Fundamentals of Artificial Intelligence

VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

COMP307/AIML420

Search 1

Dr. Heitor Murilo Gomes
heitor.gomes@vuw.ac.nz
http://www.heitorgomes.com

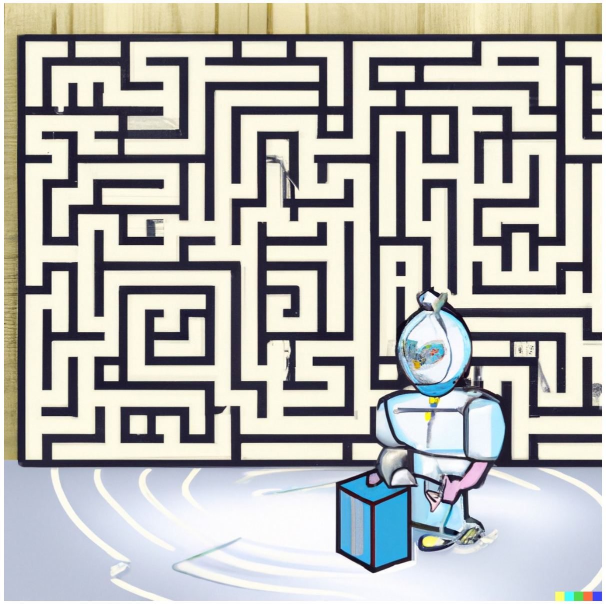
Information

- Assignment 1 (due on week 5 - 27 March 2024)
- Extension requests (use the Submission system)
- Teaching evaluation (Heitor)
- Helpdesks starting from 2pm until 4pm (Thursday until next Wednesday)

Search in AI

- Why searching is relevant in AI?
- An **agent** in an **environment**

- We are looking for a **solution** to a **problem** and we would like to know the steps (**path**) to reach such **solution**.

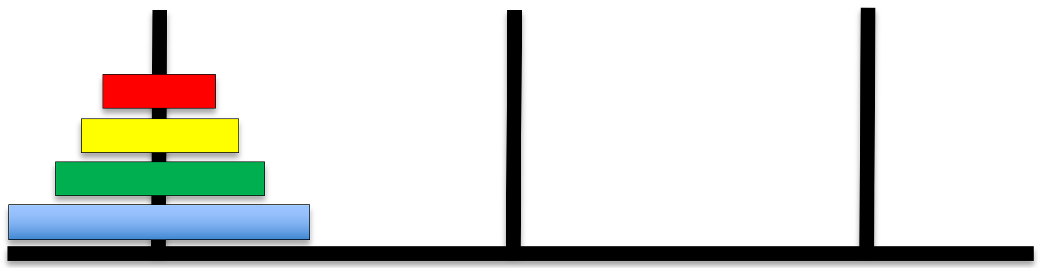


Generated with DALLE-2

- Several complex real-world problems rely on search
  - Robot navigation; University timetabling; Job shop scheduling; …
- Search is a critical step in several other AI techniques, such as machine learning and evolutionary computation.

Towers of Hanoi

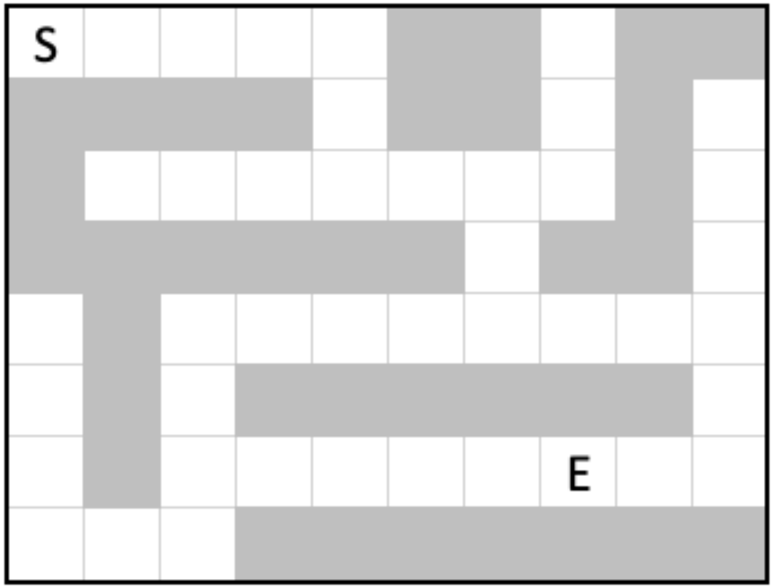
- Puzzle that consists of three pegs and a set of disks of different sizes



- Disks are **initially** stacked on one peg in decreasing order of size, with the largest at the bottom and the smallest at the top
- The **goal** is to move the entire stack to another peg, one disk at a time, <u>without placing a larger disk on top of a smaller one</u>

Maze

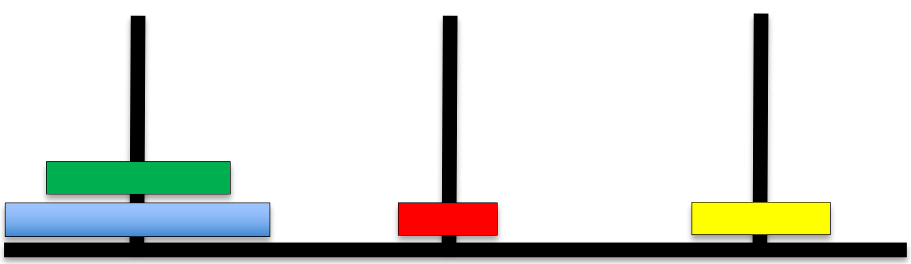
- Find a path from the Start position (S) to the End position (E)

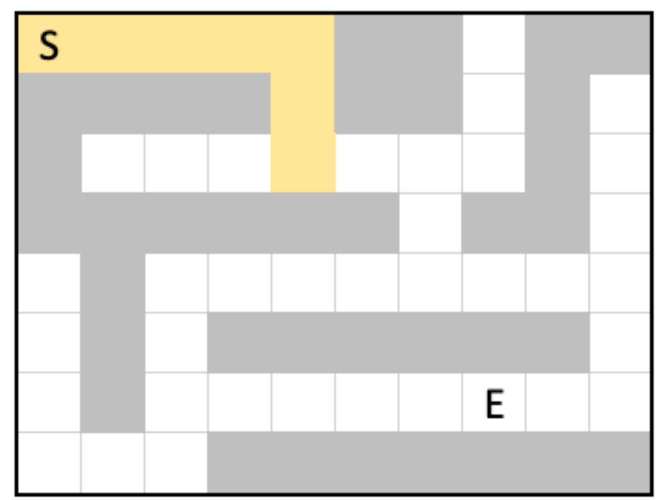


- Can’t go through walls, can only move one position at a time
- The **goal** is to move the **initial** position final position, one step at a time

Abstracting the problem

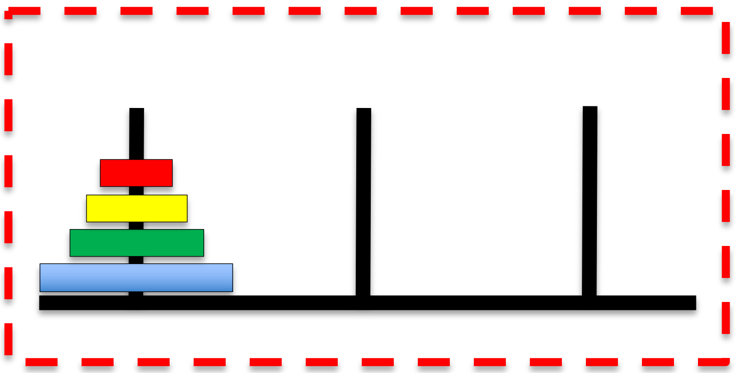
- **Agent:** entity that perceives the environment and acts upon that environment
- **State:** A configuration of the agent in its environment





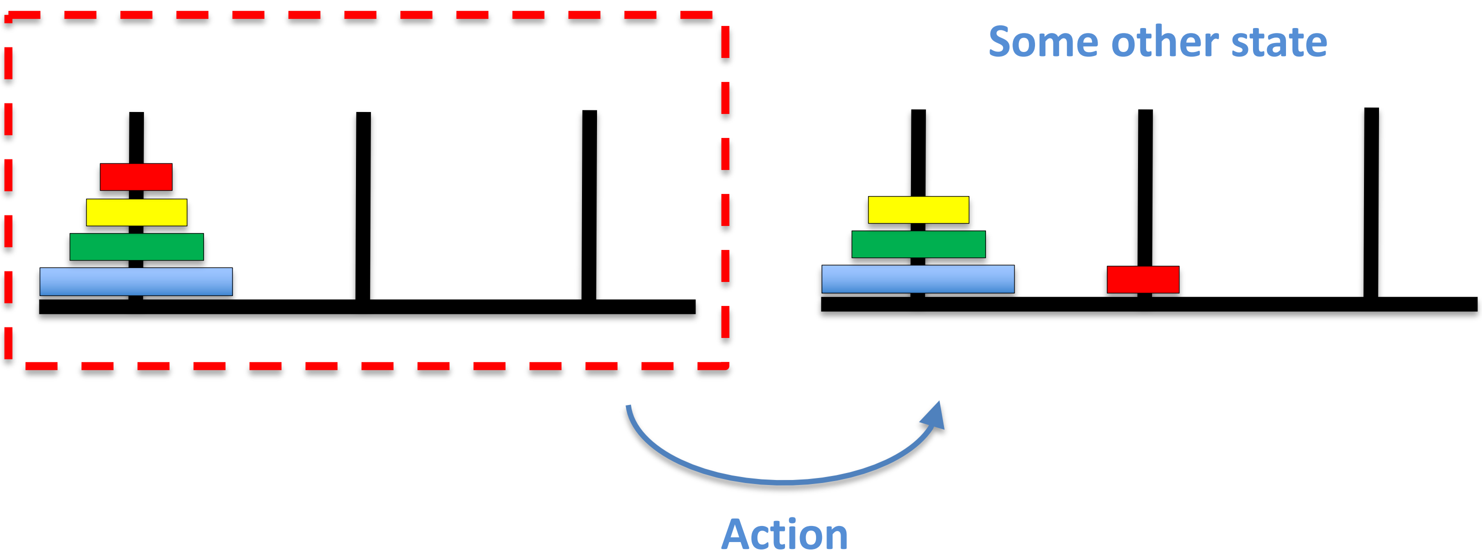
Abstracting the problem

- **Agent:** entity that perceives the environment and acts upon that environment
- **State:** A configuration of the agent in its environment
  - **Initial state**



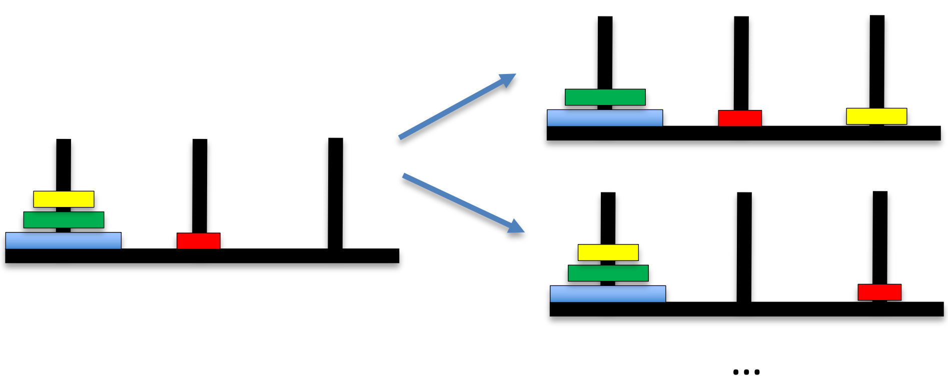
Abstracting the problem

- **Agent:** entity that perceives the <u>environment</u> and <u>acts</u> upon that environment
- **State:** A configuration of the agent in its environment
  - Initial state



Abstracting the problem

- **Agent:** entity that perceives the <u>environment</u> and <u>acts</u> upon that environment
- **State:** A configuration of the agent in its environment
  - **Initial state**
- **Actions: Choices that can be made in a state**
  - ***Action(s)*: Given a state *s*, returns all possible actions from *s***

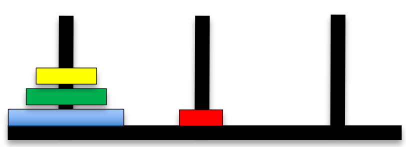
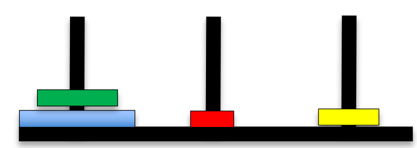


Abstracting the problem

- **Actions:** Choices that can be made in a state
  - **Action(s):** Given a state s, returns all possible actions from s
- **Transition state:** a description of the resulting state after action ***a*** is applied in state ***s***
  - ***Result(s, a) returns the state s' after action a is performed on s***

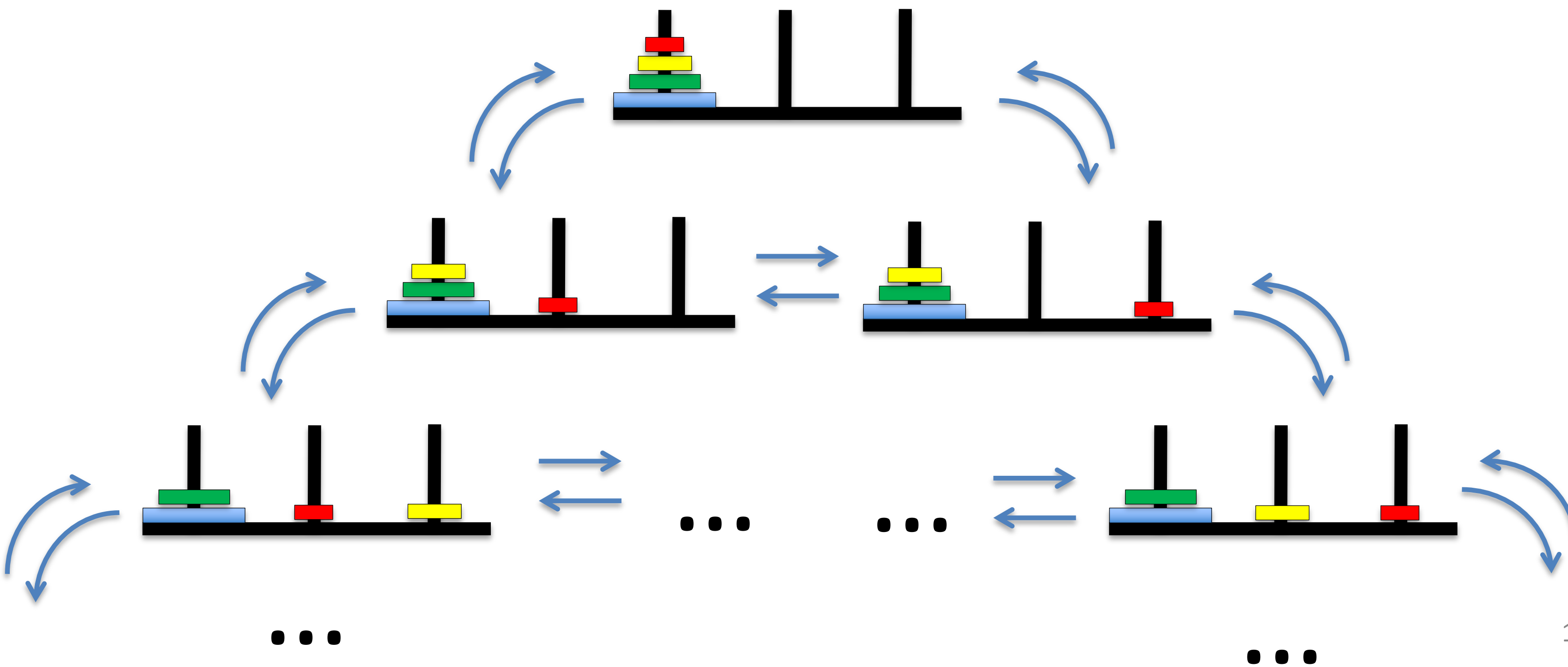
Abstracting the problem

- **Actions:** Choices that can be made in a state
  - **Action(s):** Given a state s, returns all possible actions from s
- **Transition state:** a description of the resulting state after action ***a*** is applied in state ***s***
  - ***Result(s, a) returns the state s' after action a is performed on s***

Result( ***,***  ***→ T3) =*** 

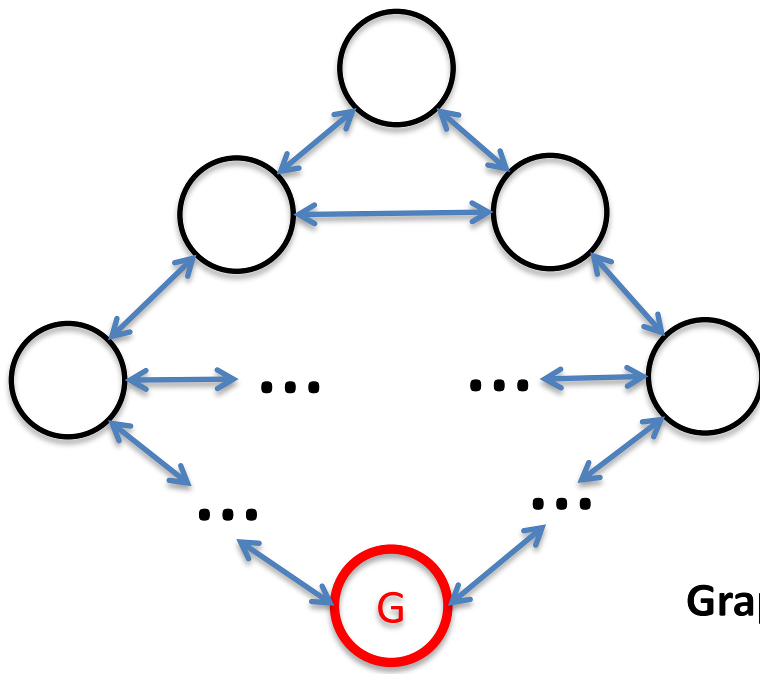
State space

- Agent, State, Actions, Transition state
- **State space:** set of <u>all possible states</u> reachable from the initial state by any sequence of actions.



State space

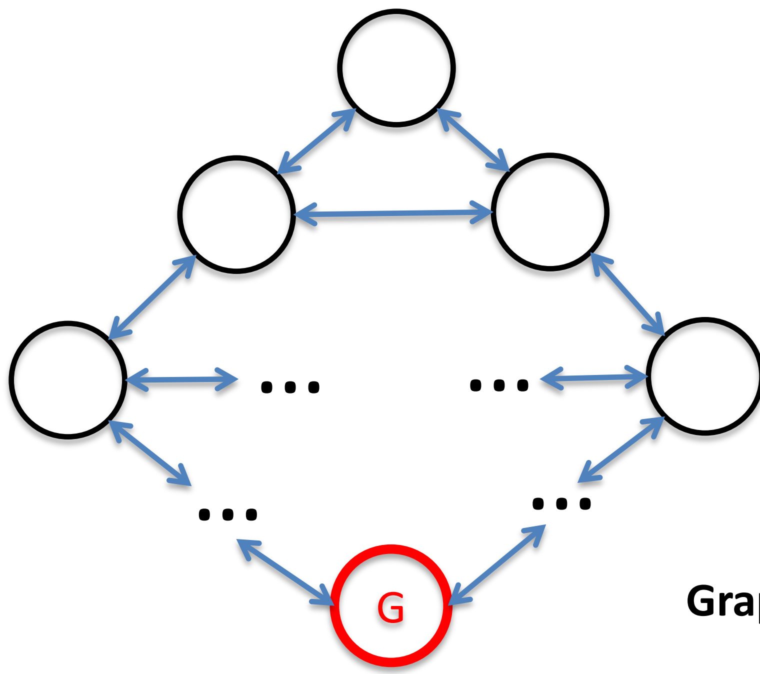
- Agent, State, Actions, Transition state
- **State space:** set of <u>all possible states</u> reachable from the initial state by any sequence of actions.
  - **Goal test:** determines whether a state is a goal



Graph abstraction

State space

- Agent, State, Actions, Transition state
- **State space:** set of <u>all possible states</u> reachable from the initial state by any sequence of actions.
  - **Goal test:** determines whether a state is a goal

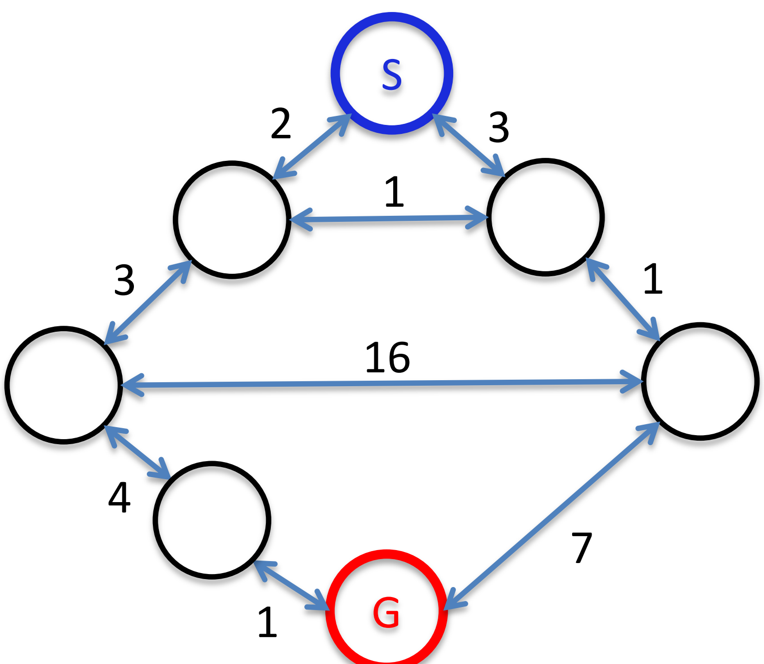


Often we are interested in the "optimal" solution

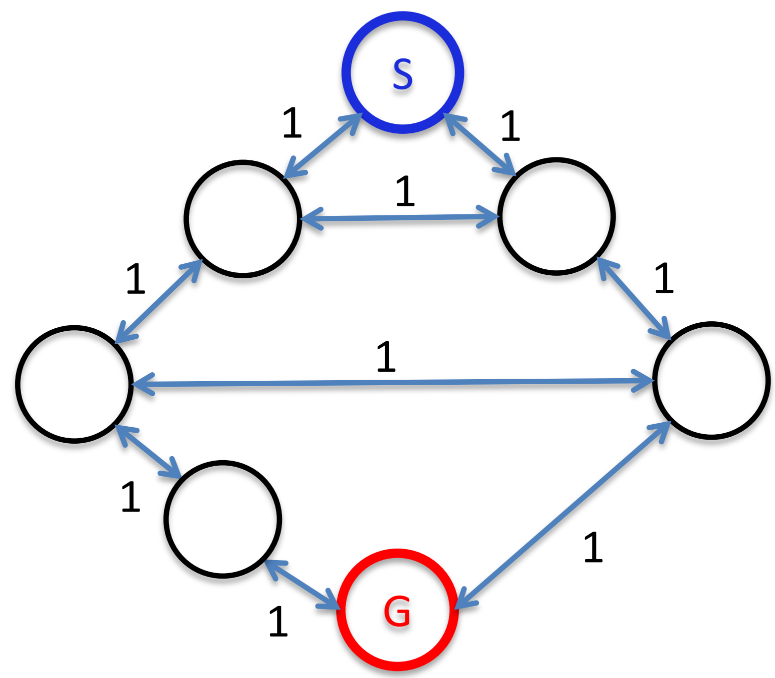
Graph abstraction

Path cost & Optimal solution

- **Path cost:** numerical cost associated with a path
- **Optimal solution:** a solution that has the lowest path cost among all solutions



Weighted



No weights (uniform cost)

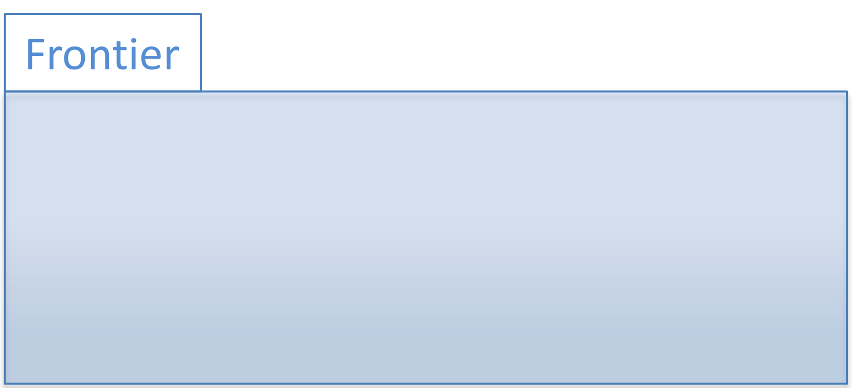
The “Frontier”

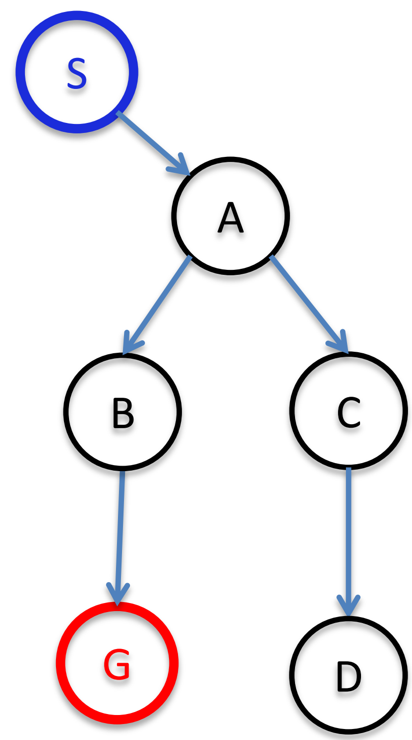
- **Frontier:** All the different options that we can explore next
- Simple algorithm:
  - Starts by adding the initial state to the Frontier
  - **Repeat**
    - If the Frontier is empty, there is no solution
    - Remove a node from the Frontier
    - **Goal test:** Node is goal? Done!
    - **Expand node:** add resulting nodes to the Frontier

Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

Find a path from S to G

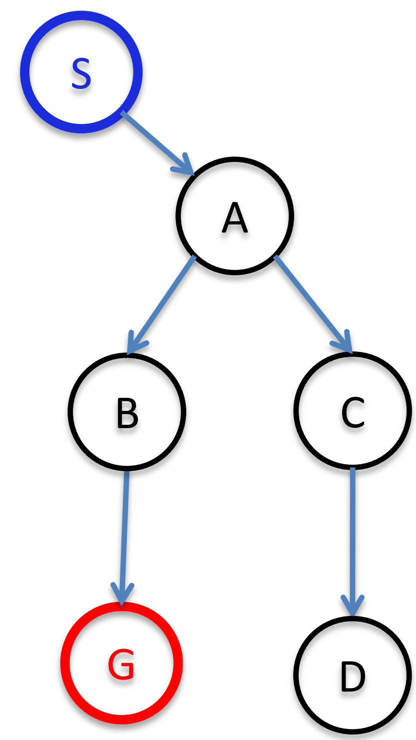


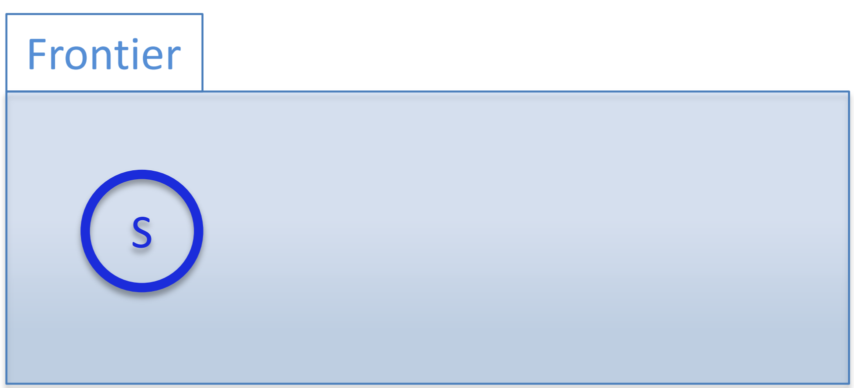


Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

Find a path from S to G

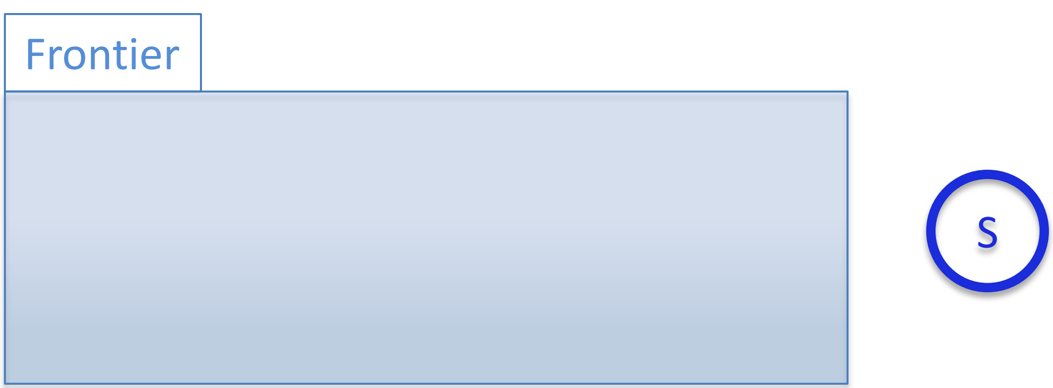


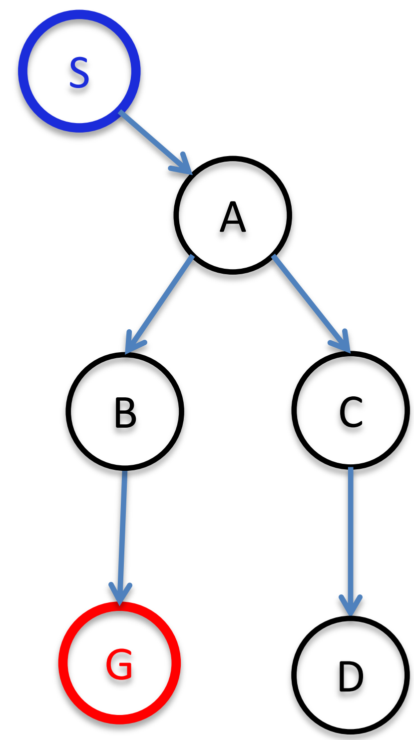


Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

Find a path from S to G

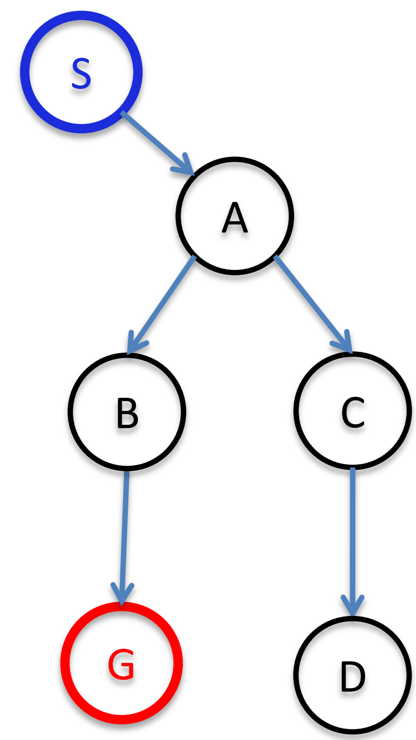


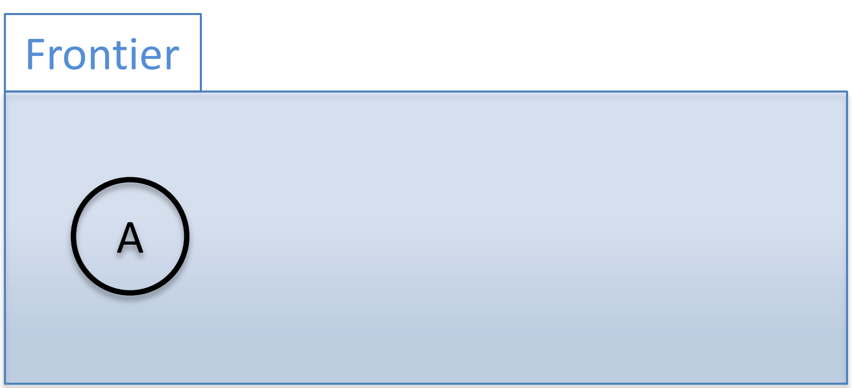


Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

Find a path from S to G

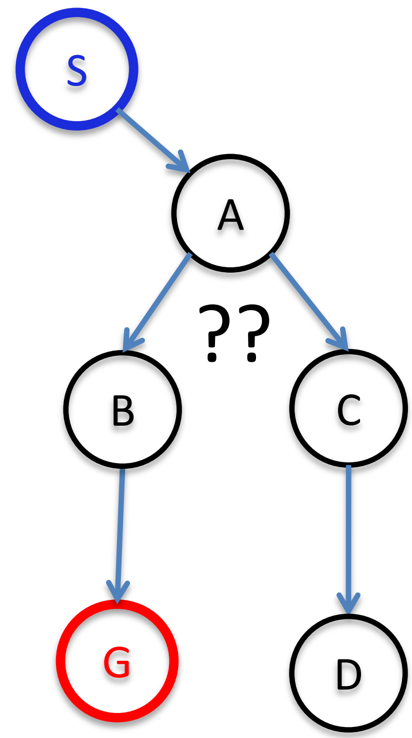


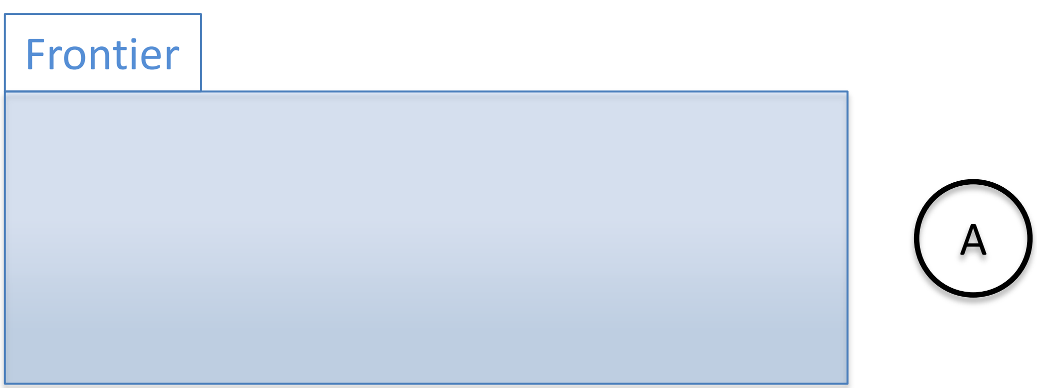


Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

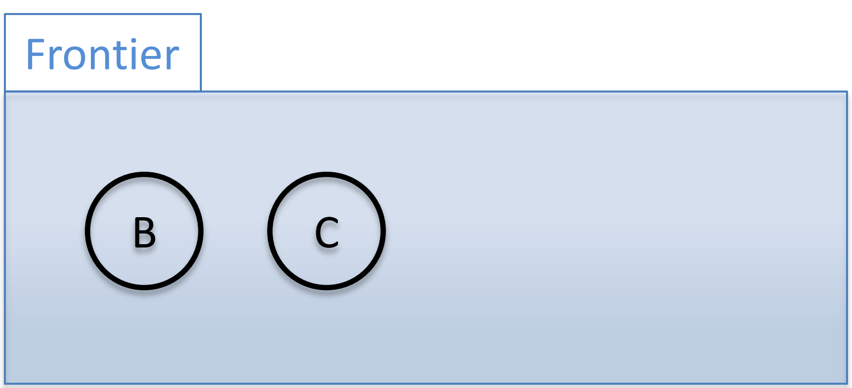
Find a path from S to G

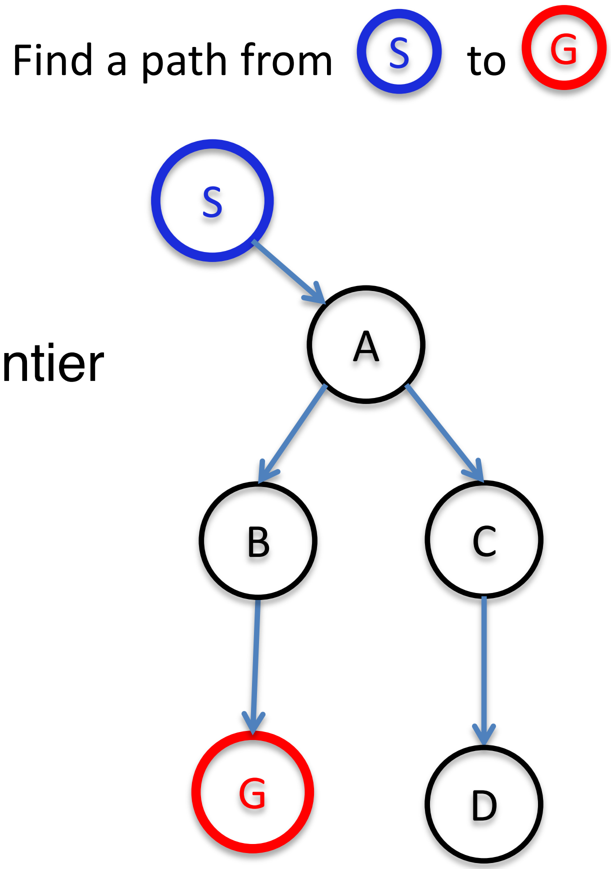




Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

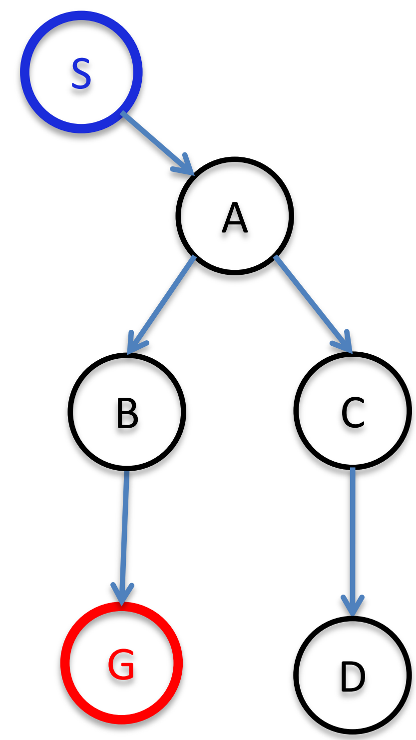


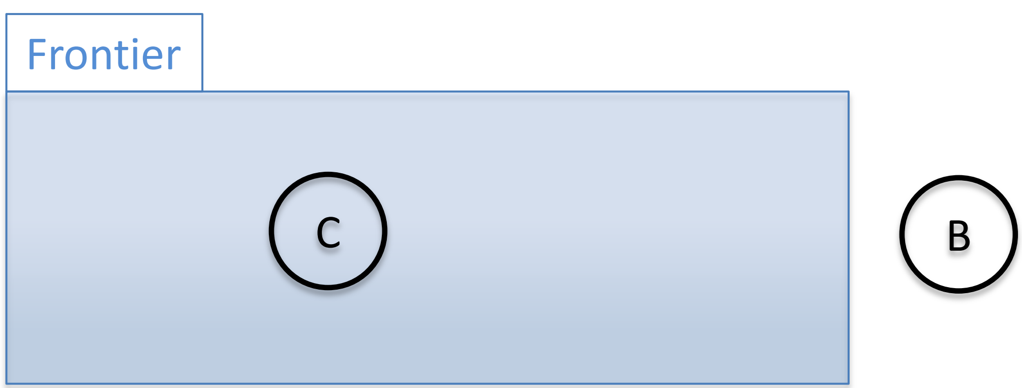


Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

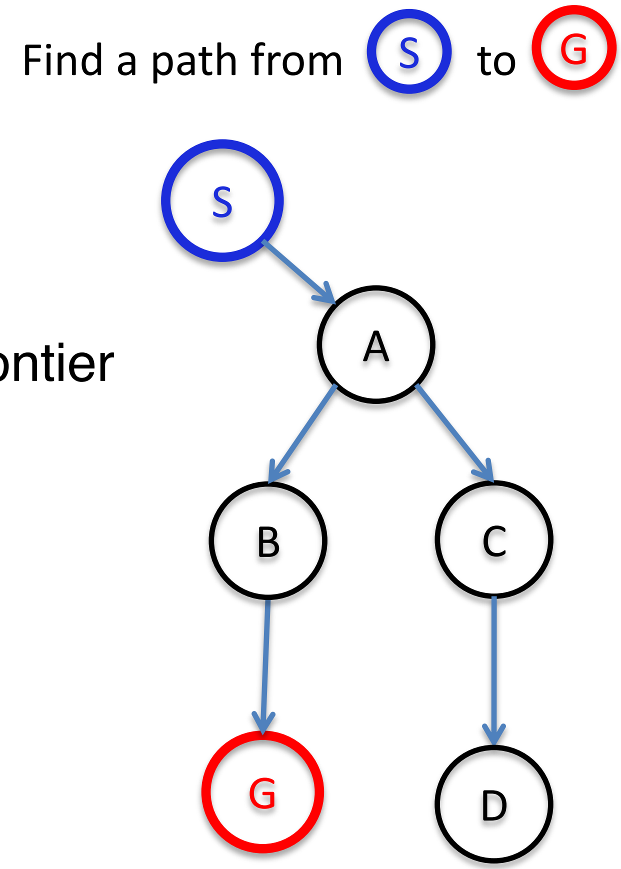
Find a path from S to G

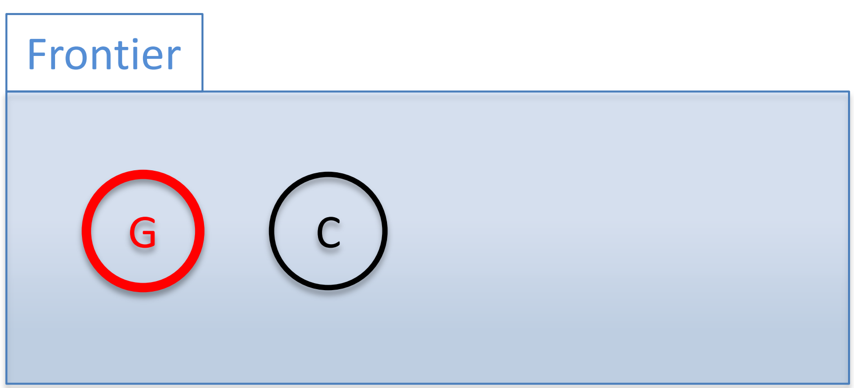




Example

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

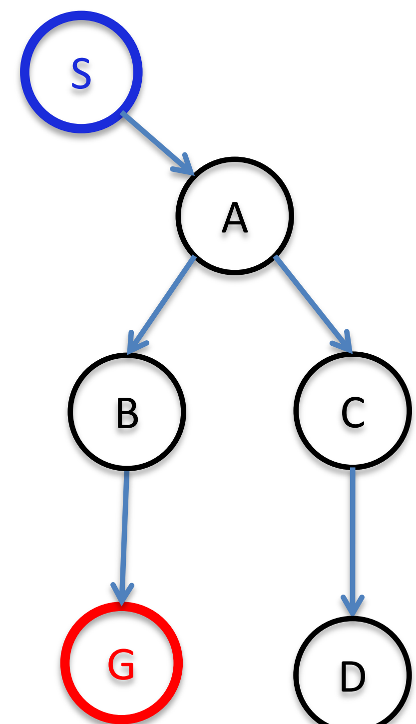




Example

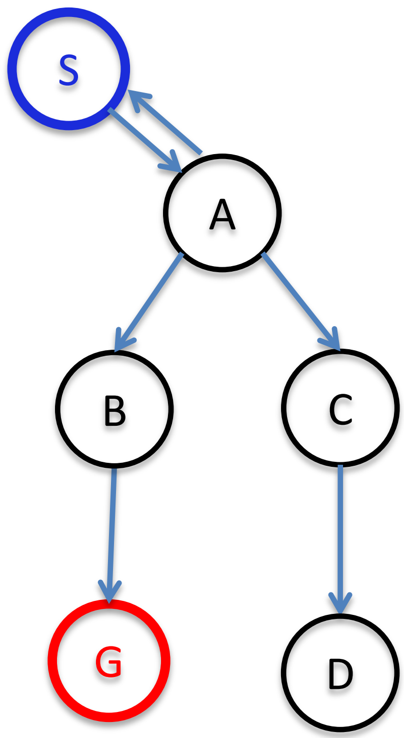
- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - **Expand node:** add resulting nodes to the Frontier

Find a path from S to G



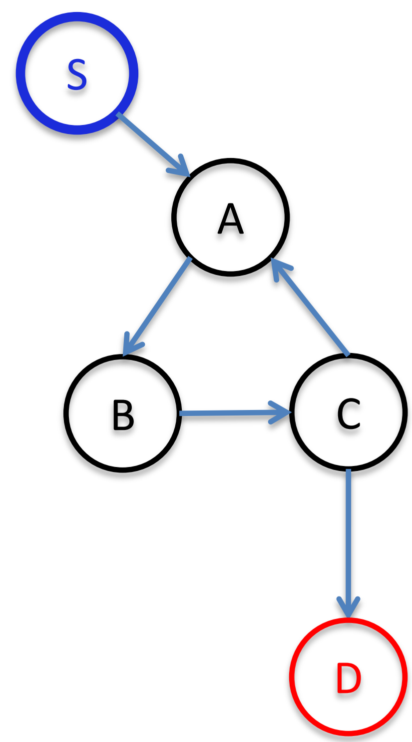
What can go wrong?

- S is removed and A is added
- A is removed and S is added
- S is removed and A is added
- A is removed and S is added
- …
- …
- …
- …



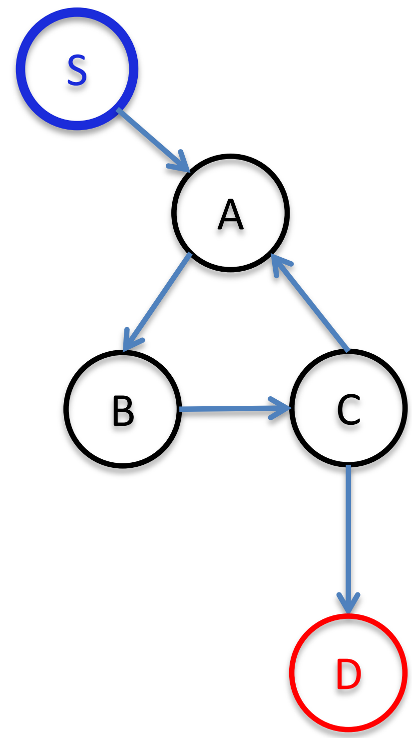
What can go wrong?

What about now?



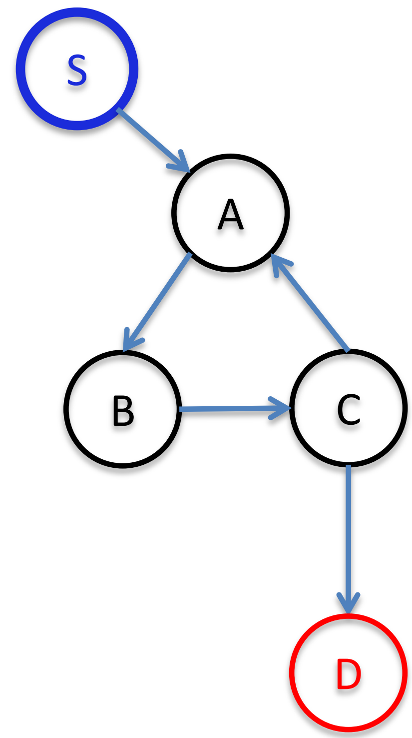
What can go wrong?

- A is removed and B is added
- B is removed and C is added
- C is removed and A and D are added
- A is removed and B is added
- B is removed and C is added
- …
- …
- …



What can go wrong?

- A is removed and B is added
- B is removed and C is added
- C is removed and A and D are added
- A is removed and B is added
- B is removed and C is added
- …
- …



We would like to avoid infinite loops!

Explored set

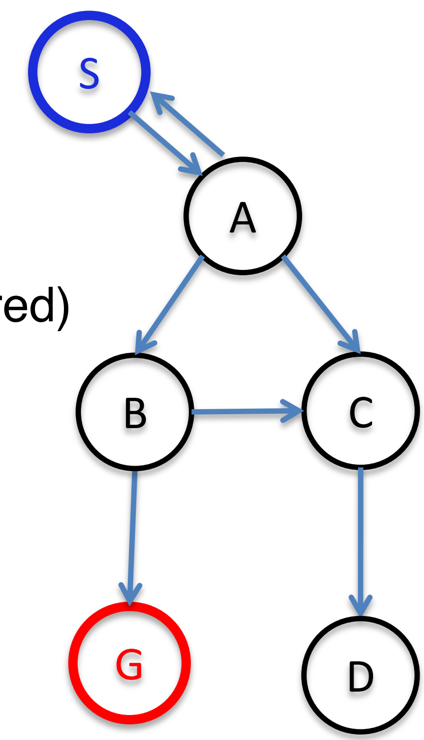
- **Explored set:** maintains a list of already explored nodes

- This allow us to avoid cycles in our basic algorithm

Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

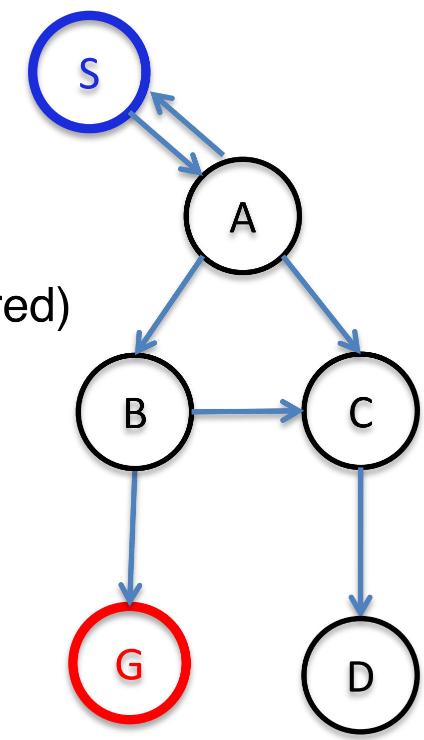
Find a path from S to G

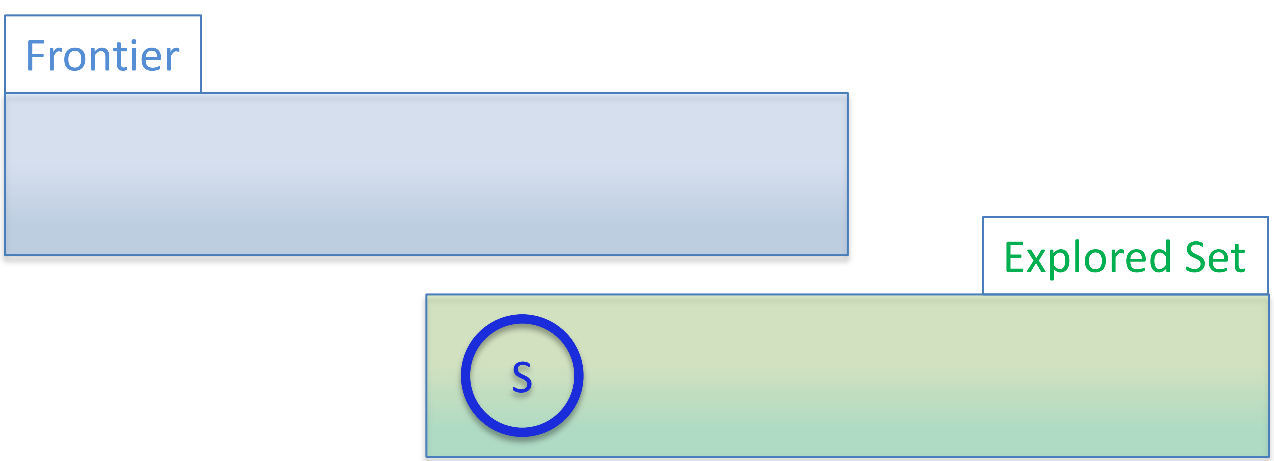


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

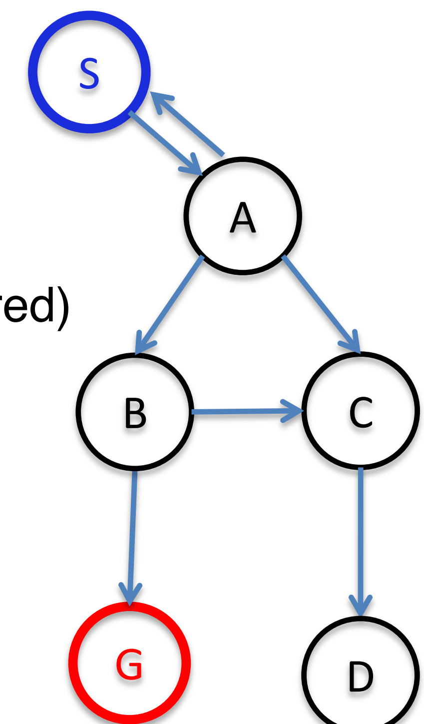


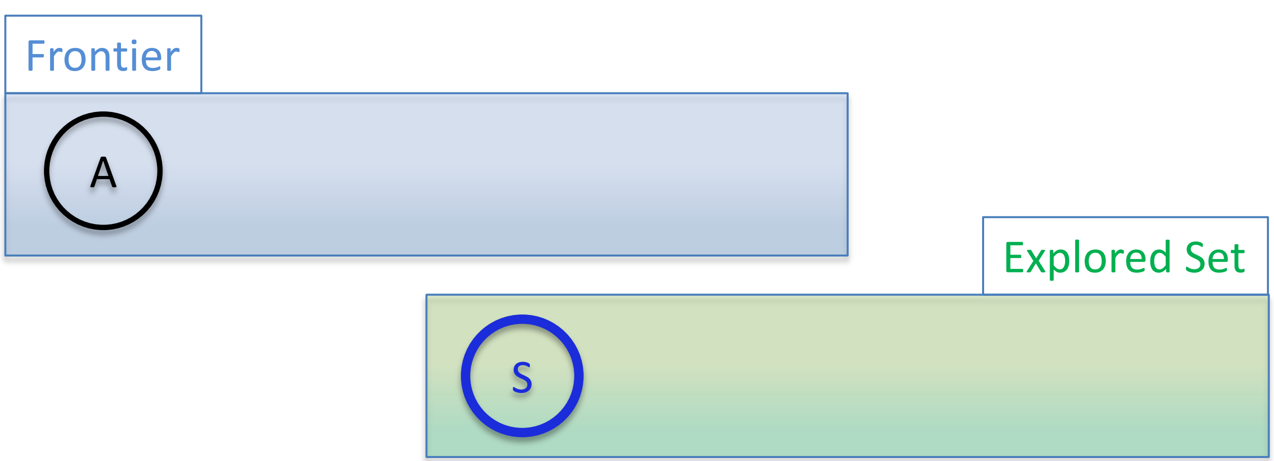


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

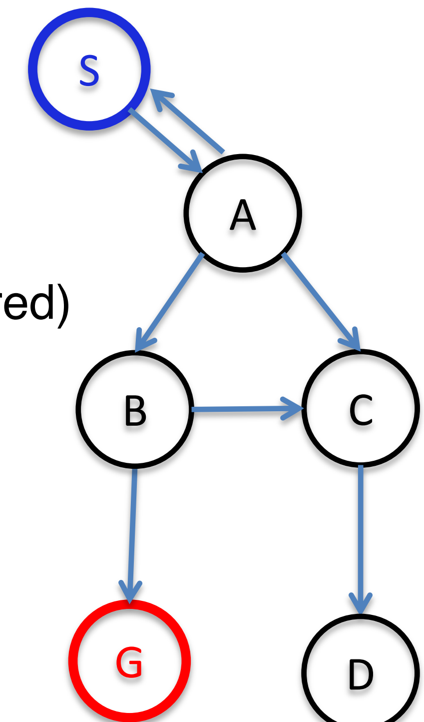


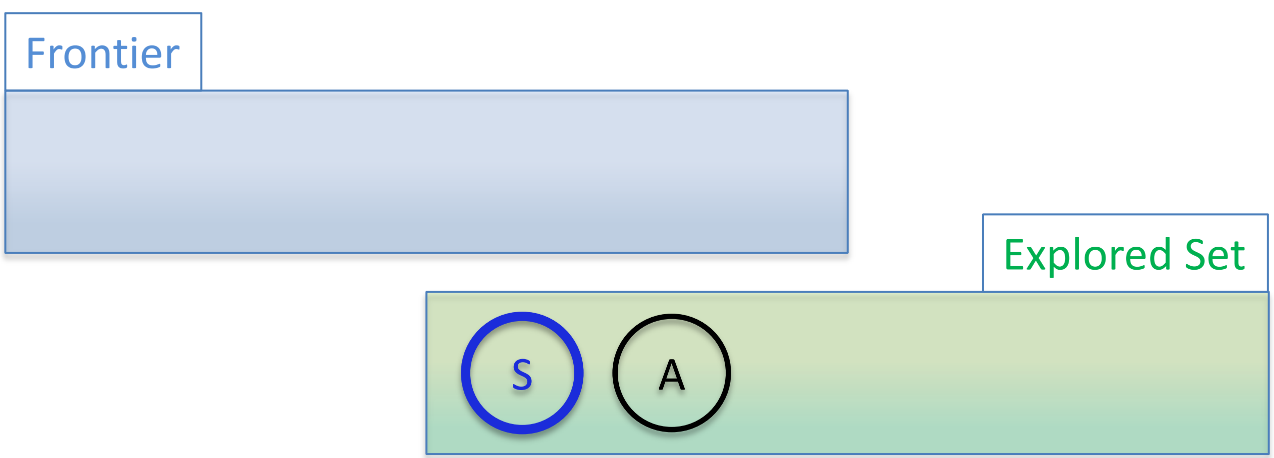


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

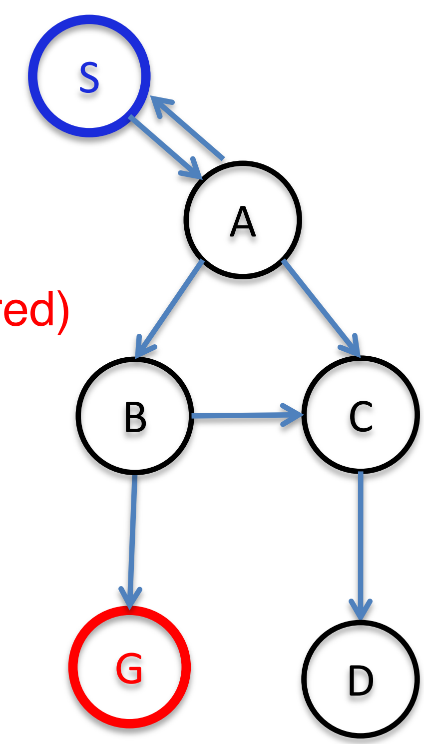


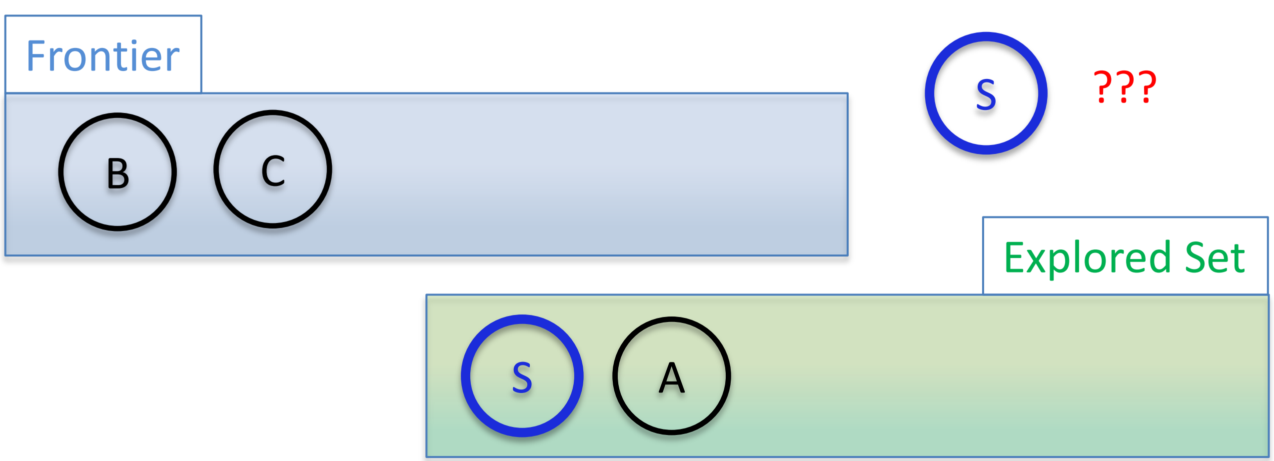


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - Add removed to the **Explored set**
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

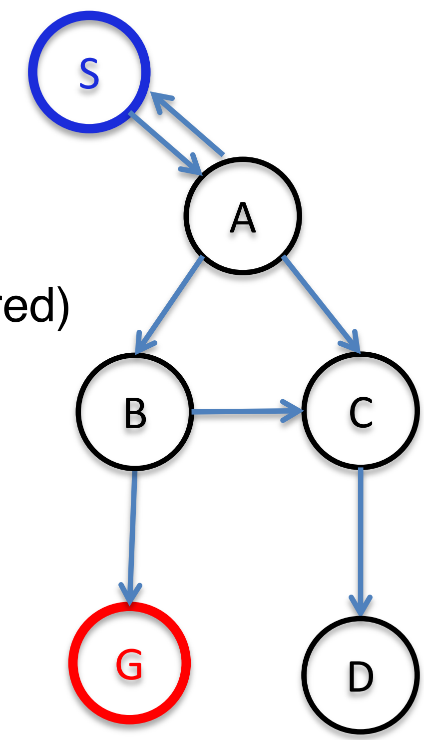


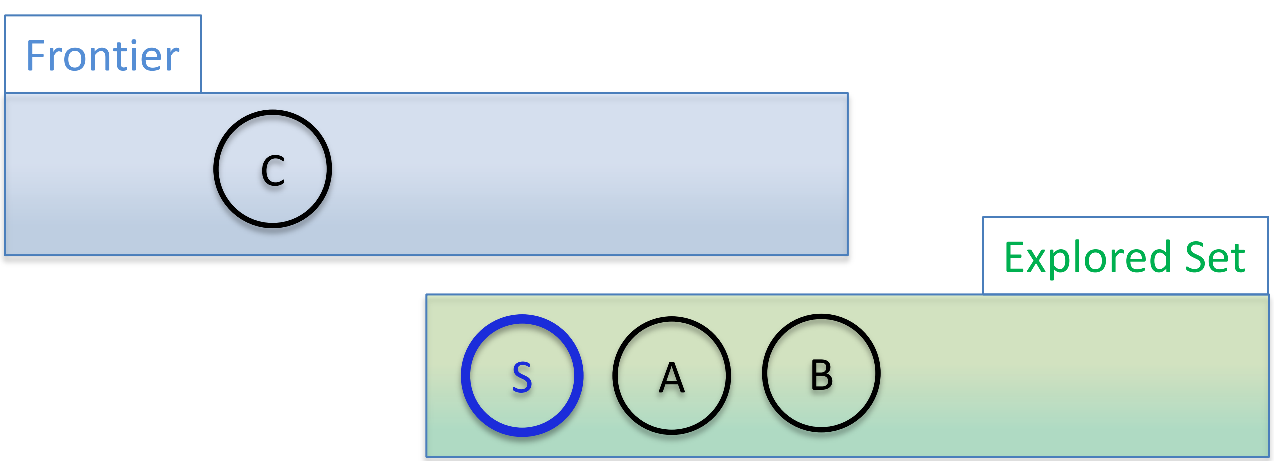


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

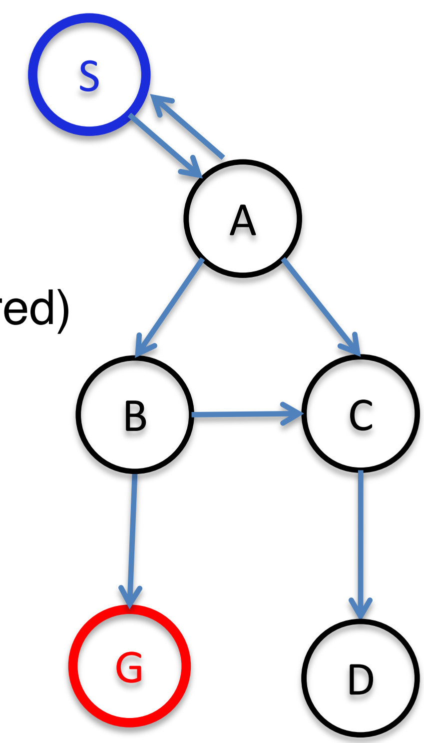


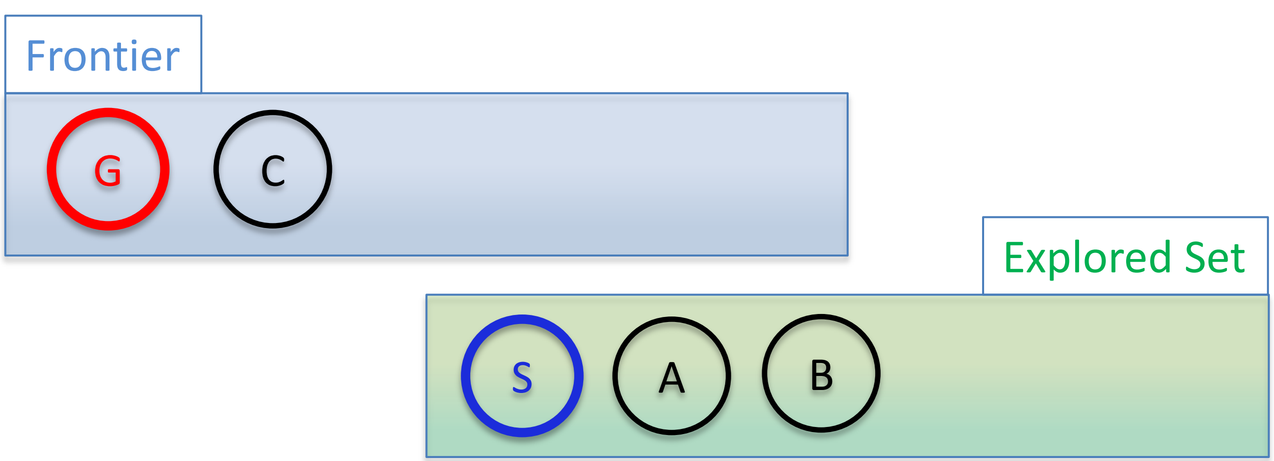


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G

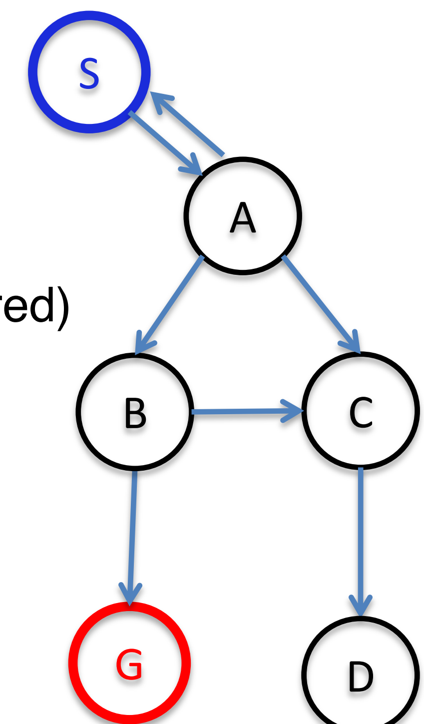


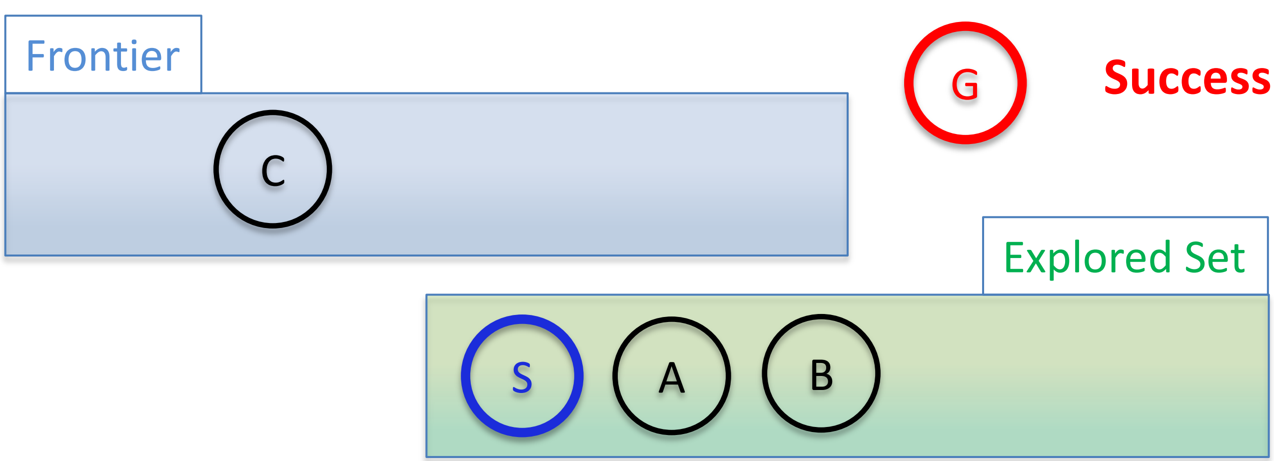


Example with Explored Set

- Starts by adding the initial state to the Frontier
- **Repeat**
  - If the Frontier is empty, there is no solution
  - Remove a node from the Frontier
  - **Goal test:** Node is goal? Done!
  - <u>Add removed to the **Explored set**</u>
  - **Expand node:** add nodes to the Frontier (if not in Explored)

Find a path from S to G





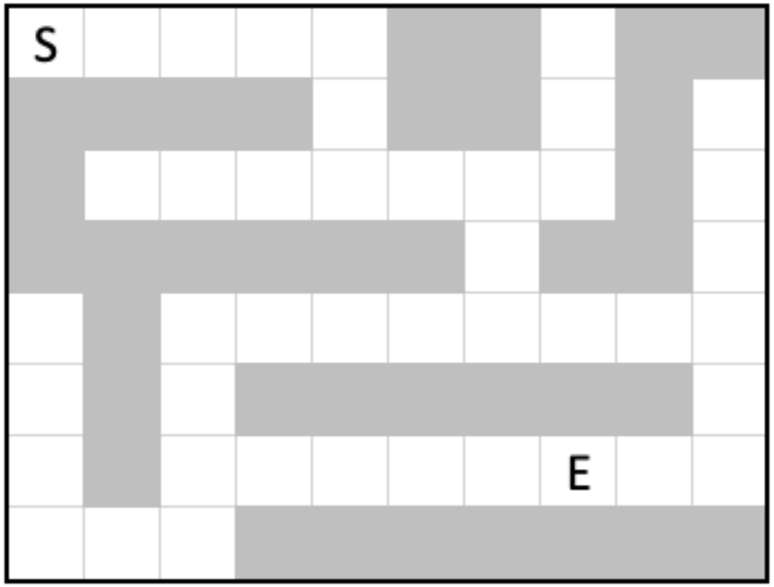
Success!!

Search algorithms

- How we explore the frontier matters!
- There are classical approaches and their variations:
  - Depth-first Search (DFS): Stack (Last in, First out)
  - Breadth-first Search (BFS): Queue (First in, First out)
- DFS variants
  - Depth Limited Search (DLS)
  - Iterative Deepening Search (IDS)
- BFS variants
  - Uniform Cost Search
  - Bidirectional Search

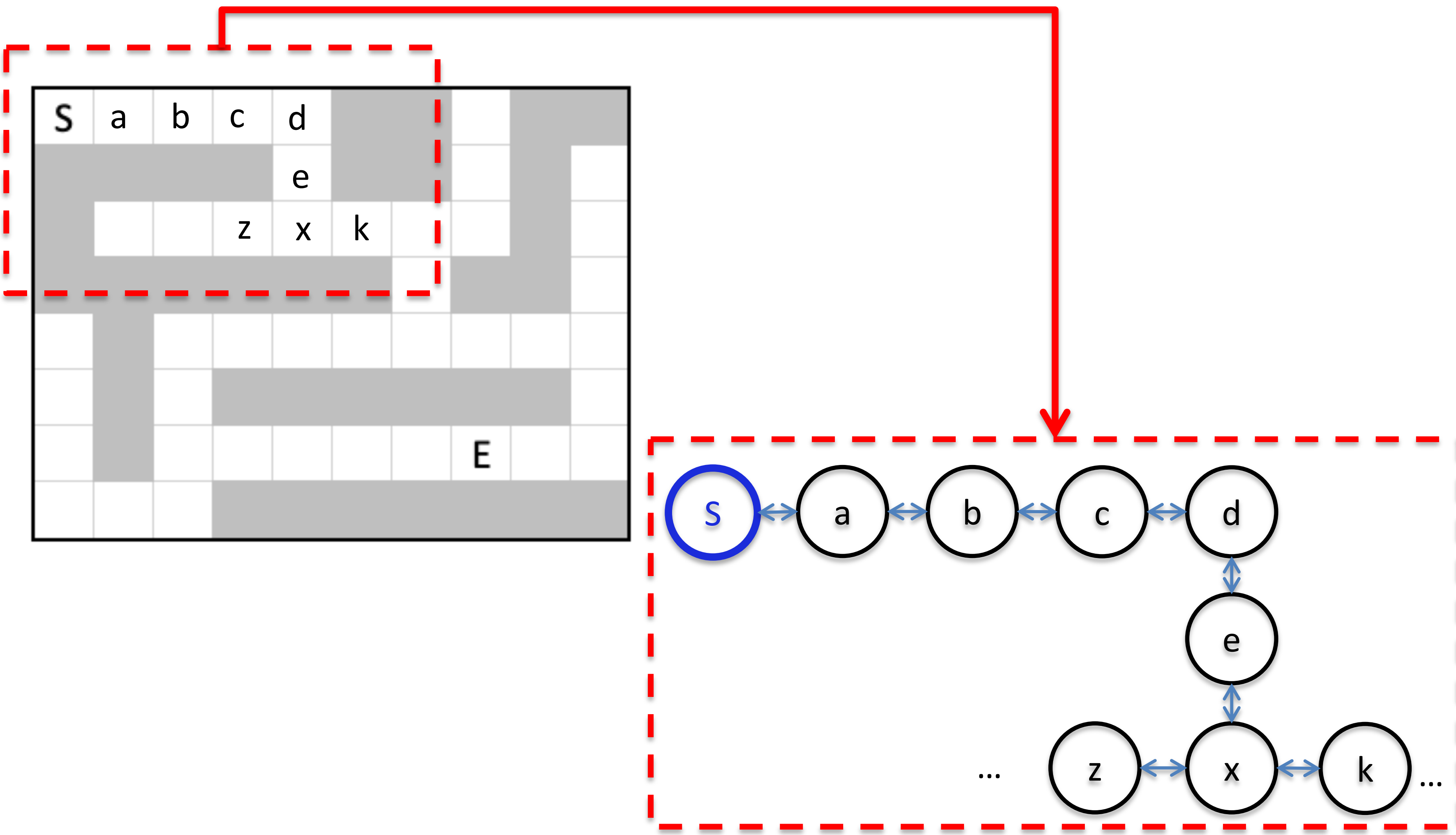
DFS x BFS - Back to the Maze

- Find a path from the Start position (S) to the End position (E)

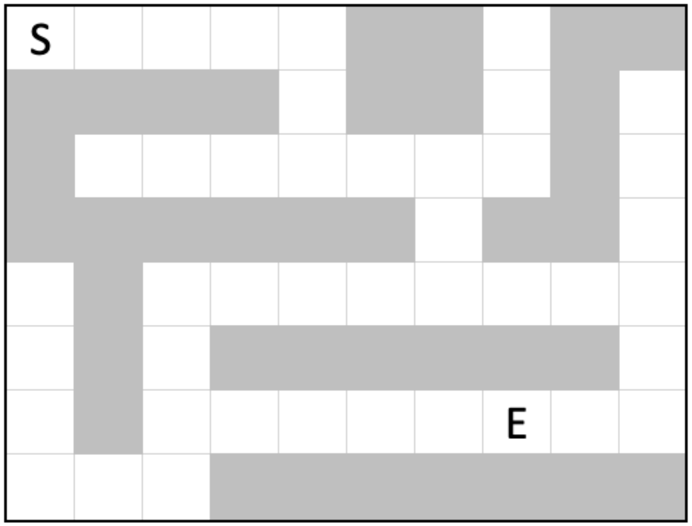


- Can’t go through walls, can only move one position at a time
- The **goal** is to move the **initial** position final position

DFS x BFS – Graph abstraction

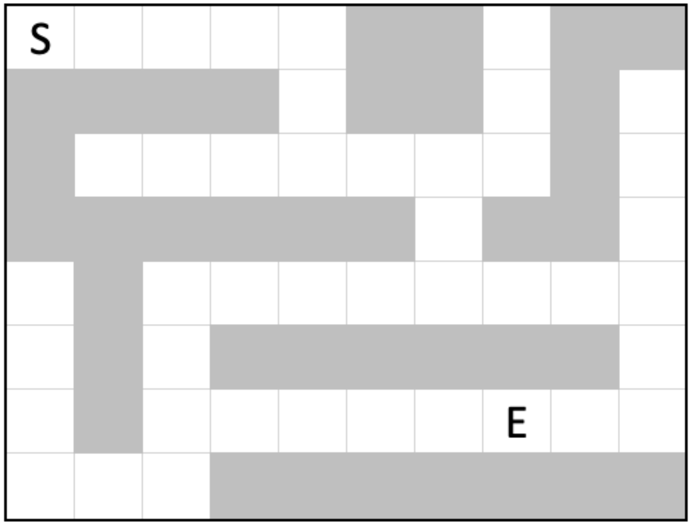


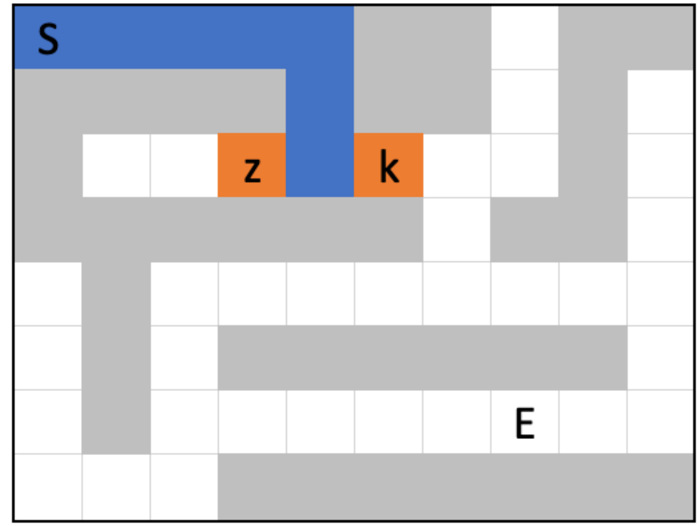
DFS example



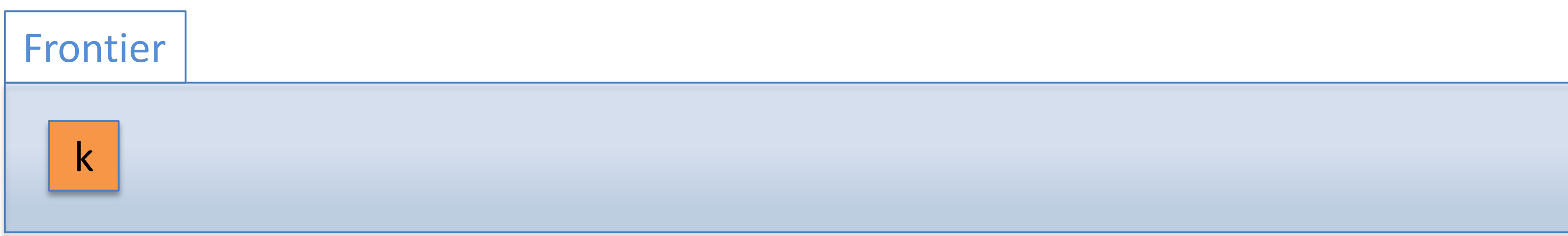
* Explored Set not shown for simplicity; assuming nodes are added as follows: right, down, left, top.

DFS example



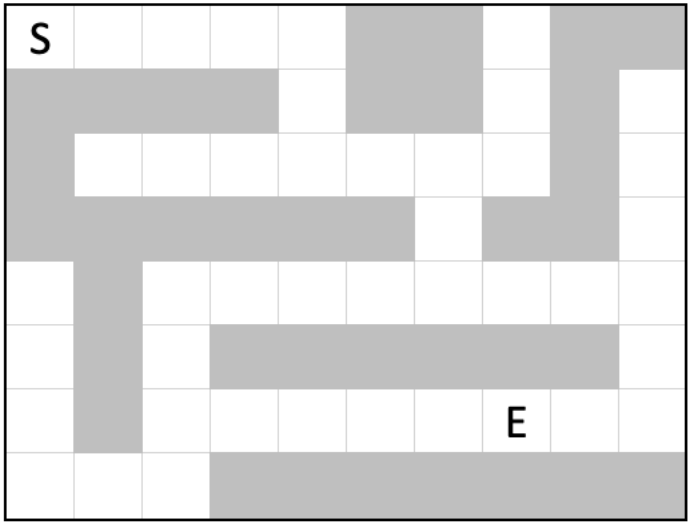


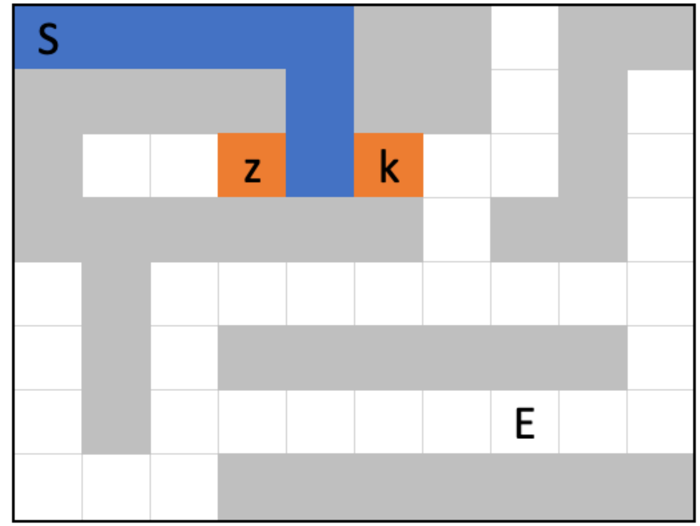
Adding **k** to the **Frontier**



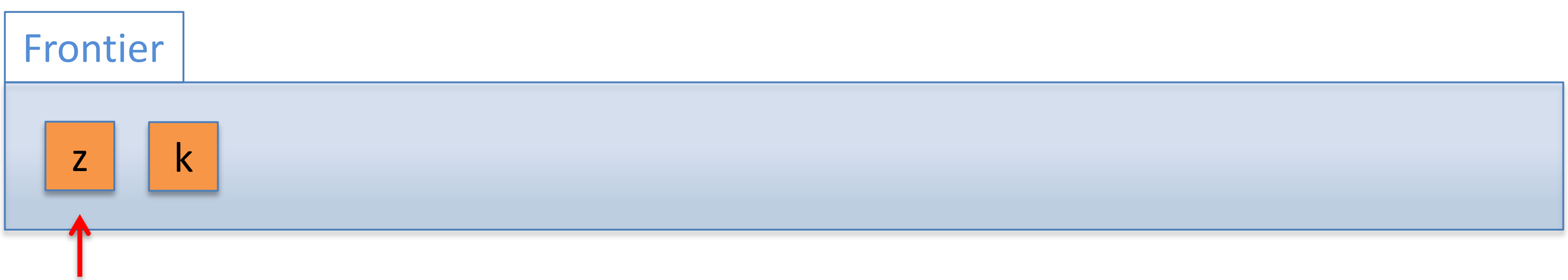
* Skipping the first trivial steps (i.e. only one path); assuming nodes are added as follows: right, down, left, top.

DFS example



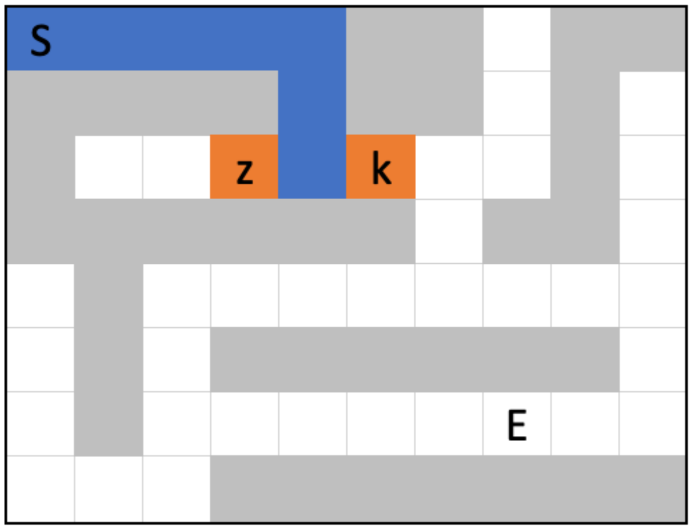


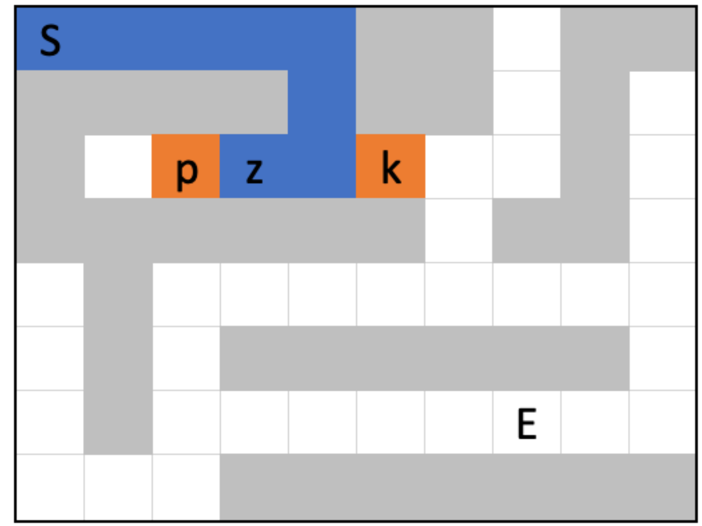
Adding **z** to the **Frontier**



Top of the **stack**

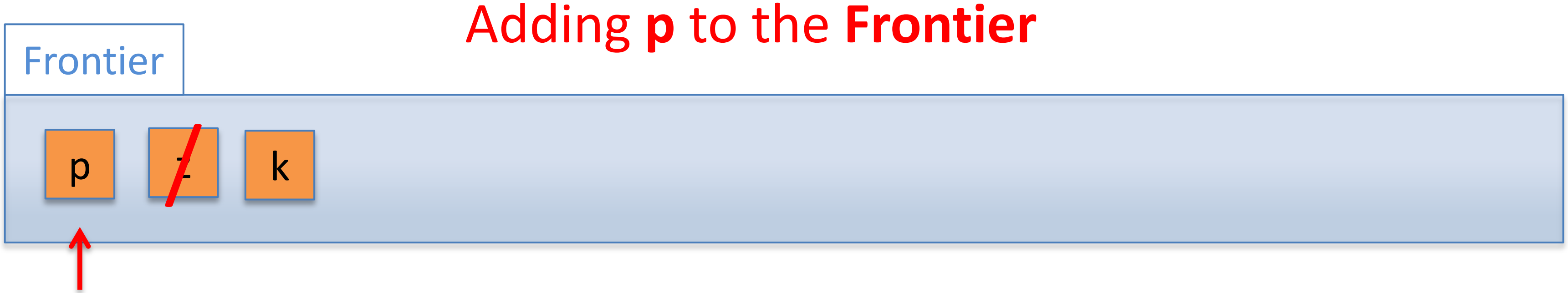
DFS example





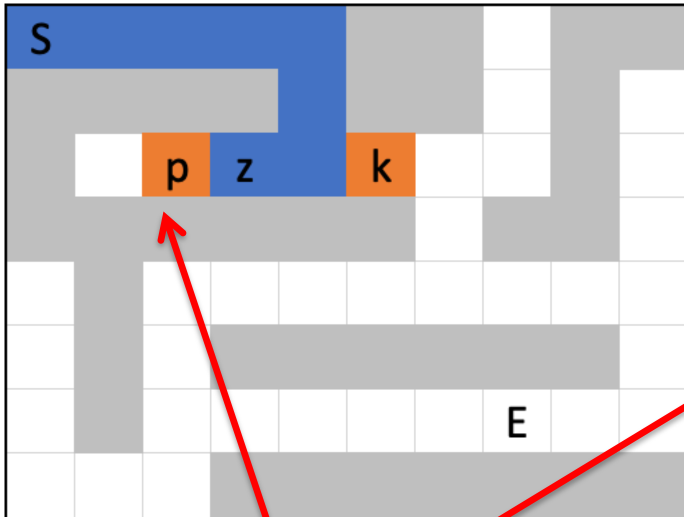
Removing **z** to the **Frontier**

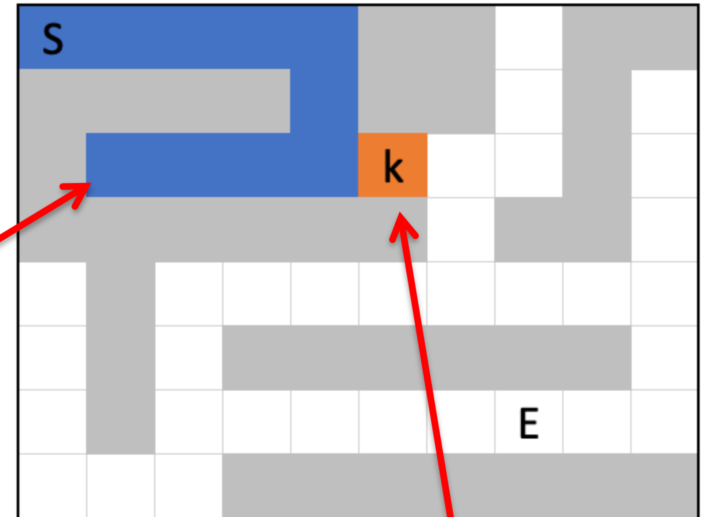
Adding **p** to the **Frontier**



Top of the **stack**

DFS example



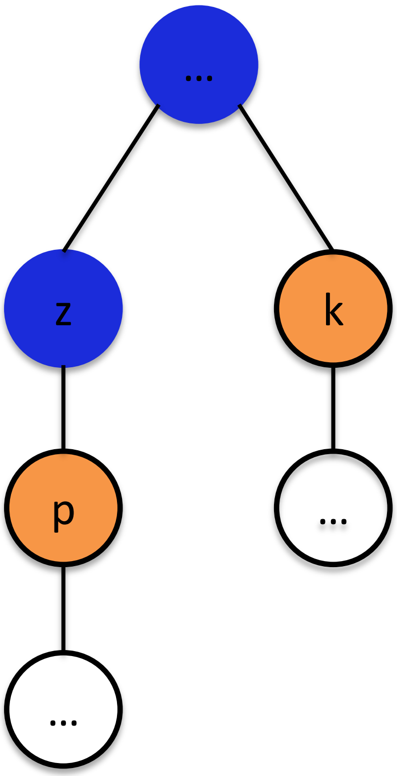


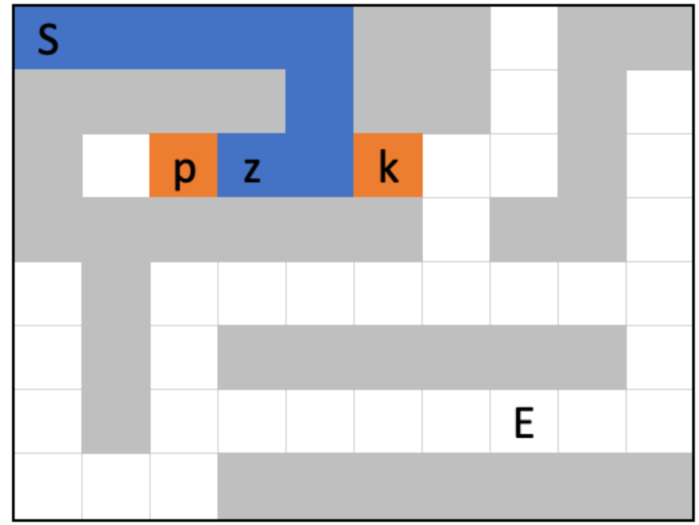
We explore everything here

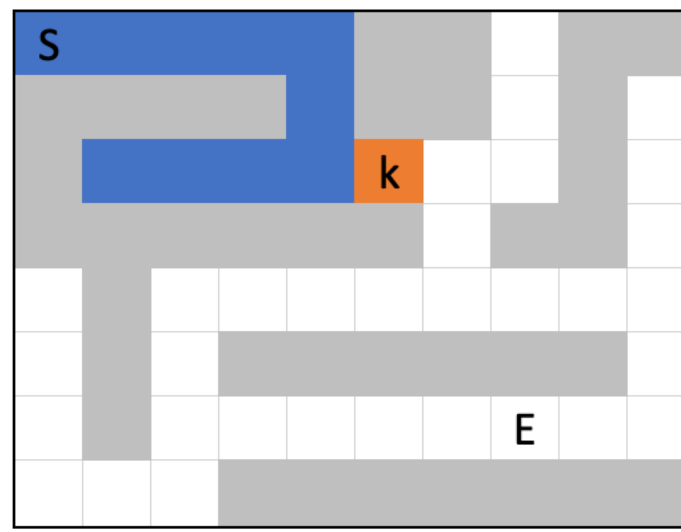
before exploring from here

* Skipping the first trivial steps (i.e. only one path)

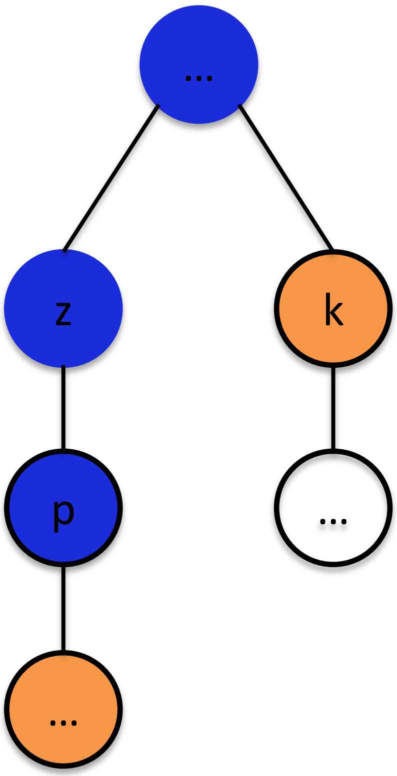
DFS example (graph view)

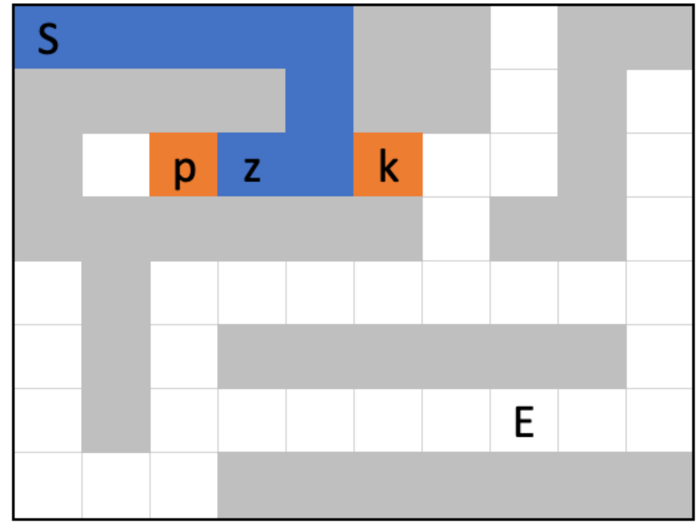


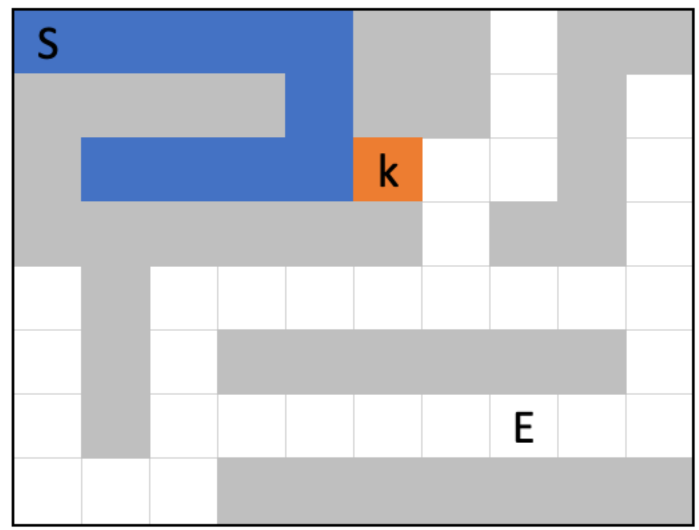




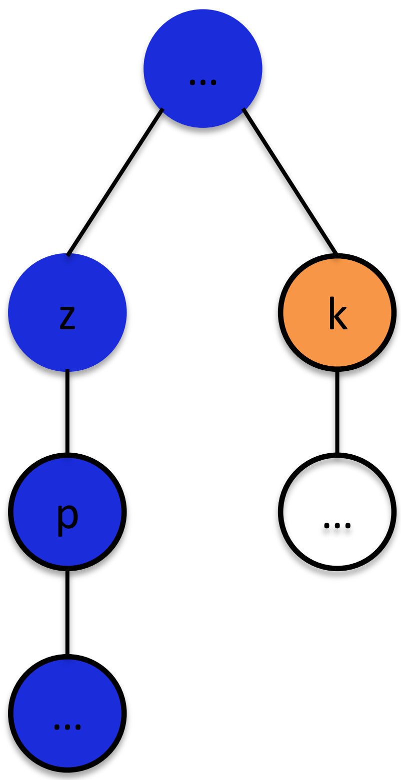
DFS example (graph view)

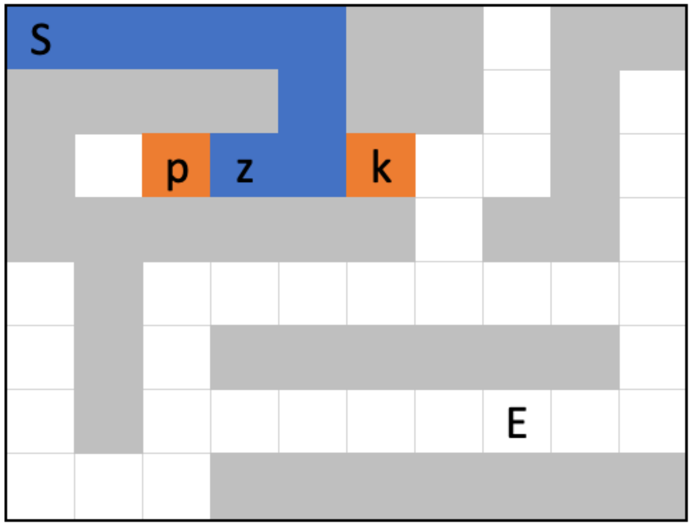


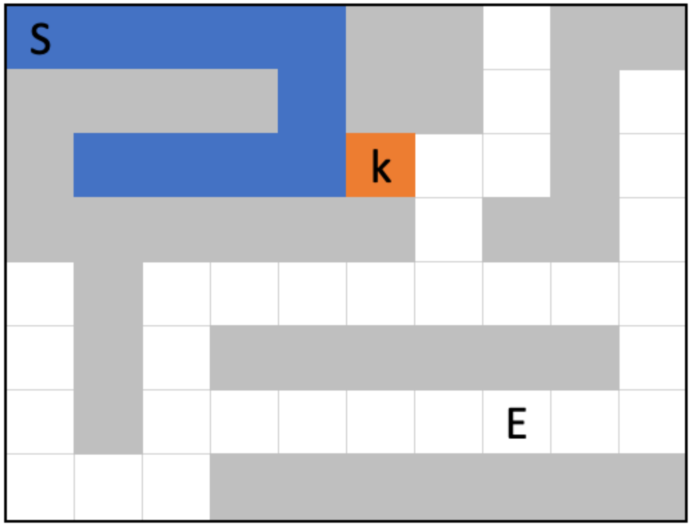




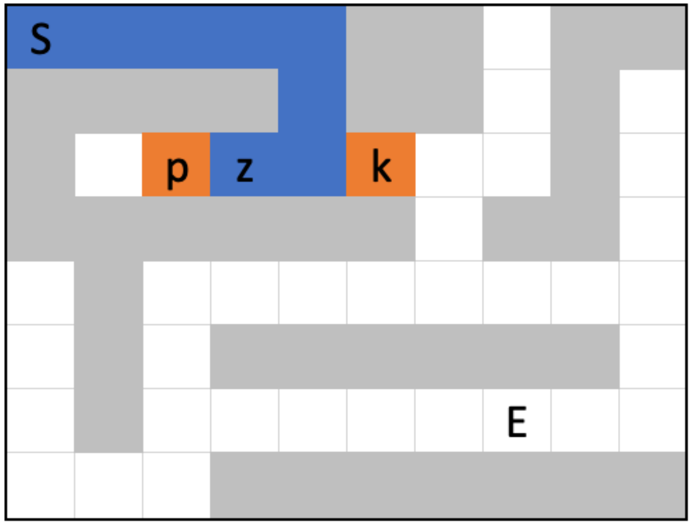
DFS example (graph view)



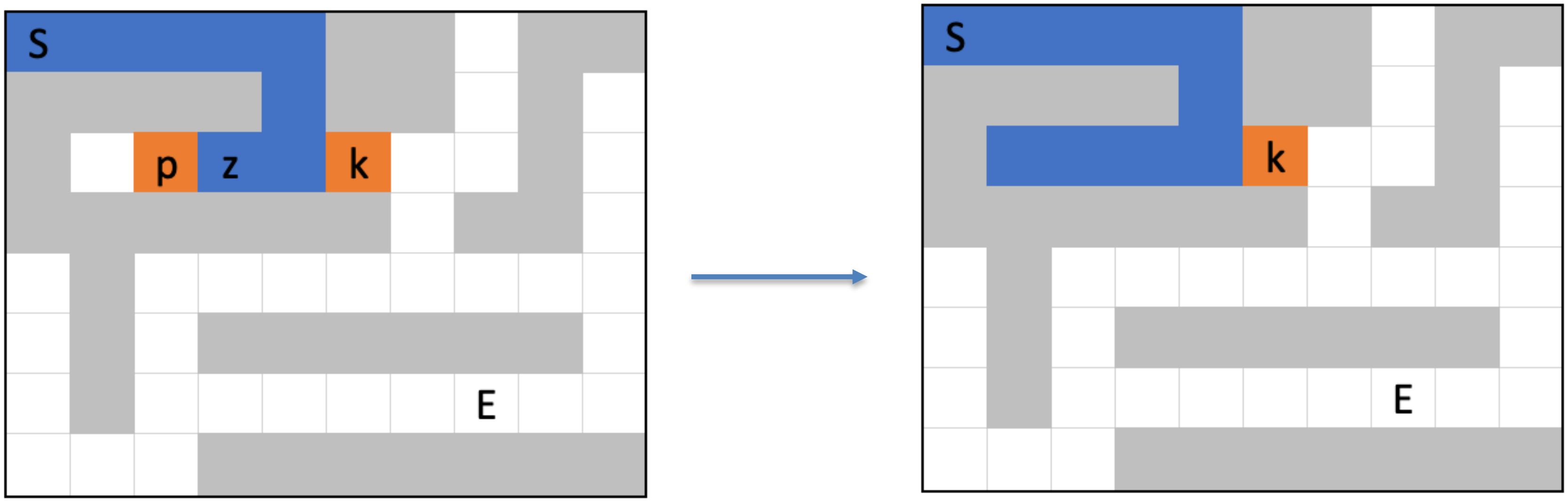




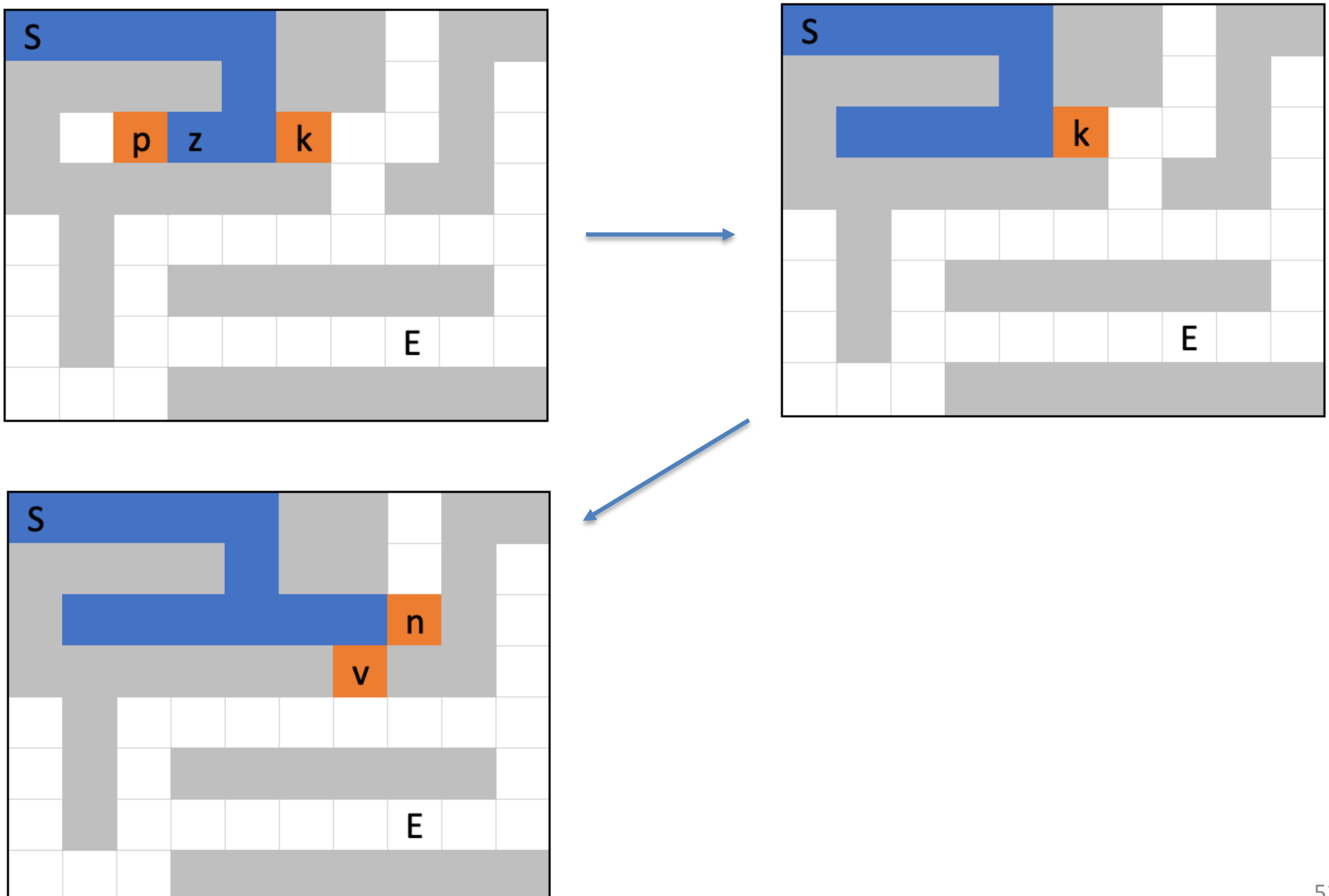
DFS example



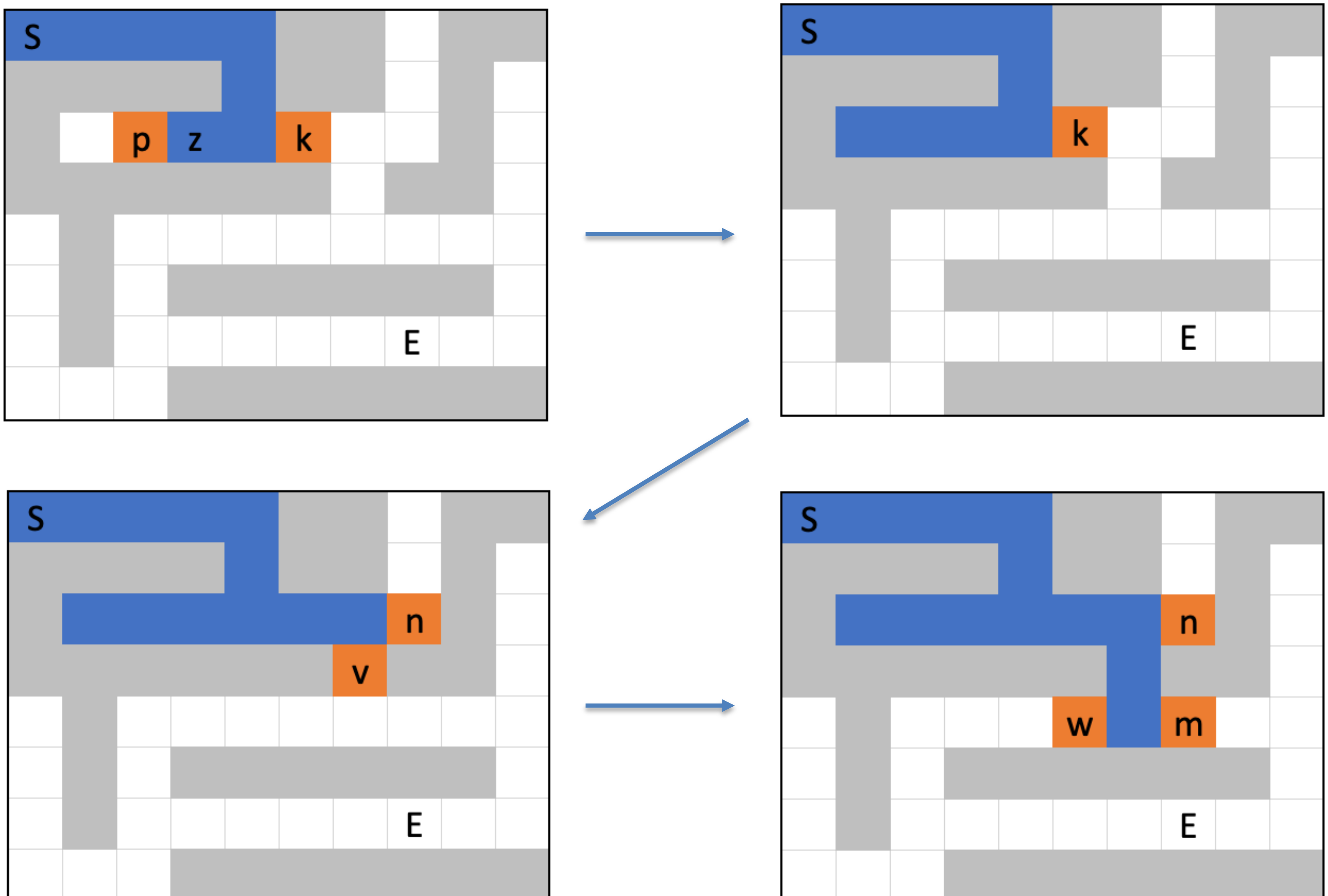
DFS example



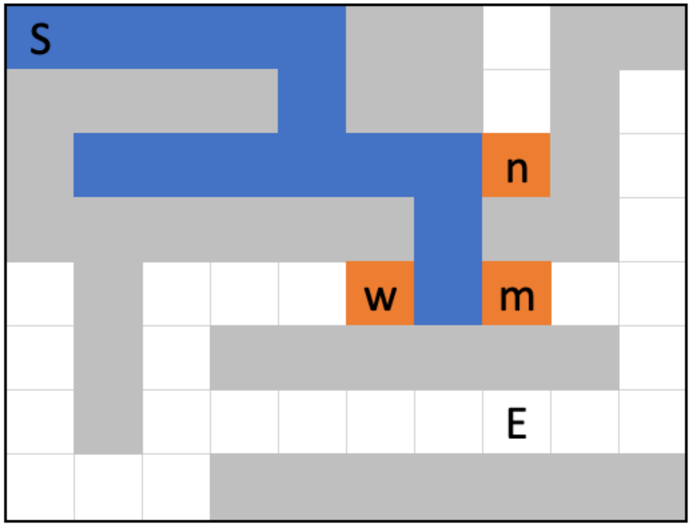
DFS example



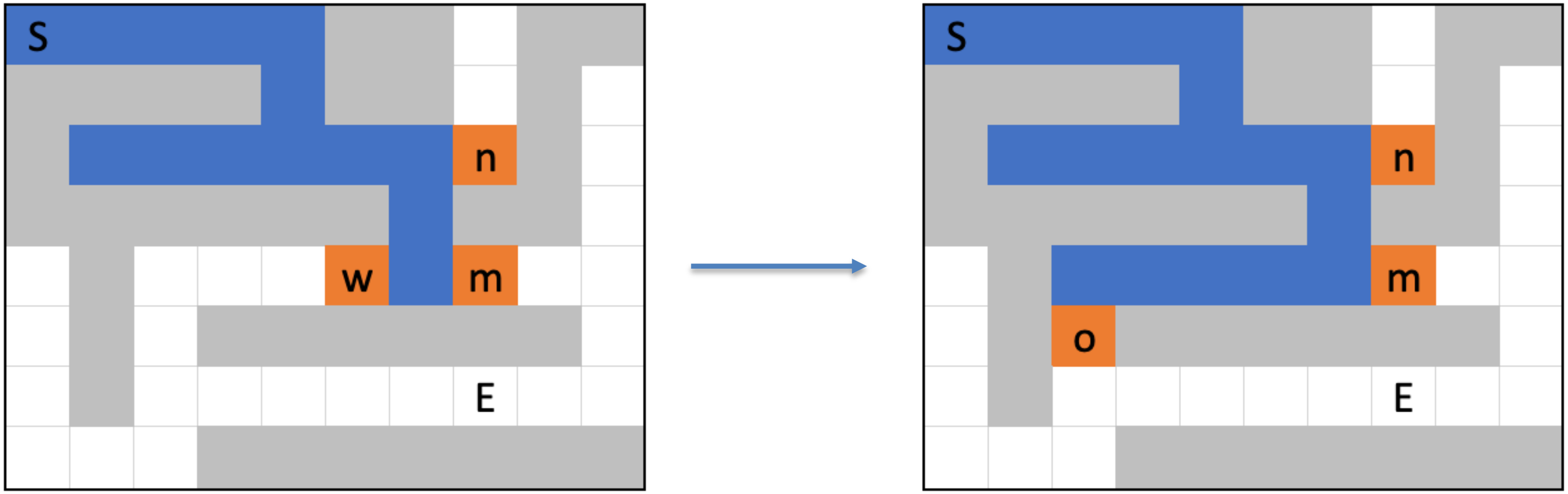
DFS example



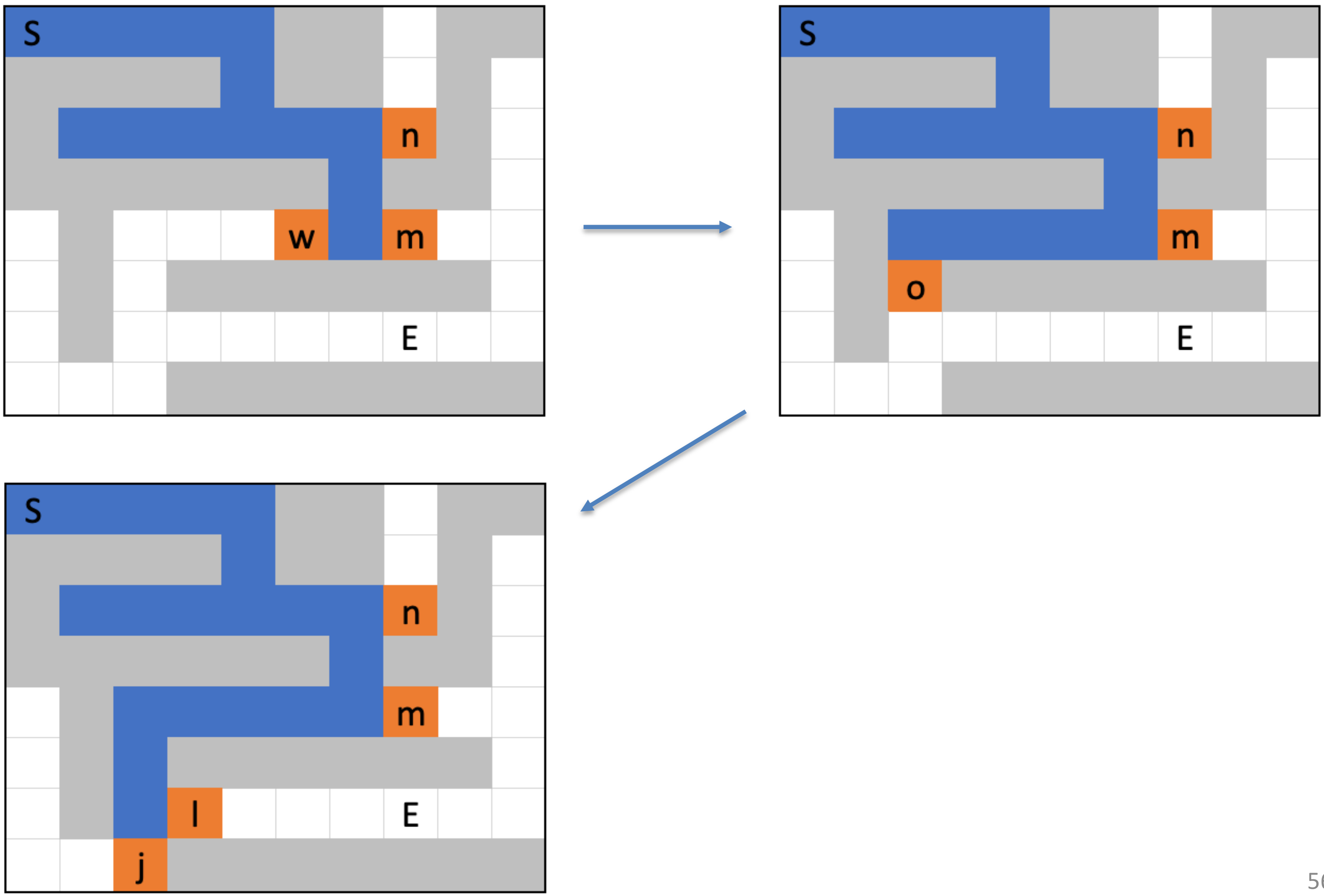
DFS example



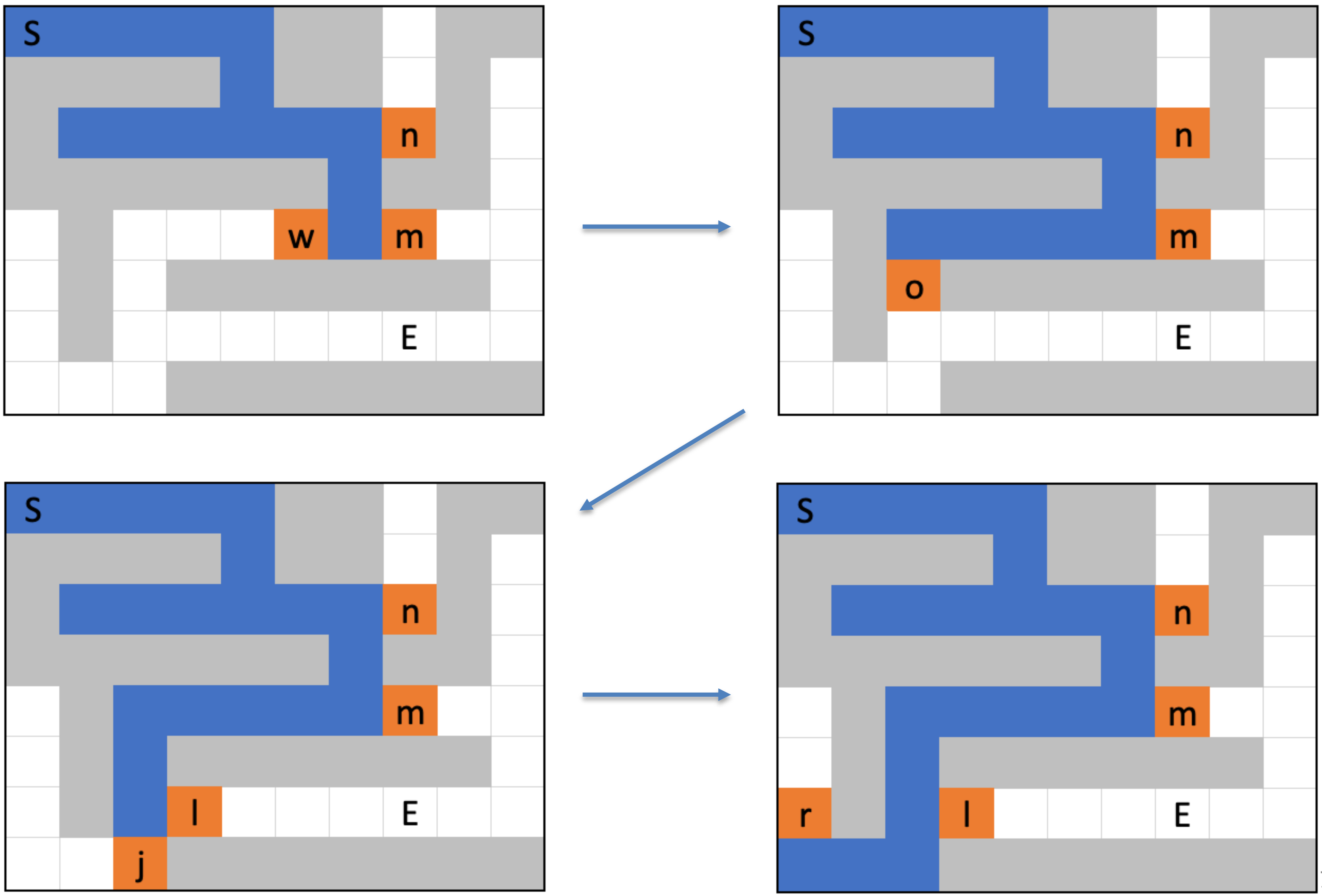
DFS example



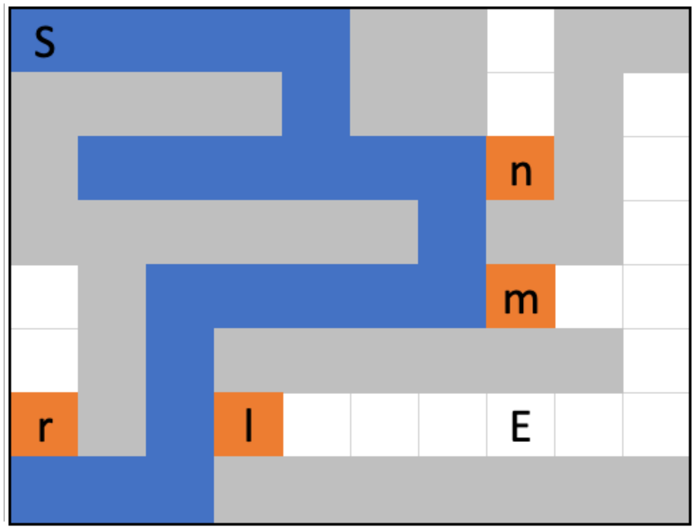
DFS example



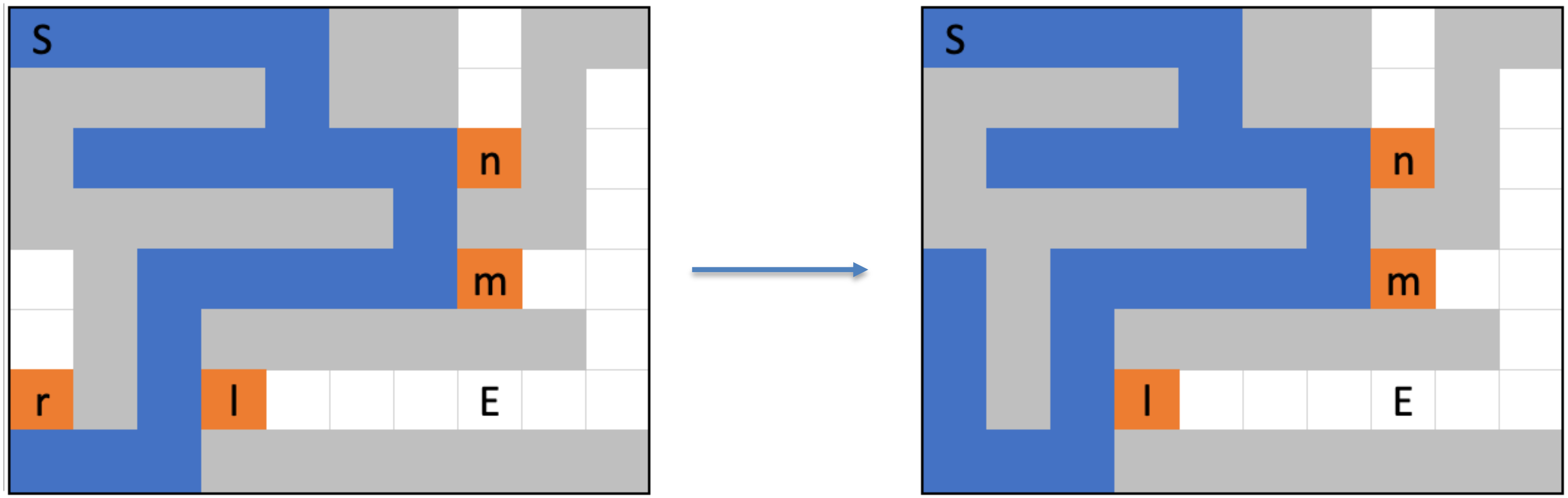
DFS example



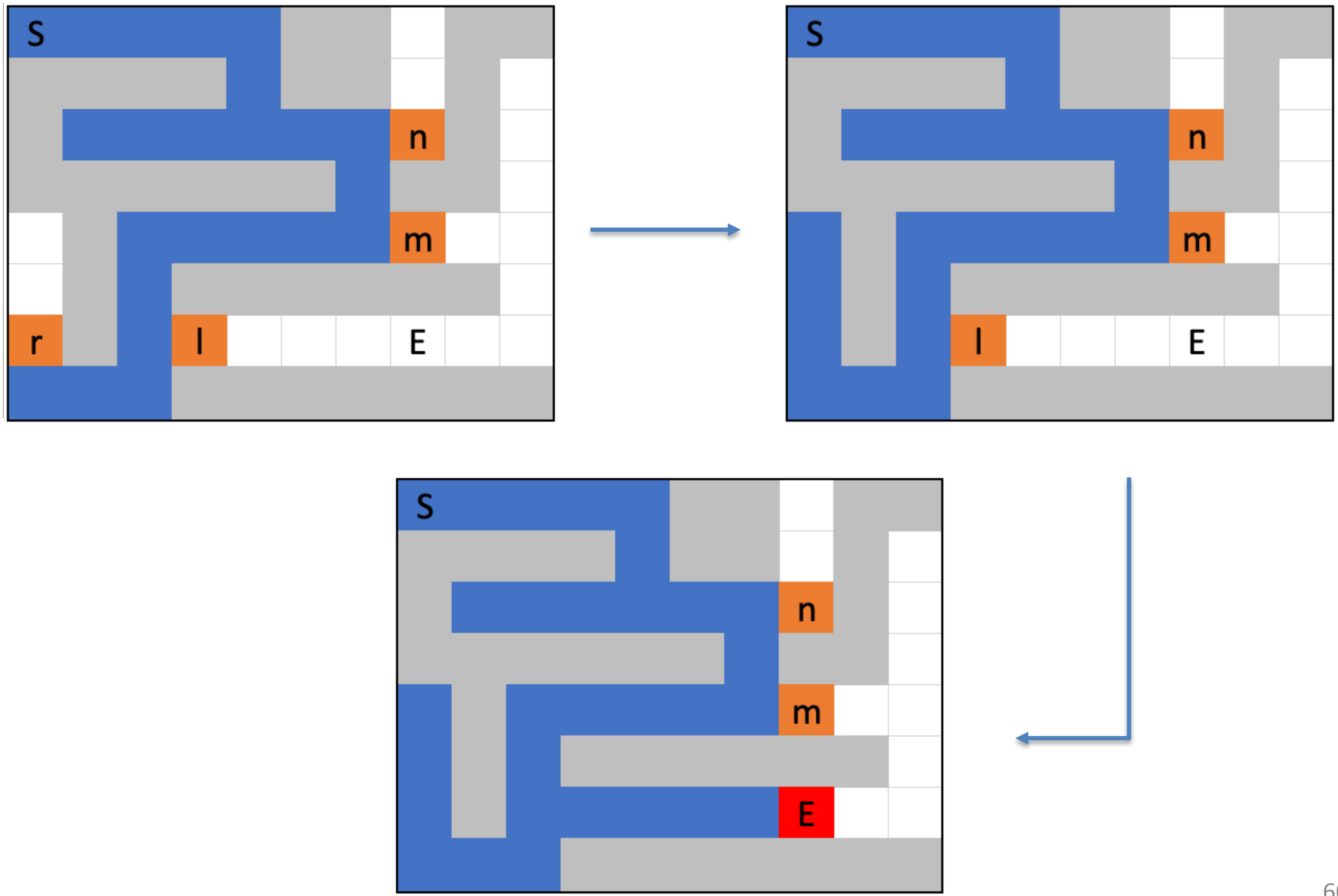
DFS example



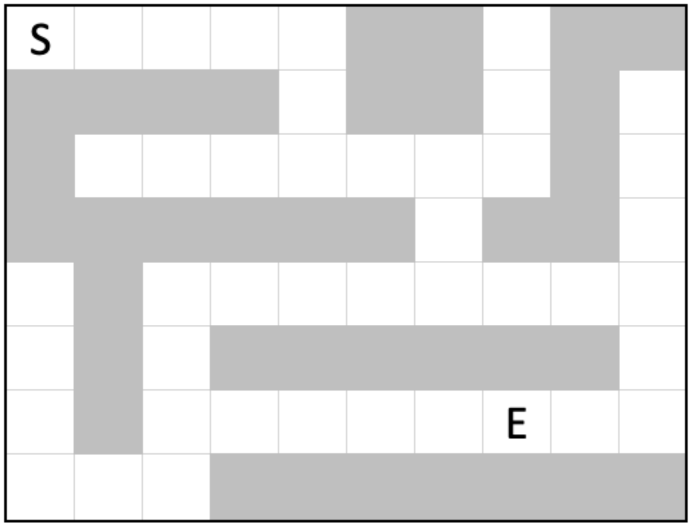
DFS example



DFS example

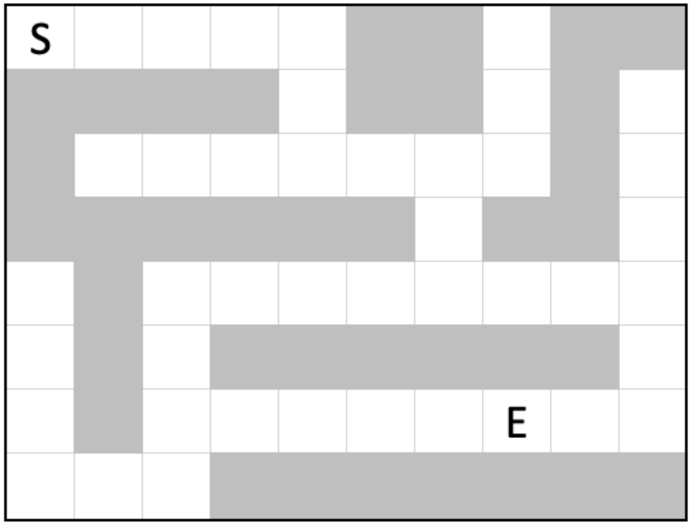


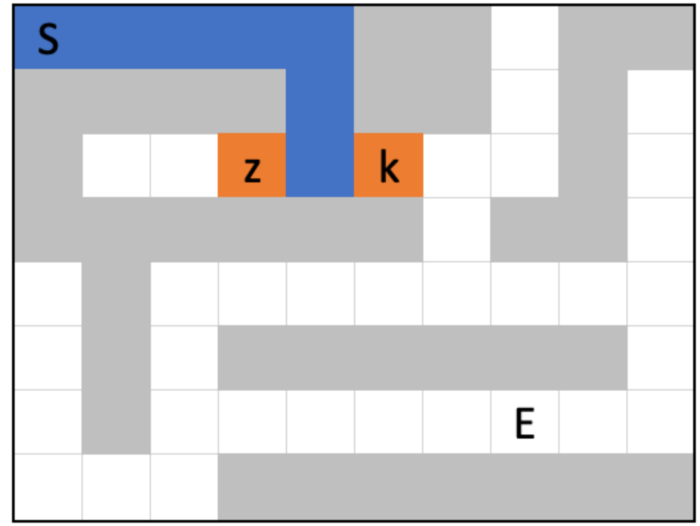
BFS example



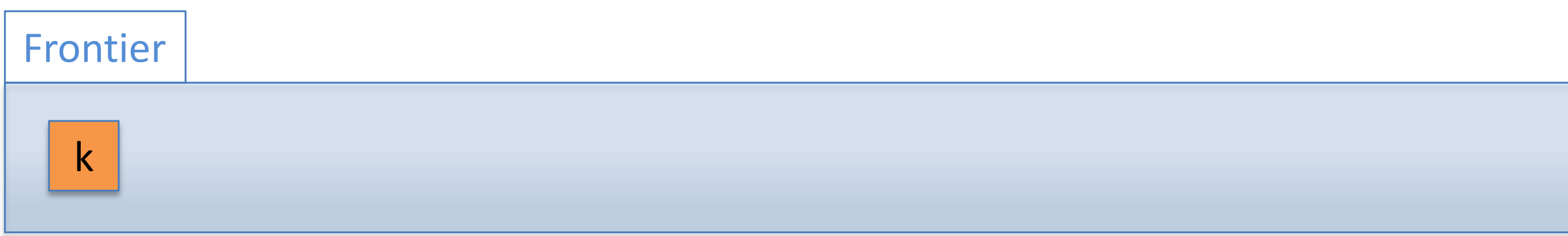
* Explored Set not shown for simplicity; assuming nodes are added as follows: right, down, left, top.

BFS example



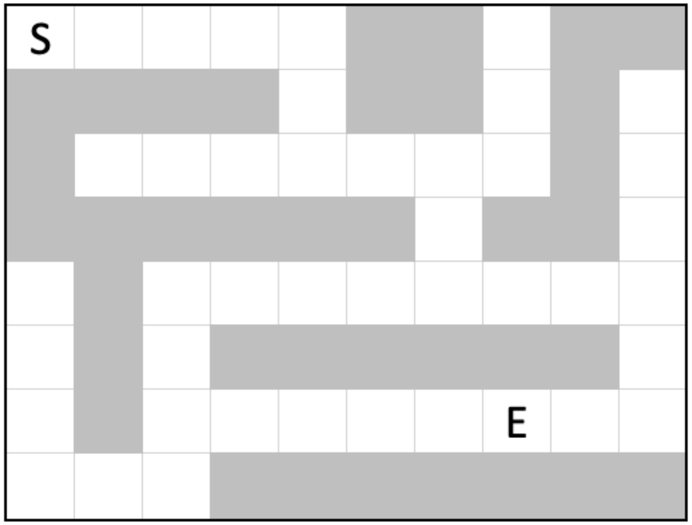


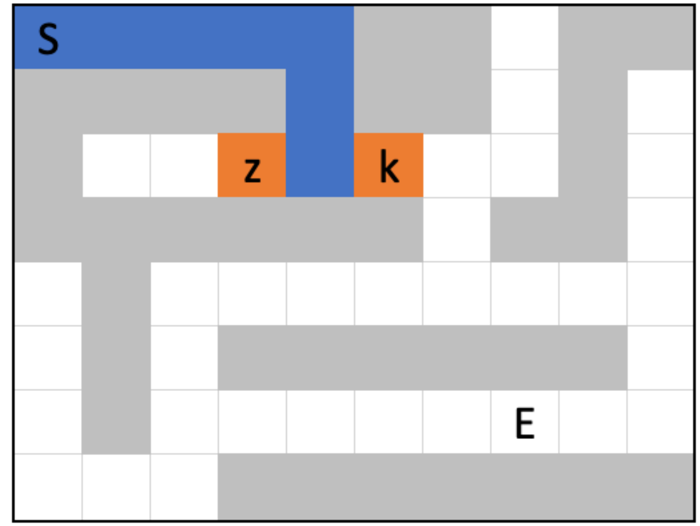
Adding **k** to the **Frontier**



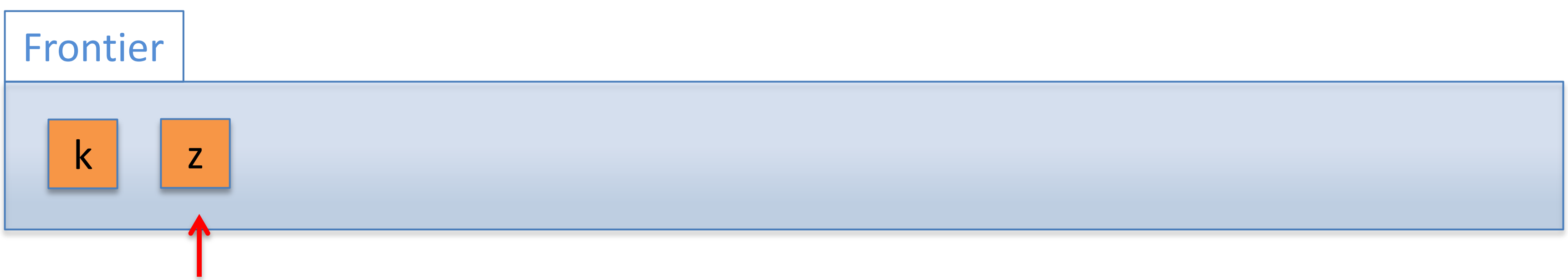
* Skipping the first trivial steps (i.e. only one path); assuming nodes are added as follows: right, down, left, top.

BFS example



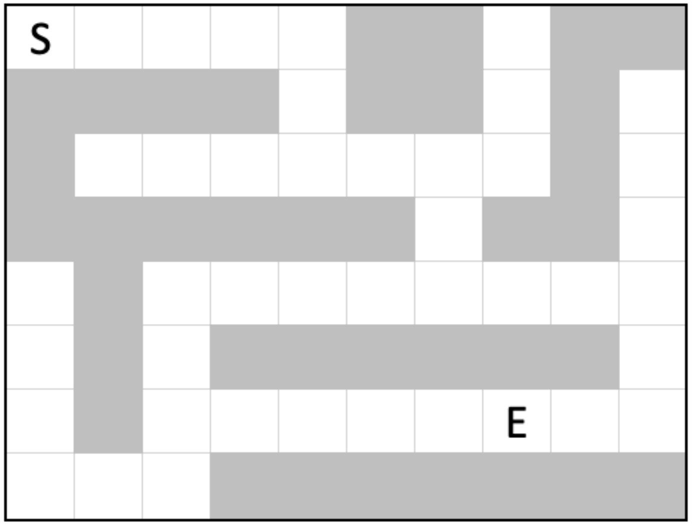


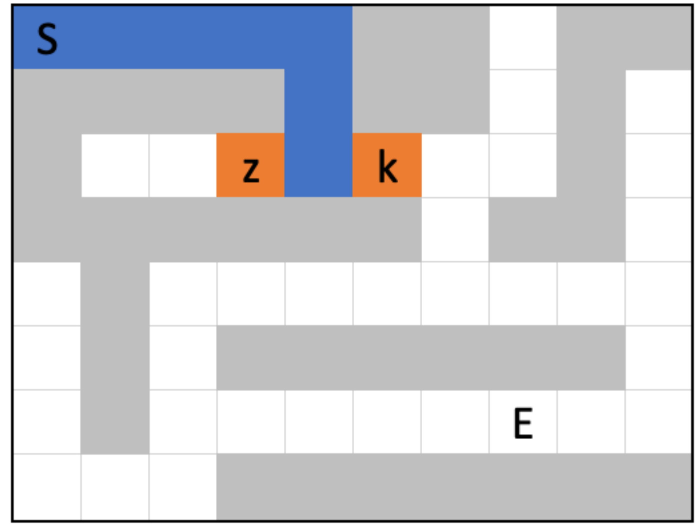
Adding **z** to the **Frontier**



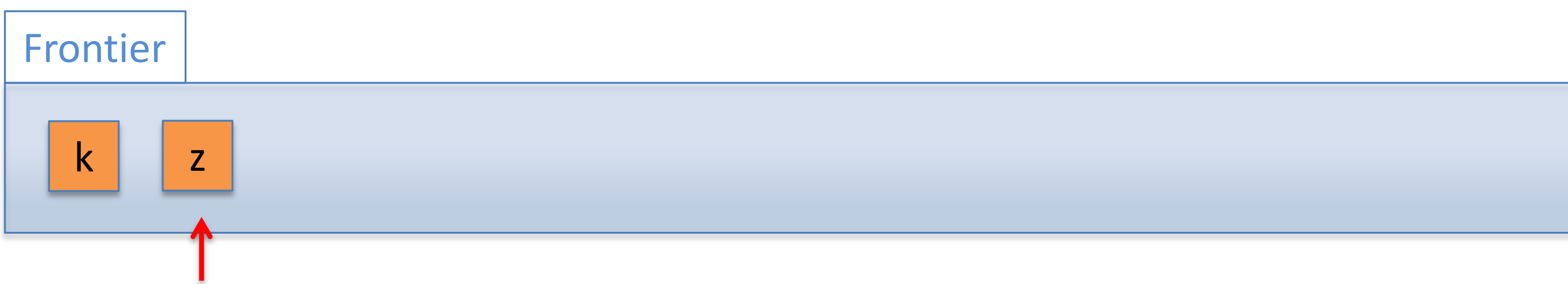
Added to the back of the **Queue**

BFS example



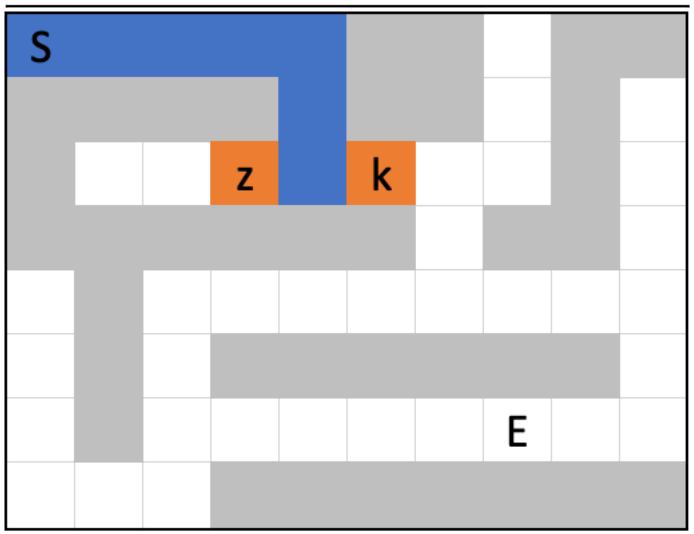


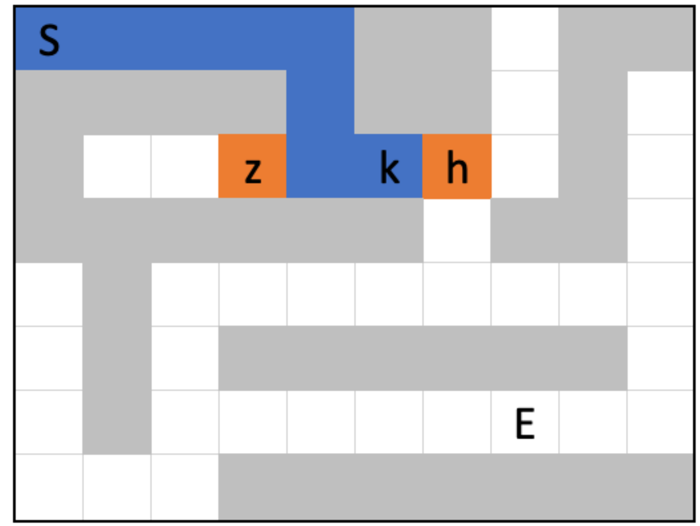
Adding **z** to the **Frontier**



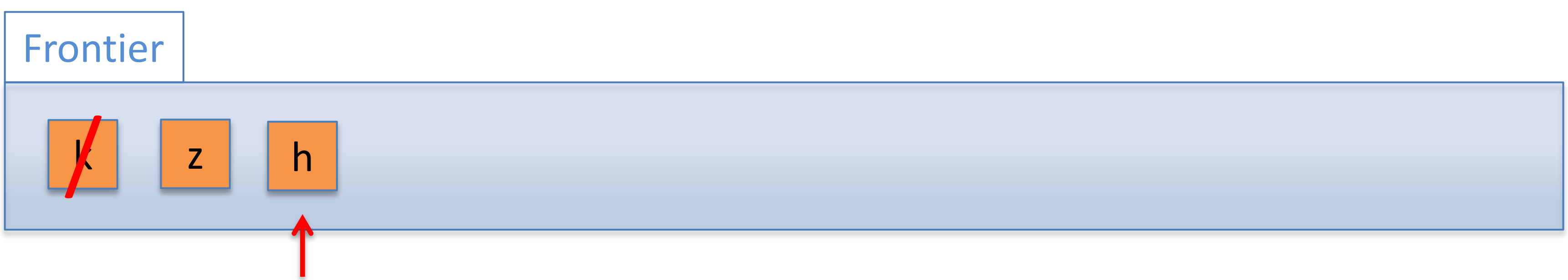
Added to the back of the **Queue**

BFS example



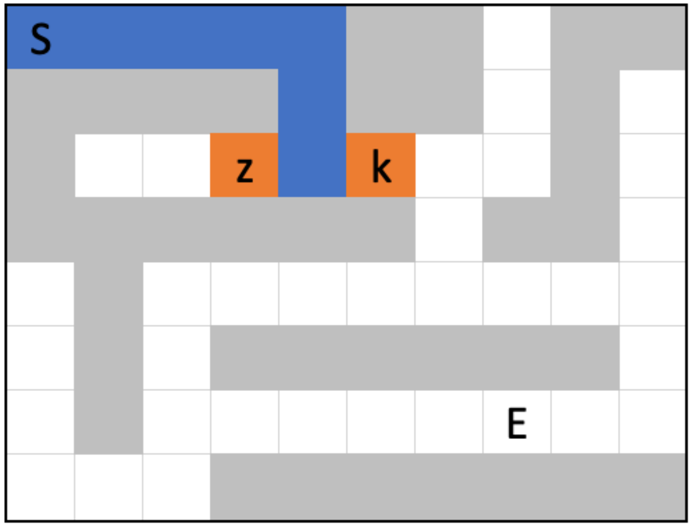


Adding **h** to the **Frontier**

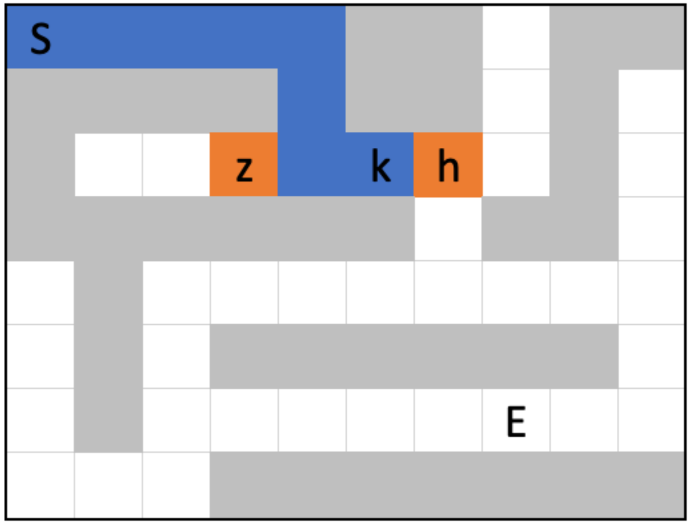


Added to the back of the **Queue**

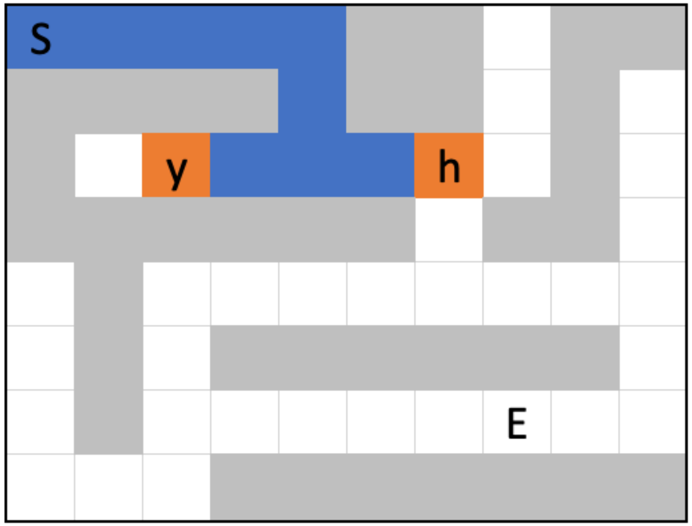
BFS example



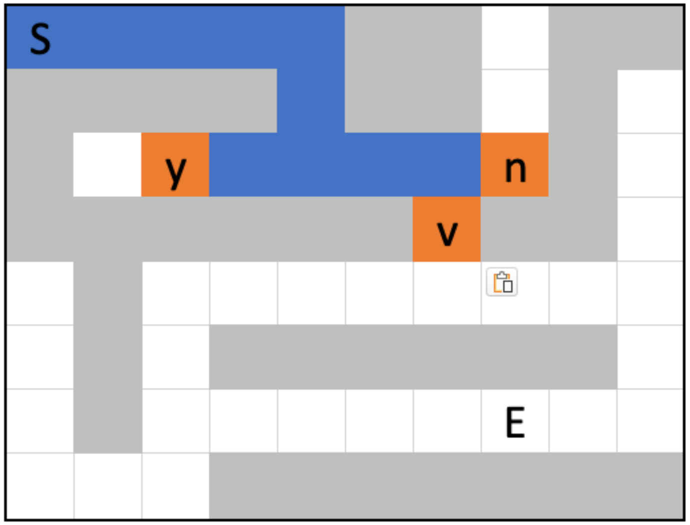
BFS example



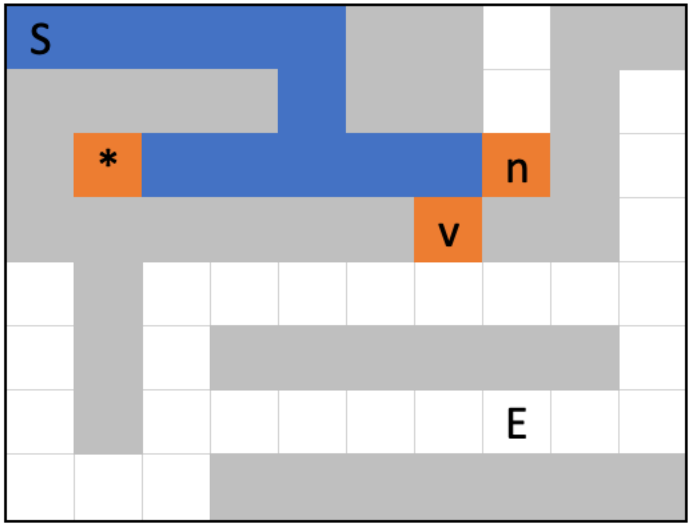
BFS example



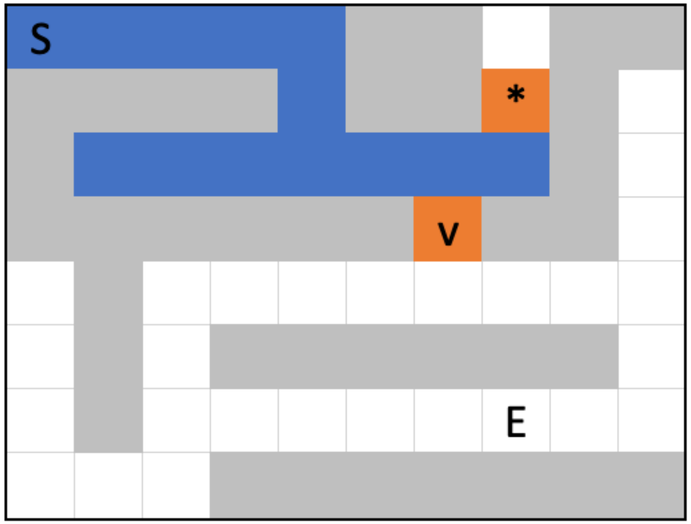
BFS example



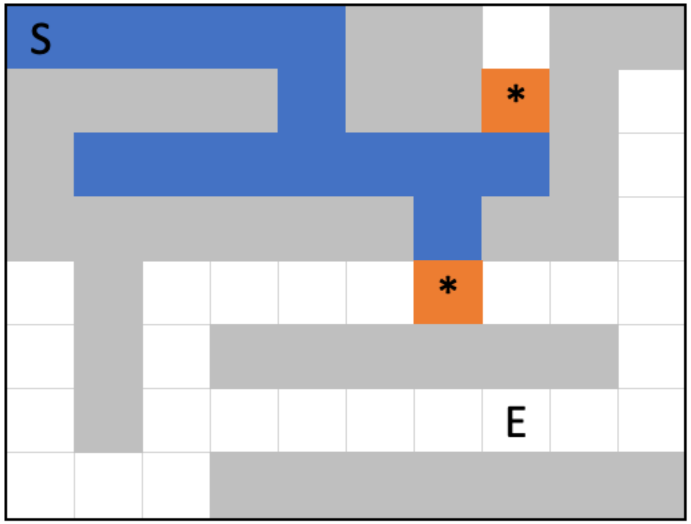
BFS example



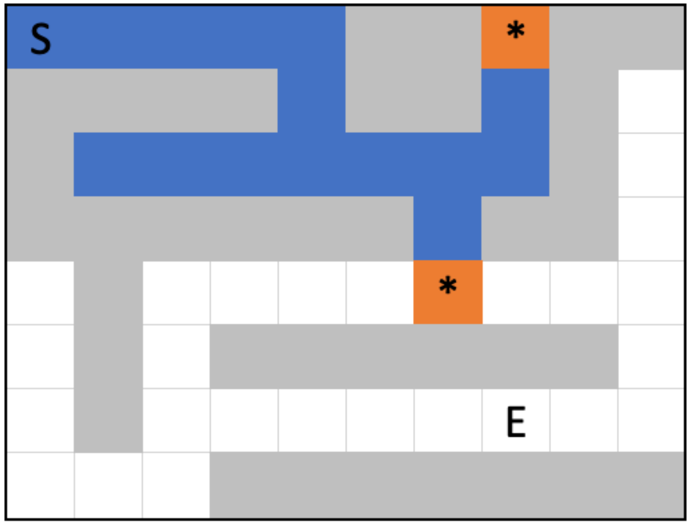
BFS example



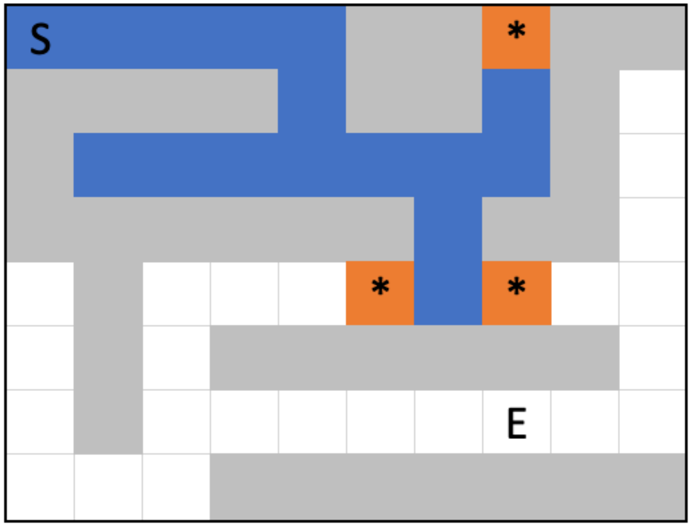
BFS example



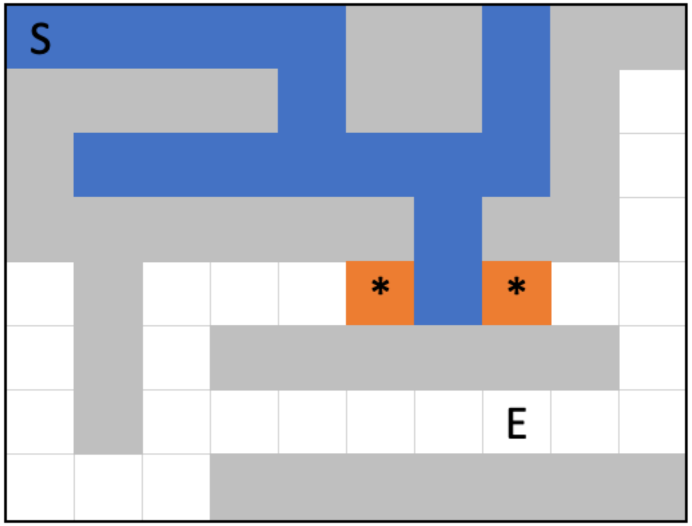
BFS example



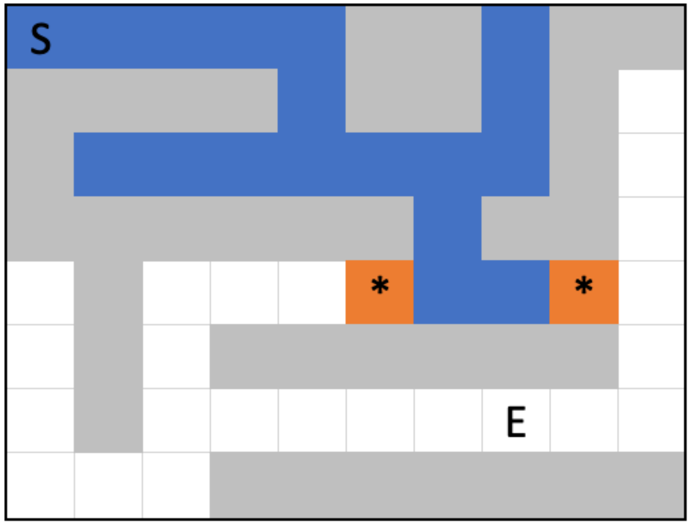
BFS example



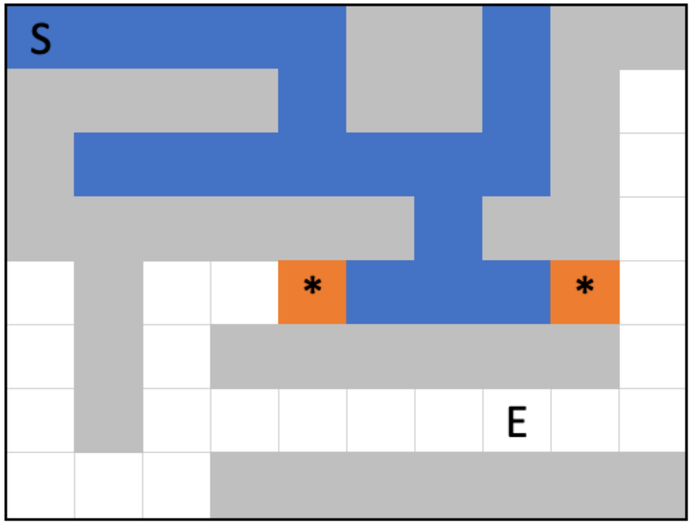
BFS example



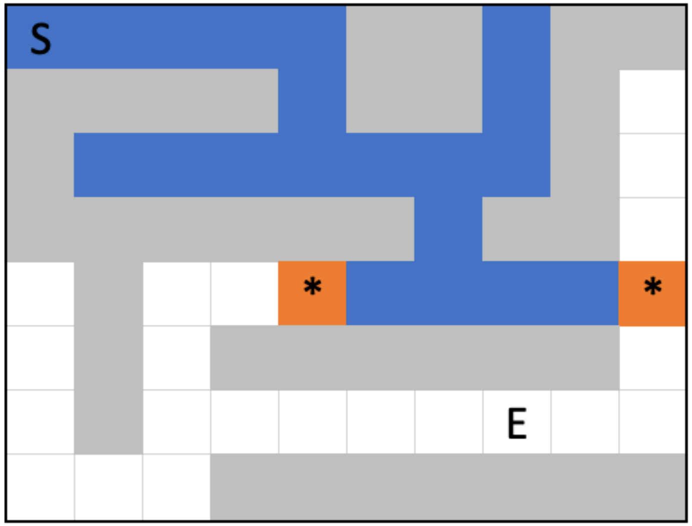
BFS example



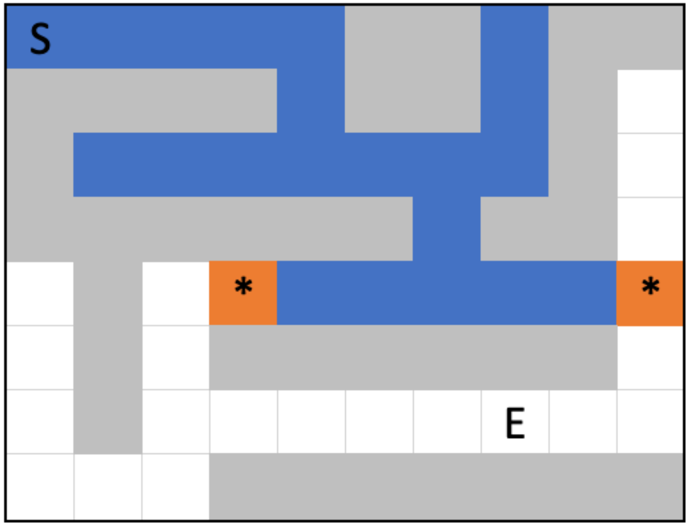
BFS example



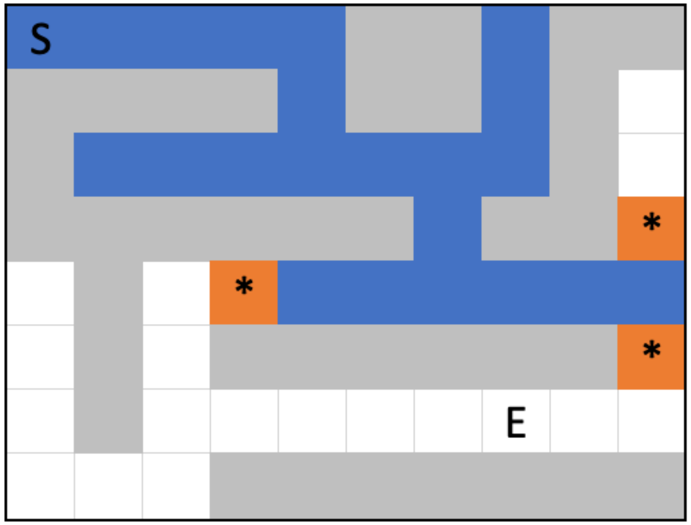
BFS example



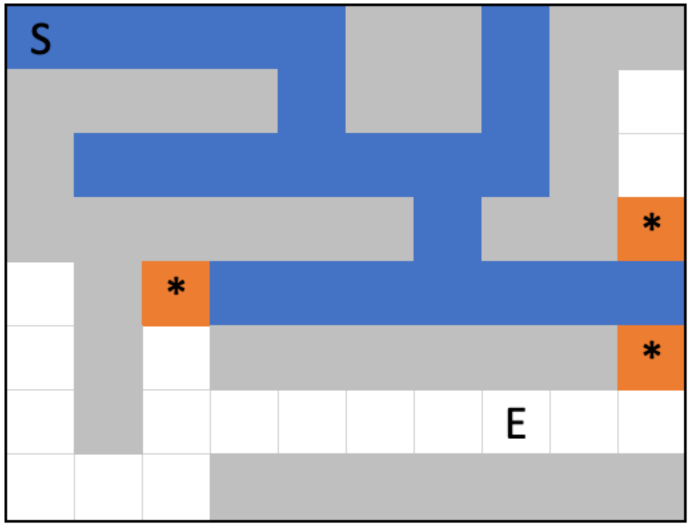
BFS example



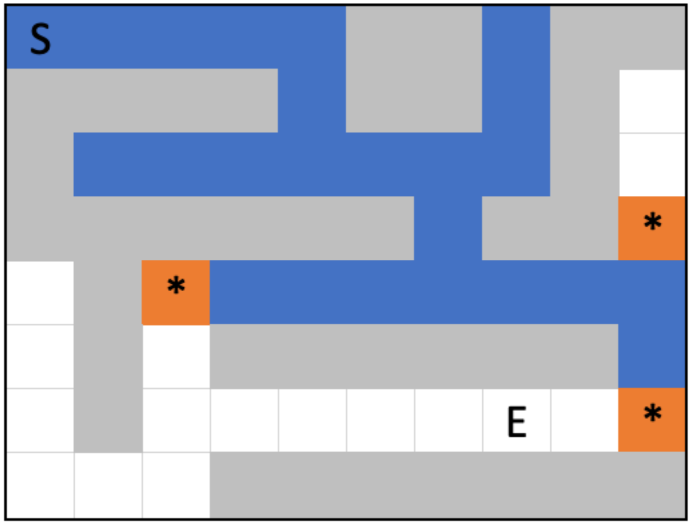
BFS example



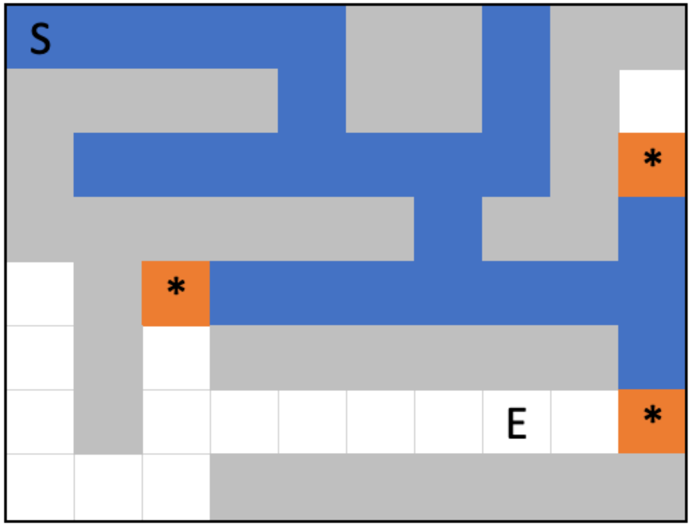
BFS example



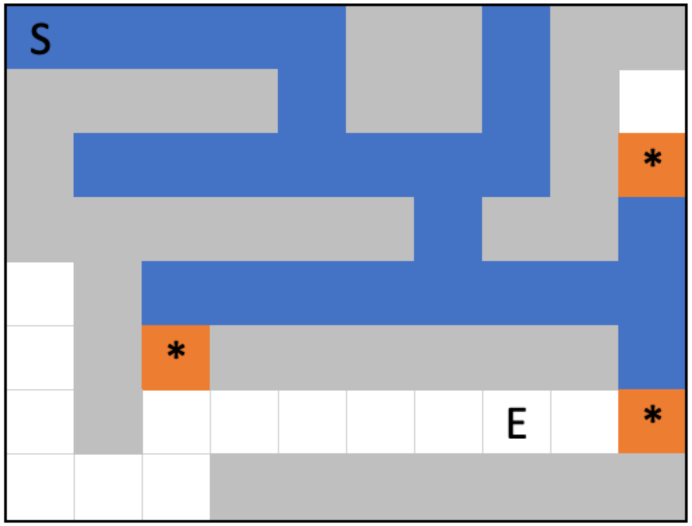
BFS example



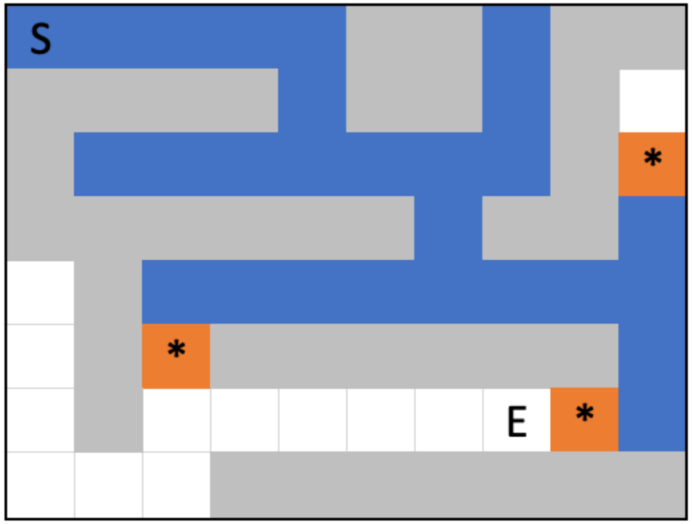
BFS example



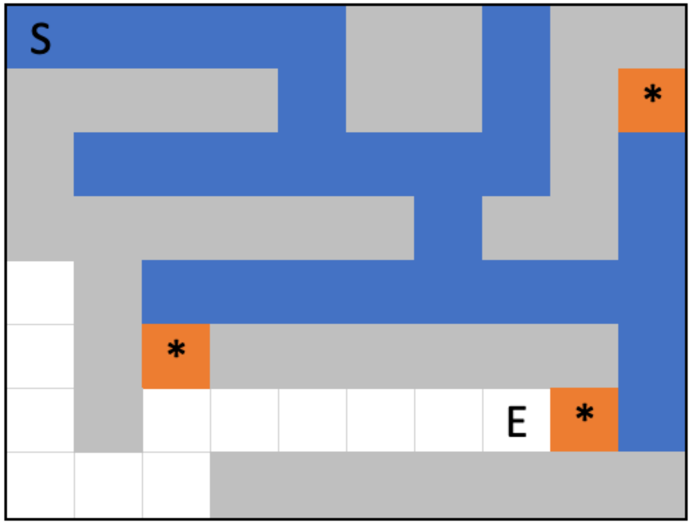
BFS example



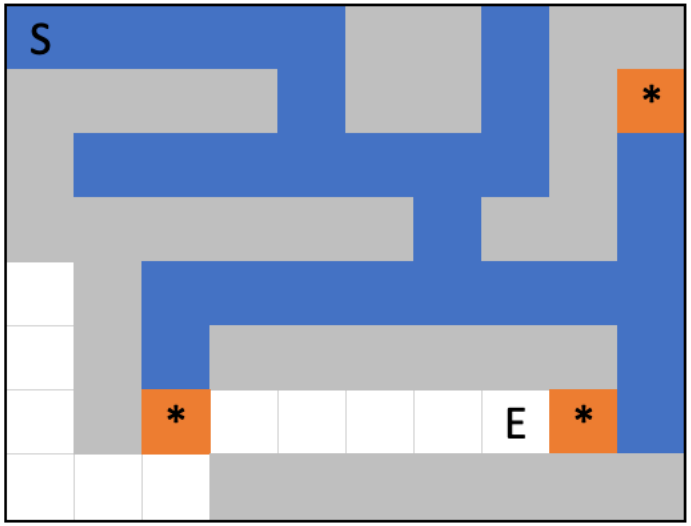
BFS example



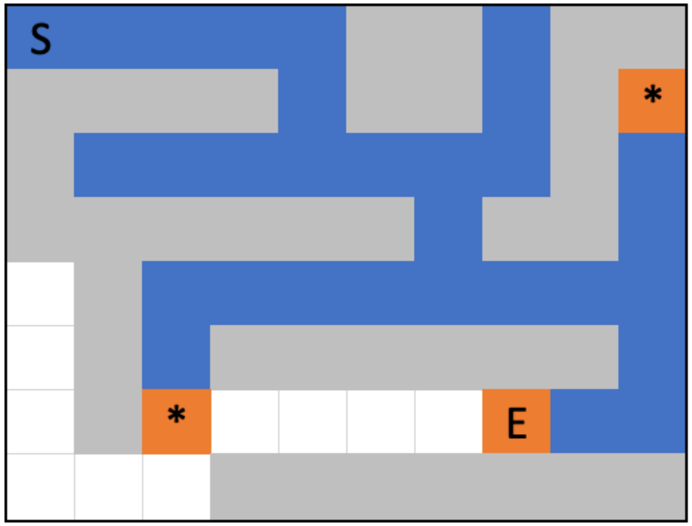
BFS example



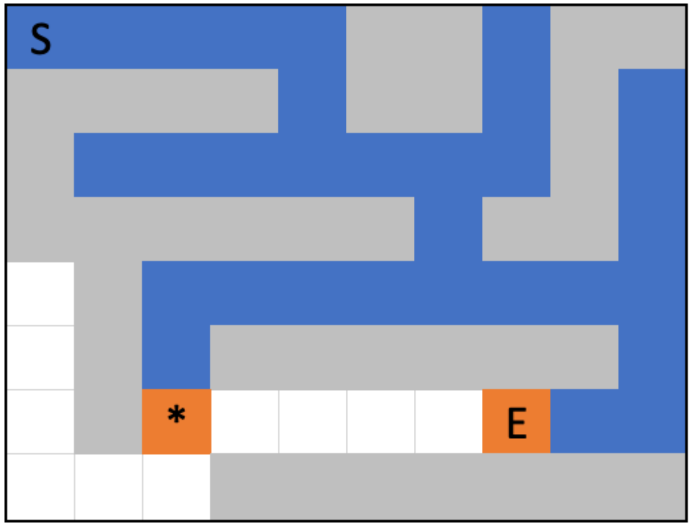
BFS example



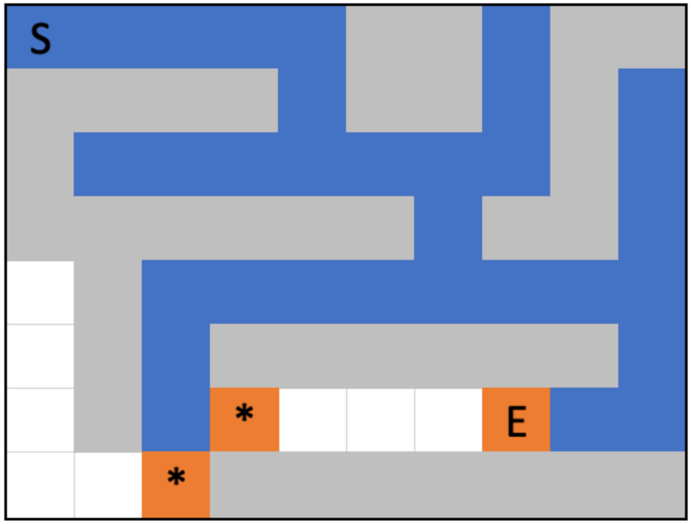
BFS example



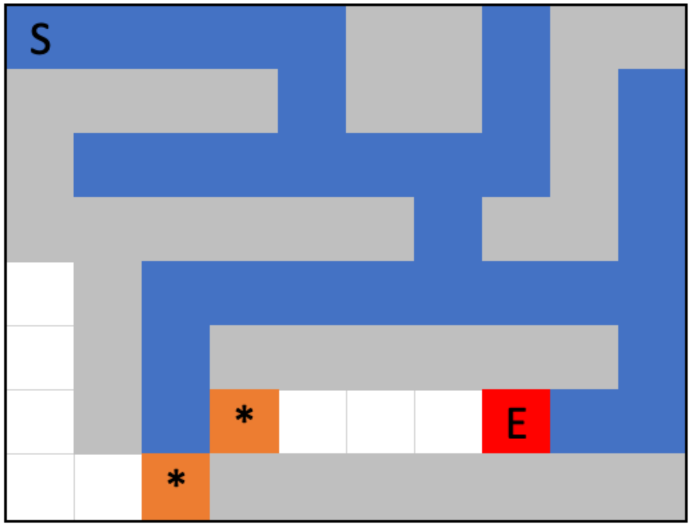
BFS example



BFS example



BFS example



We did it! We removed a node from the Frontier and it was the **Goal**

Assessing Search Strategies

- **Completeness:** Whether the strategy is guaranteed to find a solution when one exists. A complete search strategy will always find a solution if one exists

Assessing Search Strategies

- **Completeness:** Whether the strategy is guaranteed to find a solution when one exists. A complete search strategy will always find a solution if one exists
- **Optimality:** Whether the strategy finds the highest-quality solution when there are several solutions

Assessing Search Strategies

- **Completeness:** Whether the strategy is guaranteed to find a solution when one exists. A complete search strategy will always find a solution if one exists
- **Optimality:** Whether the strategy finds the highest-quality solution when there are several solutions
- **Time complexity:** the time it takes to find a solution

Assessing Search Strategies

- **Completeness:** Whether the strategy is guaranteed to find a solution when one exists. A complete search strategy will always find a solution if one exists
- **Optimality:** Whether the strategy finds the highest-quality solution when there are several solutions
- **Time complexity:** the time it takes to find a solution
- **Space complexity:** the memory the strategy needs to perform the search

Depth-First Search Analysis

- DFS is **complete** **if** the state space is **finite**
- DFS is **not** **optimal**
- In general, DFS is more efficient than BFS
  - Time Complexity: In the worst case*, DFS can explore every node, resulting in a time complexity of $O(b^d)$, where b is the **branching factor** and d is the **maximum depth**.
  - Space Complexity: depends on the maximum depth of the state space. In the worst case, uses space proportional to the maximum depth m, so it is $O(bm)$

Breadth First Search Analysis

- BFS is **complete** even if the state space is infinite*
- BFS is **optimal** if not weighted, i.e. shallowest solution
- In general, BFS can be very expensive
  - Time Complexity: Similar to DFS*, in the worst case BFS will explore all nodes, so $O(b^d)$.
  - Space Complexity: The space complexity of BFS is also $O(b^d)$ because it needs to keep track of all the nodes on the current level in memory, and the number of nodes at each level grows exponentially with the depth.

BFS vs DFS

- **Important!** The time complexity of **BFS is typically higher than that of DFS**, especially if the **branching factor** is high and the **goal** node is located **near the bottom** of the state space.

BFS vs DFS

- **Important!** The time complexity of **BFS is typically higher than that of DFS**, especially if the **branching factor** is high and the **goal** node is located **near the bottom** of the state space.
- **Why?** BFS needs to explore all nodes at a given depth before moving on to the next depth, whereas DFS can quickly move down a path until it reaches a dead end (leaf)

Depth Limited Search (DLS)

- Variant of DFS that **limit the maximum depth during exploration**
- DLS is **complete** if the **limit is greater than or equal to** the depth of the shallowest solution node
- DLS is not optimal

Depth Limited Search (DLS)

- Variant of DFS that **limit the maximum depth during exploration**
- DLS is **complete** if the **limit is greater than or equal to** the depth of the shallowest solution node
- DLS is not optimal

- **Disadvantage 1:** it **may not** be able to **find solutions** that are **deeper** than the **maximum depth limit**, even if they exist
- **Disadvantage 2:** it **may repeat** the **same path** if the **limit** is **not adequate (too low)**, leading to inefficiency

Iterative Deepening Search (IDS)

- Combines the benefits of BFS and DFS
- **Repeatedly** performs **DFS with increasing depth limits**
- IDS has the same time complexity as BFS ($O(b^d)$), but its space complexity is closer to DFS ($O(bd)$) only stores current path
- IDS is **complete** and **optimal** (if path cost is non-decreasing with depth)
- IDS is useful when space is large and goal depth is unknown

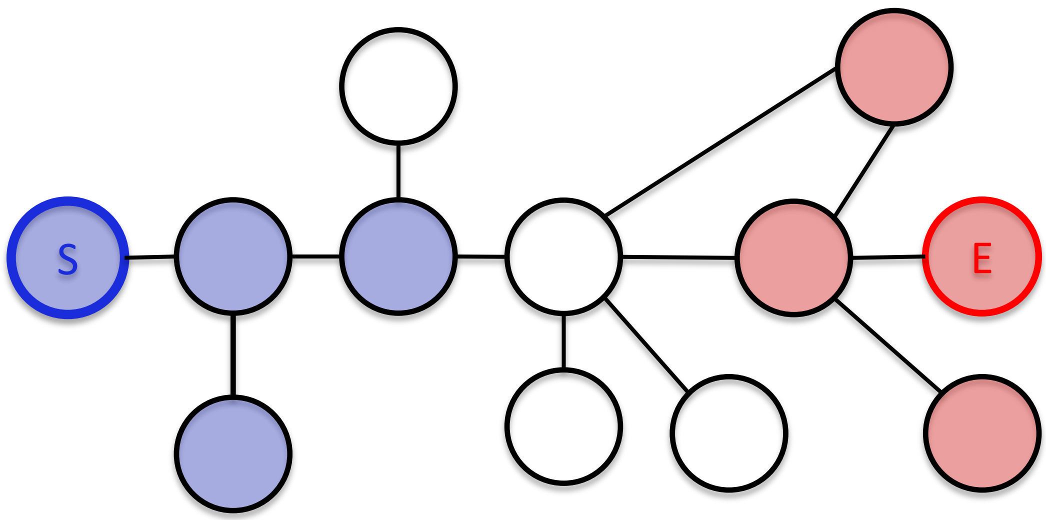
Iterative Deepening Search (IDS)

- Combines the benefits of BFS and DFS
- **Repeatedly** performs **DFS with increasing depth limits**
- IDS has the same time complexity as BFS ($O(b^d)$), but its space complexity is closer to DFS ($O(bd)$) only stores current path
- IDS is **complete** and **optimal** (if path cost is non-decreasing with depth)
- IDS is useful when space is large and goal depth is unknown

Try the maze example using IDS!

Bidirectional Search

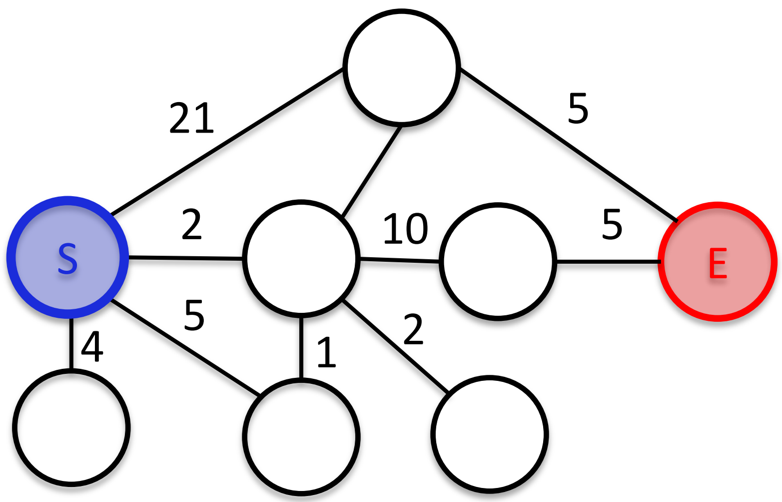
- One BFS from the **initial state** and one BFS from the **goal**
- **The searches proceed until their frontiers meet in the middle.**



- Time complexity: $O(b^{d/2})$
- Space complexity: $O(b^{d/2})$
- Issues? We need to know where is the goal state

Uniform Cost search

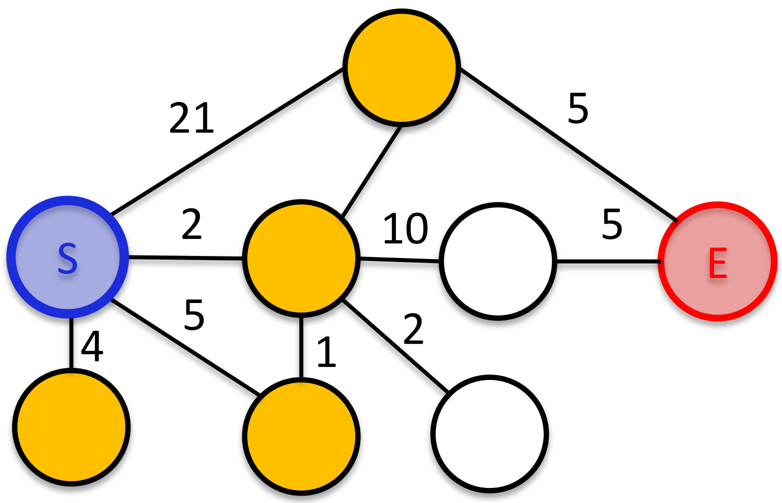
- Behaves as BFS if the actions have the same cost (no weights)
- Expand first nodes with lowest cost; Remember Dijkstra's algorithm?



- <u>Time complexity:</u> $O(b^d)$
- <u>Space complexity:</u> $O(b^d)$
- Optimal if all costs are positive

Uniform Cost search

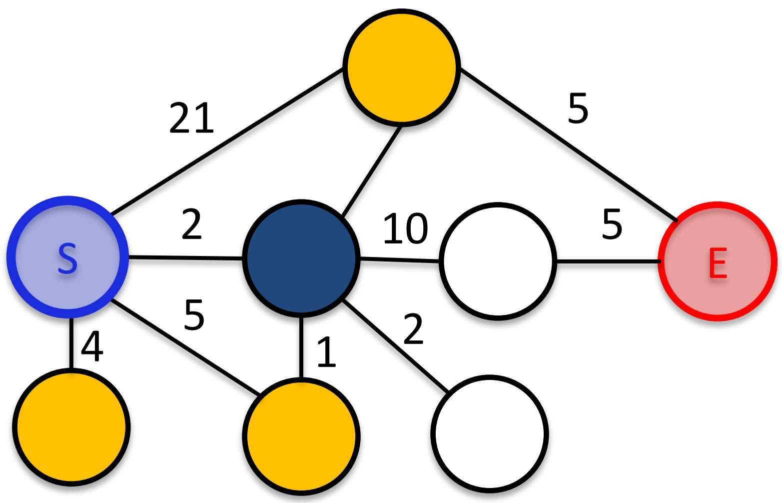
- Behaves as BFS if the actions have the same cost (no weights)
- Expand first nodes with lowest cost; Remember Dijkstra's algorithm?



- Time complexity: $O(b^d)$
- Space complexity: $O(b^d)$
- Optimal if all costs are positive

Uniform Cost search

- Behaves as BFS if the actions have the same cost (no weights)
- Expand first nodes with lowest cost; Remember Dijkstra's algorithm?



- Time complexity: $O(b^d)$
- Space complexity: $O(b^d)$
- Optimal if all costs are positive

Uninformed & Informed

- **Uninformed Search:** The algorithm does not consider specific knowledge related to the problem
  - DFS, DLS, IDS, BFS, Bidirectional Search, Uniform Cost
- **Informed Search:** Exploit knowledge specific to the problem (e.g. Greedy BFS, A*)

Uninformed & Informed

- **Uninformed Search:** The algorithm does not consider specific knowledge related to the problem
  - DFS, DLS, IDS, BFS, Bidirectional Search

- **Informed Search:** Exploit knowledge specific to the problem (e.g. Greedy BFS, A*)

Question: What is the disadvantage of using an Explored Set?

Informed Search

A **heuristic func** ***h(n)*** estimates the cheapest **cost** from **node** ***n*** to the **goal**

h(n) must be **admissible**, i.e. never overestimates the cost

- *h(n)* is defined by relaxing the problem
- E.g. Ignoring walls and using the **Manhattan distance** in a maze

Why/When do we need heuristics?

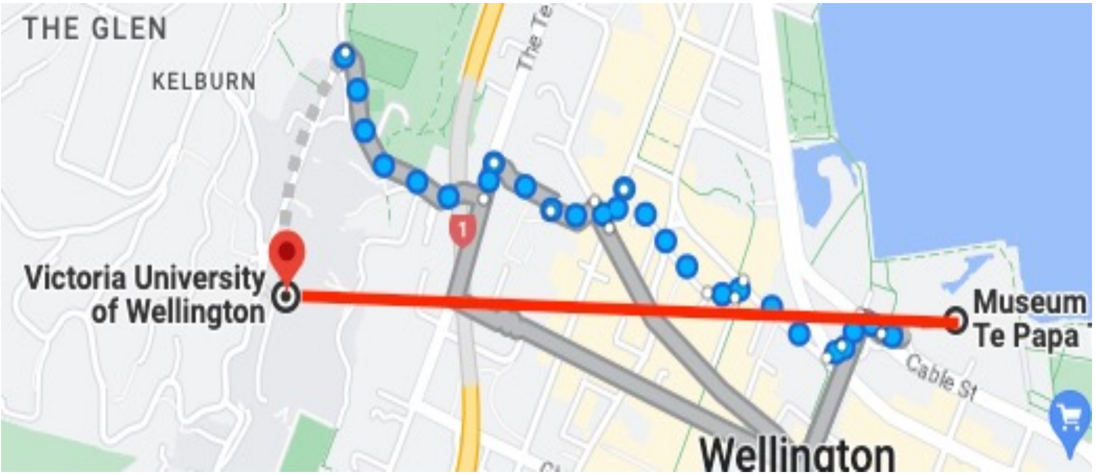
- When the **search space (state space) is too large**!
- Example: chess has a branching factor of 35…

Greedy (Best First) Search

- Always expand node whose state appears to be closer to the goal state
- The data structure is a ***priority queue***
  - Priority is given by ***f(n)***, such that ***f(n) = h(n)***

Greedy (Best First) Search

- Always expand node whose state appears to be closer to the goal state
- The data structure is a ***priority queue***
  - Priority is given by ***f(n)***, such that ***f(n) = h(n)***
- ***Example:*** *Route finding in a map*



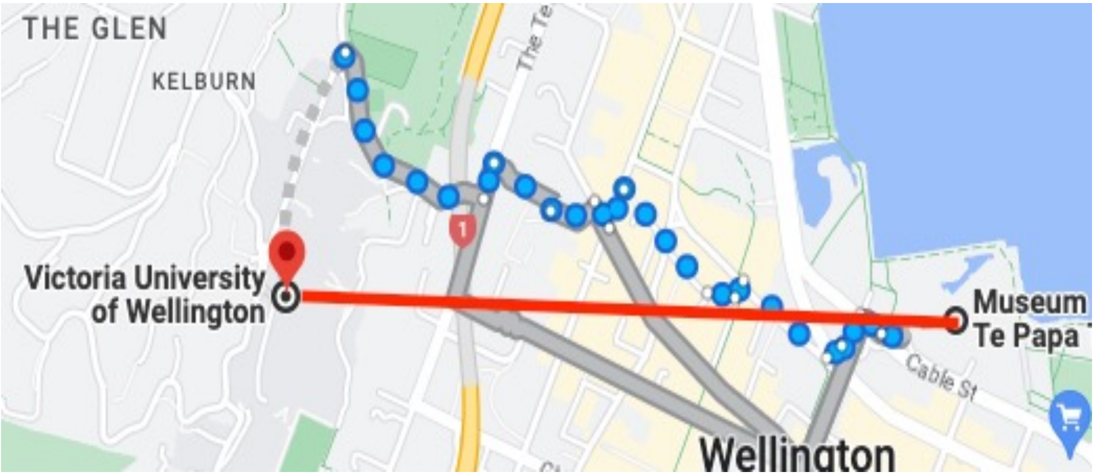
At each node, h(n) gives the estimated dist.

Initial state: Te Papa

Goal state: VuW

h(Te Papa) **=** straight line distance, e.g. **1km**

Greedy (Best First) Search



At each node, h(n) gives the estimated dist.

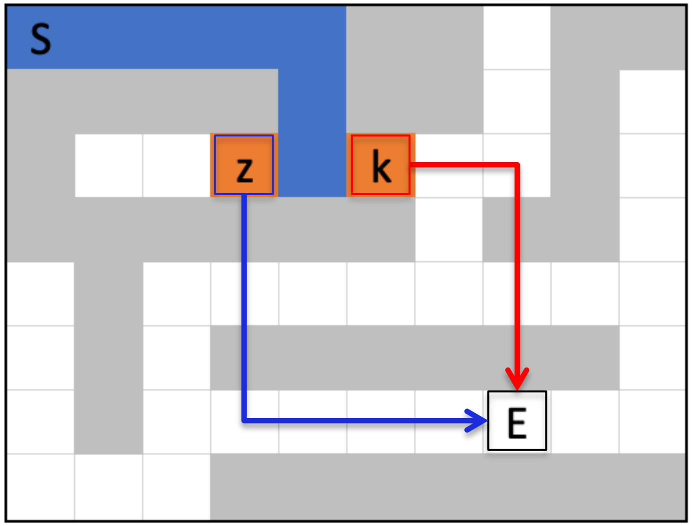
Initial state: Te Papa

Goal state: VuW

h(Te Papa) **=** straight line distance, e.g. **1km**

- Actual path is longer (1.6km)
- ***h(n)*** **informs** the algorithm to avoid unnecessary actions:
  - Reaching Victoria Street (in Hamilton) during the search (DFS)
  - Exploring the whole CBD first (BFS)

Greedy (Best First) Search - Maze



$h(z) = 8$

$h(k) = 6$

- *Which node should the algorithm explore next?*

Greedy (Best First) Search - Maze

- *Not* ***optimal*** *and not* ***complete****!*
- *May explore 'false' paths*
- *Time and Space Complexity: $O(b^m)$*

Crucial issue: Ignores the path cost ***g(n)***, which is the cost from the initial state up to node **n**

Greedy (Best First) Search - Maze

- *Not* ***optimal*** *and not* ***complete****!*
- *May explore 'false' paths*
- *Time and Space Complexity: $O(b^m)$*

Can you think of a Maze where this algorithm would explore a "false" path?

Crucial issue: Ignores the path cost ***g(n)***, which is the cost from the initial state up to node **n**

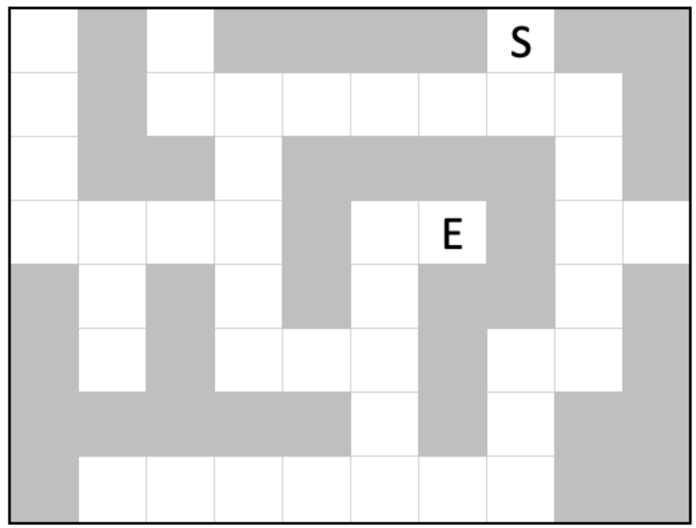
A*

Estimates the **total path cost f(n)**

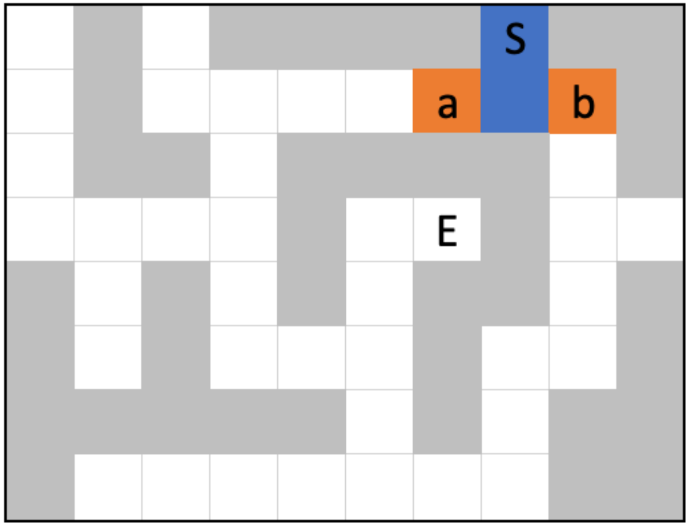
- ***f(n)*** *= g(n) + h(n)*
- *g(n): from the initial node to node n*
- *h(n): estimated cost of "relaxed" path from n to goal*

f(n) *represents the estimated cost of the cheapest solution through n*

A* - maze

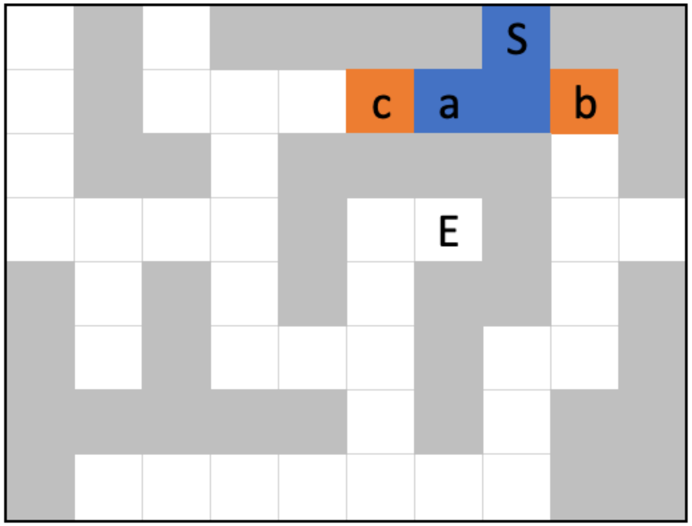


A* - maze



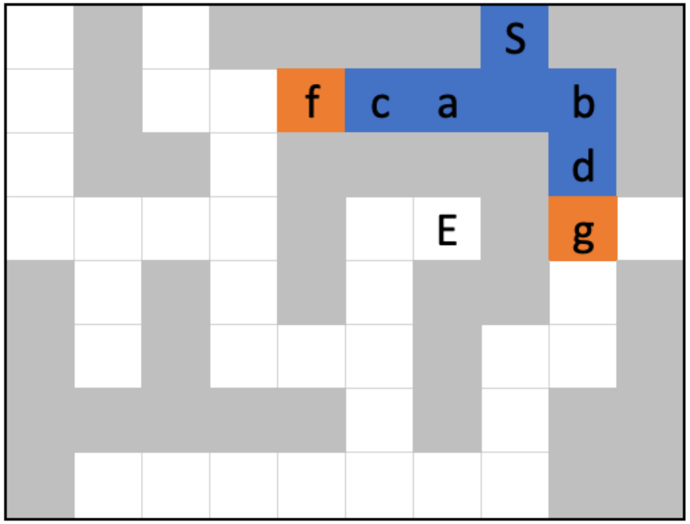
	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4

A* - maze



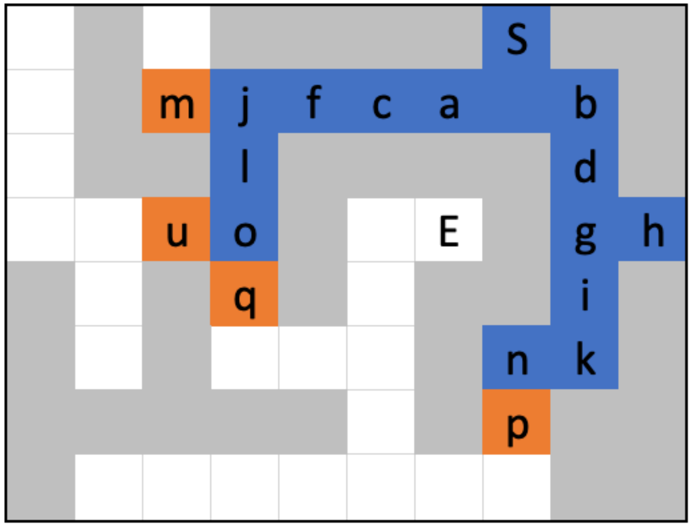
	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4
c	6	3	3

A* - maze



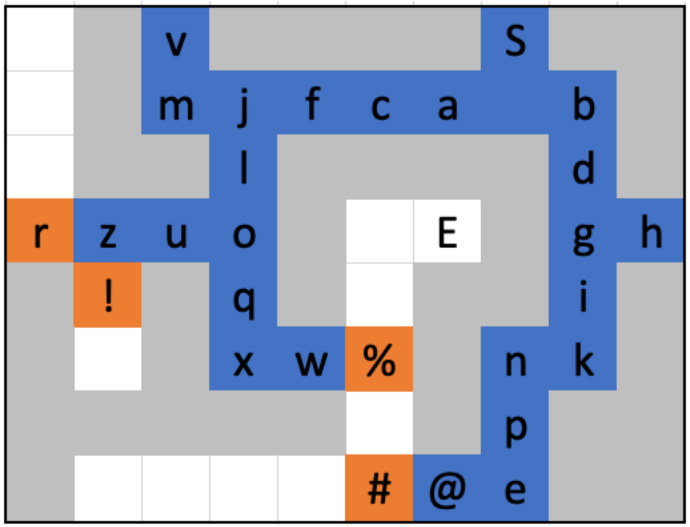
	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4
c	6	3	3
d	6	3	3
g	6	4	2
f	8	4	4

A* - maze



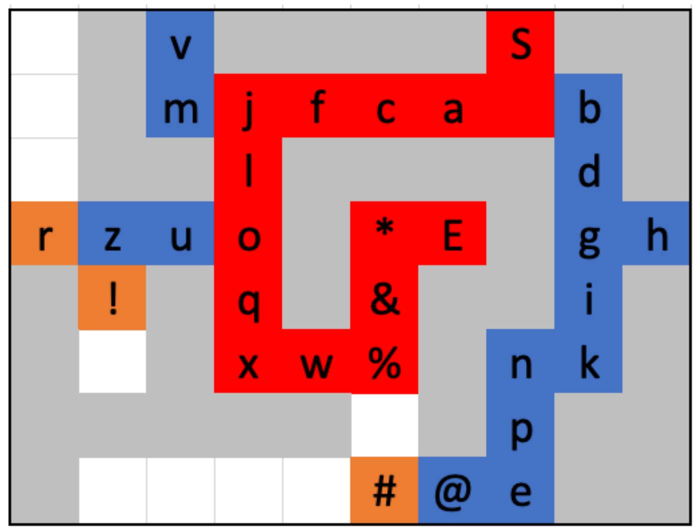
	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4
c	6	3	3
d	6	3	3
g	6	4	2
f	8	4	4
…	…	…	…
o	10	7	3
m	12	6	6
p	12	8	4
q	12	8	4
u	12	8	4

A* - maze



	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4
c	6	3	3
d	6	3	3
…	…	…	…
m	12	6	6
p	12	8	4
…	…	…	…
w	14	10	4
%	14	11	3
!	16	10	6
r	16	10	6
#	16	11	5

A* - maze



	f(n)	g(n)	h(n)
a	4	2	2
b	6	2	4
…	…	…	…
%	14	11	3
!	16	10	6
r	16	10	6
#	16	11	5
…	…	…	…
*	14	13	1
E	14	14	0
!	16	10	6
r	16	10	6
#	16	11	5

Summary

- Search is an important part of several other algorithms
- **Abstracting** the problem is fundamental
- **Selecting** an appropriate Search algorithm
- **Defining** ***h(n)*** may not always be trivial
- We were focusing **on finding the *path* from S to E**

Coming up next…

- Search 2 (next lecture)
- History AI (Friday Tutorial) – Prof Mengjie Zhang
- Tip: Try out the Search algorithms!