Uninformed search

Christos Dimitrakakis

February 26, 2026

Outline

Search to find a solution

- Input: Problem specification (e.g. route-finding)
- Output: Solution (e.g. a policy)

Algorithms for finding solutions

- Must search through solution space
- Must return an feasible solution
- Will ideally return an optimal solution

Graphs in Search Algorithms

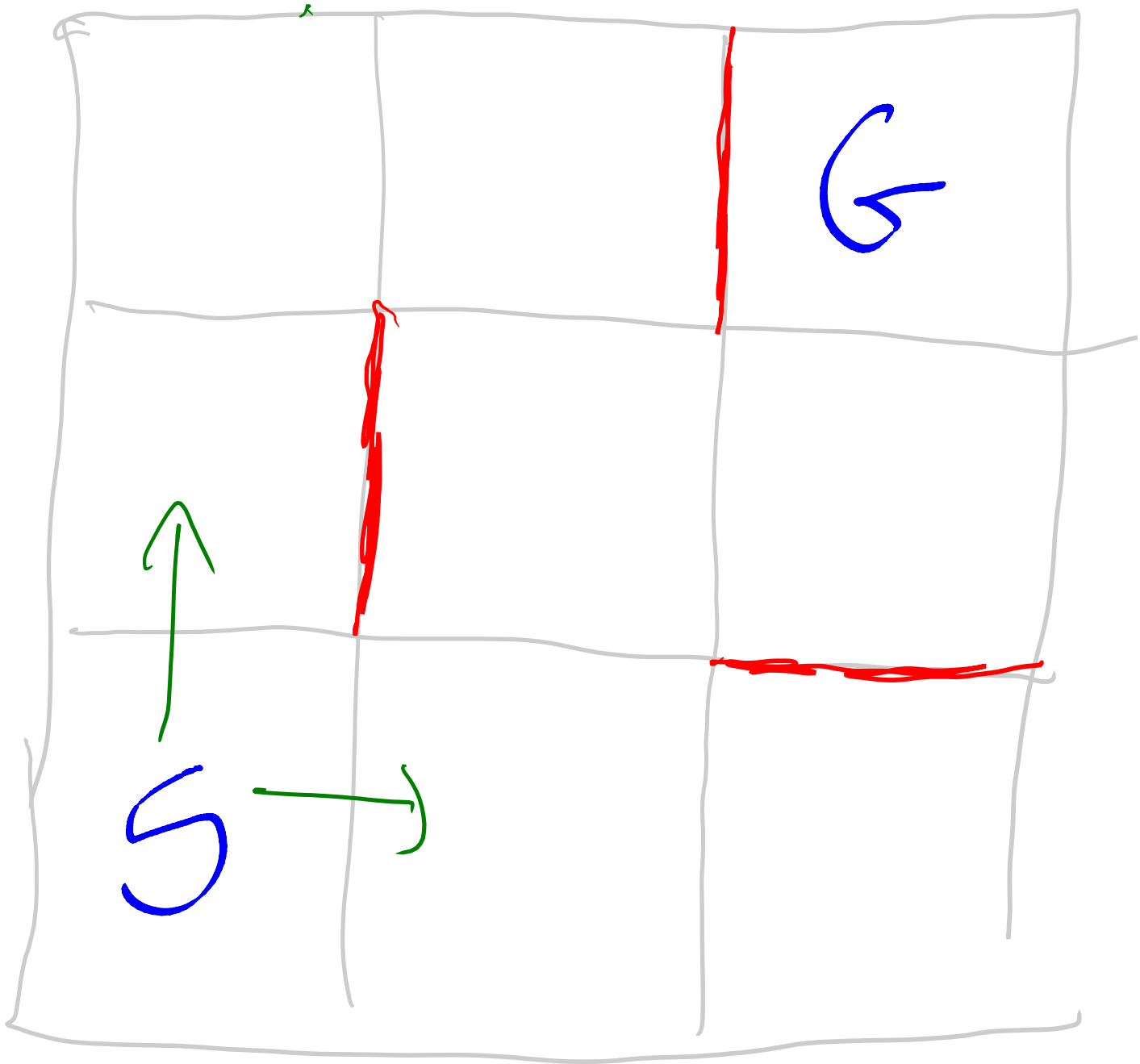
Problem Graph

- Specifies the problem.
- An abstraction of a real-world problem

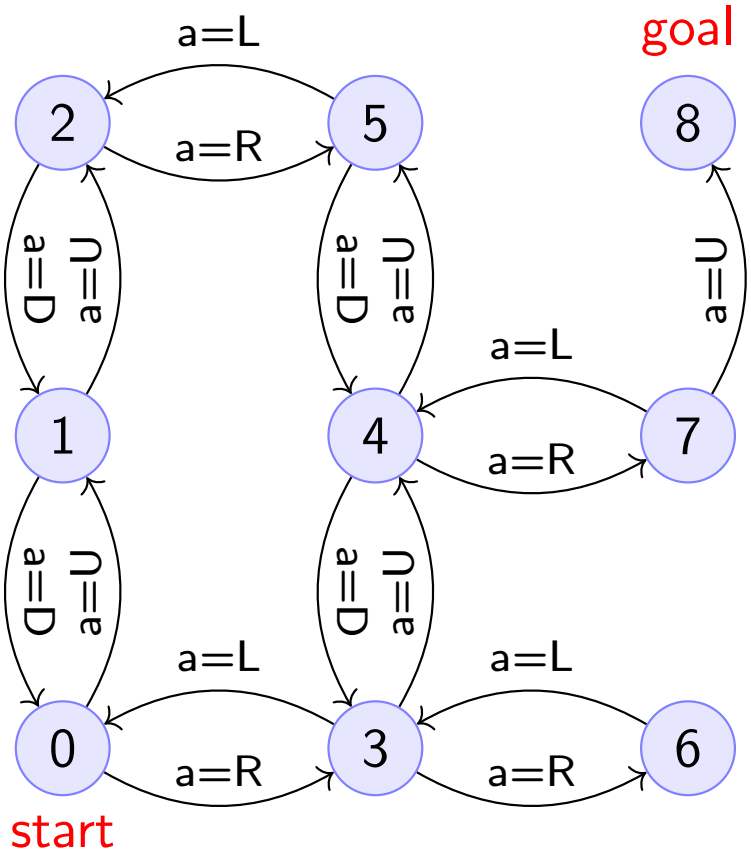
Search Graph

- Saves the state of the search.

A gridworld

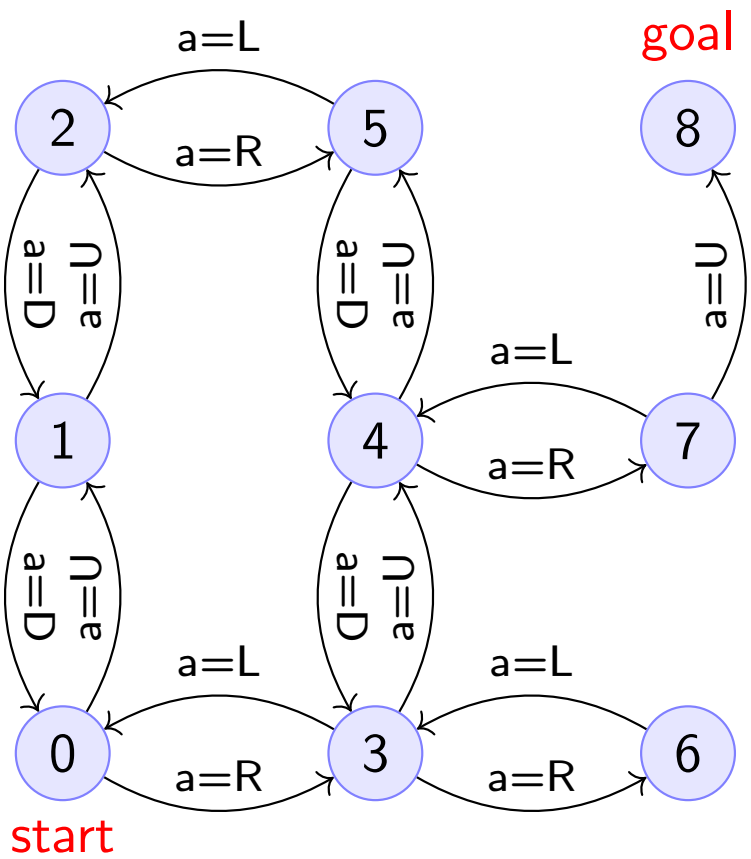


A gridworld-maze problem



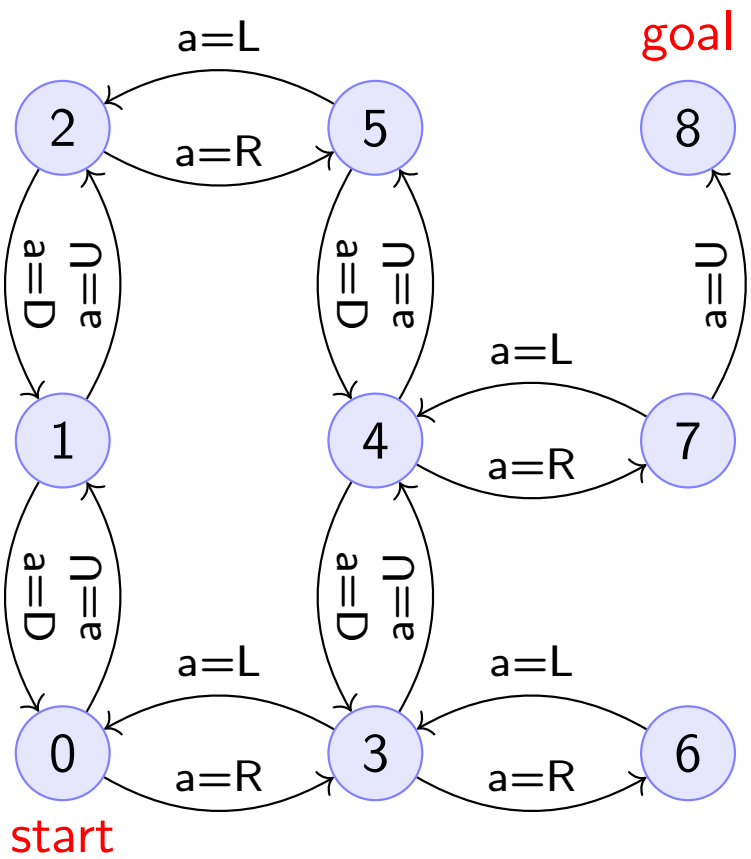
- States (États): $S = \{0, \ldots, 8\}$

A gridworld-maze problem



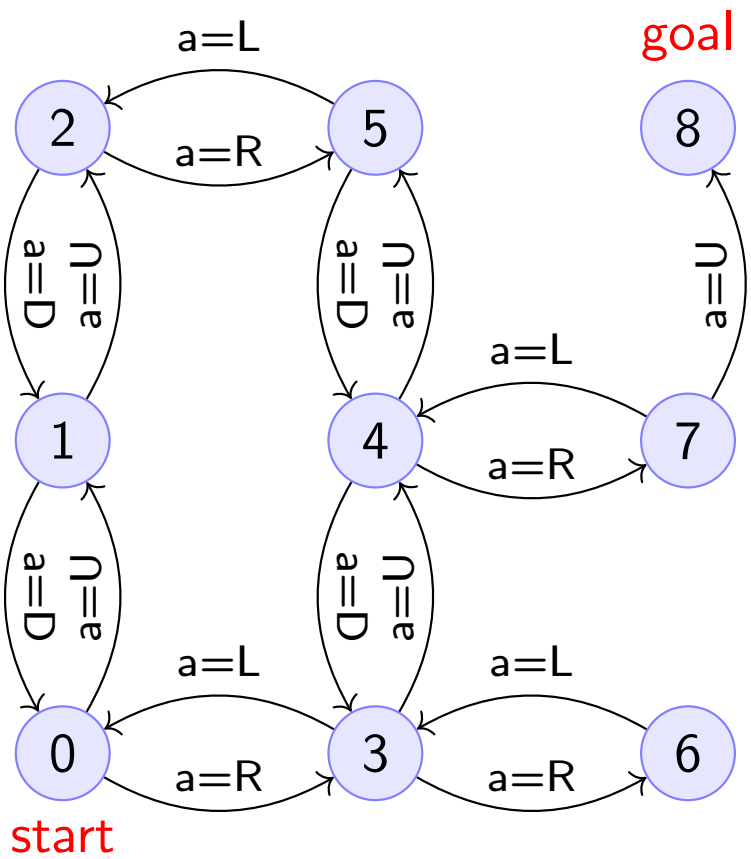
- States (États): $S = \{0, \ldots, 8\}$
- Actions: $A = \{U, D, L, R\}$.

A gridworld-maze problem



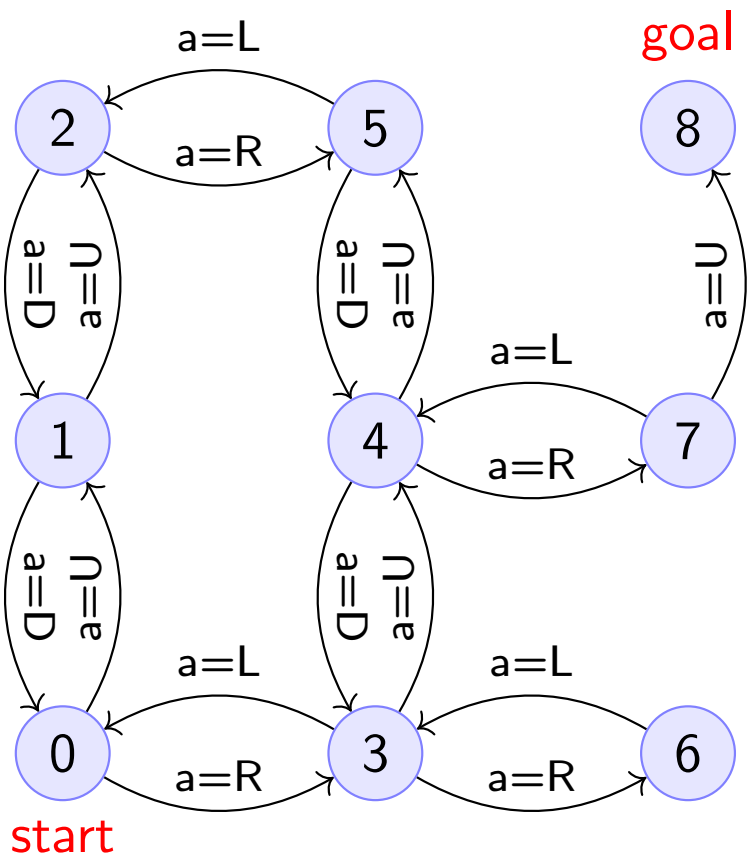
- States (États): $S = \{0, \ldots, 8\}$
- Actions: $A = \{U, D, L, R\}$.
- Policy (Politique): $\pi : S \to A$ chooses an action in every state.

A gridworld-maze problem



- States (États): $S = \{0, \ldots, 8\}$
- Actions: $A = \{U, D, L, R\}$.
- Policy (Politique): $\pi : S \to A$ chooses an action in every state.

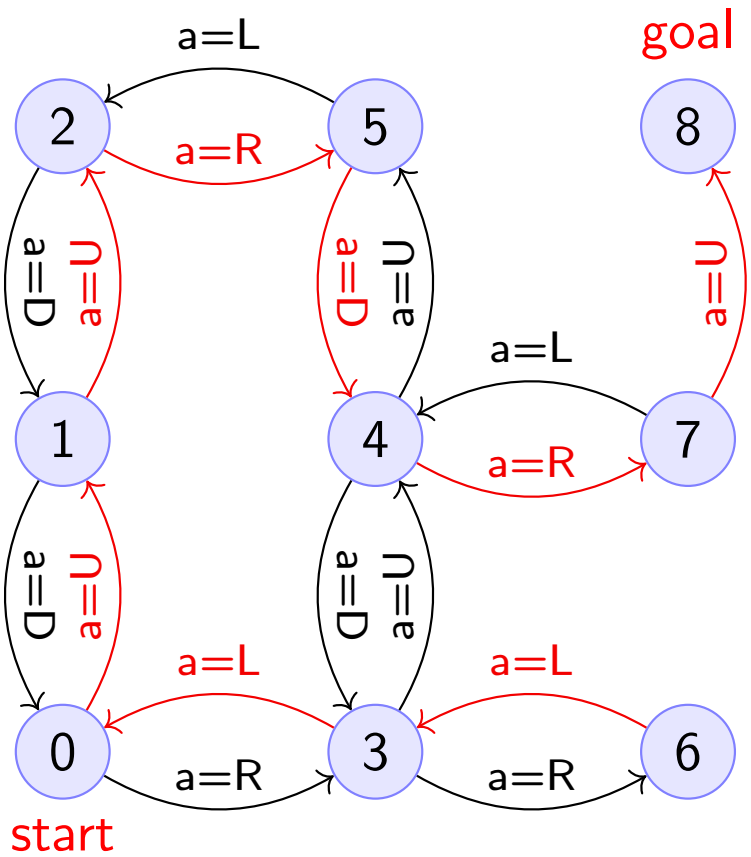
A gridworld-maze problem



- States (États): $S = \{0, \ldots, 8\}$
- Actions: $A = \{U, D, L, R\}$.
- Policy (Politique): $\pi : S \to A$ chooses an action in every state.

S	A
0	
1	
2	
3	
4	
5	
6	
7	
8	

A gridworld-maze solution



- States: $S = \{0, \ldots, 8\}$
- Actions: $A = \{U, D, L, R\}$.
- Policy: $\pi : S \to A$ chooses an action in every state.

S	A
0	U
1	U
2	R
3	L
4	R
5	D
6	L
7	U
8	

Graph definitions

Graph $G = \langle N, E \rangle$ (*graphe*)

A graph G is defined by:

- Set of nodes N (un sommet)
- Set of edges E (une arête), with $\langle x, y \rangle \in E$ and $x, y \in N$

Graph definitions

Graph $G = \langle N, E \rangle$ (*graphe*)

A graph G is defined by:

- Set of nodes N (un sommet)
- Set of edges E (une arête), with $\langle x, y \rangle \in E$ and $x, y \in N$

Labels and costs/rewards (*ettiquettes, coutes/récompenses*)

- Nodes can be labelled as e.g. start and goal states.
- Edges can be labelled with actions or costs/rewards

Graph definitions

Graph $G = \langle N, E \rangle$ (*graphe*)

A graph G is defined by:

- Set of nodes N (un sommet)
- Set of edges E (une arête), with $\langle x, y \rangle \in E$ and $x, y \in N$

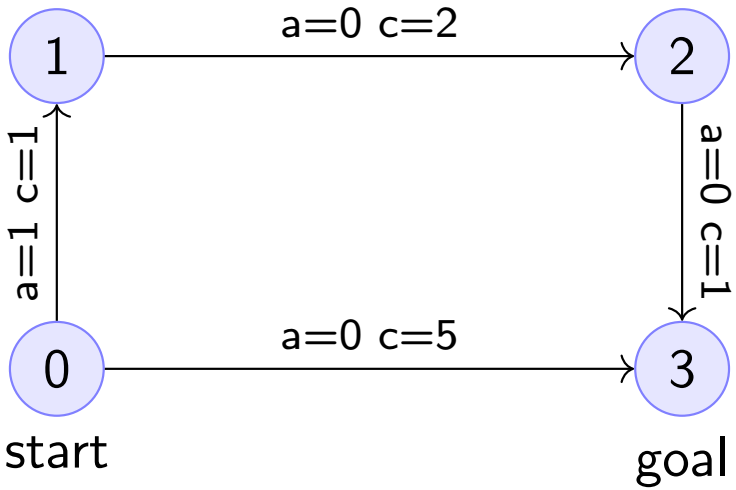
Labels and costs/rewards (*ettiquettes, coutes/récompenses*)

- Nodes can be labelled as e.g. start and goal states.
- Edges can be labelled with actions or costs/rewards

Paths and cycles (*chemins, circuits*)

- A path h, from x to y, in N, is a sequence: $\langle h_0, \ldots, h_k \rangle$ so that $h_0 = x, h_k = y$ and $\langle h_t, h_{t+1} \rangle \in E$.
- We write h_j for the j-th element of the path h.
- A cycle is a path $\langle h_0, \ldots, h_k \rangle$ where $h_0 = h_k$.
- If a graph has no cycles, it is acyclic
- A state with no outgoing edges to other states is terminal

Graph labels



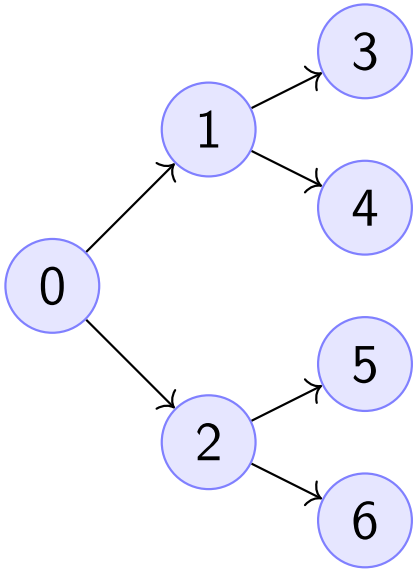
- Action labels a: Tell you which action traverses an edge
- Edge costs c: How much it costs to traverse an edge
- Node labels: Some nodes are goal or start nodes.

Notation

- For a node $i \in N$, we write $g(i) = 1$ if it's a goal label.
- For an edge $(i, j) \in E$, we write $c(i, j) \in \mathbb{R}$ for its cost.
- The total cost for a path $h = \langle x, \ldots, y \rangle$ from node x to a node y is

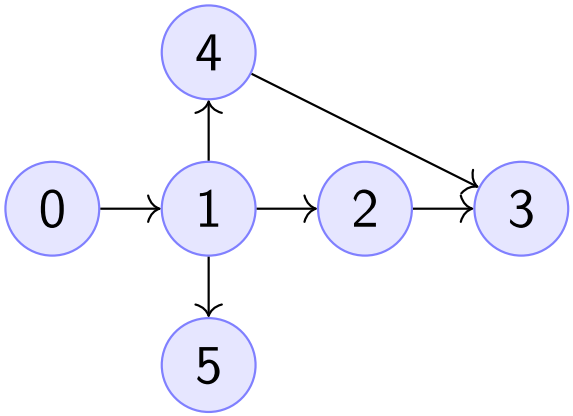
$$\sum_{k=0}^{|h|-1} c(h_k, h_{k+1}), \qquad \text{where } |h| \text{ is the length of } h$$

Tree example



- No matter where the goal is, there is a unique path to it.
- Depending on how the edges are ordered, finding it may require up to 6 steps.
- For binary trees of depth D, the worst-case complexity is 2^D.
- The branching factor is the number of children that each node can have.

Shortcut example



- There are two paths to node 3.
- Depending on how we search, we may find the longest path first.
- Why would we prefer one path to another?
- How should we search to find the *best* path?

Two types of graphs

State-space graphs: describe the problem

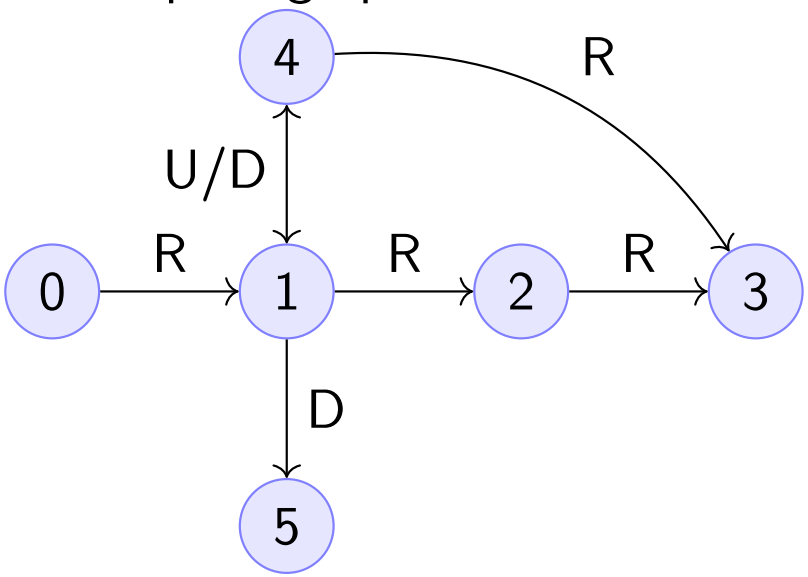
- Shows how to get from one state to the other
- Shows how much cost/reward we incur/obtain
- Shows what are start and goal sates

Search graphs: summarise the solution state

- Show which nodes we visited
- Summarise the best solution
- Allow us to choose the next node to visit in a clever way

Graph visualisation

State-space graph

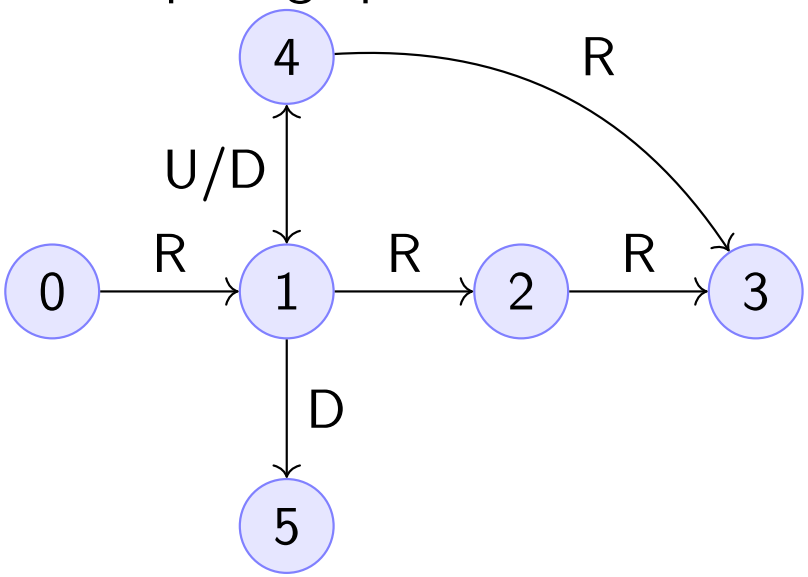


Search graph

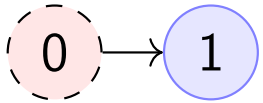
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



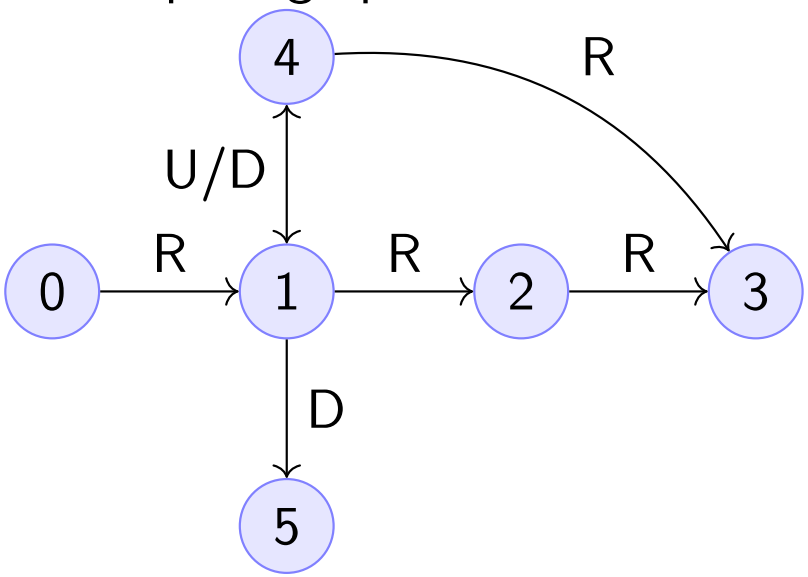
Search graph



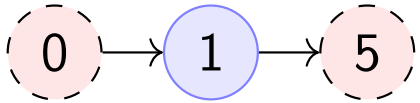
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



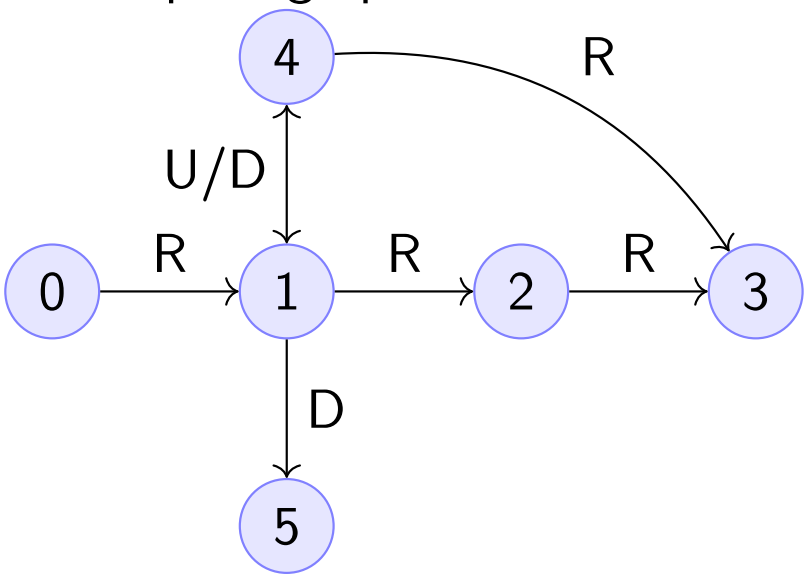
Search graph



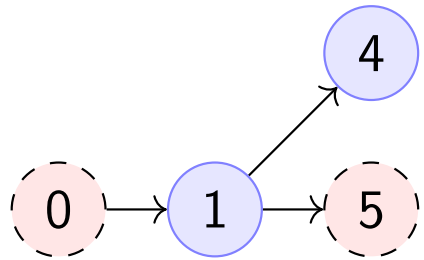
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



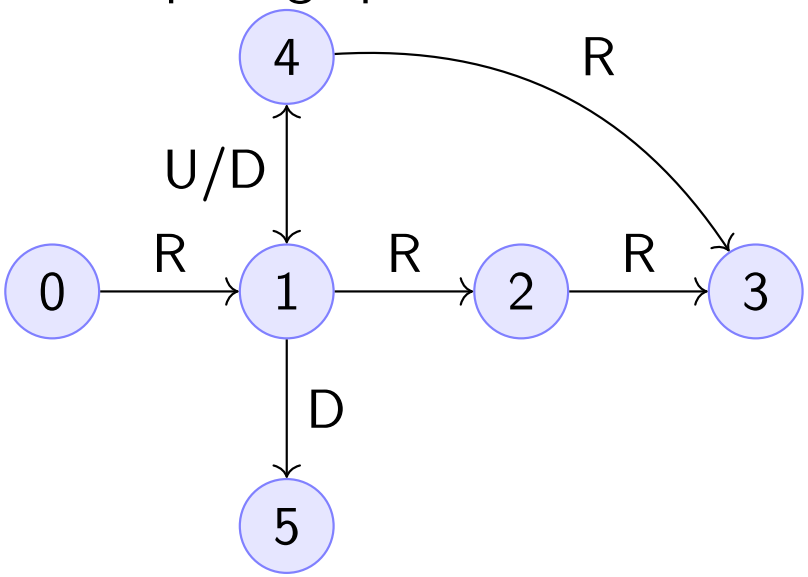
Search graph



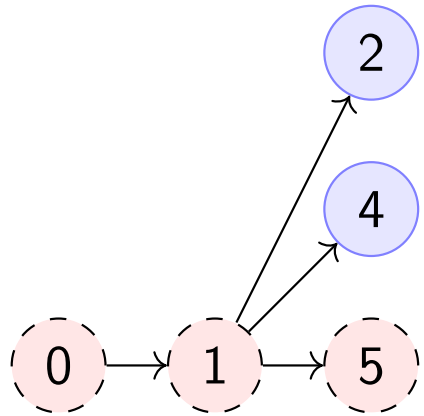
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



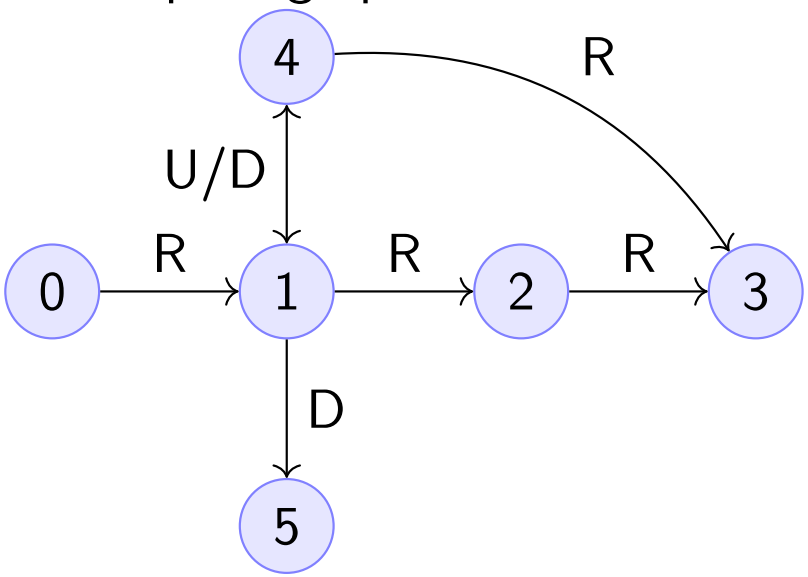
Search graph



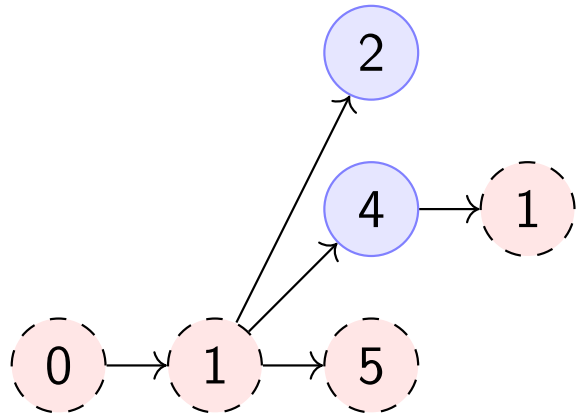
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



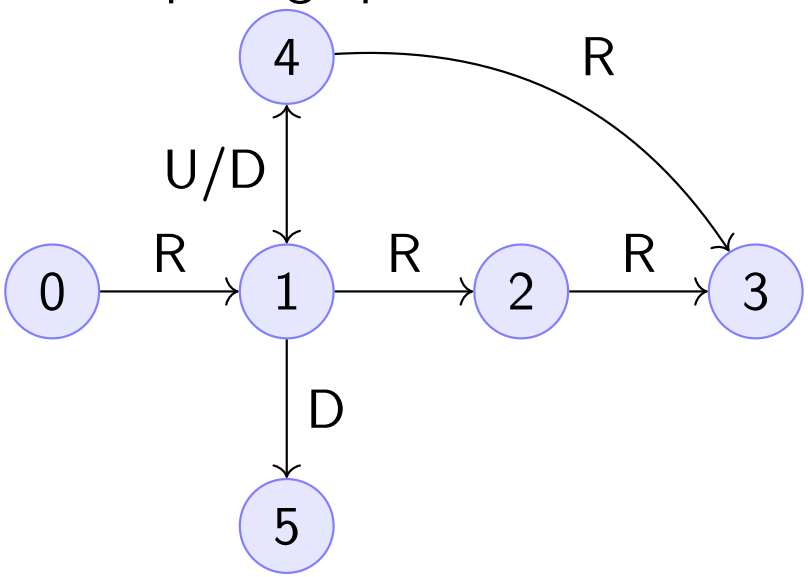
Search graph



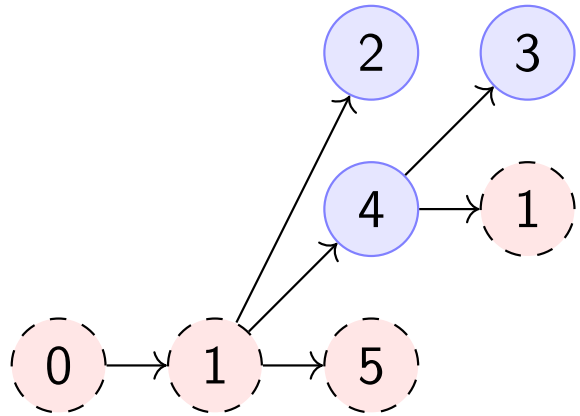
Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

Graph visualisation

State-space graph



Search graph



Our search algorithm selects the oldest node in the frontier first, and actions in the order D,U,L,R.

State-action problem specification

It is sometimes more convenient to specify a state-action graph.

- $s \in S$: state space
- $a \in A$: action space (with $A_s \subset S$ available actions in state s)
- $\tau : S \times A \to S$: transition model (deterministic)

State-action problem specification

It is sometimes more convenient to specify a state-action graph.

- $s \in S$: state space
- $a \in A$: action space (with $A_s \subset S$ available actions in state s)
- $\tau : S \times A \to S$: transition model (deterministic)

Resulting graph

- Nodes: S
- Edges from node i: $\{(i, \tau(i, a) | a \in A_s\}$

State-action problem specification

It is sometimes more convenient to specify a state-action graph.

- $s \in S$: state space
- $a \in A$: action space (with $A_s \subset S$ available actions in state s)
- $\tau : S \times A \to S$: transition model (deterministic)

Resulting graph

- Nodes: S
- Edges from node i: $\{(i, \tau(i, a) | a \in A_s\}$

Problem specifications

One or more of the following:

- $g : S \to \{0, 1\}$: goal indicator
- $c : S \times A \to \mathbb{R}$: step cost or constraint.
- $r : S \times A \to \mathbb{R}$: step reward.

State-action problem specification

It is sometimes more convenient to specify a state-action graph.

- $s \in S$: state space
- $a \in A$: action space (with $A_s \subset S$ available actions in state s)
- $\tau : S \times A \to S$: transition model (deterministic)

Resulting graph

- Nodes: S
- Edges from node i: $\{(i, \tau(i, a) | a \in A_s\}$

Problem specifications

One or more of the following:

- $g : S \to \{0, 1\}$: goal indicator
- $c : S \times A \to \mathbb{R}$: step cost or constraint.
- $r : S \times A \to \mathbb{R}$: step reward.

Solution specification

- $\pi : S \to A$ deterministic policy
- For deterministic problem and policy, the policy is open loop

The search graph S'.

- Node 0 is root of the search graph S'.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.
- state-action model: select action a in node i, and set $s' = \tau(s^i, a)$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.
- state-action model: select action a in node i, and set $s' = \tau(s^i, a)$.
- node-edge model: select edge (s^i, j) in node i, and set $s' = j$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.
- state-action model: select action a in node i, and set $s' = \tau(s^i, a)$.
- node-edge model: select edge (s^i, j) in node i, and set $s' = j$.
- $i + 1$ is now a child of i $[i + 1 \in \mathtt{Ch}(i)]$, with $s^{i+1} = s'$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.
- state-action model: select action a in node i, and set $s' = \tau(s^i, a)$.
- node-edge model: select edge (s^i, j) in node i, and set $s' = j$.
- $i + 1$ is now a child of i $[i + 1 \in \mathtt{Ch}(i)]$, with $s^{i+1} = s'$.
- Update the frontier $F_{k+1} = F_k \cup \{i + 1\} \setminus \{i\}$ and searched $S'_{k+1} = S'_k \cup \{i\}$.

The search graph S'.

- Node 0 is root of the search graph S'.
- Each node $i \in S'$ corresponds to a state $s^i \in S$.
- It also corresponds to a path $s^0, \ldots, \mathtt{Pa}(\mathtt{Pa}(s^i)), \mathtt{Pa}(s^i), s^i$.
- Node depth: $d_i = 1 + d_{\mathtt{Pa}(i)}$, with $d_0 = 0$.

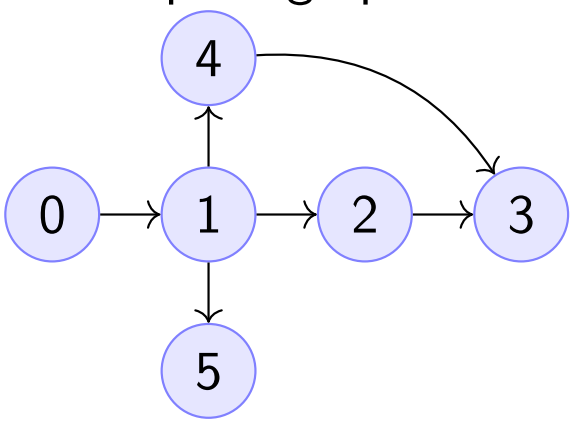
Frontier: what we can search.

Set $F_0 = \{0\}$ and set of searched nodes $S'_0 = \emptyset$.
At step $k = 0, 1, \ldots$:

- The frontier is F_k, and searched states S'_k.
- Select a node i, where $s^i \in F_k$.
- state-action model: select action a in node i, and set $s' = \tau(s^i, a)$.
- node-edge model: select edge (s^i, j) in node i, and set $s' = j$.
- $i + 1$ is now a child of i $[i + 1 \in \mathtt{Ch}(i)]$, with $s^{i+1} = s'$.
- Update the frontier $F_{k+1} = F_k \cup \{i + 1\} \setminus \{i\}$ and searched $S'_{k+1} = S'_k \cup \{i\}$.
- In the end, no more nodes can be added: $F_k = \emptyset$ and $S'_k = S'_{k+1}$

Depth-first search example

State-space graph

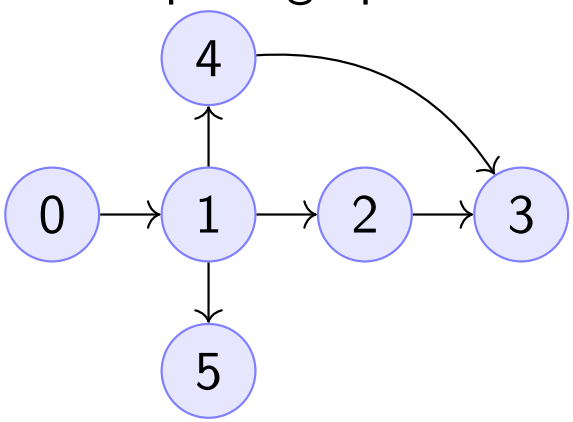


Search graph

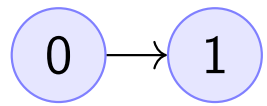
- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search example

State-space graph



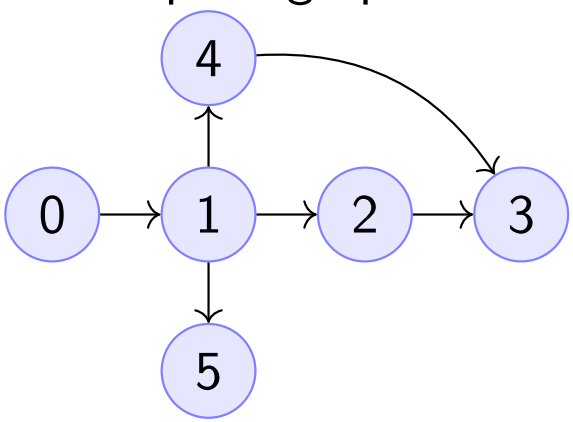
Search graph



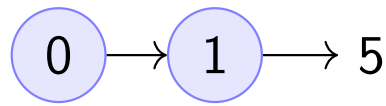
- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search example

State-space graph



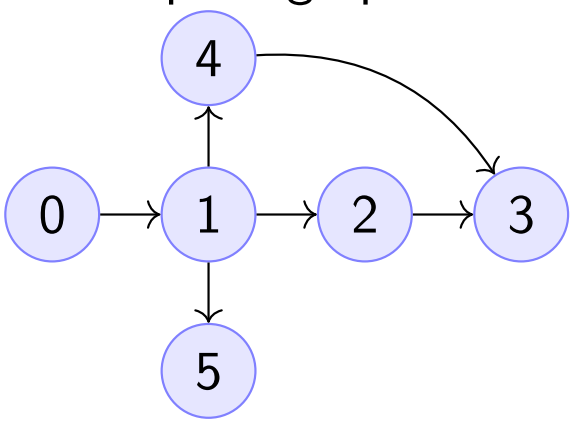
Search graph



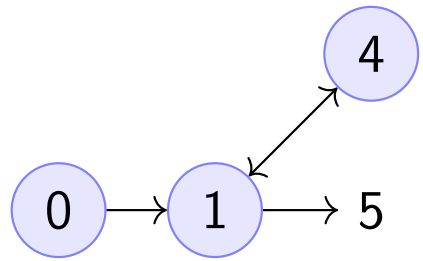
- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search example

State-space graph



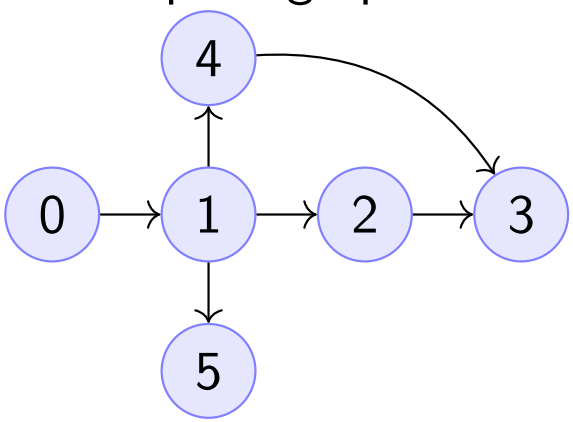
Search graph



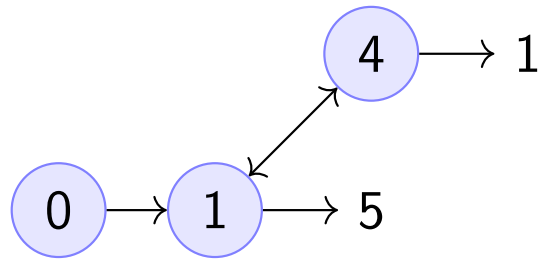
- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search example

State-space graph



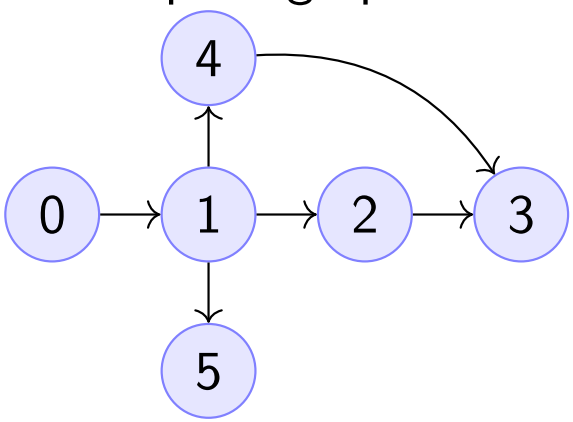
Search graph



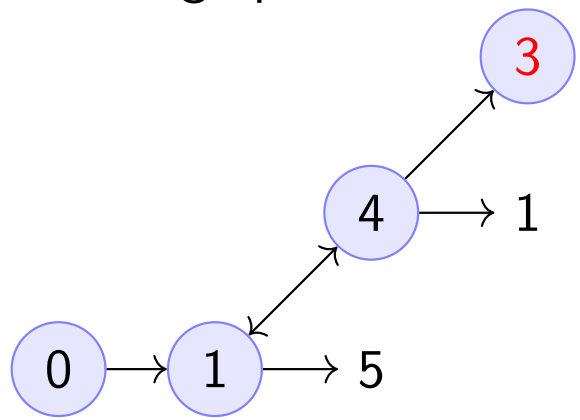
- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search example

State-space graph



Search graph



- Our search algorithm selects the deepest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Depth-first search

Generic depth-first search

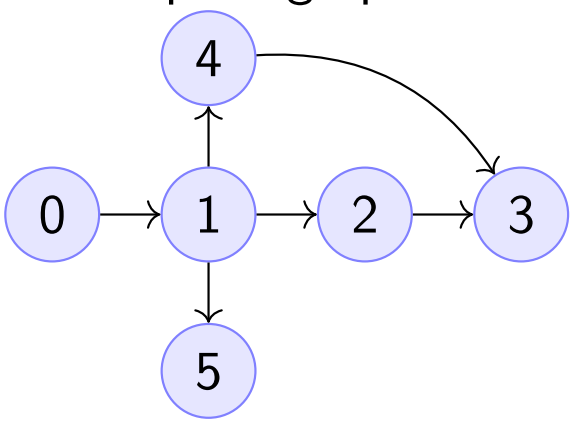
global $S' = \emptyset$: Nodes searched
input $G = \langle N, E \rangle$: Graph.
input n : Current node
function `DepthFirst`(G, n)
$S' = S' \cup \{n\}$: mark n as searched
for $c \notin F : \langle c, j \rangle \in E$ **do**
  if `DepthFirst`(G, j) **then**
    return 1.
  end if
end for

Discussion

- This function goes through all the nodes in the graph
- How can we use it to identify a paths to the goal?
- How can we modify it to identify all paths to the goal?
- How can we modify it to identify the shortest path to the goal?

Breadth-first search example

State-space graph

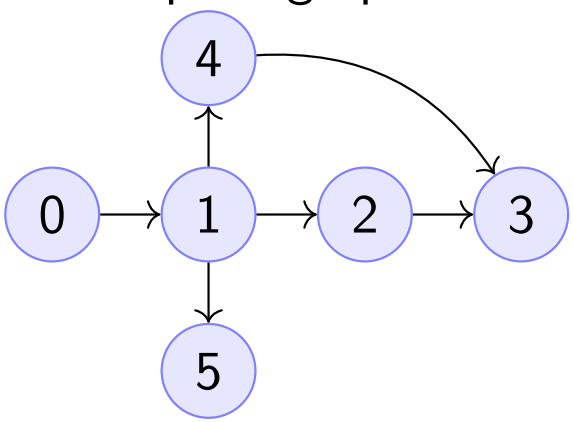


Search graph

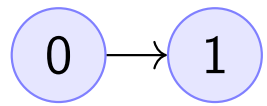
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search example

State-space graph



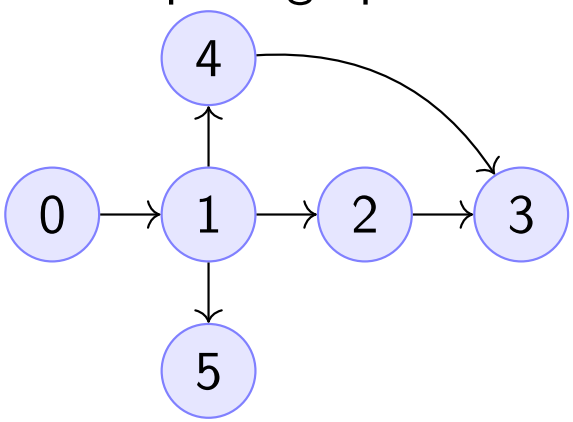
Search graph



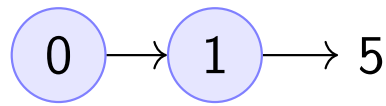
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search example

State-space graph



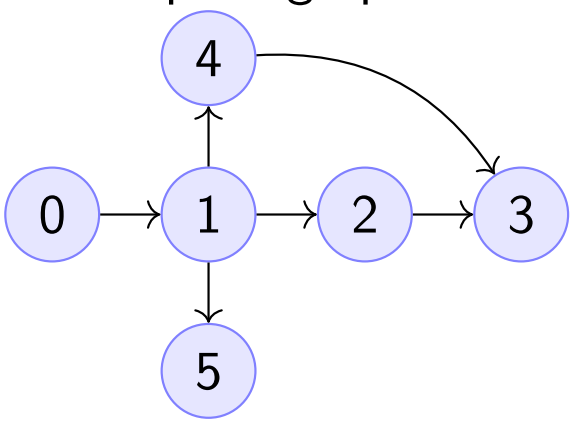
Search graph



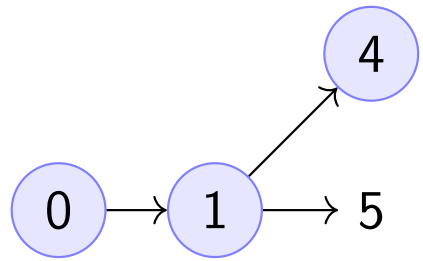
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search example

State-space graph



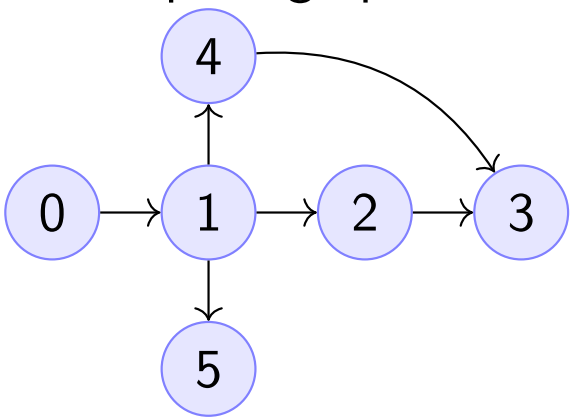
Search graph



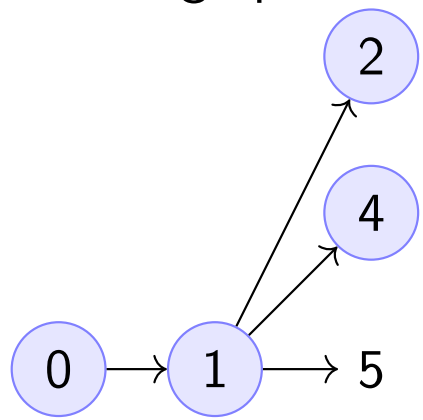
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search example

State-space graph



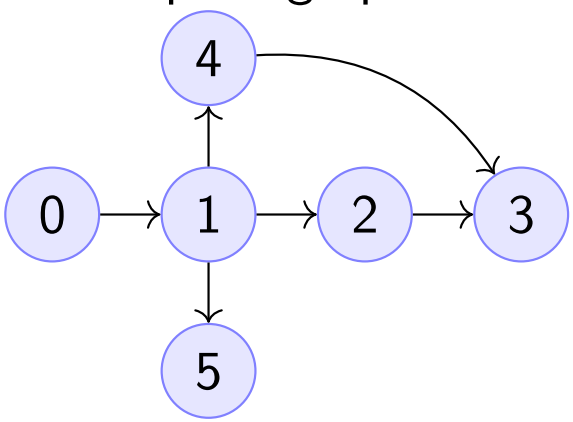
Search graph



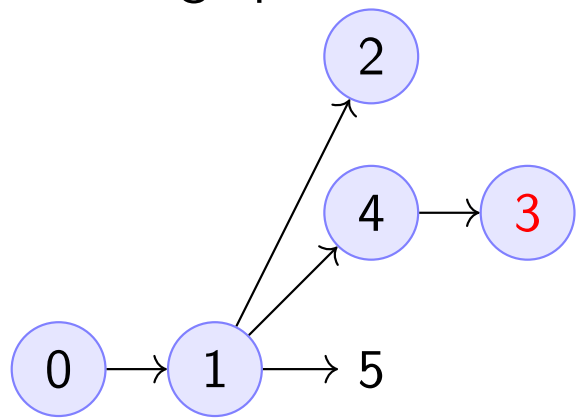
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search example

State-space graph



Search graph



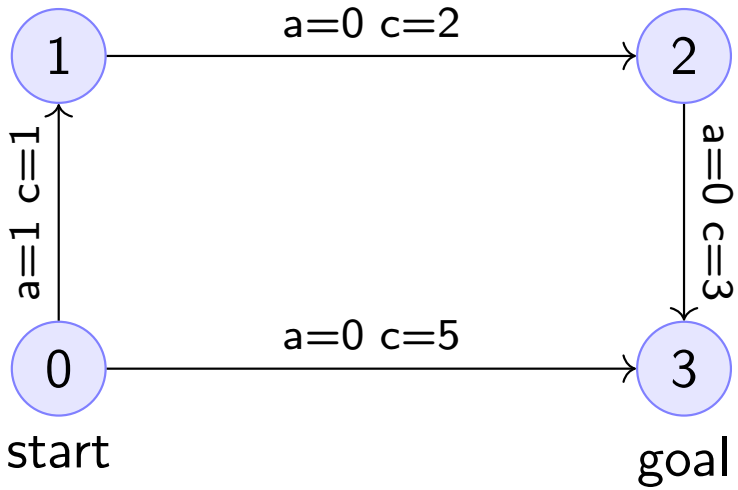
- Our search algorithm selects the shallowest node in the frontier first, and actions in the order D,U,L,R.
- What if our action search was ordered R, L, U, D ?

Breadth-first search

Unlike Depth-First search, this cannot easily use a recursive function call implementation.

input $G = \langle N, E \rangle$: Graph.
input x : Start node
function `BreadthFirst`(G, x)
$S' = \emptyset$: Nodes searched.
$F = \{x\}$. Initialise the frontier
while $F \neq \emptyset$ **do**
  $s = \arg\min_{i \in F} d_i$. Select minimum depth node.
  $S' = S' \cup \{s\}$. Add s to the list of searched nodes.
  $F = F \setminus \{s\}$. Remove s from the frontier.
  $F = F \cup \mathtt{Ch}(s)$. Add s's children to the frontier.
end while

Minimum-cost search example

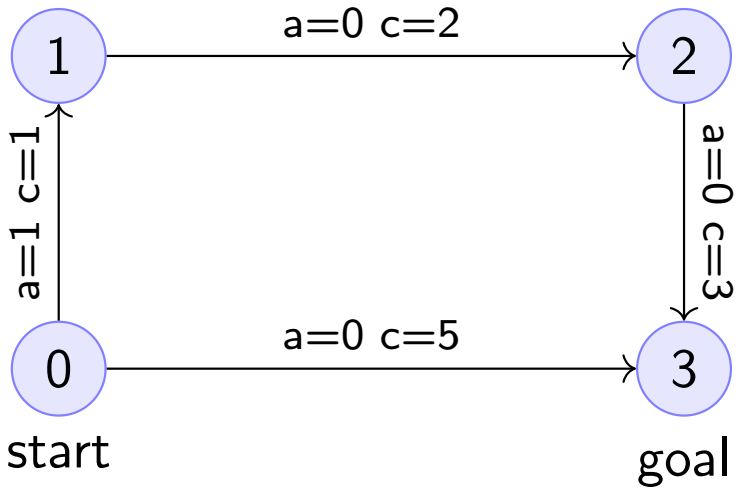


Search graph

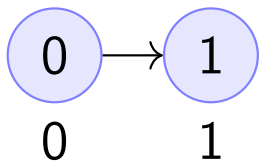
0

We have to keep track of the total cost at each search node.

Minimum-cost search example

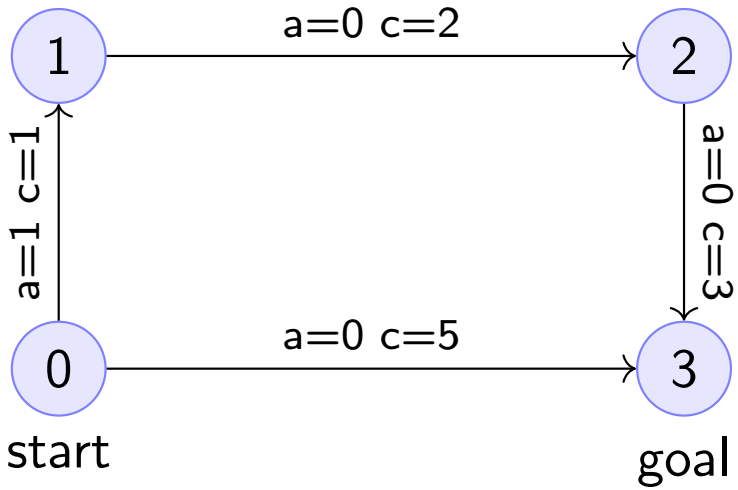


Search graph

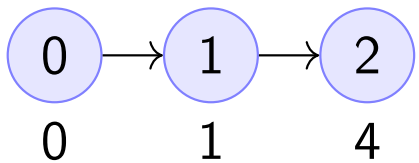


We have to keep track of the total cost at each search node.

Minimum-cost search example

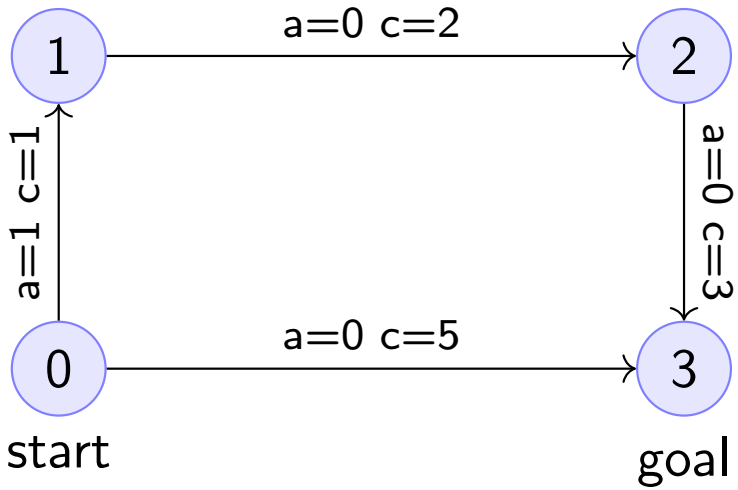


Search graph

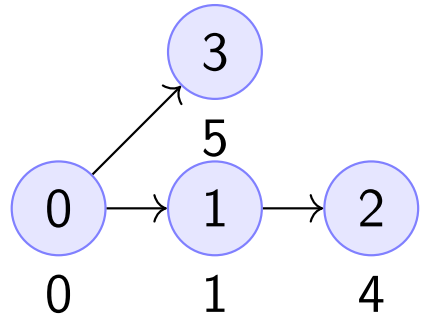


We have to keep track of the total cost at each search node.

Minimum-cost search example



Search graph



We have to keep track of the total cost at each search node.

Minimum-cost search

Note that BFS always adds the minimum depth node. We can instead add the minimum-cost node.

input $G = \langle N, E \rangle$: Graph.
input x : Start node
function $\texttt{BreadthFirst}(G, x)$
$S' = \emptyset$: Nodes searched.
$F = \{x\}$. Initialise the frontier
$c_x = 0$. Initialise the cost of node x
while $F \neq \emptyset$ **do**
  $n = \arg\min_{f \in F} c_f$. Select minimum cost node.
  $F = F \setminus \{n\}$. Remove n from the frontier.
  if $n \notin S'$ **then**
    $B = \{i \in \texttt{Ch}(n) : i \notin \texttt{An}(n)\}$. Get the set of unsearched children of n.
    $\forall b \in B$, $b_i = c_n + c(n, b)$. Calculate the total cost to each child b.
    $S' = S' \cup \{n\}$. Add n to the list of searched nodes.
    $F = F \cup B$. Add n's children to the frontier.
  end if
end while