goal states: in Bucharest;
- Actions: drive from a to b for neighbouring cities a and b, for example drive from Arad to Zerind. Note that there is no single action that takes you from a city to a non-neighbouring city (for example, from Arad to Oradea);
- Cost function: km you drive.

Search Graph for holiday in Romania

- Set S of states: being in a city from the map of Romania: in Arad, in Zerind, in Oradea, etc
- Start state s_{start}: in Arad;
- Set S_{goal} of goal states: in Bucharest;
- Actions: drive from a to b for neighbouring cities a and b, for example drive from Arad to Zerind. Note that there is no single action that takes you from a city to a non-neighbouring city (for example, from Arad to Oradea);
- Cost function: km you drive.

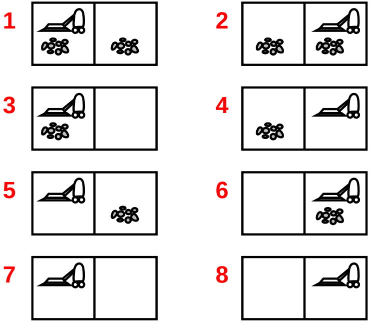
In this case, the map from the previous slide is already a representation of the search graph!

Example 3: Vacuum cleaner world



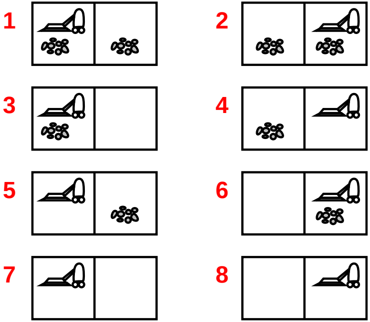
- Set S of states: the 8 situations depicted in figure;

Example 3: Vacuum cleaner world



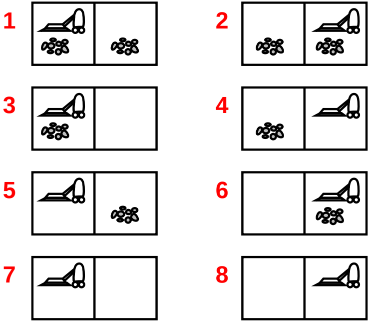
- Set S of states: the 8 situations depicted in figure;
- Start state s_{start}: any state;

Example 3: Vacuum cleaner world



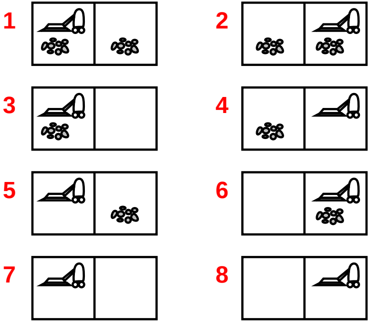
- Set S of states: the 8 situations depicted in figure;
- Start state s_{start}: any state;
- Set S_{goal} of goal states: states 7 and 8;

Example 3: Vacuum cleaner world



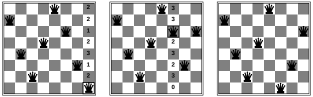
- Set S of states: the 8 situations depicted in figure;
- Start state s_{start}: any state;
- Set S_{goal} of goal states: states 7 and 8;
- Actions: move left/right, suck up the dirt, do nothing;

Example 3: Vacuum cleaner world



- Set S of states: the 8 situations depicted in figure;
- Start state s_{start}: any state;
- Set S_{goal} of goal states: states 7 and 8;
- Actions: move left/right, suck up the dirt, do nothing;
- Cost function: assume all actions equally expensive (cost=1).

Example 4: 8 Queens Problem



- Place 8 queens on chess board so that no queen can be taken by another.

(A queen attacks any piece in the same row, column or diagonal.)

Search graph for 8 Queens Problem

- Set S of states: any arrangement of 0 to 8 queens on the board;
- Start state s_{start}: empty chess board;
- Set S_{goal} of goal states: 8 queens on chess board such that no queen attacks another;
- Actions: add a queen to an empty square;
- Cost function: none needed.

Some comments on defining search graph: Abstraction

- Real world is absurdly complex: state space must be *abstracted* for problem solving

Some comments on defining search graph: Abstraction

- Real world is absurdly complex: state space must be *abstracted* for problem solving
- Abstract state = set of real states

Some comments on defining search graph: Abstraction

- Real world is absurdly complex: state space must be *abstracted* for problem solving
- Abstract state = set of real states
- Abstract action = complex combination of real actions

  "Arad $\rightarrow$ Zerind"

  represents a complex set of possible routes, detours, rest stops, etc.

Some comments on defining search graph: Abstraction

- Real world is absurdly complex: state space must be *abstracted* for problem solving
- Abstract state = set of real states
- Abstract action = complex combination of real actions

  "Arad $\rightarrow$ Zerind"

  represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realisability, *any* real state "in Arad" must get to real state "in Zerind"

Some comments on defining search graph: Abstraction

- Real world is absurdly complex: state space must be *abstracted* for problem solving
- Abstract state = set of real states
- Abstract action = complex combination of real actions

  "Arad $\rightarrow$ Zerind"

  represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realisability, *any* real state "in Arad" must get to real state "in Zerind"
- Abstract solution = set of real paths that are solutions in the real world

Abstraction should be "easier" than the original problem!

Size of the search graph

In the toy examples given above the search space is still rather small. In real search problems the state space is huge:

Problem	States	Brute-Force Search (10 million states/sec)
8 Puzzle	10^6	0.01 seconds
15 Puzzle	10^{13}	6 days
Rubik's Cube	10^{18}	68, 000 years
24 Puzzle	10^{26}	12 billion years
Checkers	10^{40}	
Chess	10^{120}	

In what follows we assume we can't store the whole search graph explicitly. It is imaginary.

Search

We assume

- the start state s_{start} is given,

Search

We assume

- the start state s_{start} is given,
- a description of the actions is given in the following form: if in state s, then states $s_1, \ldots, s_n$ can be reached by an action (they are called successor states of s),

Search

We assume

- the start state s_{start} is given,
- a description of the actions is given in the following form: if in state s, then states $s_1, \ldots, s_n$ can be reached by an action (they are called successor states of s),
- a description of the goal states is given.

Search

We assume

- the start state s_{start} is given,
- a description of the actions is given in the following form: if in state s, then states $s_1, \ldots, s_n$ can be reached by an action (they are called successor states of s),
- a description of the goal states is given.

Search is the systematic exploration of the search tree consisting of all possible paths of states starting with the start state using the actions available until a path to a goal state is reached.

Search

We assume

- the start state s_{start} is given,
- a description of the actions is given in the following form: if in state s, then states $s_1, \ldots, s_n$ can be reached by an action (they are called successor states of s),
- a description of the goal states is given.

Search is the systematic exploration of the search tree consisting of all possible paths of states starting with the start state using the actions available until a path to a goal state is reached.
Note that the whole resulting search tree is imaginary. Typically it cannot be stored.

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Thus we have

- a single path of length 0 (just Arad),

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Thus we have

- a single path of length 0 (just Arad),
- three paths of length 1,

Search Tree for holiday in Romania

The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Thus we have

- a single path of length 0 (just Arad),
- three paths of length 1,
- eight paths of length 2,
- and so on

Search Tree for holiday in Romania

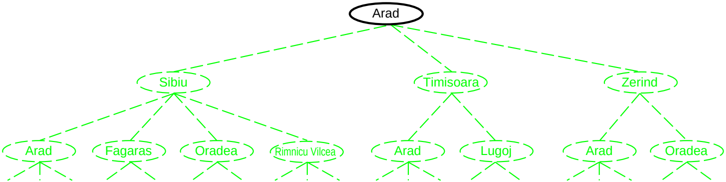
The paths starting in Arad are:

- Arad
- Arad, Sibiu and Arad, Timisoara and Arad, Zerind
- Arad, Sibiu, Arad and Arad, Sibiu, Figaras and so on.

Thus we have

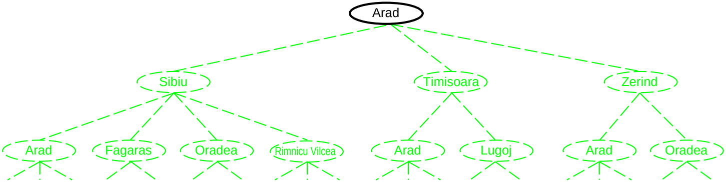
- a single path of length 0 (just Arad),
- three paths of length 1,
- eight paths of length 2,
- and so on

The set of all paths can be depicted as a tree with root Arad:



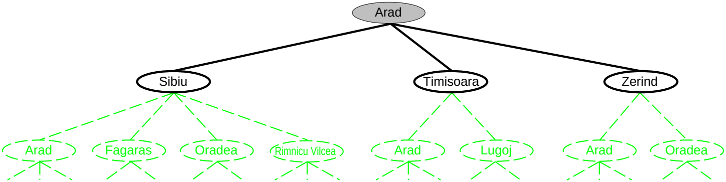
Search Algorithm

- When describing search algorithms state how the search tree is explored by the search algorithm.
- The root of the tree is the start state s_{start}
- Applying actions, we generate longer paths by adding successor states to a path (also called expanding a path/state)



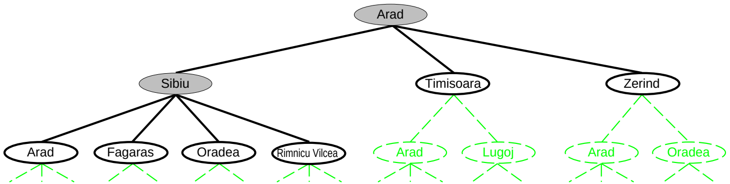
Search Algorithm

- When describing search algorithms state how the search tree is explored by the search algorithm.
- The root of the tree is the start state s_{start}
- Applying actions, we generate longer paths by adding successor states to a path (also called expanding a path/state)

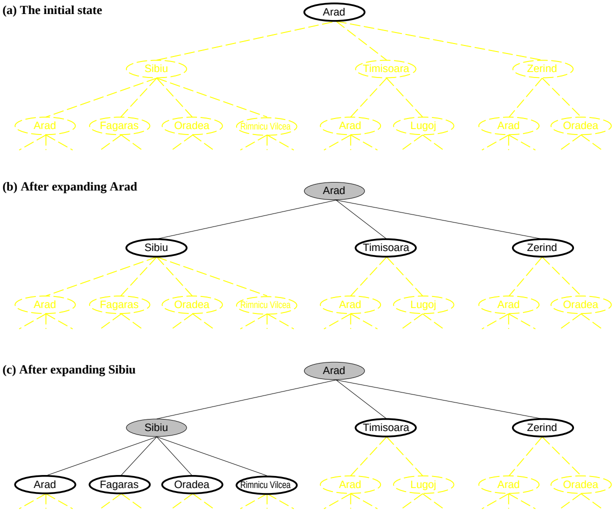


Search Algorithm

- When describing search algorithms state how the search tree is explored by the search algorithm.
- The root of the tree is the start state s_{start}
- Applying actions, we generate longer paths by adding successor states to a path (also called expanding a path/state)



The Search Tree



Pseudocode

In this module we give algorithms (such as seach algorithms) in pseudocode, informal high-level descriptions of algorithms that are intended for human reading rather than machine reading.

Pseudocode

In this module we give algorithms (such as seach algorithms) in pseudocode, informal high-level descriptions of algorithms that are intended for human reading rather than machine reading.

Algorithms in pseudocode often abstract from implementation details and leave certain steps (or subroutines) unspecified.

Pseudocode

In this module we give algorithms (such as seach algorithms) in pseudocode, informal high-level descriptions of algorithms that are intended for human reading rather than machine reading.

Algorithms in pseudocode often abstract from implementation details and leave certain steps (or subroutines) unspecified.

We now give a generic algorithm for search in pseudocode that, in fact, leaves a very important step unspecified.

Generic Algorithm for Search

Input: a start state s_0
for each state s the successors of s
a test goal(s) checking whether s is a goal state

Set frontier := $\{s_0\}$
while frontier is not empty **do**
select and remove from frontier a path $s_0....s_k$
if goal(s_k) **then**
return $s_0 \dots s_k$ (and terminate)
else for every successor s of s_k add $s_0 \dots s_k s$ to frontier
end if
end while

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.
- The algorithm maintains a set called frontier containing paths. In the beginning it only contains the path s_0. The frontier is also called agenda.

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.
- The algorithm maintains a set called frontier containing paths. In the beginning it only contains the path s_0. The frontier is also called agenda.
- As long as frontier is not empty, it selects and removes a path from frontier. We also say that this path is expanded.

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.
- The algorithm maintains a set called frontier containing paths. In the beginning it only contains the path s_0. The frontier is also called agenda.
- As long as frontier is not empty, it selects and removes a path from frontier. We also say that this path is expanded.
- It terminates and returns the path if its final state is a goal state.

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.
- The algorithm maintains a set called frontier containing paths. In the beginning it only contains the path s_0. The frontier is also called agenda.
- As long as frontier is not empty, it selects and removes a path from frontier. We also say that this path is expanded.
- It terminates and returns the path if its final state is a goal state.
- Otherwise it adds to frontier all paths obtained by expanding (adding a successor to) the removed path.

Generic Algorithm for Search: comments

- The input are a start state s_0, a method to produce the successor states of a state (expand a state), and a method to check whether a state is a goal state.
- The algorithm maintains a set called frontier containing paths. In the beginning it only contains the path s_0. The frontier is also called agenda.
- As long as frontier is not empty, it selects and removes a path from frontier. We also say that this path is expanded.
- It terminates and returns the path if its final state is a goal state.
- Otherwise it adds to frontier all paths obtained by expanding (adding a successor to) the removed path.
- It then repeats the same steps by selecting and removing a path from the updated frontier.

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \ldots s_k$ do we select and remove from the frontier?

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \dots s_k$ do we select and remove from the frontier? We also call this:
  - which path $s_0 \dots s_k$ do we expand or select for expansion?
  - Or simply: which state s_k do we expand? (Note: sometimes not precise enough.)

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \ldots s_k$ do we select and remove from the frontier? We also call this:
  - which path $s_0 \ldots s_k$ do we expand or select for expansion?
  - Or simply: which state s_k do we expand? (Note: sometimes not precise enough.)
- Given $s_0 \ldots s_k$, in which order do we add the expansions $s_0 \ldots s_k s$ to the frontier?

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \dots s_k$ do we select and remove from the frontier? We also call this:
  - which path $s_0 \dots s_k$ do we expand or select for expansion?
  - Or simply: which state s_k do we expand? (Note: sometimes not precise enough.)
- Given $s_0 \dots s_k$, in which order do we add the expansions $s_0 \dots s_k s$ to the frontier?

We do not specify the order in which we add $s_0 \dots s_k s$ (often called don't care non-determinism), but we address the problem which path we select (expand) and remove from the frontier.

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \ldots s_k$ do we select and remove from the frontier? We also call this:
  - which path $s_0 \ldots s_k$ do we expand or select for expansion?
  - Or simply: which state s_k do we expand? (Note: sometimes not precise enough.)
- Given $s_0 \ldots s_k$, in which order do we add the expansions $s_0 \ldots s_k s$ to the frontier?

We do not specify the order in which we add $s_0 \ldots s_k s$ (often called don't care non-determinism), but we address the problem which path we select (expand) and remove from the frontier.

- Two obvious strategies:
  - breadth first search, BFS, (always select the path first added to frontier);

Generic Tree Search Algorithm

The generic tree search algorithm leaves the following steps open:

- Which path $s_0 \ldots s_k$ do we select and remove from the frontier? We also call this:
  - which path $s_0 \ldots s_k$ do we expand or select for expansion?
  - Or simply: which state s_k do we expand? (Note: sometimes not precise enough.)
- Given $s_0 \ldots s_k$, in which order do we add the expansions $s_0 \ldots s_k s$ to the frontier?

We do not specify the order in which we add $s_0 \ldots s_k s$ (often called don't care non-determinism), but we address the problem which path we select (expand) and remove from the frontier.

- Two obvious strategies:
  - breadth first search, BFS, (always select the path first added to frontier);
  - depth first search, DFS, (always select the path last added to frontier).

Breadth First Search (BFS)

- Start by selecting (expanding) start state — gives tree of depth 1.

Breadth First Search (BFS)

- Start by selecting (expanding) start state — gives tree of depth 1.
- Then select (expand) all paths that resulted from previous step — gives tree of depth 2.

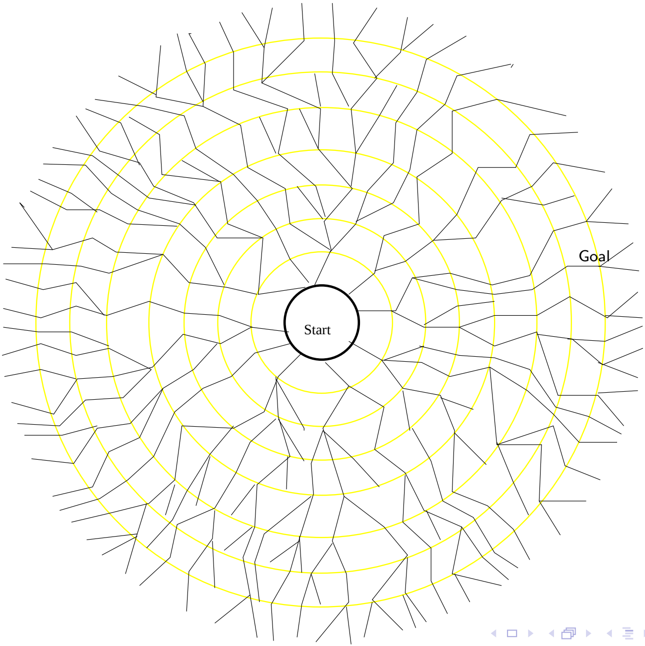
Breadth First Search (BFS)

- Start by selecting (expanding) start state — gives tree of depth 1.
- Then select (expand) all paths that resulted from previous step — gives tree of depth 2.
- Then select (expand) all paths that resulted from previous step — gives tree of depth 3.

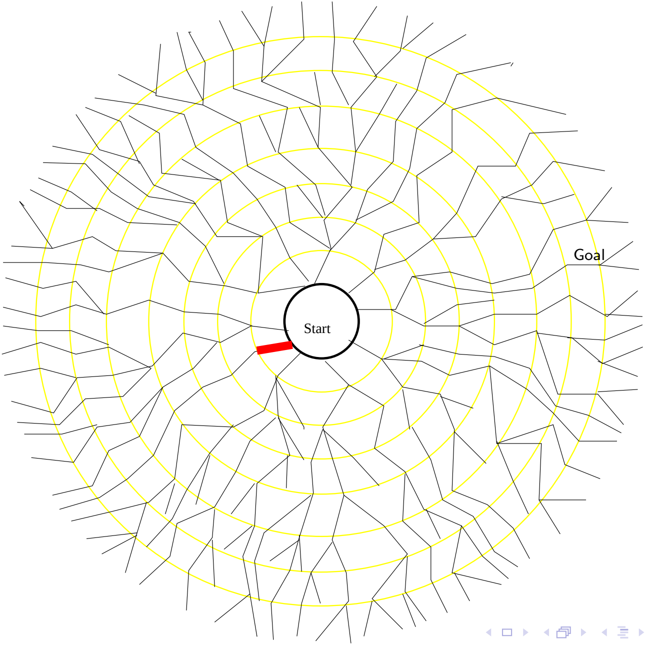
Breadth First Search (BFS)

- Start by selecting (expanding) start state — gives tree of depth 1.
- Then select (expand) all paths that resulted from previous step — gives tree of depth 2.
- Then select (expand) all paths that resulted from previous step — gives tree of depth 3.
- and so on.
- In general: select (expand) all paths of depth n before depth $n+1$.

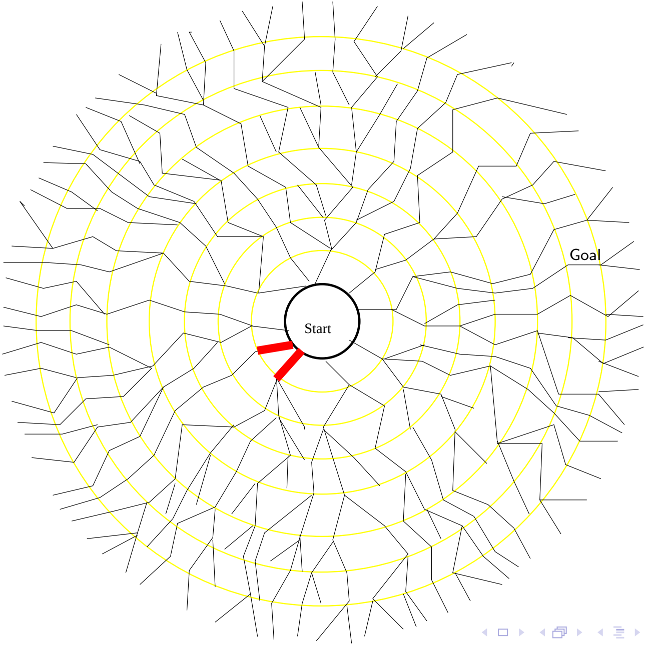
Example: BFS in a Maze



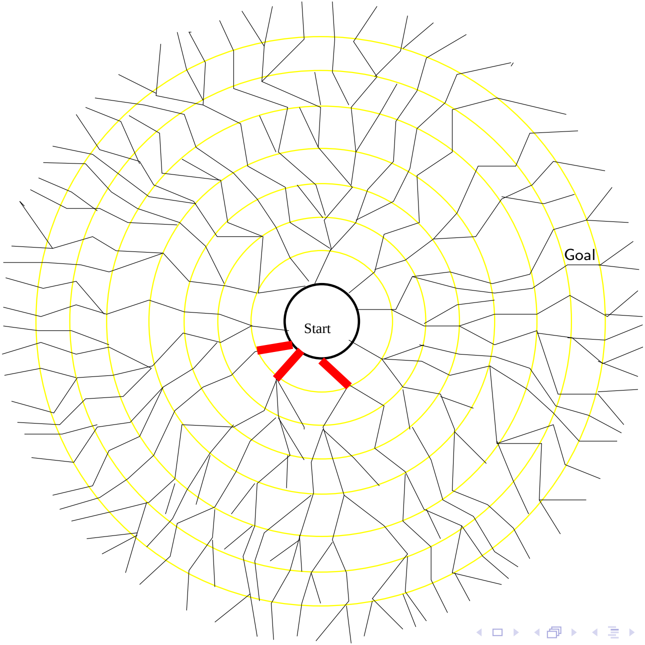
Example: BFS in a Maze



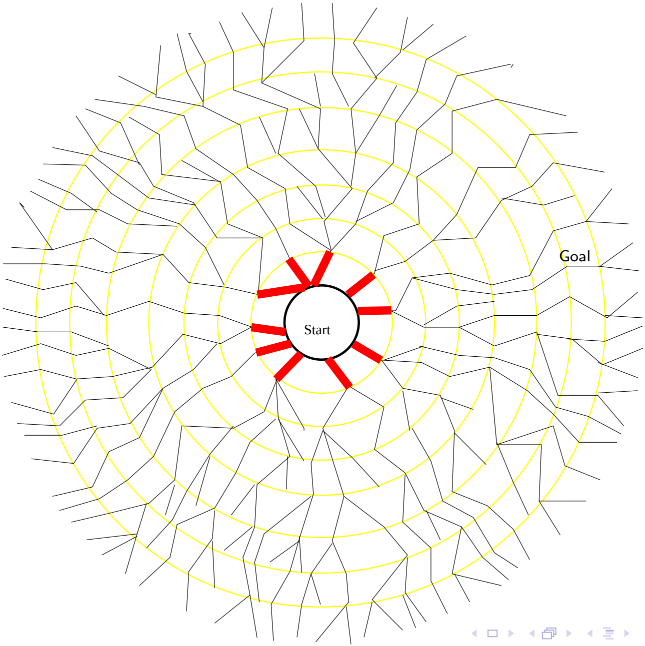
Example: BFS in a Maze



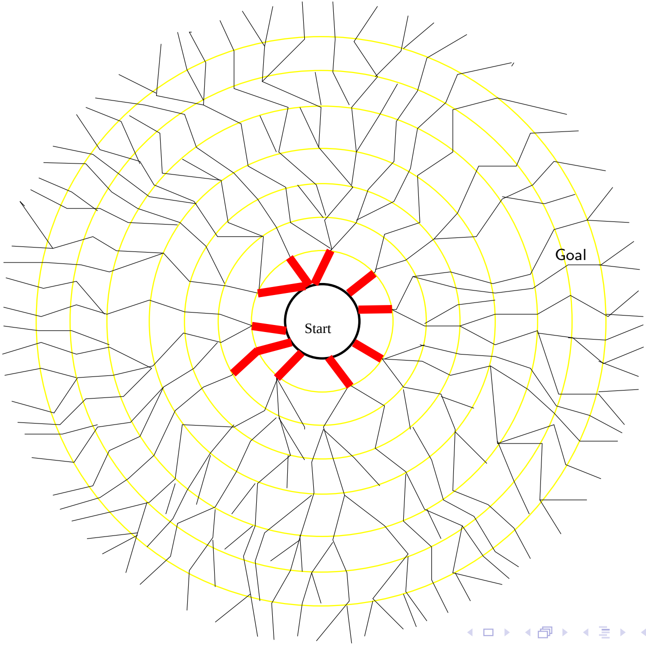
Example: BFS in a Maze



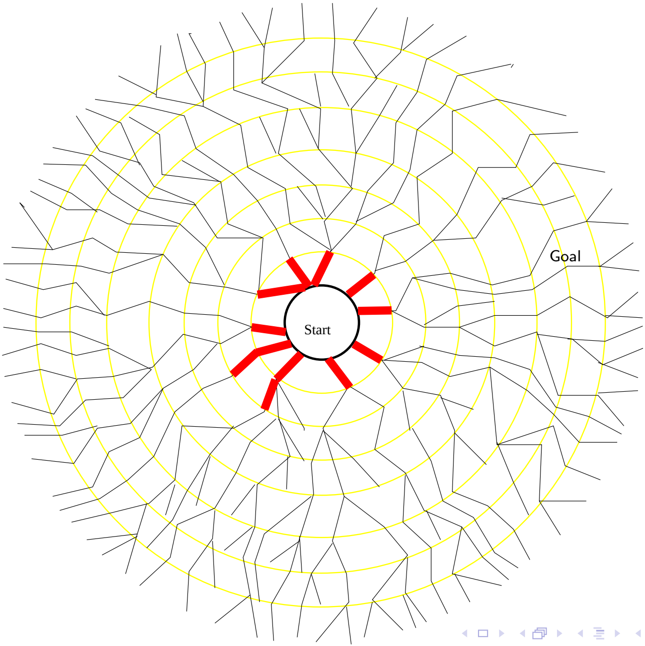
Example: BFS in a Maze



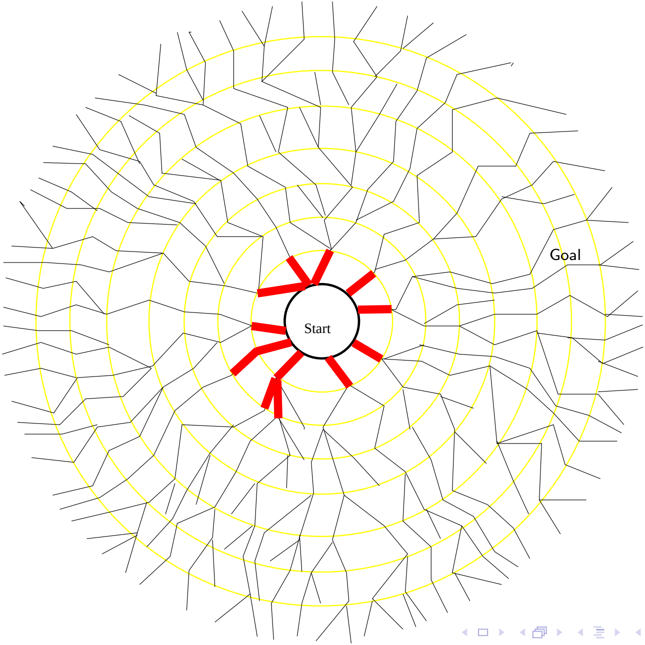
Example: BFS in a Maze



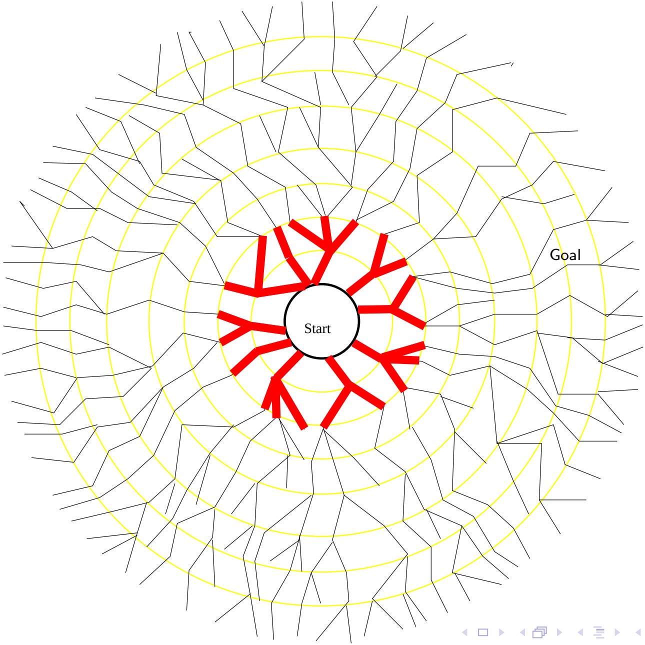
Example: BFS in a Maze



Example: BFS in a Maze



Example: BFS in a Maze



Breadth First Search

For the select step always select a path that was first added to the frontier:

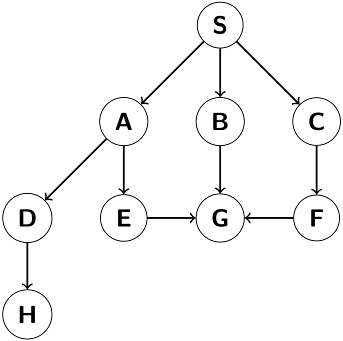
Input: a start state s_0
    for each state s the successors of s
    a test goal(s) checking whether s is a goal state

Set frontier $:= \{s_0\}$
while frontier is not empty **do**
    select and remove from frontier the path $s_0....s_k$ that was
    first added to frontier
    if goal(s_k) **then**
        return $s_0 \dots s_k$ (and terminate)
    else for every successor s of s_k add $s_0 \dots s_k s$ to frontier
    end if
end while

Breadth-First Example

Reaching G from S

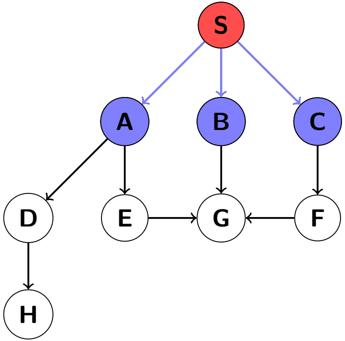
Exp. paths	Frontier
	$\{S\}$



Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$

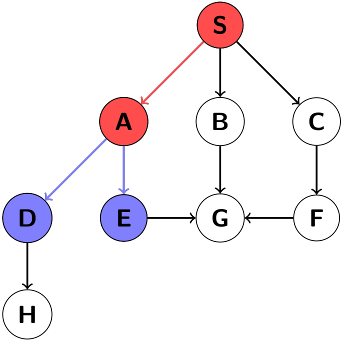


Selected path: S
Is the last state in S_{goal}? No
Expand S: add $SA, SB,$ and SC to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$



Selected path: SA

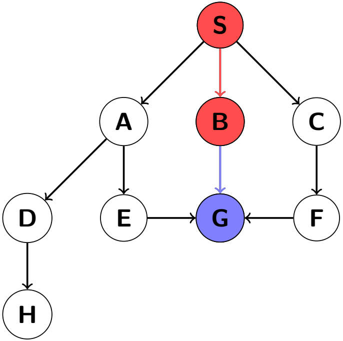
Is the last state in S_{goal}? No

Expand SA: add SAD and SAE to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SB not goal	$\{SC, SAD, SAE\ SBG\}$

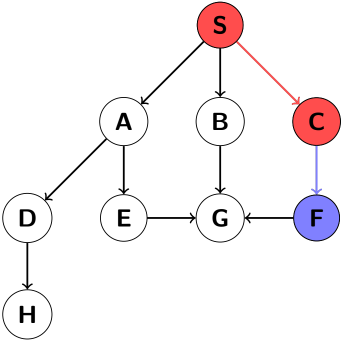


Selected path: SB
Is the last state in S_{goal}? No
Expand SB: add SBG to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SB not goal	$\{SC, SAD, SAE\ SBG\}$
SC not goal	$\{SAD, SAE, SBG, SCF\}$



Selected path: SC

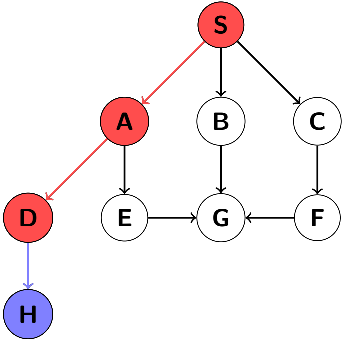
Is the last state in S_{goal}? No

Expand SC: add SCF to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SB not goal	$\{SC, SAD, SAE\ SBG\}$
SC not goal	$\{SAD, SAE, SBG, SCF\}$
SAD not goal	$\{SAE, SBG, SCF, SADH\}$

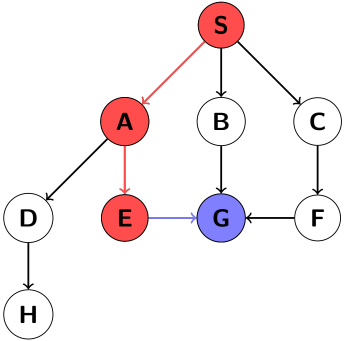


Selected path: SAD
Is the last state in S_{goal}? No
Expand SAD: add $SADH$ to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SB not goal	$\{SC, SAD, SAE\ SBG\}$
SC not goal	$\{SAD, SAE, SBG, SCF\}$
SAD not goal	$\{SAE, SBG, SCF, SADH\}$
SAE not goal	$\{SBG, SCF, SADH, SAEG\}$

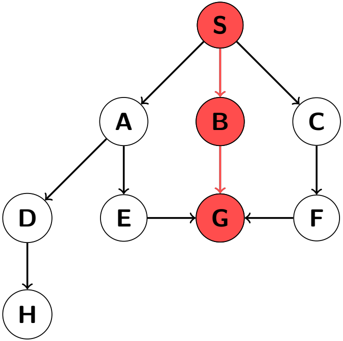


Selected path: SAE
Is the last state in S_{goal}? No
Expand SAE: add $SAEG$ to the frontier

Breadth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SB not goal	$\{SC, SAD, SAE\ SBG\}$
SC not goal	$\{SAD, SAE, SBG, SCF\}$
SAD not goal	$\{SAE, SBG, SCF, SADH\}$
SAE not goal	$\{SBG, SCF, SADH, SAEG\}$
SBG goal!	$\{SCF, SADH, SAEG\}$



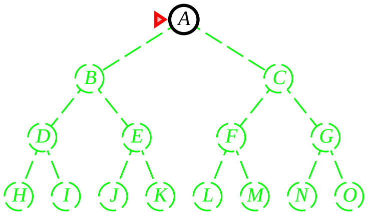
Selected path: SBG

Is the last state in S_{goal}? Yes!

Path found: SBG

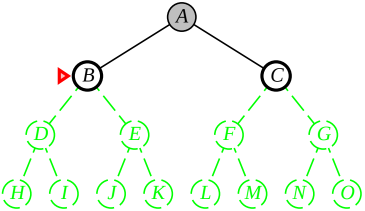
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one “branch” of search tree.



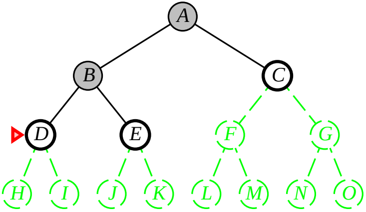
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



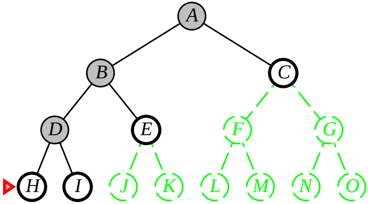
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



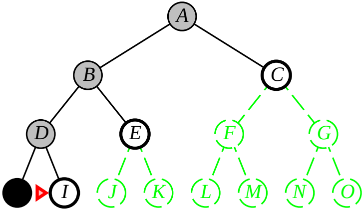
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



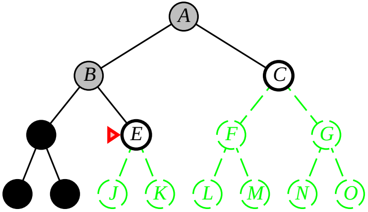
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



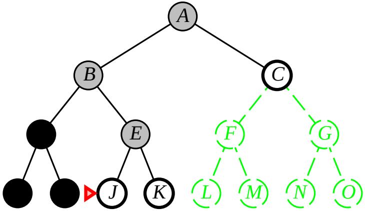
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one “branch” of search tree.



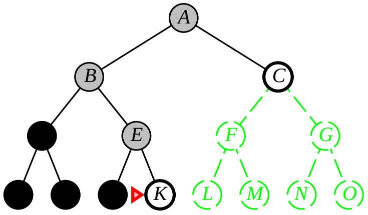
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



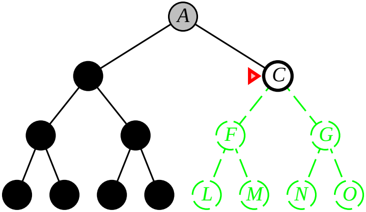
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



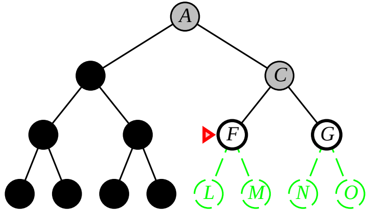
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one “branch” of search tree.



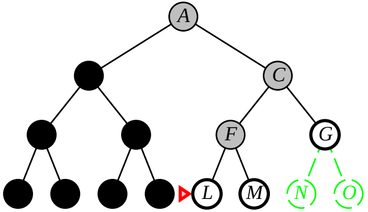
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



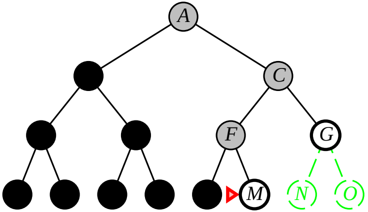
Depth First Search (DFS)

- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.

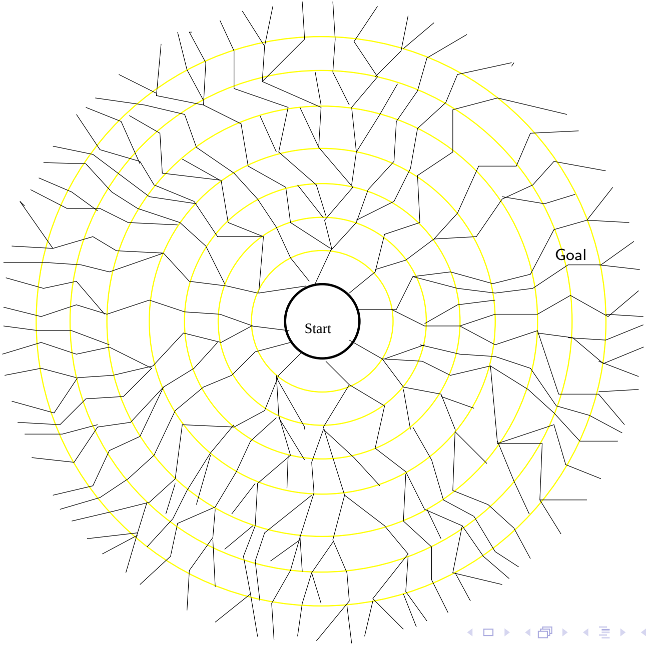


Depth First Search (DFS)

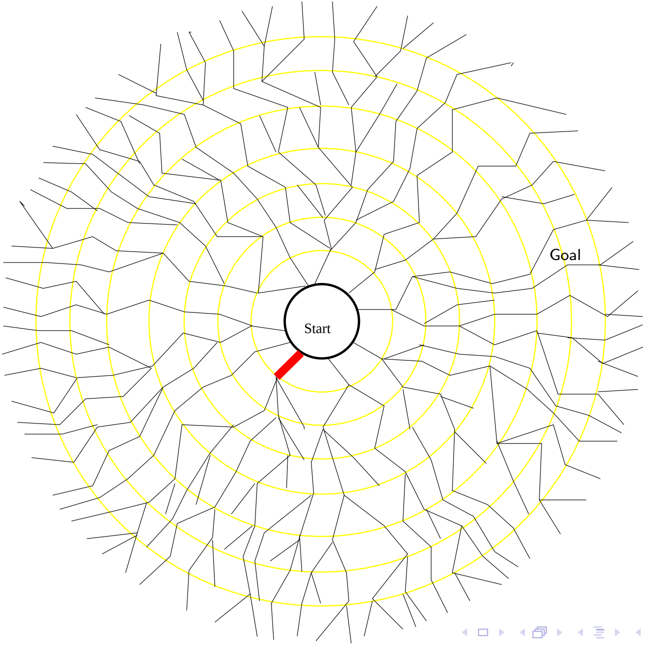
- Start by selecting start state.
- Select one of the paths resulting from 1st step.
- Select one of the paths resulting from 2nd step.
- Select one of the paths resulting from 3nd step. And so on.
- Always select longest (also called deepest) path.
- Follow one "branch" of search tree.



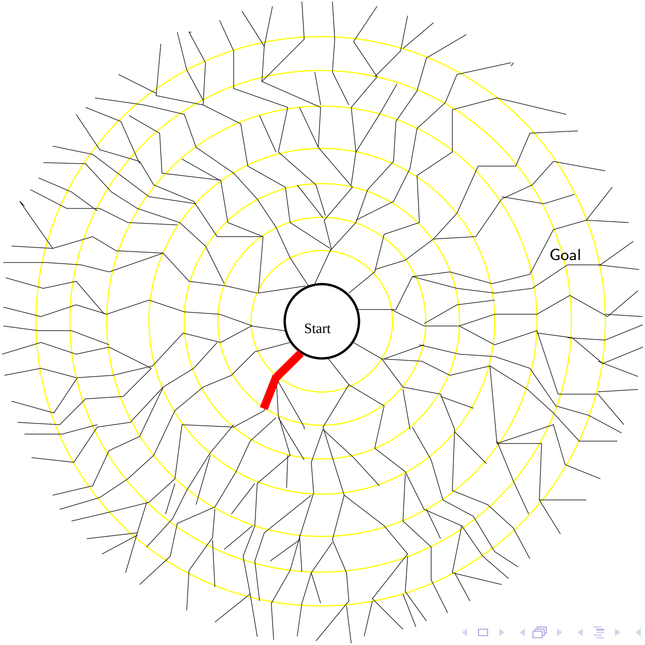
Example: DFS in a Maze



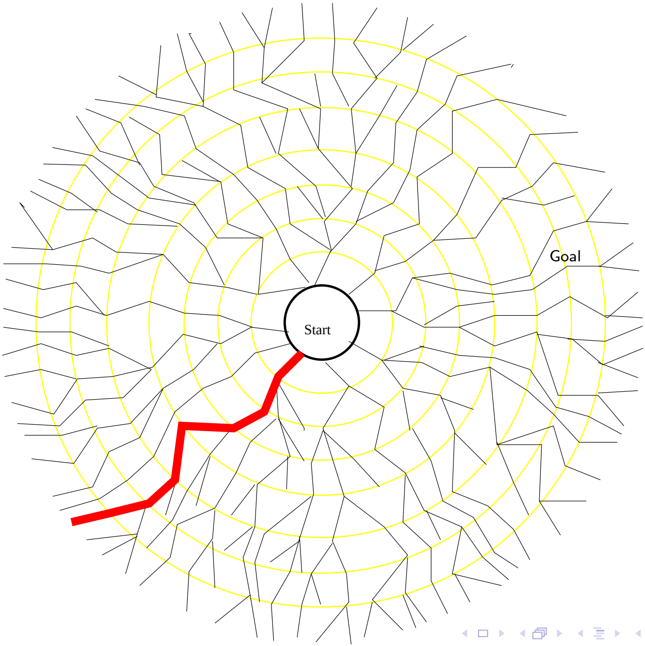
Example: DFS in a Maze



Example: DFS in a Maze



Example: DFS in a Maze



Depth First Search

In the select step always select a path that was last added to the frontier:

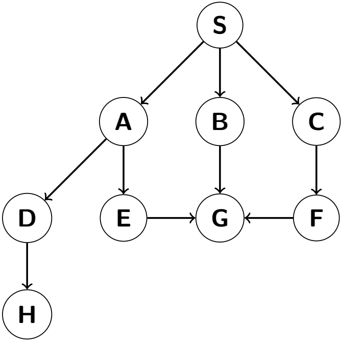
Input: a start state s_0
    for each state s the successors of s
    a test goal(s) checking whether s is a goal state

Set frontier $:= \{s_0\}$
while frontier is not empty **do**
    select and remove from frontier the path $s_0....s_k$ that was
    last added to frontier
    if goal(s_k) **then**
        return $s_0 \ldots s_k$ (and terminate)
    else for every successor s of s_k add $s_0 \ldots s_k s$ to frontier
    end if
end while

Depth-First Example

Reaching G from S

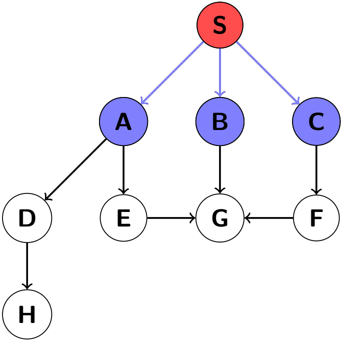
Exp. paths	Frontier
	$\{S\}$



Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$



Selected path: S

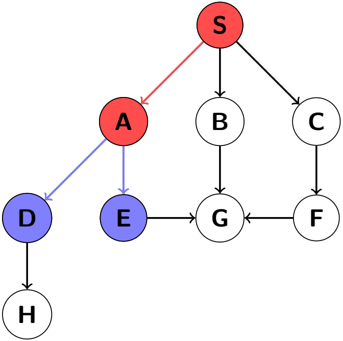
Is the last state in S_{goal}? No

Expand S: add $SA, SB,$ and SC to the frontier

Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$



Selected path: SA

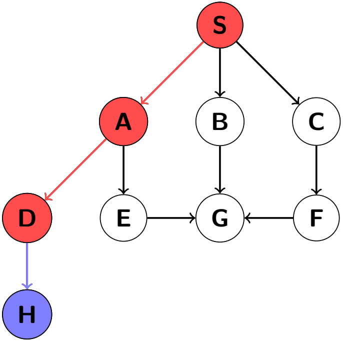
Is the last state in S_{goal}? No

Expand SA: add SAD and SAE to the frontier

Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SAD not goal	$\{SB, SC, SAE, SADH\}$



Selected path: SAD

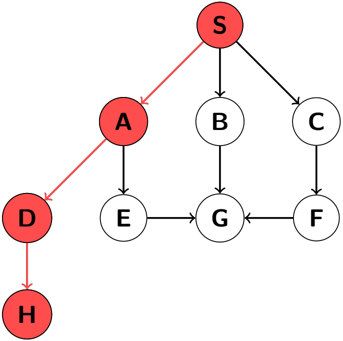
Is the last state in S_{goal}? No

Expand SAD: add $SADH$ to the frontier

Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SAD not goal	$\{SB, SC, SAE, SADH\}$
$SADH$ not goal	$\{SB, SC, SAE\}$

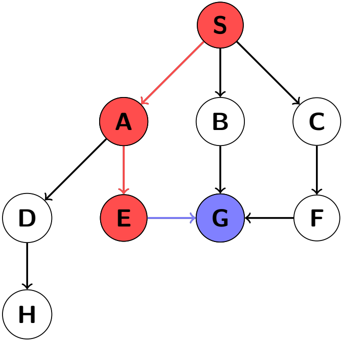


Selected path: $SADH$
Is the last state in S_{goal}? No
Expand $SADH$: nothing to add

Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SAD not goal	$\{SB, SC, SAE, SADH\}$
$SADH$ not goal	$\{SB, SC, SAE\}$
SAE not goal	$\{SB, SC, SAEG\}$



Selected path: SAE

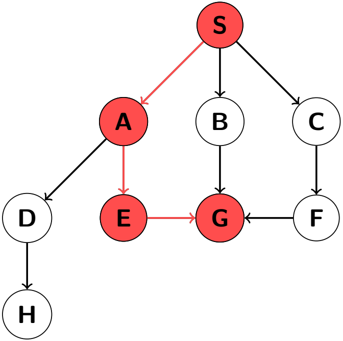
Is the last state in S_{goal}? No

Expand SAE: add $SAEG$ to the frontier

Depth-First Example

Reaching G from S

Exp. paths	Frontier
	$\{S\}$
S not goal	$\{SA, SB, SC\}$
SA not goal	$\{SB, SC, SAD, SAE\}$
SAD not goal	$\{SB, SC, SAE, SADH\}$
$SADH$ not goal	$\{SB, SC, SAE\}$
SAE not goal	$\{SB, SC, SAEG\}$
$SAEG$ goal	$\{SB, SC\}$



Selected path: $SAEG$
Is the last state in S_{goal}? Yes!
Path found: $SAEG$

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

  YES for BFS and NO for DFS

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

  YES for BFS and NO for DFS

- Time complexity: what is the number of paths generated?

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

  YES for BFS and NO for DFS

- Time complexity: what is the number of paths generated? No answer yet: let's discuss how to compute this.

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

  YES for BFS and NO for DFS

- Time complexity: what is the number of paths generated? No answer yet: let's discuss how to compute this.
- Space complexity: what is the maximum number of paths in memory (in frontier)?

Properties of Search Strategies

- Completeness: does the algorithm always find a solution if one exists?

  YES for BFS and NO for DFS (consider cycles in the search graph)

- Optimality: does the algorithm always find a shortest path (path of lowest cost)?

  YES for BFS and NO for DFS

- Time complexity: what is the number of paths generated?
  No answer yet: let's discuss how to compute this.
- Space complexity: what is the maximum number of paths in memory (in frontier)?
  No answer yet: let's discuss how to compute this.

Properties of Search Strategies

- Time and space complexity are measured in terms of:
  - b—maximum branching factor of the search tree: this is the maximal number of successor states of a state in the state space.

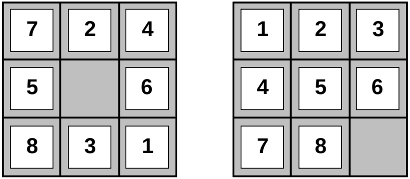
Properties of Search Strategies

- Time and space complexity are measured in terms of:
  - b—maximum branching factor of the search tree: this is the maximal number of successor states of a state in the state space.
  - d—depth of the optimal solution: this is the length of the shortest path from the start state to a goal state.

Properties of Search Strategies

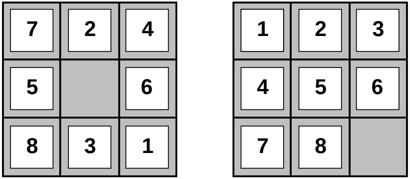
- Time and space complexity are measured in terms of:
  - b—maximum branching factor of the search tree: this is the maximal number of successor states of a state in the state space.
  - d—depth of the optimal solution: this is the length of the shortest path from the start state to a goal state.
  - m—maximum depth of the state space: this is the length of the longest path in the state space (may be ∞)

Example 1: 8-Puzzle



Start State Goal State

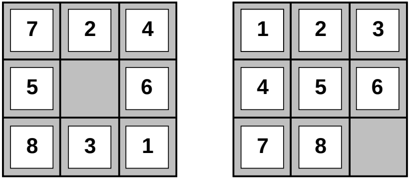
Example 1: 8-Puzzle



Start State **Goal State**

- the maximum branching factor is $b = 4$.

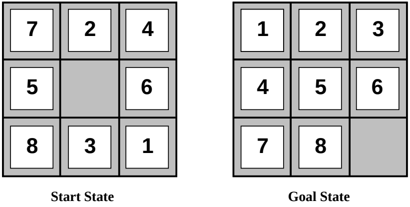
Example 1: 8-Puzzle



Start State **Goal State**

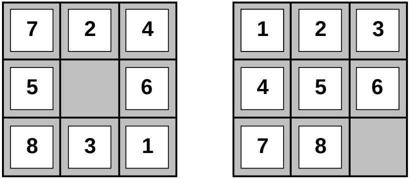
- the maximum branching factor is $b = 4$.
- for certain start states the length of the shortest solution path to a goal state is $d = 31$.

Example 1: 8-Puzzle



- the maximum branching factor is $b = 4$.
- for certain start states the length of the shortest solution path to a goal state is $d = 31$.
- the length of the longest path in the state space is $m = \infty$.

Example 1: 8-Puzzle



Start State **Goal State**

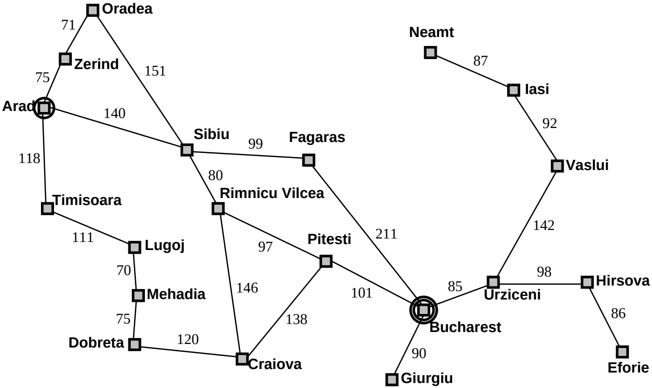
- the maximum branching factor is $b = 4$.
- for certain start states the length of the shortest solution path to a goal state is $d = 31$.
- the length of the longest path in the state space is $m = \infty$.

In fact, for the following two start states the minimal solution paths have length 31.

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

Example 2: Holiday in Romania

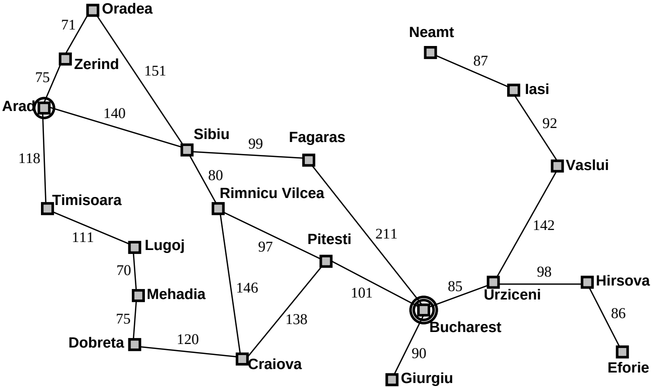
Drive as quickly as possible from Arad to Bucharest.



- the maximum branching factor is $b = 4$.

Example 2: Holiday in Romania

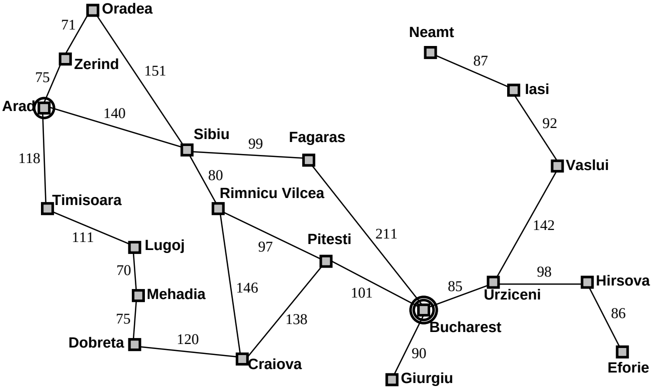
Drive as quickly as possible from Arad to Bucharest.



- the maximum branching factor is $b = 4$.
- for start state Arad and goal state Bucharest the length of the shortest path is $d = 3$.

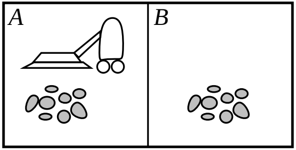
Example 2: Holiday in Romania

Drive as quickly as possible from Arad to Bucharest.

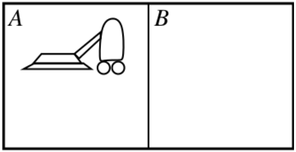


- the maximum branching factor is $b = 4$.
- for start state Arad and goal state Bucharest the length of the shortest path is $d = 3$.
- the length of the longest path in the state space is $m = \infty$.

Example 3: Vacuum cleaner world



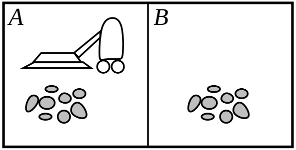
Possible Start



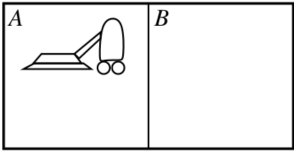
Possible Goal

- the maximum branching factor is $b = 2$.

Example 3: Vacuum cleaner world



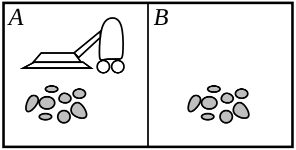
Possible Start



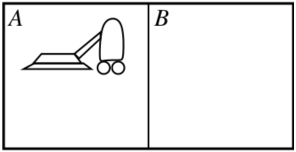
Possible Goal

- the maximum branching factor is $b = 2$.
- for some start states the length of the shortest path to a goal state is $d = 3$.

Example 3: Vacuum cleaner world



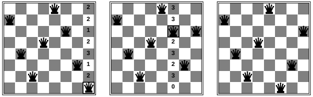
Possible Start



Possible Goal

- the maximum branching factor is $b = 2$.
- for some start states the length of the shortest path to a goal state is $d = 3$.
- the length of the longest path in the state space is $m = \infty$.

Example 4: The 8 Queens Problem

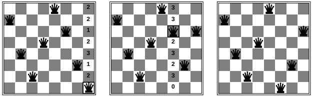


- This is a problem from chess.
- Place 8 queens on chess board such that no queen attacks any other.

(A queen attacks any piece in the same row, column or diagonal.)

- the maximum branching factor is $b = 8 \times 8$.

Example 4: The 8 Queens Problem

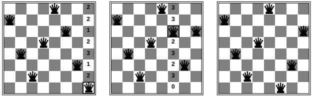


- This is a problem from chess.
- Place 8 queens on chess board such that no queen attacks any other.

(A queen attacks any piece in the same row, column or diagonal.)

- the maximum branching factor is $b = 8 \times 8$.
- the length of all paths from the start state to a goal state is $d = 8$.

Example 4: The 8 Queens Problem



- This is a problem from chess.
- Place 8 queens on chess board such that no queen attacks any other.

(A queen attacks any piece in the same row, column or diagonal.)

- the maximum branching factor is $b = 8 \times 8$.
- the length of all paths from the start state to a goal state is $d = 8$.
- the length of all paths from the start state is at most 8. Thus $m = 8$.

Properties of Breadth First Search

Facts. (1) The number of paths of length d in a search tree with maximum branching factor b is at most b^d.
(2) The number of paths of length at most d in a search tree with maximum branching factor b is at most $1 + b + b^2 + \cdots + b^d$.

Properties of Breadth First Search

Facts. (1) The number of paths of length d in a search tree with maximum branching factor b is at most b^d.
(2) The number of paths of length at most d in a search tree with maximum branching factor b is at most $1 + b + b^2 + \cdots + b^d$.

- Time complexity. If the shortest path to a goal state has length d, then in the worst case BFS will look at

$$1 + b + b^2 + \cdots + b^d$$

paths before reaching a goal state (exponential).

Properties of Breadth First Search

Facts. (1) The number of paths of length d in a search tree with maximum branching factor b is at most b^d.
(2) The number of paths of length at most d in a search tree with maximum branching factor b is at most $1 + b + b^2 + \cdots + b^d$.

- Time complexity. If the shortest path to a goal state has length d, then in the worst case BFS will look at

$$1 + b + b^2 + \cdots + b^d$$

paths before reaching a goal state (exponential).

- Space complexity. If the shortest path to a goal state has length d, then in the worst case the frontier can contain b^d paths and so the memory requirement is b^d (exponential).

Properties of Breadth First Search

Facts. (1) The number of paths of length d in a search tree with maximum branching factor b is at most b^d.
(2) The number of paths of length at most d in a search tree with maximum branching factor b is at most $1 + b + b^2 + \cdots + b^d$.

- Time complexity. If the shortest path to a goal state has length d, then in the worst case BFS will look at

  $$1 + b + b^2 + \cdots + b^d$$

  paths before reaching a goal state (exponential).
- Space complexity. If the shortest path to a goal state has length d, then in the worst case the frontier can contain b^d paths and so the memory requirement is b^d (exponential).
- Advantage of BFS: it is complete (guaranteed to reach a solution if one exists).

Properties of Breadth First Search

Facts. (1) The number of paths of length d in a search tree with maximum branching factor b is at most b^d.
(2) The number of paths of length at most d in a search tree with maximum branching factor b is at most $1 + b + b^2 + \cdots + b^d$.

- Time complexity. If the shortest path to a goal state has length d, then in the worst case BFS will look at

$$1 + b + b^2 + \cdots + b^d$$

  paths before reaching a goal state (exponential).
- Space complexity. If the shortest path to a goal state has length d, then in the worst case the frontier can contain b^d paths and so the memory requirement is b^d (exponential).
- Advantage of BFS: it is complete (guaranteed to reach a solution if one exists).
- Advantage of BFS: it is optimal (always finds the shortest solution).

Complexity

Depth	paths	Time
0	1	1 msec
1	11	.01 sec
2	111	.1 sec
4	11,111	11 secs
6	10^6	18 mins
8	10^8	31 hours
10	10^{10}	128 days
12	10^{12}	35 years

Combinatorial explosion!

Time for breadth first search, assuming a branching factor of 10 and approximately 1000 states are expanded per second.

Properties of Depth First Search (DFS)

- Not complete: DFS is not guaranteed to find a solution.

Properties of Depth First Search (DFS)

- Not complete: DFS is not guaranteed to find a solution.
- Not optimal: Solution found is not guaranteed to be the shortest path.

Properties of Depth First Search (DFS)

- Not complete: DFS is not guaranteed to find a solution.
- Not optimal: Solution found is not guaranteed to be the shortest path.
- Time complexity: If the length of the longest path from the start state is m, then in the worst case DFS will look at

  $$1 + b + b^2 + \cdots + b^m$$

  paths before reaching a goal state (exponential). Note that m can be infinite, in which case in the worst case it does not terminate.

Properties of Depth First Search (DFS)

- Not complete: DFS is not guaranteed to find a solution.
- Not optimal: Solution found is not guaranteed to be the shortest path.
- Time complexity: If the length of the longest path from the start state is m, then in the worst case DFS will look at

$$1 + b + b^2 + \cdots + b^m$$

  paths before reaching a goal state (exponential). Note that m can be infinite, in which case in the worst case it does not terminate.
- Space complexity: If the length of the longest path starting at a goal state is m, then in the worst case the frontier can contain

$$b \times m$$

  paths. If m is infinite, then the space requirement is infinite in the worst case. But if DFS finds a path to the goal state, then memory requirement is much less than for BFS.

Basic Search Strategies

- BFS is complete but expensive.
- DFS is cheap in space complexity but incomplete
- Can't we do better than this?

Next Lectures

Search strategies:

- *Blind* search strategies
  no additional information about states beyond that provided in the problem definition
  - Breadth First Search
  - Depth First Search
  - ...
- *Heuristic* search strategies
  Know whether one non-goal state is "more promising" than another
  - greedy search
  - A^* search