Today

Uninformed Search Methods:
Uninformed Search Methods:

Depth-First Search
Today

Uninformed Search Methods:

Depth-First Search
Breadth-First Search
Today

Uninformed Search Methods:

Depth-First Search
Breadth-First Search
Uniform-Cost Search
Uninformed Search Methods:
  Depth-First Search
  Breadth-First Search
  Uniform-Cost Search
Informed Search:
Today

Uninformed Search Methods:
- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:
- A* or “A star”.
Today

Uninformed Search Methods:
  Depth-First Search
  Breadth-First Search
  Uniform-Cost Search

Informed Search:
  A* or “A star”.
  Sirius?
Today

Uninformed Search Methods:

- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:

- A* or “A star”.
- Sirius? Brightest star in sky.
Uninformed Search Methods:
- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:
- A* or “A star”.
  - Sirius? Brightest star in sky.
  - No!
Uninformed Search Methods:

- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:

- A* or “A star”.
  - Sirius? Brightest star in sky.
  - No! search!

Main idea:
Uninformed Search Methods:

Depth-First Search
Breadth-First Search
Uniform-Cost Search

Informed Search:

A* or “A star”.
Sirius? Brightest star in sky.
No! search! Main idea: Heuristics.
Uninformed Search Methods:
  Depth-First Search
  Breadth-First Search
  Uniform-Cost Search

Informed Search:
  A* or “A star”.
    Sirius? Brightest star in sky.
    No! search! Main idea: Heuristics.
    Admissible.
Uninformed Search Methods:
- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:
- A* or “A star”.
  - Sirius? Brightest star in sky.
  - No! search!
  - Main idea: Heuristics.
  - Admissible.

Graph Search:
Uninformed Search Methods:

Depth-First Search
Breadth-First Search
Uniform-Cost Search

Informed Search:

A* or “A star”.
Sirius? Brightest star in sky.
No! search! Main idea: Heuristics.
Admissible.
Graph Search: Consistent.
State space graph: A mathematical representation of a search problem

State Space Graphs

Tiny search graph for a tiny search problem
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)

Tiny search graph for a tiny search problem
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- The goal test is a set of goal nodes (maybe only one)
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- The goal test is a set of goal nodes (maybe only one)

In a state space graph, each state occurs only once!
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- The goal test is a set of goal nodes (maybe only one)

In a state space graph, each state occurs only once!

We can rarely build this full graph in memory (it’s too big),
State Space Graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successors (action results)
- The goal test is a set of goal nodes (maybe only one)

In a state space graph, each state occurs only once!

We can rarely build this full graph in memory (it’s too big), but it’s a useful idea.
Search Trees

This is now / start

A search tree:

Possible futures.
Search Trees

This is now / start

A search tree:
A “what if” tree of plans and their outcomes

Possible futures.
Search Trees

A search tree:
A “what if” tree of plans and their outcomes
The start state is the root node

This is now / start

“N”

“A”

Possible futures.

“E”

“N”

“E”

“N”

Possible futures.
Search Trees

A search tree:
A “what if” tree of plans and their outcomes
The start state is the root node
Children correspond to successors

Possible futures.
Search Trees

This is now / start

A search tree:
A “what if” tree of plans and their outcomes
The start state is the root node
Children correspond to successors
Nodes show states, but correspond to PLANS that achieve those states

Possible futures.
Search Trees

This is now / start

A search tree:
A “what if” tree of plans and their outcomes
The start state is the root node
Children correspond to successors
Nodes show states, but correspond to PLANS that achieve those states
For most problems, we can never actually build the whole tree

Possible futures.
State Space Graphs vs. Search Trees

Each NODE in search tree is an entire PATH in state space graph.
State Space Graphs vs. Search Trees

Each NODE in search tree is an entire PATH in state space graph.
State Space Graphs vs. Search Trees

Each NODE in search tree is an entire PATH in state space graph. We construct both on demand – and we construct as little as possible.
Tree Search: example.
Depth-First Search
Depth-First Search

Strategy: expand a deepest node first.
Depth-First Search

Strategy: expand a deepest node first.
Implementation: Fringe is a LIFO stack
Tree Search: example.

Strategy: expand a deepest node first

Implementation:
Fringe is a LIFO stack
Search Algorithm Properties
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

![Sketch of search tree]

- **b** is the branching factor
- **m** is the maximum depth
- **solutions at various depths**

Number of nodes in entire tree:

\[ 1 + b + b^2 + \cdots = O(b^m) \]
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

![Sketch of search tree]

- **b** is the branching factor
- **m** is the maximum depth
- **Solutions at various depths**
- **Number of nodes in entire tree:** $1 + b + b^2 + \cdots = O(b^m)$
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?
Search Algorithm Properties

**Complete**: Guaranteed to find a solution if one exists?

**Optimal**: Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $b^m$ nodes
- $b^2$ nodes
- $b$ nodes
- 1 node
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth

Number of nodes in entire tree:
- $1 + b + b^2 + \cdots = O(b^m)$
Search Algorithm Properties

Complete: Guaranteed to find a solution if one exists?

Optimal: Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- b is the branching factor
- m is the maximum depth
- solutions at various depths

\[ 1 + b + b^2 + \cdots = O(b^m) \]
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- \( b \) is the branching factor
- \( m \) is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?
- $1$
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?
- $1 + b$
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?
$$1 + b + b^2 + \cdots$$
Search Algorithm Properties

**Complete:** Guaranteed to find a solution if one exists?

**Optimal:** Guaranteed to find the least cost path?

Time complexity?

Space complexity?

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?
$$1 + b + b^2 + \cdots = O(b^m)$$
Depth-First Search (DFS) Properties

Which nodes expanded?

- Some left prefix of the tree.
- Could process the whole tree!
- If $m$ is finite, takes time $O(b^m)$.
- How much space does the fringe take? Only has siblings on path to root, so $O(bm)$.
- Is it complete? $m$ could be infinite, so only if we prevent cycles (more later).
- Is it optimal? No, it finds the "leftmost" solution, regardless of depth or cost.
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!

- If \( m \) is finite, takes time \( O(b^m) \).
- How much space does the fringe take?
  Only has siblings on path to root, so \( O(b^m) \).
- Is it complete?
  \( m \) could be infinite, so only if we prevent cycles (more later).
- Is it optimal?
  No, it finds the "leftmost" solution, regardless of depth or cost.
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If $m$ is finite, takes time $O(b^m)$

How much space does the fringe take?
Only has siblings on path to root, so $O(b^m)$

Is it complete?
$m$ could be infinite, so only if we prevent cycles (more later)

Is it optimal?
No, it finds the "leftmost" solution, regardless of depth or cost
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If \( m \) is finite, takes time \( O(b^m) \)
How much space does the fringe take?

If \( m \) could be infinite, only if we prevent cycles (more later)
Is it optimal?
No, it finds the "leftmost" solution, regardless of depth or cost
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If \( m \) is finite, takes time \( O(b^m) \)

How much space does the fringe take?
Only has siblings on path to root, so \( O(bm) \)
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If \( m \) is finite, takes time \( O(b^m) \)

How much space does the fringe take?
Only has siblings on path to root, so \( O(bm) \)

Is it complete?

m could be infinite, so only if we prevent cycles (more later)

Is it optimal?
No, it finds the "leftmost" solution, regardless of depth or cost
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If $m$ is finite, takes time $O(b^m)$

How much space does the fringe take?
Only has siblings on path to root, so $O(bm)$

Is it complete?
$m$ could be infinite, so only if we prevent cycles (more later)
Depth-First Search (DFS) Properties

Which nodes expanded?
Some left prefix of the tree.
Could process the whole tree!
If \( m \) is finite, takes time \( O(b^m) \)

How much space does the fringe take?
Only has siblings on path to root, so \( O(bm) \)

Is it complete?
\( m \) could be infinite, so only if we prevent cycles (more later)

Is it optimal?
No, it finds the "leftmost" solution, regardless of depth or cost
Depth-First Search (DFS) Properties

Which nodes expanded?
- Some left prefix of the tree.
- Could process the whole tree!
- If $m$ is finite, takes time $O(b^m)$

How much space does the fringe take?
- Only has siblings on path to root, so $O(bm)$

Is it complete?
- $m$ could be infinite, so only if we prevent cycles (more later)

Is it optimal?
- No, it finds the "leftmost" solution, regardless of depth or cost
Breadth-First Search
Breadth-First Search

Strategy: expand a shallowest node first
Breadth-First Search

Strategy: expand a shallowest node first
Implementation: Fringe is a FIFO queue
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
Let depth of shallowest solution be $s$
Search takes time $O(b^s)$
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
Let depth of shallowest solution be \( s \)
Search takes time \( O(b^s) \)

How much space does the fringe take?

- \( 1 \) node
- \( b \) nodes
- \( b^2 \) nodes
- \( b^3 \) nodes
- \( b^s \) nodes
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
Let depth of shallowest solution be $s$
Search takes time $O(b^s)$

How much space does the fringe take? Has roughly the last tier, so $O(b^s)$
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution

Let depth of shallowest solution be $s$

Search takes time $O(b^s)$

How much space does the fringe take?

Has roughly the last tier, so $O(b^s)$

Is it complete?

$s$ must be finite if a solution exists, so yes!

Is it optimal?

Only if costs are all 1 (more on costs later).
Breadth-First Search (BFS) Properties

What nodes does BFS expand?
Processes all nodes above shallowest solution
Let depth of shallowest solution be $s$
Search takes time $O(b^s)$

How much space does the fringe take?
Has roughly the last tier, so $O(b^s)$

Is it complete?
s must be finite if a solution exists,
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
Let depth of shallowest solution be $s$
Search takes time $O(b^s)$

How much space does the fringe take? Has roughly the last tier, so $O(b^s)$

Is it complete?
$s$ must be finite if a solution exists, so yes!
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
Let depth of shallowest solution be $s$
Search takes time $O(b^s)$

How much space does the fringe take? Has roughly the last tier, so $O(b^s)$

Is it complete?
$s$ must be finite if a solution exists, so yes!

Is it optimal?

Only if costs are all 1 (more on costs later).
Breadth-First Search (BFS) Properties

What nodes does BFS expand? Processes all nodes above shallowest solution
   Let depth of shallowest solution be $s$
   Search takes time $O(b^s)$

How much space does the fringe take? Has roughly the last tier, so $O(b^s)$

Is it complete? $s$ must be finite if a solution exists, so yes!

Is it optimal? Only if costs are all 1 (more on costs later).
Quiz: DFS vs BFS
DFS vs BFS

When will BFS outperform DFS?
When will DFS outperform BFS?

[Demo:
    dfs
    /
    bfs
    maze water (L2D6)]
DFS vs BFS

When will BFS outperform DFS?
When will DFS outperform BFS?

[Demo
:
dfs
/
bfs
maze water (L2D6)]

Space versus Time or Quality of Solution.
Video of Demo Maze Water DFS/BFS (part 1)
Video of Demo Maze Water DFS/BFS (part 2)
Next Up.

Iterative Deepening.
Next Up.

Iterative Deepening.
Uniform Cost Search.
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

Run a DFS with depth limit 1.
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

Run a DFS with depth limit 1.
If no solution...
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

Run a DFS with depth limit 1.
If no solution...
Run a DFS with depth limit 2.
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

- Run a DFS with depth limit 1.
  - If no solution...
- Run a DFS with depth limit 2.
  - If no solution ...

Isn’t that wastefully redundant?
Generally most work happens in the lowest level searched, so not so bad!
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

Run a DFS with depth limit 1.
If no solution...
Run a DFS with depth limit 2.
If no solution ...
Run a DFS with depth limit 3. ....
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

- Run a DFS with depth limit 1.
  - If no solution...
- Run a DFS with depth limit 2.
  - If no solution...
- Run a DFS with depth limit 3. ....

Isn’t that wastefully redundant?
Iterative Deepening

Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages

Run a DFS with depth limit 1.  
If no solution...  
Run a DFS with depth limit 2.  
If no solution ...  
Run a DFS with depth limit 3.  ....

Isn’t that wastefully redundant?

Generally most work happens in the lowest level searched, so not so bad!
Cost-Sensitive Search

BFS finds the shortest path in terms of number of actions.
Cost-Sensitive Search

BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.
BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.
Cost-Sensitive Search

BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.

How?
Uniform Cost Search (Djikstra’s algorithm.)
Uniform Cost Search

Strategy: expand a cheapest node first:

Diagram of a graph with labeled edges and nodes.
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)
Uniform Cost Search

Strategy: expand a cheapest node first:
Fringe is a priority queue (priority: cumulative cost)
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?

C/ε “tiers”

b

c ≤ 1

c ≤ 2

c ≤ 3
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!

How much space does the fringe take?
Last Tier Space: $O\left(\frac{bC^*/\epsilon}{\epsilon}\right)$.

Is it complete?
Finite solution cost/positive arc weights? Then yes.

Is it optimal?
Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost \( C^* \) and arc cost \( \geq \epsilon \):

\[
\text{depth } O\left(\frac{C^*}{\epsilon}\right)
\]

\[
\text{Time: } O\left(\frac{b}{\epsilon} \frac{C^*}{\epsilon}\right)
\]

\[
\text{Last Tier Space: } O\left(\frac{b}{\epsilon} \frac{C^*}{\epsilon}\right)
\]

Is it complete?
Finite solution cost/positive arc weights?
Then yes.

Is it optimal?
Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
depth $O(C^*/\varepsilon)$. 

Time: $O(bC^*/\varepsilon)$.

How much space does the fringe take?
Last Tier Space: $O(bC^*/\varepsilon)$.

Is it complete?
Finite solution cost/positive arc weights? Then yes.

Is it optimal?
Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
derthephepse $O(C^*/\varepsilon)$.
Time: $O(b^{C^*/\varepsilon})$. 

Is it complete?
Finite solution cost/positive arc weights?
Then yes.
Is it optimal?
Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
- depth $O(C^*/\varepsilon)$.
- Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
  depth $O(C^*/\varepsilon)$.
Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\varepsilon})$. 
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
  depth $O(C^*/\varepsilon)$.
  Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\varepsilon})$.

Is it complete?
Finite solution cost/positive arc weights?
  Then yes.
Is it optimal?
  Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost \( C^* \) and arc cost \( \geq \varepsilon \):
depth \( O(C^*/\varepsilon) \).
Time: \( O(b^{C^*/\varepsilon}) \).

How much space does the fringe take?
Last Tier Space: \( O(b^{C^*/\varepsilon}) \).

Is it complete?
Finite solution cost/positive arc weights?

Is it optimal?
Yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
   depth $O(C^*/\varepsilon)$.
Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\varepsilon})$.

Is it complete?
Finite solution cost/positive arc weights?
Then yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
depth $O(C^*/\varepsilon)$.
Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\varepsilon})$.

Is it complete?
Finite solution cost/positive arc weights?
Then yes.
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \epsilon$:
  depth $O(C^*/\epsilon)$.
Time: $O(b^{C^*/\epsilon})$.

How much space does the fringe take?
  Last Tier Space: $O(b^{C^*/\epsilon})$.

Is it complete?
  Finite solution cost/positive arc weights?
  Then yes.

Is it optimal?
Uniform Cost Search (UCS) Properties

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \varepsilon$:
depth $O(C^*/\varepsilon)$.
Time: $O(b^{C^*/\varepsilon})$.

How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\varepsilon})$.

Is it complete?
Finite solution cost/positive arc weights?
Then yes.

Is it optimal?
Yes.
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
Goes in every "direction".

The ugly?
Huh?
No information about goal location.
We'll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
No information about goal location.
We'll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]
Uniform Cost Issues

- **Remember:** UCS explores increasing cost contours.
- **The good:**
  - UCS is complete and optimal!
- **The bad:**
  - Goes in every “direction”.

The ugly?
- No information about goal location.
  - We’ll fix that soon!

**[Demo: empty grid UCS (L2D5)]**

**[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]**
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
Goes in every “direction”.

The ugly?

[Diagram: Empty grid UCS (L2D5)]

[Diagram: Maze with deep/shallow water DFS/BFS/UCS (L2D7)]
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
Goes in every “direction”.

The ugly?
Huh?

[Diagram of cost contours and a maze with deep/shallow water DFS/BFS/UCS (L2D7)]
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
Goes in every “direction”.

The ugly?
Huh?

No information about goal location.
Uniform Cost Issues

Remember: UCS explores increasing cost contours.

The good:
UCS is complete and optimal!

The bad:
Goes in every “direction”.

The ugly?
Huh?
No information about goal location.
We’ll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]
Video of Demo Empty UCS
Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 1)
Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 2)
Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 3)
The One Queue

All these search algorithms are the same except for fringe strategies.
The One Queue

All these search algorithms are the same except for fringe strategies. Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
The One Queue

All these search algorithms are the same except for fringe strategies. Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)

Practically, for DFS and BFS, you can avoid the $\log(n)$ overhead from an actual priority queue, by using stacks and queues
The One Queue

All these search algorithms are the same except for fringe strategies. Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)

Practically, for DFS and BFS, you can avoid the $\log(n)$ overhead from an actual priority queue, by using stacks and queues

Can even code one implementation that takes a variable queuing object.
The One Queue

All these search algorithms are the same except for fringe strategies. Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities).

Practically, for DFS and BFS, you can avoid the $\log(n)$ overhead from an actual priority queue, by using stacks and queues.

Can even code one implementation that takes a variable queuing object.
Search and Models

Search operates over models (state spaces) of the world
Search and Models

Search operates over models (state spaces) of the world

The agent doesn’t actually try all the plans out in the real world!
Search and Models

Search operates over models (state spaces) of the world.
The agent doesn’t actually try all the plans out in the real world!
Planning is all “in simulation”
Search and Models

Search operates over models (state spaces) of the world.
The agent doesn’t actually try all the plans out in the real world!
Planning is all “in simulation”
Your search is only as good as your models. . .
Search and Models

Search operates over models (state spaces) of the world.
The agent doesn’t actually try all the plans out in the real world!
Planning is all “in simulation”
Your search is only as good as your models. . .
Search Gone Wrong?
Search Gone Wrong?

Model or State Space gone wrong!
<table>
<thead>
<tr>
<th><strong>Start</strong></th>
<th><strong>End</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Address in: Norway</td>
<td>Address in: Norway</td>
</tr>
<tr>
<td>Place name in: North America</td>
<td>Place name in: Europe</td>
</tr>
<tr>
<td>Street Address</td>
<td>Street Address</td>
</tr>
<tr>
<td>City: Haugesund</td>
<td>City:</td>
</tr>
<tr>
<td>Postal Code:</td>
<td>Postal Code:</td>
</tr>
</tbody>
</table>

**Search Tips**

**Units**
- Miles
- Kilometers

**Route Type**
- Quickest
- Shortest

**Map Style**
- Standard
- LineDrive™
Informed Search
Informed Search

Heuristics
Greedy Search
A* Search
Graph Search
Recap: Search
Recap: Search

Search problem:
States (configurations of the world)
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

Search tree:
- Nodes: represent plans for reaching states
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

Search tree:
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Nodes: represent plans for reaching states
Plans have costs (sum of action costs)
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Nodes: represent plans for reaching states
Plans have costs (sum of action costs)

Search algorithm:
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

Search tree:
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

Search algorithm:
- Systematically builds a search tree
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Nodes: represent plans for reaching states
Plans have costs (sum of action costs)

Search algorithm:
Systematically builds a search tree
Orders the fringe (unexplored nodes)
Recap: Search

**Search problem:**
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

**Search tree:**
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

**Search algorithm:**
- Systematically builds a search tree
- Orders the fringe (unexplored nodes)
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

Search tree:
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

Search algorithm:
- Systematically builds a search tree
- Orders the fringe (unexplored nodes)

Complete: finds a plan.
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Nodes: represent plans for reaching states
Plans have costs (sum of action costs)

Search algorithm:
Systematically builds a search tree
Orders the fringe (unexplored nodes)

Complete: finds a plan.
Optimal: finds least-cost plans
Recap: Search

Search problem:
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

Search tree:
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

Search algorithm:
- Systematically builds a search tree
- Orders the fringe (unexplored nodes)

Complete: finds a plan.
Optimal: finds least-cost plans
Example: Pancake Problem

Cost: Number of pancakes flipped
Example: Pancake Problem

BOUND S FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let f(σ) be the smallest number of prefix reversals that will transform σ to the identity permutation, and let f(n) be the largest such f(σ) for all σ in (the symmetric group) S_n. We show that f(n) ≤ (5n + 5)/3, and that f(n) ≥ 17n/16 for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function g(n) is shown to obey 3n/2 − 1 ≤ g(n) ≤ 2n + 3.
Example: Pancake Problem

State space graph with costs as weights.
General Tree Search

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end

General Tree Search

Action: flip top two
Cost: 2
Action: flip all four

function Tree-Search(problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
General Tree Search

**Action:** flip top two  
**Cost:** 2

**Action:** flip all four  
**Cost:** 4

---

```plaintext
function Tree-Search(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```
General Tree Search

Action: flip top two
Cost: 2

Action: flip all four
Cost: 4

Path to reach goal:

```
def Tree-Search(problem, strategy):
    return a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```
General Tree Search

Action: flip top two
Cost: 2

Action: flip all four
Cost: 4

Path to reach goal:
Flip four, flip three

function Tree-Search(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
end
General Tree Search

Action: flip top two  
Cost: 2

Action: flip all four  
Cost: 4

Path to reach goal:  
Flip four, flip three  
Total cost: 7
The One Queue

Our search algorithms are the same except for fringe strategies.
The One Queue

Our search algorithms are the same except for fringe strategies

- All fringes are priority queues: states with priorities.
Our search algorithms are the same except for fringe strategies

- All fringes are priority queues: states with priorities.
- DFS, BFS a bit faster using simple stack/queues.
The One Queue

Our search algorithms are the same except for fringe strategies

- All fringes are priority queues: states with priorities.
- DFS, BFS a bit faster using simple stack/queues.
- Can code one implementation with variable queuing object.
Uninformed Search
Strategy: expand lowest path cost.

The good: Complete and optimal!

The bad: Explores options in every "direction"
No information about goal location
Informed Search
A heuristic is:

▶ A function that estimates how close a state is to a goal
▶ Designed for a particular search problem
▶ Examples: Euclidean distance for pathing. Manhattan distance.
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples:
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Euclidean distance for pathing.
Search Heuristics

A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Euclidean distance for pathing. Manhattan distance.
Example: Heuristic Function

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

Heuristic: the number of the largest pancake that is still out of place
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place