

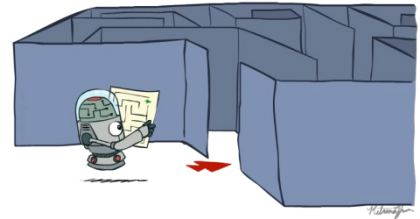
UCCD2063

# Artificial Intelligence Techniques

---

Unit 10:

## Search Algorithms





# Outline

---

- **Search Problems**
  - **Problem formulation**
- Search Algorithms
  - Uninformed search
  - Informed search
- Adversarial Search
  - Minimax
  - Alpha-Beta Pruning

## Reference material:

- [CS188 AI Note 1](#)

# Search Problem

- **Search problem** is the task of finding the **sequence of actions** or steps that leads to the desired **goal**
- Example search problem:

Suppose a man wants to transport his **cabbage**, **goat** and **wolf** across the river. Unfortunately, his boat is very small to carry more than 2 passengers across. He cannot leave his **cabbage** and **goat** alone (goat eats cabbage), or his **wolf** and **goat** alone (wolf eats goat).

**Find the solution that allows the man to cross the river without losing anything in the shortest time possible.**



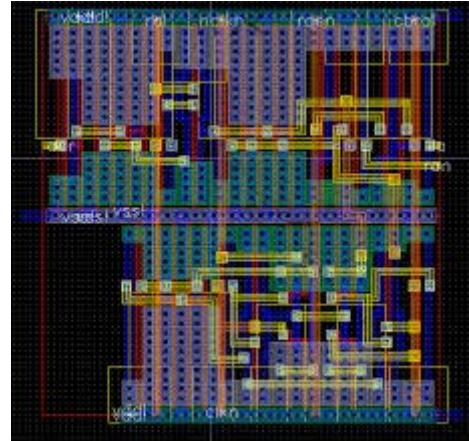
## Example Real-world Search Problems

- **Automatic assembly sequencing:** find the best order to assemble parts of an object. The process involves a difficult and expensive geometric search.
- **Protein design:** find the sequence of amino acids that will fold into a 3D protein with the right properties to cure some disease.



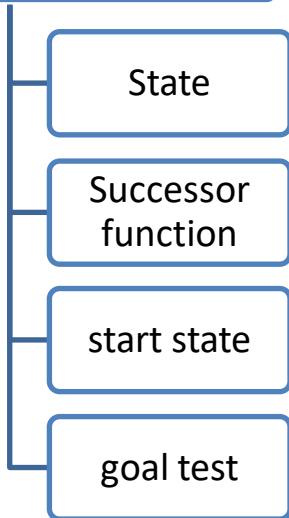
# Real-world Search Problems

- **VLSI layout:** find the best way to layout millions of components and connections on a chip which minimizes area, shorten delay and leave space for wiring.
- **Traveling salesman problem:** Find the shortest route to visit each city exactly once



# Definitions

## Search Problem



A **search problem** comprises the following:

- The set of all possible configurations
- Given a *state*, defines the list of possible *actions* and their corresponding *costs* and *resultant states*
- The (initial) state that agent starts in
- Determines whether a given state is a goal state.

- A **solution** is a sequence of **actions** (a plan) which transforms the **start** state to a **goal** state

# The 4 stages of solving search problems

## 1. Goal Formulation

- The agent adopts a **goal**.
- A goal is a set of states in which the goal is satisfied



## 2. Problem formulation

- Define **state**, **successor function** (actions, cost), **start state** and **goal state**



## 3. Search for solution

- Use **search algorithms** to find the best sequence of actions that leads from start state to goal state in the search



## 4. Execution

- The agent executes the actions specified by the solution found

# Problem formulation

- Transforming the task into a search problem

## Example: Crossing the river problem

Let's use the following notation:

**M**: man      **G**: goat      **C**: cabbage      **W**: wolf

**States:**

{those at the left bank} || {those at the right bank}

CW    MG	MCW    G	W    MCG
MGW    C	...	C    MGW

Invalid: CGW || M    GW || MC    CG || MW    ...

**Start state:** MCGW ||

**Goal state:** || MGCW



# Successor functions:

## List of possible actions:

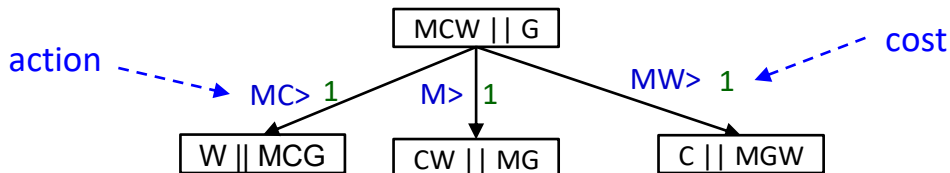
M> man crosses alone  
MC> man crosses with cabbage  
MG> man crosses with goat  
MW> man crosses with wolf  
M< man returns alone  
MC< man returns with cabbage  
MG< man returns with goat  
MW< man returns with wolf

## Cost of action

- 1 for each trip

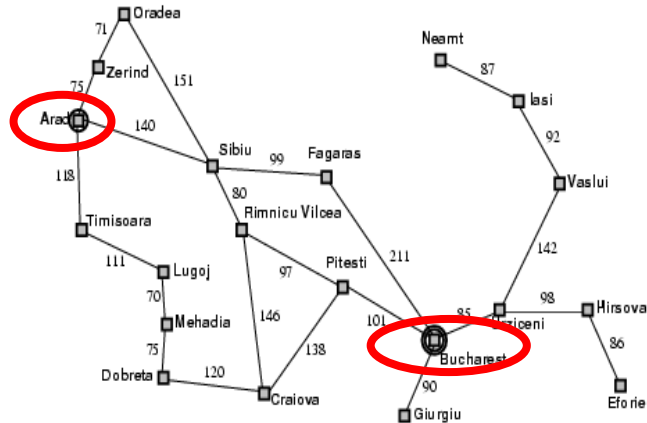
## Transition model:

Given a state, returns the list of possible actions, their corresponding cost and resultant states. For example:



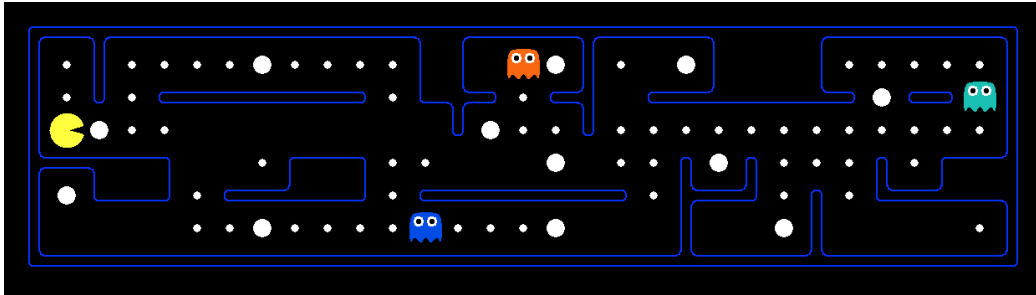
# Problem formulation example: Route finding

**Route finding problem:** Given the map of Romania, find the way to go from Arad to Bucharest



States	Cities ( $n$ =number of cities)
Successor function	Action: Move to any adjacent cities Cost: distance to the adjacent cities
Start state	Arad
Goal test	Bucharest

# Problem formulation example: Pacman



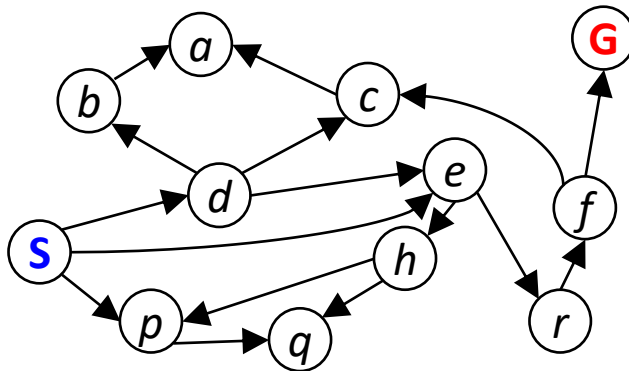
**Problem:** Find the steps for Pacman to eat all food.

Assume the ghost cannot hurt the Pacman (or it will be an Adversarial Game problem)

States	$\{(x, y), \text{food\_eaten status}\} (n=n\text{Pos} \times 2^{n\text{Food}})$
Start state	Random
Successor	Action: move N/S/E/W to a new location Cost: number of steps per movement
Goal test	Food_eaten status all <b>True</b>

# State Space Graph

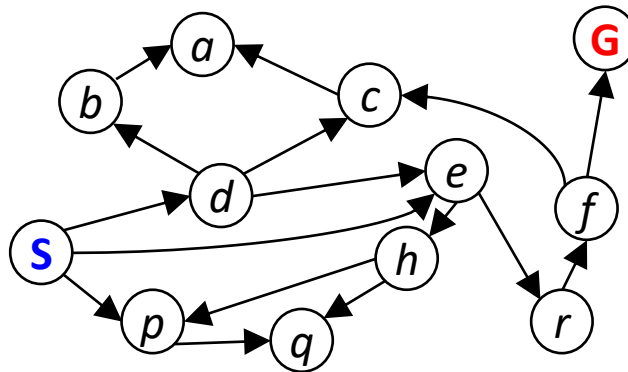
- **State space graph**: A graphical representation of a search problem that shows how states are related to each other
  - **Nodes** represents (abstracted) states
  - **Arcs** represent successors (possible actions and costs)
  - The **goal** is a set of goal nodes (maybe only one)



*Tiny state space graph for a tiny search problem*

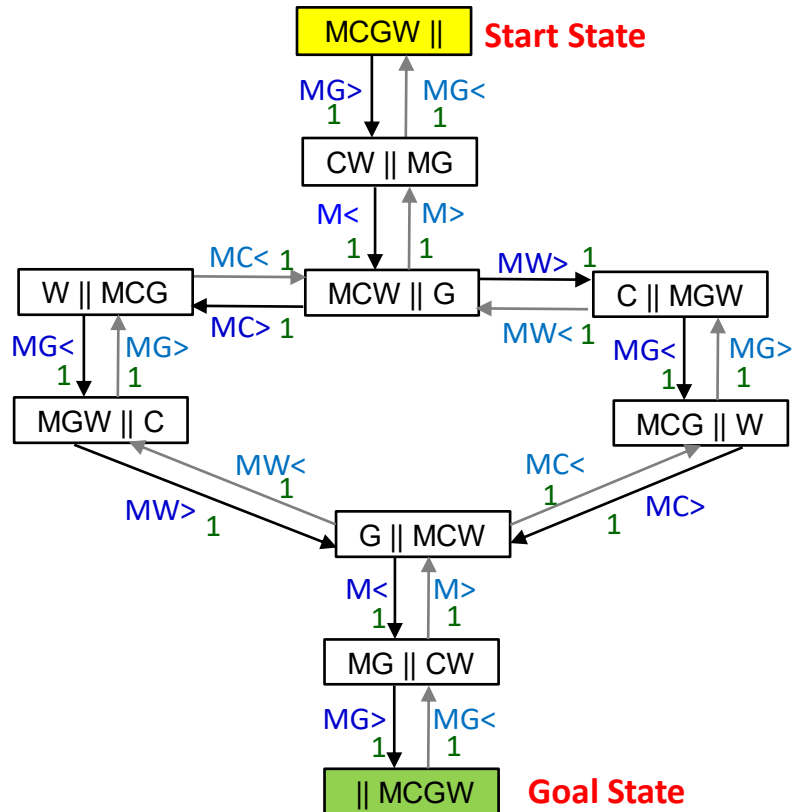
# State Space Graph (cont.)

- In a state space graph, **each state occurs only once!**
- We rarely build this full graph in memory (for most cases, it is too big), but it's a useful idea.



*Tiny state space graph for a tiny search problem*

## State space graph example: Crossing the river problem



### List of possible actions:

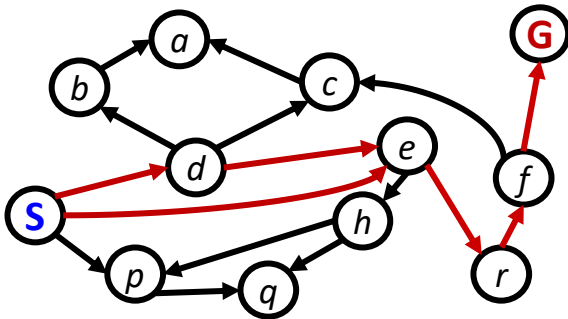
M>	man crosses alone
MC>	man crosses with cabbage
MG>	man crosses with goat
MW>	man crosses with wolf
M<	man returns alone
MC<	man returns with cabbage
MG<	man returns with goat
MW<	man returns with wolf



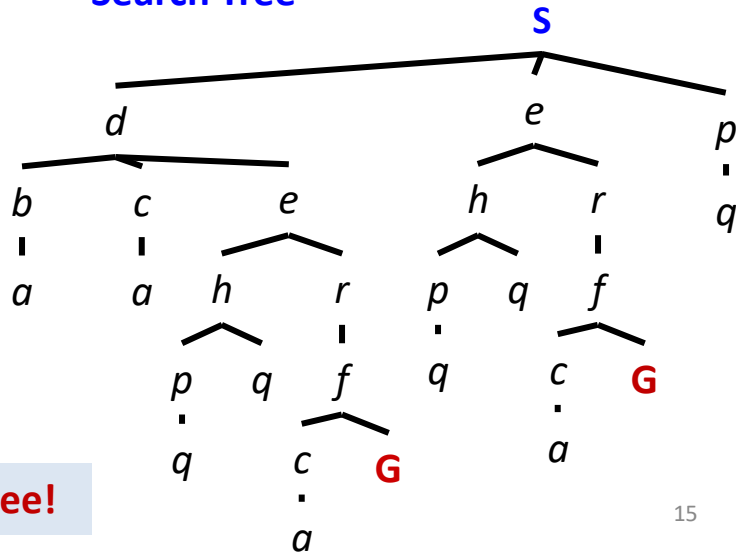
# Search Tree

- To solve a search problem, instead of building a full state space graph (too big), we construct a **search tree** to find the best route leading from the **start state** to the **goal state**.
- A search tree is constructed as we search for solution using an appropriate **search algorithm** (discuss next).

## State Space Graph



## Search Tree



**Note: repeated states in search tree!**



# Search Algorithms

---

- **General Search Framework**
  - General Tree Search
  - General Graph Search
- Uninformed Search Algorithms
  - Breadth-First Search (BFS)
  - Depth first search (DFS)
  - Uniform cost search (UCS)
- Informed Search
  - Informed Search strategy
  - A\* Search



# General Search Framework

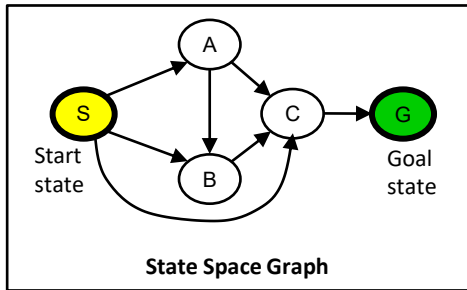
## Search strategy:

- Use an array called **frontier** to store unexplored states.
- Use an array called **explored** to store explored states if do not want to explore previously explored states

## General search framework

1. Insert **start state** into **frontier**
2. Select a **state** from **frontier**
3. Explore the **selected state**
  - Generate the **children states** of **selected state**
  - Insert all **children states** into **frontier**
4. Repeat step 2 and 3 until solution is found (success) or the frontier is empty (failure)

## Example

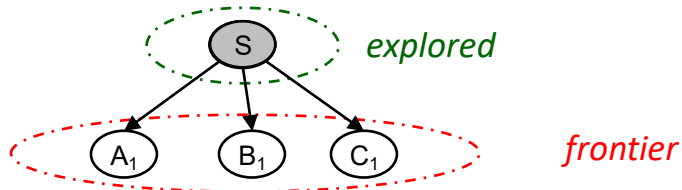


Find the way from S to G

1. First iteration:

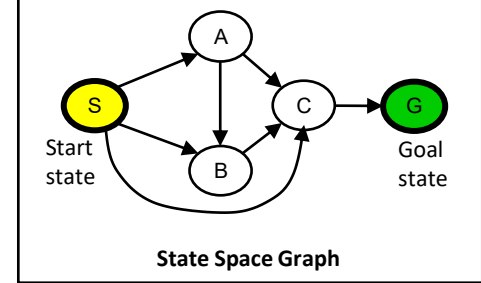
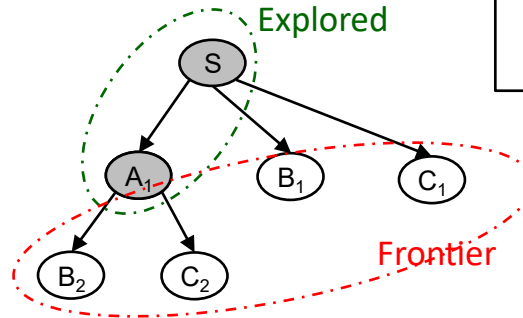


2. Second iteration:



# Searching in a Search Tree

## 3. Third iteration:



## 4. Continue until the explored state is a goal state, or failure

### Notes:

- All search algorithms share this same basic framework
- The main difference between different **search strategies** (BFS or DFS or IDS or UCS) is in terms of **how to select states from the frontier** list to be explored next
- Want to expand **as few tree nodes** as possible

# General Tree Search

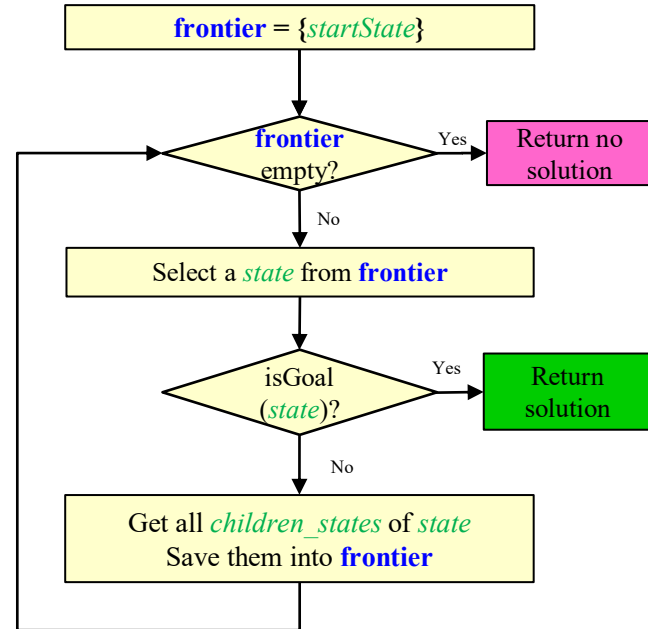
- Tree Search allows repeated state

## Problem definition

(provided by user, different for each task)

```
function TreeSearch(startState, ISGOAL, GETSUCCESSOR)
  returns SOLUTION or FAILURE
  initialize frontier with startState
  while not frontier.ISEMPTY():
    state = frontier.POP() # depends on search strategy
    if ISGOAL(state)
      return SOLUTION(state)
    for child in GETSUCCESSOR(state)
      frontier.ADD(child)
  return FAILURE
```

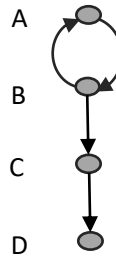
Pseudo-code for Tree Search



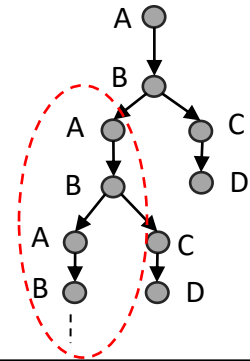
# Potential issues with Tree Search

- Loops in state space graph (repeated states) results in search tree with infinite size – some search algorithm may get lost (not complete)

State Space Graph



Tree Search



# General **Graph Search**

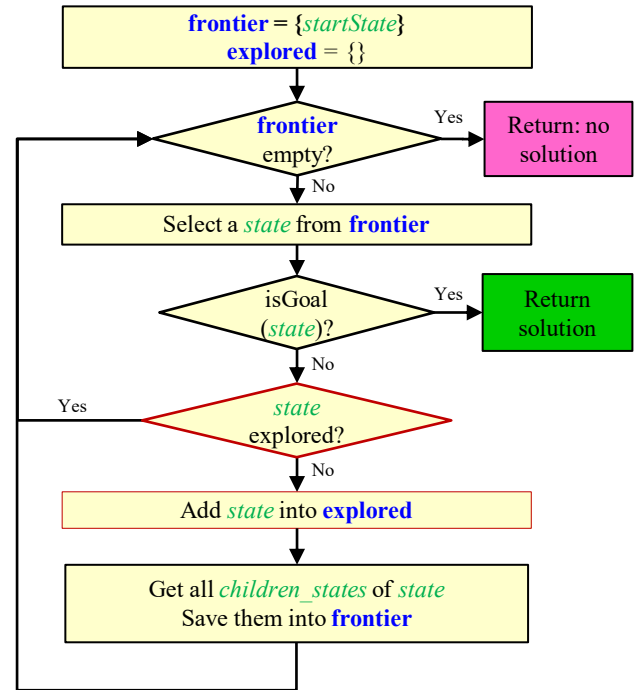
- Different from tree search, **graph search** does not explore states that have already been explored previously
- Use the array *explored* to keep track of states that have been explored
- Similarly expand the search tree layer by layer
- But only explore states (from *frontier*) that have not been explored (or found in *explored*)
- *explored* should be implemented using a **set**(items are unique), not a **list**(items can repeat)

# General Graph Search

- Graph search ignore states that **have been explored before**

## Pseudo-code for Graph Search

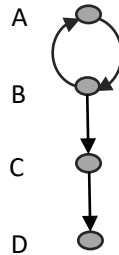
```
function GraphSearch (startState, ISGOAL,  
GETSUCCESSOR)  
returns SOLUTION or FAILURE  
  initialize frontier with startState  
  initialize explored as empty set  
  
  while not frontier.ISEMPTY():  
    state = frontier.POP()  
    if ISGOAL (state)  
      return SOLUTION (state)  
  
    if state not in explored  
      explored.ADD(state)  
      for child in GETSUCCESSOR (state)  
        frontier.ADD(child)  
  
  return FAILURE
```



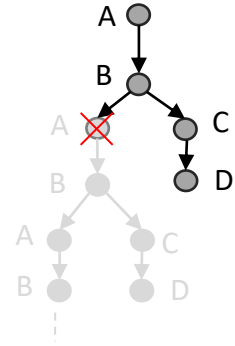
# General Graph Search

- No infinite path caused by loops in search tree

State Space Graph



Graph Search



Graph search requires more memory!





# Search Algorithms

---

- General Search Framework
  - General Tree Search
  - General Graph Search
- **Uninformed Search Algorithms**
  - **Breadth-First Search (BFS)**
  - Depth first search (DFS)
  - Uniform cost search (UCS)
- Informed Search
  - Informed Search strategy
  - A\* Search

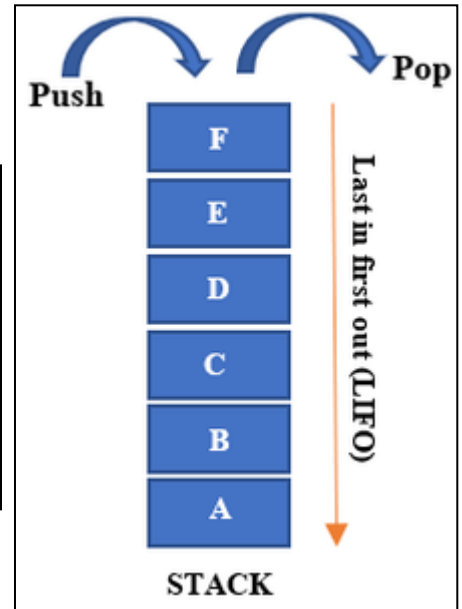
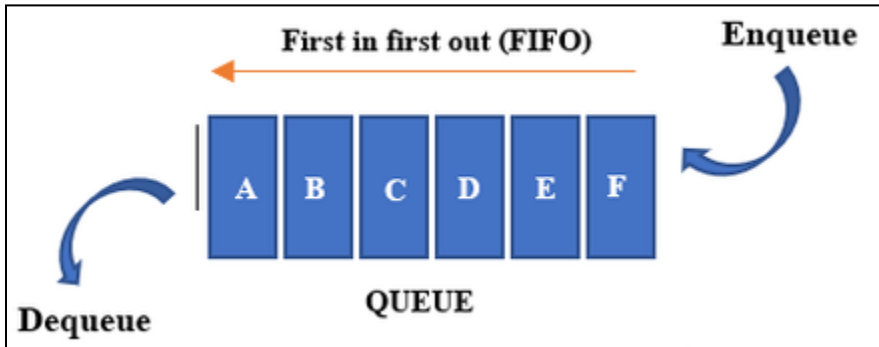
# Breadth-first search (BFS)

- Breadth-First Search explores the **shallowest** nodes in the frontier first
- Nodes are explored **level by level**
- In practice, the **shallowest** node is selected by processing the frontier list using a **first-in-first-out (FIFO)** strategy
- To implement the FIFO strategy, the *frontier* is implemented using a **Queue**
- Stops when the goal node is detected (successful) *or* no more node to expand (failure)

Cost is not considered in BFS

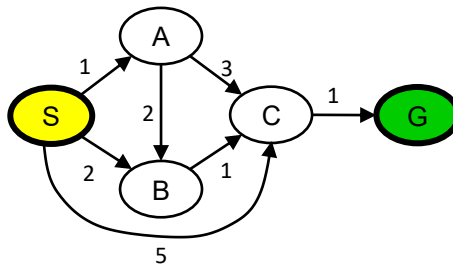


# Queue vs Stack



# BFS Tree Search Example

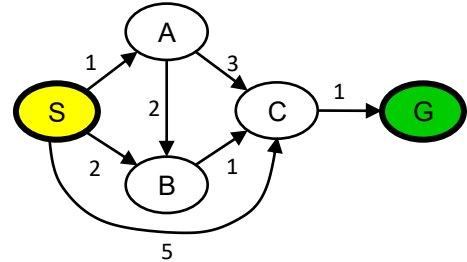
Use **BFS Tree Search** to find your way from S to G. When there are several options, select based on alphabetical order.



**Notes:**

Optimal path is  $S \rightarrow B \rightarrow C \rightarrow G$   
Cost = 4

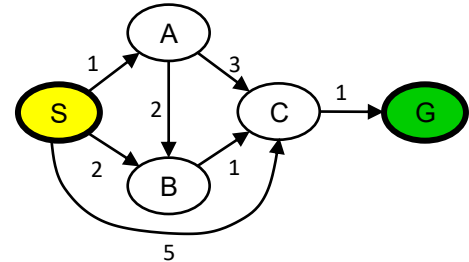
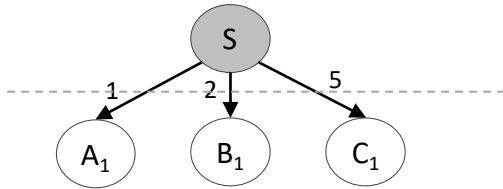
1. Find the solution path and its cost found by **BFS Tree Search**
2. Is the solution optimal?
3. How many states are explored by BFS Tree Search?
4. What is the memory used (max size of frontier) ?



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

- Insert start state S into frontier

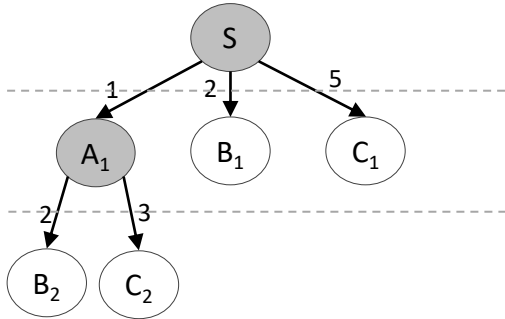
(front)	Frontier	(back)	Selected Node
	S		-



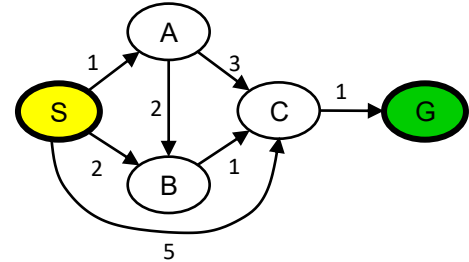
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

- Selects state S from frontier
- Three children states (A, B, C) generated and inserted into frontier

(front)	Frontier	(back)	Selected Node
	S		-
	A <sub>1</sub> , B <sub>1</sub> , C <sub>1</sub> ←		S

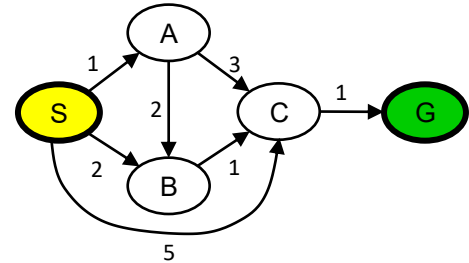
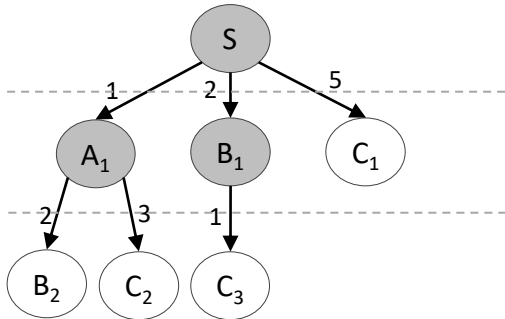


- Selects  $A_1$  (**First in First Out, FIFO**)
- B and C is already in the frontier but repeated states are allowed in Tree Search.



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

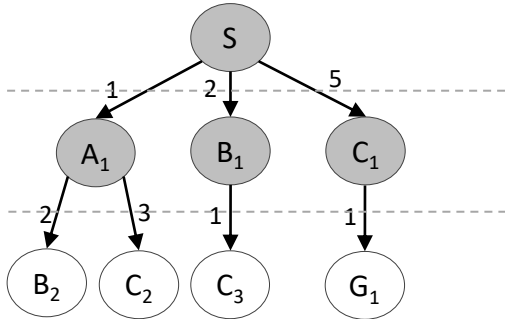
(front)	Frontier	(back)	Selected Node
	<b>S</b>		-
	<b><math>A_1, B_1, C_1</math></b>		S
	$B_1, C_1, \mathbf{B_2, C_2}$ ←		$A_1$



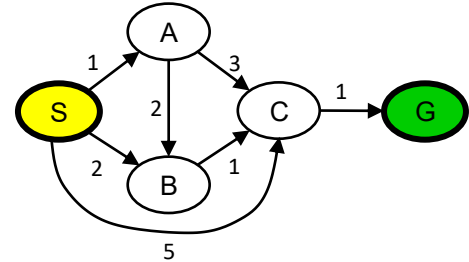
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>



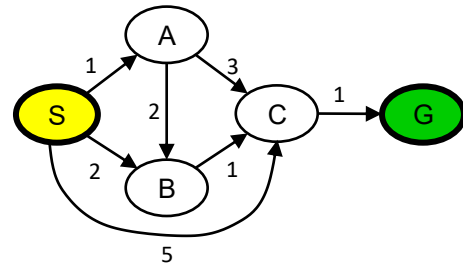
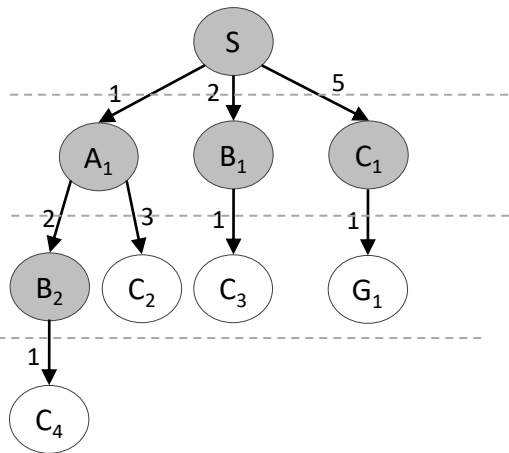


- Goal state generated  
(For general tree search,  
goal states are checked  
when a node is explored,  
not when generated)



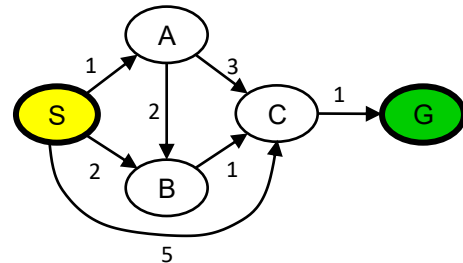
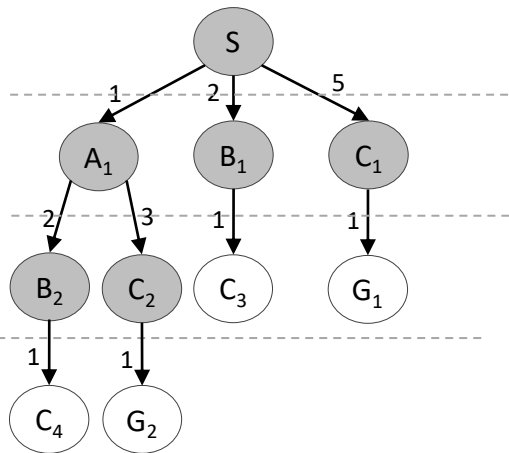
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G<sub>1</sub></b>			C <sub>1</sub>



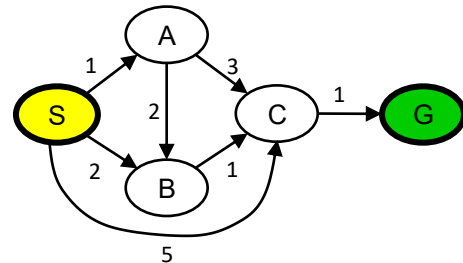
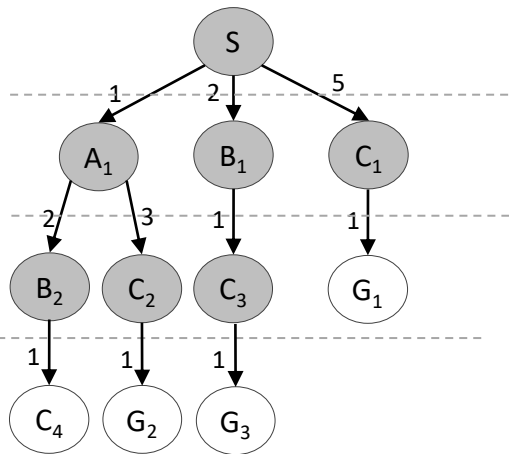
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G<sub>1</sub></b>			C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G <sub>1</sub> , <b>C<sub>4</sub></b>			B <sub>2</sub>



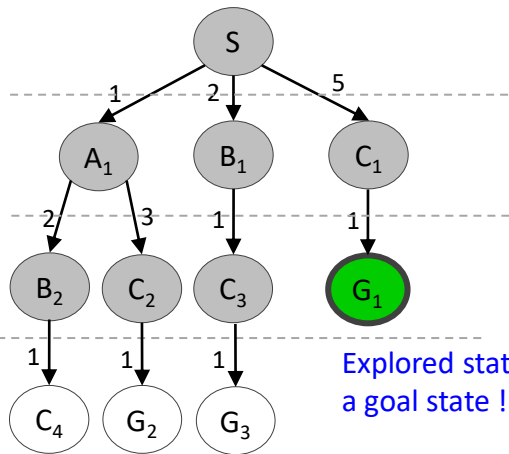
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G<sub>1</sub></b>			C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G <sub>1</sub> , <b>C<sub>4</sub></b>			B <sub>2</sub>
C <sub>3</sub> , G <sub>1</sub> , C <sub>4</sub> , <b>G<sub>2</sub></b>			C <sub>2</sub>



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G<sub>1</sub></b>			C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G <sub>1</sub> , <b>C<sub>4</sub></b>			B <sub>2</sub>
C <sub>3</sub> , G <sub>1</sub> , C <sub>4</sub> , <b>G<sub>2</sub></b>			C <sub>2</sub>
G <sub>1</sub> , C <sub>4</sub> , G <sub>2</sub> , <b>G<sub>3</sub></b>			C <sub>3</sub>



Explored state is a goal state !!

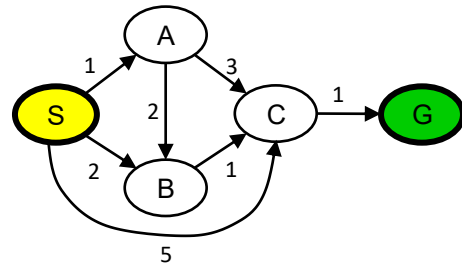
Solution Path: **S → C → G**

Cost: **6**

Optimal: **No**

Nodes expanded: **8**

Memory (Frontier): **4**

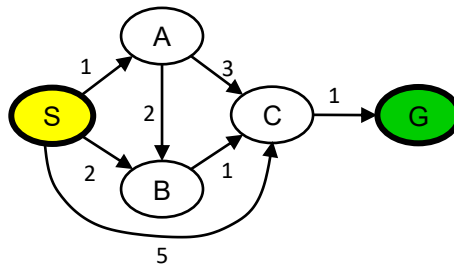


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>			A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>			B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G<sub>1</sub></b>			C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G <sub>1</sub> , <b>C<sub>4</sub></b>			B <sub>2</sub>
C <sub>3</sub> , G <sub>1</sub> , C <sub>4</sub> , <b>G<sub>2</sub></b>			C <sub>2</sub>
G <sub>1</sub> , C <sub>4</sub> , G <sub>2</sub> , <b>G<sub>3</sub></b>			C <sub>3</sub>
C <sub>4</sub> , G <sub>2</sub> , G <sub>3</sub>			G <sub>1</sub> (Goal)

# BFS Graph Search Example

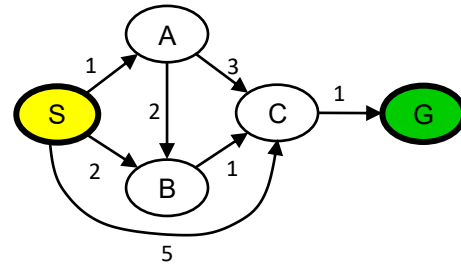
Use **BFS Graph Search** to find your way from S to G. When there are several options, select based on alphabetical order.



**Notes:**

Optimal path is  $S \rightarrow B \rightarrow C \rightarrow G$   
Cost = 4

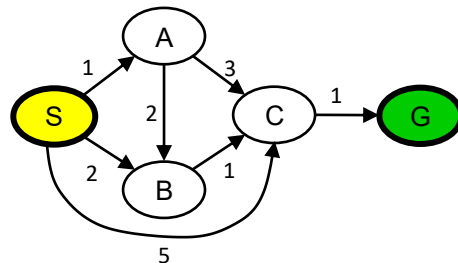
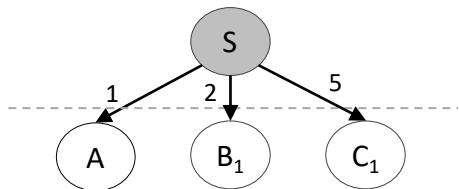
1. Find the solution path and its cost found by **BFS Graph Search**
2. Is the solution optimal?
3. How many nodes are expanded?
4. What is the memory used (frontier + explored) ?



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

- For graph search, use *explored* to keep track of states that have been explored

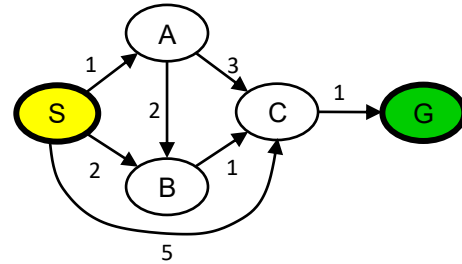
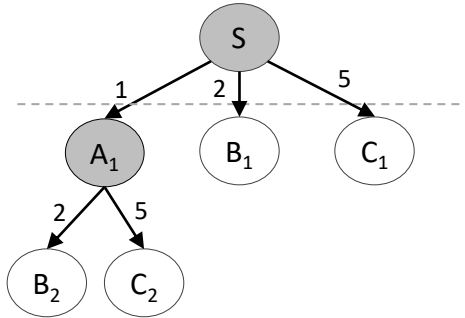
(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

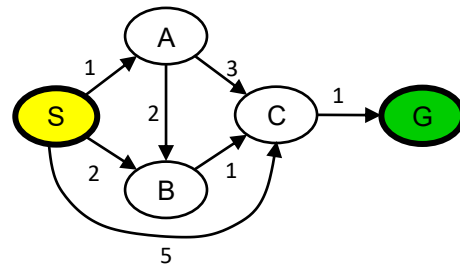
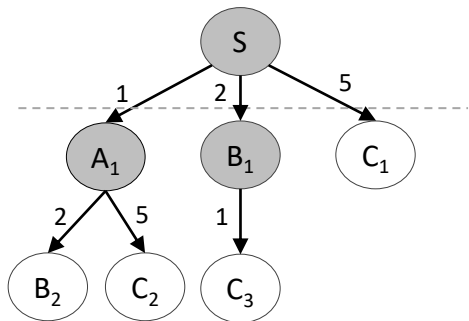
(front)	Frontier	(back)	Explored	Selected Node
	<b>S</b>			-
	<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S





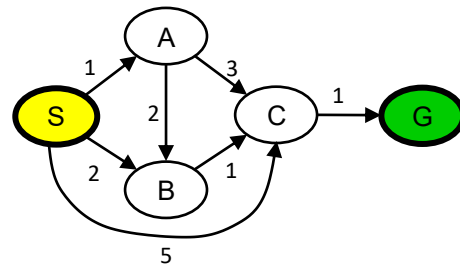
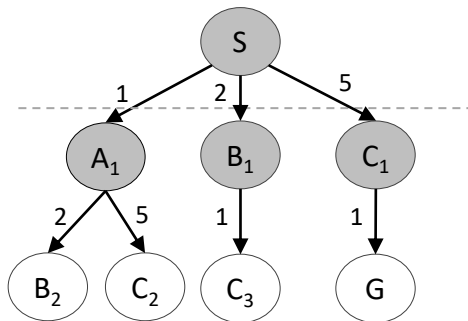
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S		S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A		A <sub>1</sub>



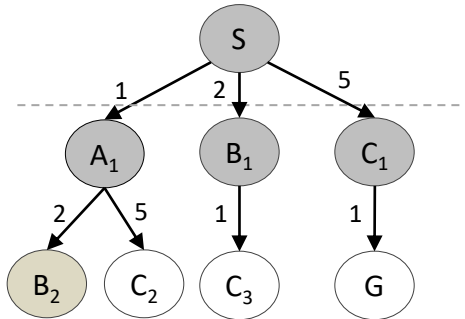
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S	S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A	A <sub>1</sub>	A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B	B <sub>1</sub>	B <sub>1</sub>

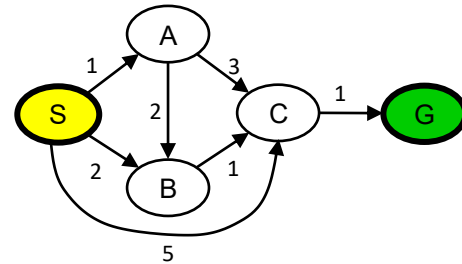


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S		S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A		A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B		B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G</b>		S, A, B, C		C <sub>1</sub>

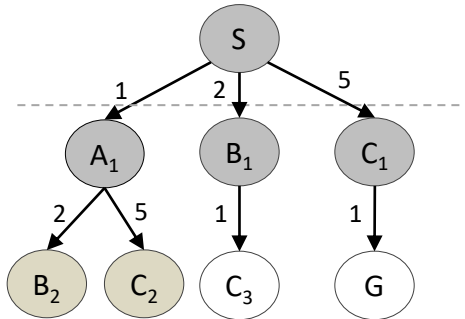


State B is in **Explored** (has been explored before) and therefore will not be explored again

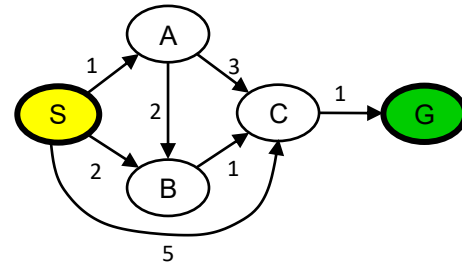


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S	S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A	S, A	A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B	S, A, B	B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G</b>		S, A, B, C	S, A, B, C	C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>B<sub>2</sub></del>

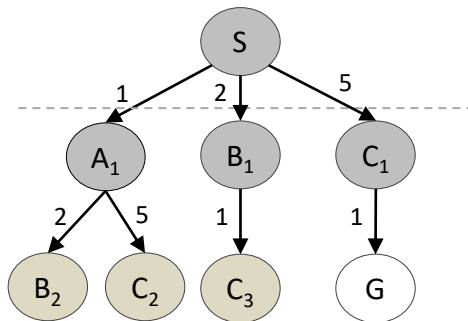


State C is in **Explored** (has been explored before) and therefore will not be explored again

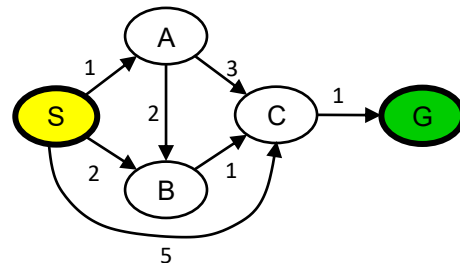


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S	S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A	S, A	A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B	S, A, B	B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G</b>		S, A, B, C	S, A, B, C	C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>B<sub>2</sub></del>
C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>C<sub>2</sub></del>

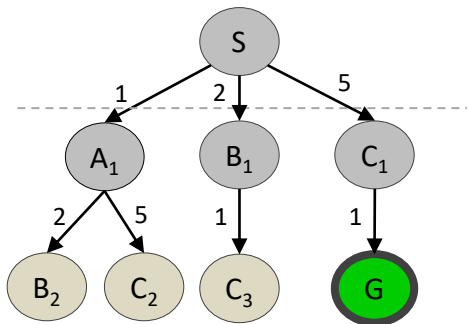


State C is in **Explored** (has been explored before) and therefore will not be explored again

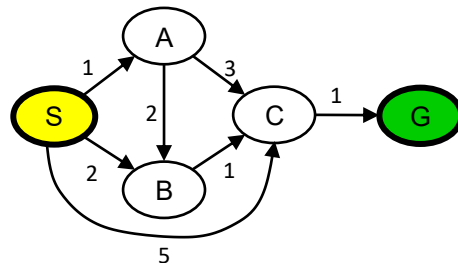


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S	S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A	S, A	A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B	S, A, B	B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G</b>		S, A, B, C	S, A, B, C	C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>B<sub>2</sub></del>
C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>C<sub>2</sub></del>
G		S, A, B, C	S, A, B, C	<del>C<sub>3</sub></del>

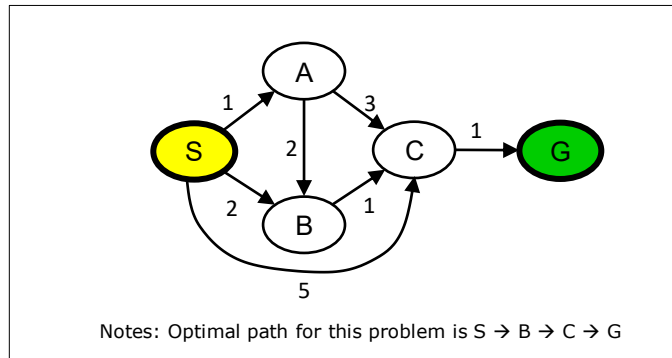
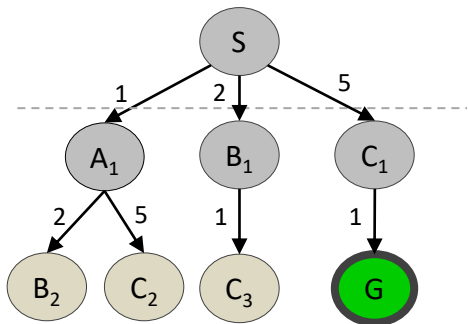


Expanded node is a goal state! Solution found.



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S	S
B <sub>1</sub> , C <sub>1</sub> , <b>B<sub>2</sub>, C<sub>2</sub></b>		S, A	S, A	A <sub>1</sub>
C <sub>1</sub> , B <sub>2</sub> , C <sub>2</sub> , <b>C<sub>3</sub></b>		S, A, B	S, A, B	B <sub>1</sub>
B <sub>2</sub> , C <sub>2</sub> , C <sub>3</sub> , <b>G</b>		S, A, B, C	S, A, B, C	C <sub>1</sub>
C <sub>2</sub> , C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>B<sub>2</sub></del>
C <sub>3</sub> , G		S, A, B, C	S, A, B, C	<del>C<sub>2</sub></del>
G		S, A, B, C	S, A, B, C	<del>C<sub>3</sub></del>
-		S, A, B, C	S, A, B, C	G



Solution Path:  **$S \rightarrow C \rightarrow G$**

Cost: **6**

Optimal: **No**

Nodes explored: **5**

Memory (frontier +  
explored):  $4+4=$ **8**

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>				-
<b><math>A_1, B_1, C_1</math></b>		S	S	S
$B_1, C_1, $ <b><math>B_2, C_2</math></b>		S, A	S, A	$A_1$
$C_1, B_2, C_2, $ <b><math>C_3</math></b>		S, A, B	S, A, B	$B_1$
$B_2, C_2, C_3, $ <b>G</b>		S, A, B, C	S, A, B, C	$C_1$
$C_2, C_3, G$		S, A, B, C	S, A, B, C	<del><math>B_2</math></del>
$C_3, G$		S, A, B, C	S, A, B, C	<del><math>C_2</math></del>
G		S, A, B, C	S, A, B, C	<del><math>C_3</math></del>
-		S, A, B, C	S, A, B, C	G





# Search Algorithms

---

- General Search Framework
  - General Tree Search
  - General Graph Search
- Uninformed Search Algorithms
  - Breadth-First Search (BFS)
  - **Depth first search (DFS)**
  - Uniform cost search (UCS)
- Informed Search
  - Informed Search strategy
  - A\* Search

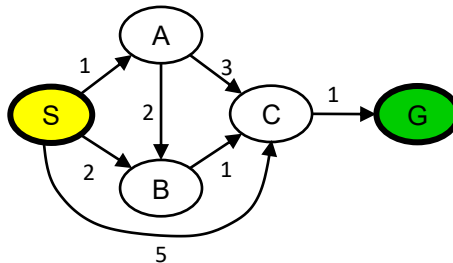
# Depth First Search (DFS)

- DFS explores the **deepest** node in the frontier first
- As a consequence, search proceeds immediately to the leaf nodes before turning back in search of other branches
- In practice, the **deepest** node can be selected by processing the nodes in the frontier list using a **last-in-first-out (LIFO)** strategy
- To implement the LIFO strategy, the *frontier* is implemented using a **stack**



# DFS Tree Search: Example

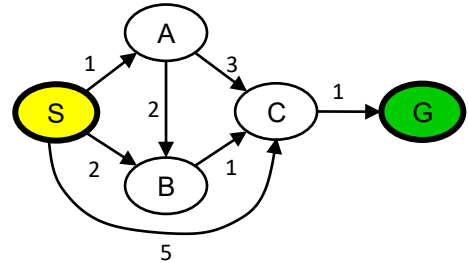
Use **DFS Tree Search** to find your way from S to G. When there are several options, select based on alphabetical order.



**Notes:**

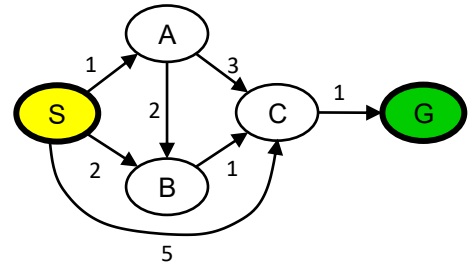
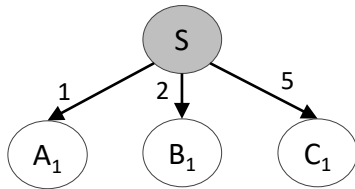
Optimal path is  $S \rightarrow B \rightarrow C \rightarrow G$   
Cost = 4

1. Find the solution path and its cost found by DFS Tree Search
2. Is the solution optimal?
3. How many nodes are expanded?
4. What is the memory used (frontier) ?



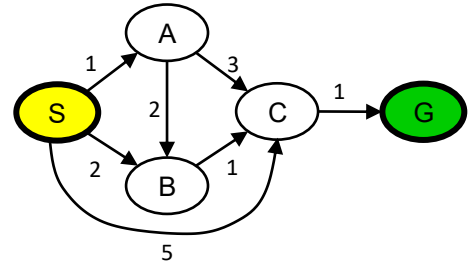
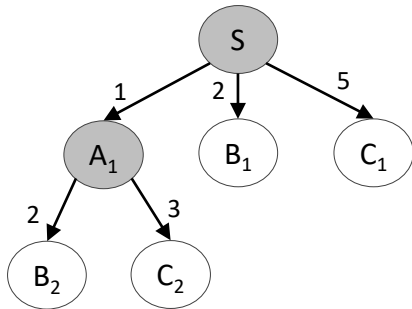
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

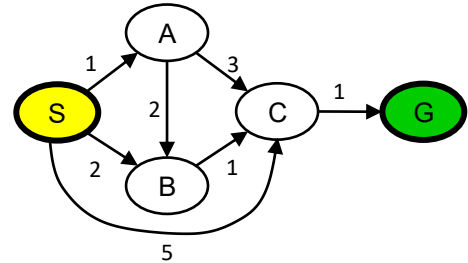
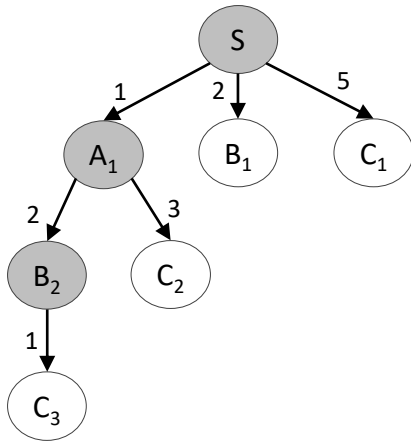
(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

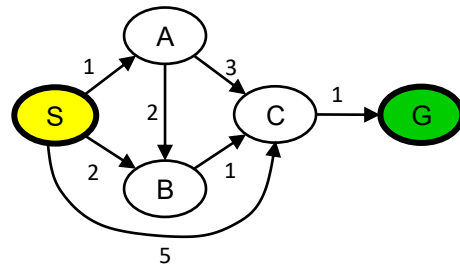
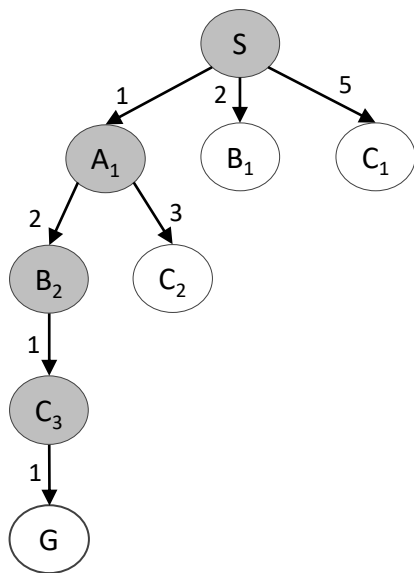
- Nodes in the frontier list is **last-in-first-out (LIFO)**

(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
<b>B<sub>2</sub>, C<sub>2</sub></b> , B <sub>1</sub> , C <sub>1</sub>			A <sub>1</sub>



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

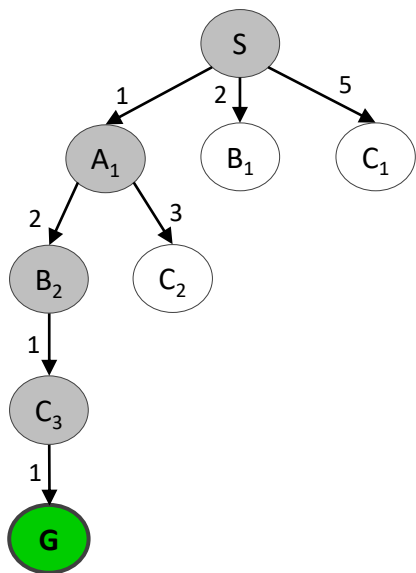
(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
<b>B<sub>2</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			A <sub>1</sub>
<b>C<sub>3</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			B <sub>2</sub>



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
<b>B<sub>2</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			A <sub>1</sub>
<b>C<sub>3</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			B <sub>2</sub>
<b>G, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			C <sub>3</sub>





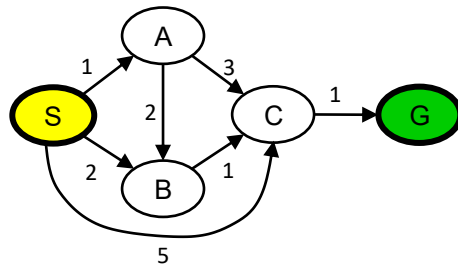
Solution Path: **S → A → B → C → G**

Cost: **5**

Optimal: **No**

Nodes explored: **5**

Memory required (frontier): **4**

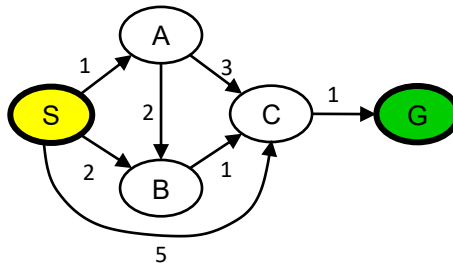


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(Top)	Frontier	(Bottom)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S
<b>B<sub>2</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			A <sub>1</sub>
<b>C<sub>3</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			B <sub>2</sub>
<b>G, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			C <sub>3</sub>
C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			G (Goal)

# DFS Graph Search: Example

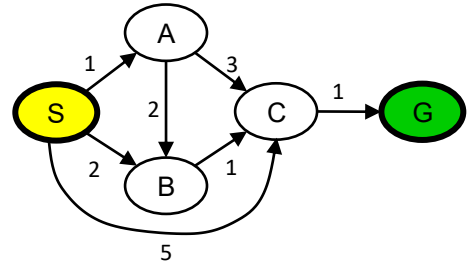
Use **DFS Graph Search** to find your way from S to G. When there are several options, select based on alphabetical order.



**Notes:**

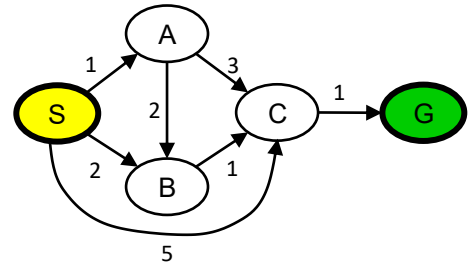
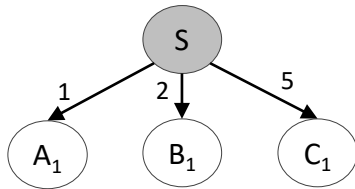
Optimal path is  $S \rightarrow B \rightarrow C \rightarrow G$   
Cost = 4

1. Find the solution path and its cost found by DFS Graph Search
2. Is the solution optimal?
3. How many nodes are expanded?
4. What is the memory used (frontier + explored)?



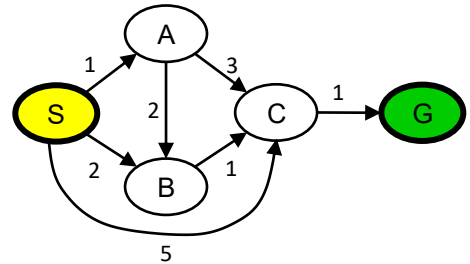
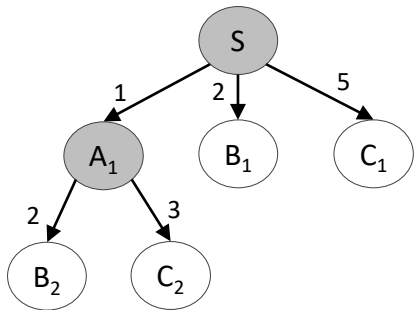
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>			-	-



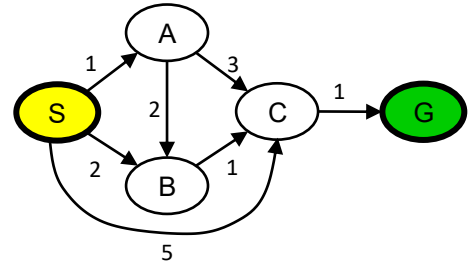
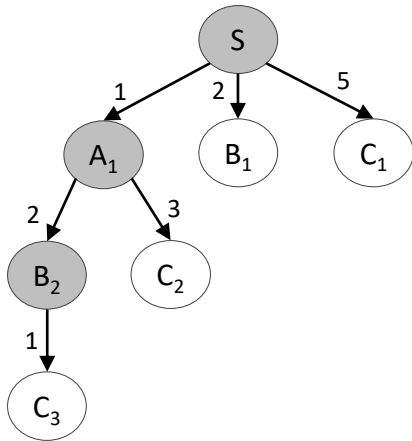
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
	<b>S</b>		-	-
	<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>		S	S



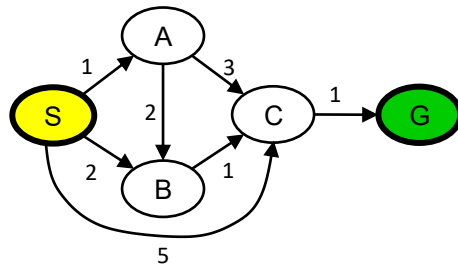
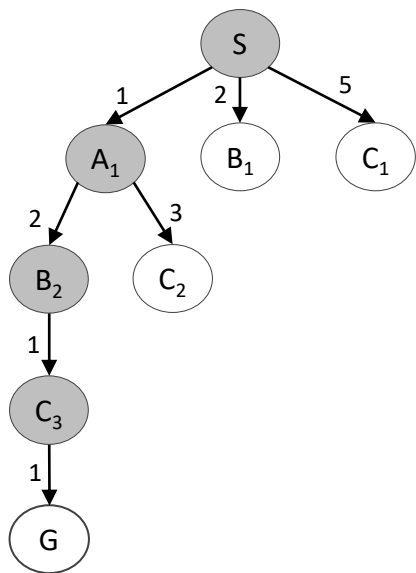
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
<b>S</b>			-	-
<b>A<sub>1</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S	S
<b>B<sub>2</sub>, C<sub>2</sub>, B<sub>1</sub>, C<sub>1</sub></b>			S, A	A <sub>1</sub>



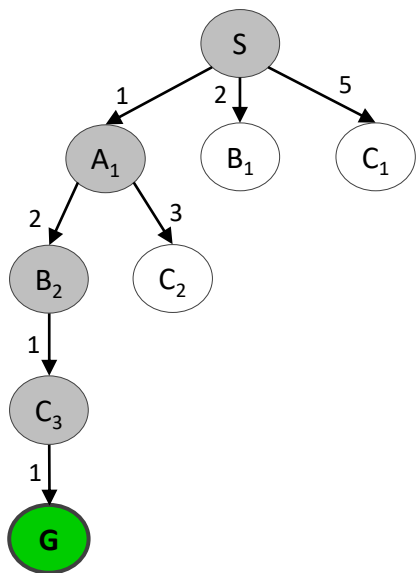
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
S			-	-
A <sub>1</sub> , B <sub>1</sub> , C <sub>1</sub>			S	S
B <sub>2</sub> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A	A <sub>1</sub>
<b>C<sub>3</sub></b> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B	B <sub>2</sub>



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
S			-	-
A <sub>1</sub> , B <sub>1</sub> , C <sub>1</sub>			S	S
B <sub>2</sub> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A	A <sub>1</sub>
C <sub>3</sub> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B	B <sub>2</sub>
<b>G</b> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B, C	C <sub>3</sub>



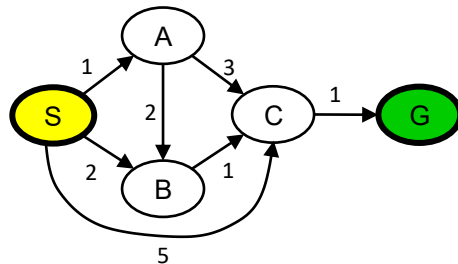
Solution Path: **S → A → B → C → G**

Cost: **5**

Optimal: **No**

Nodes explored: **5**

Memory required (frontier + explored): **8**



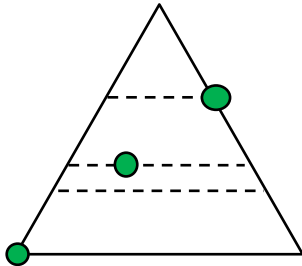
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Explored	Selected Node
S			-	-
A <sub>1</sub> , B <sub>1</sub> , C <sub>1</sub>			S	S
B <sub>2</sub> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A	A <sub>1</sub>
C <sub>3</sub> , C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B	B <sub>2</sub>
G, C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B, C	C <sub>3</sub>
C <sub>2</sub> , B <sub>1</sub> , C <sub>1</sub>			S, A, B, C	G (Goal)

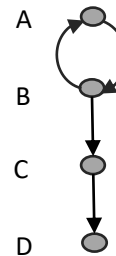


# DFS vs BFS

- When there exist shallow solutions, BFS is a better solution.
- If memory is a concern, DFS is better since it requires less storage in the Frontier.
- DFS might stuck in infinite loop if there are loops in the state space. Therefore, if completeness (guarantee that a solution exists) is important, use BFS or try graph search.



State Space Graph





# Search Algorithms

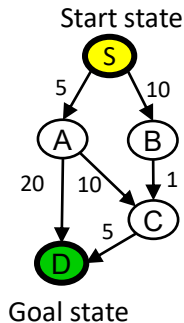
---

- General Search Framework
  - General Tree Search
  - General Graph Search
- Uninformed Search Algorithms
  - Breadth-First Search (BFS)
  - Depth first search (DFS)
  - **Uniform cost search (UCS)**
- Informed Search
  - Informed Search strategy
  - A\* Search

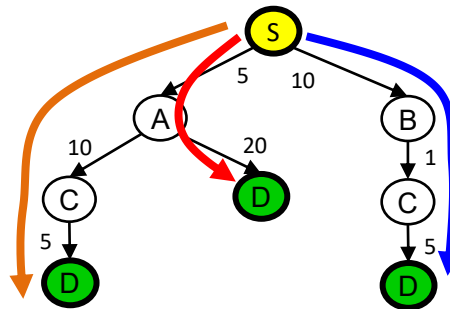
# Optimal solution

- Both BFS and DFS do not consider path cost
- BFS finds the *shallowest* solution. The shallowest solution is not necessarily the *optimal* solution
- DFS finds the *left-most* or *right-most* solution. The left-most or right-most solution is not necessarily the *optimal* solution

State space graph



Search Tree



Solution by DFS if  
start at left side of  
tree. Cost = 20 (not  
optimal)

Solution by BFS.  
Cost = 25 (not  
optimal)

Optimal solution.  
Cost = 16

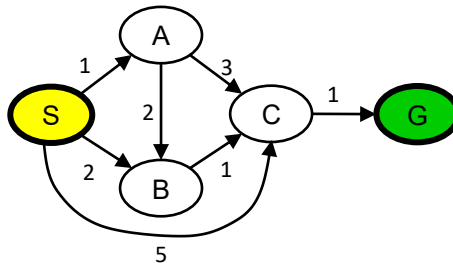
# Uniform Cost Search (UCS)

- Selects the frontier node **with the lowest path cost**
- The frontier is implemented using a **priority queue** that sorts the states based on **ascending** cost



# UCS Tree Search Example

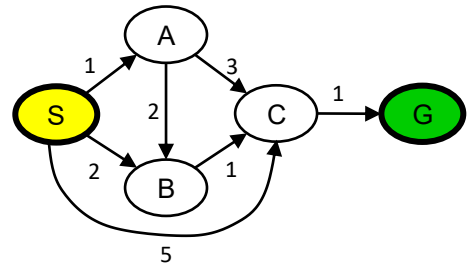
Use **UCS Tree Search** to find your way from S to G. When there are several options, select based on alphabetical order.



**Notes:**

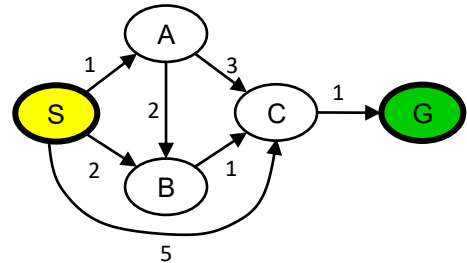
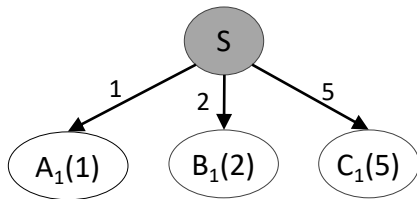
Optimal path is  $S \rightarrow B \rightarrow C \rightarrow G$   
Cost = 4

1. Find the solution path and its cost found by UCS Tree Search.
2. Is the solution optimal?
3. How many nodes are expanded?
4. What is the memory used (frontier) ?



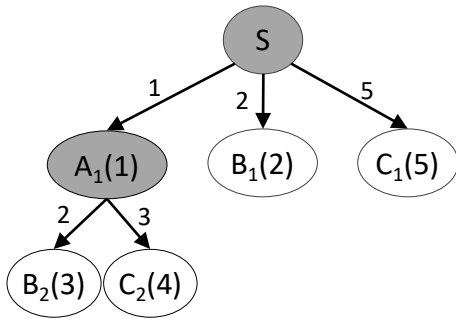
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
S			-

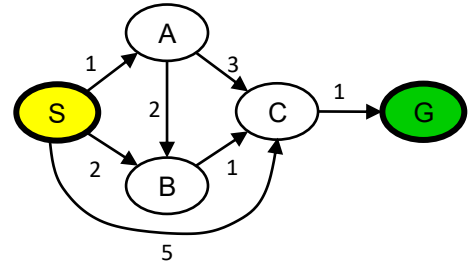


Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
	S		-
	A <sub>1</sub> (1), B <sub>1</sub> (2), C <sub>1</sub> (5)		S



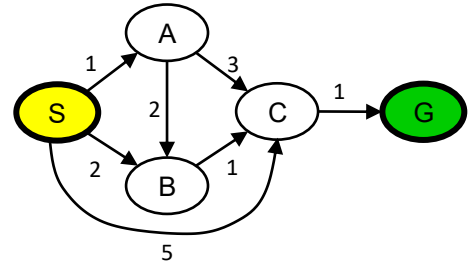
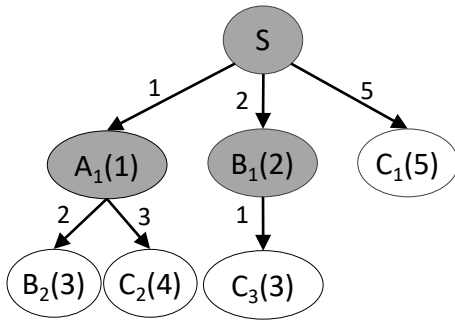
- Nodes in the frontier list is sorted based on action cost using **priority queue**



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

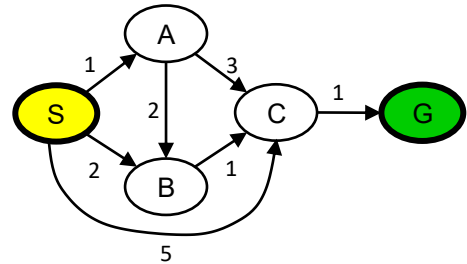
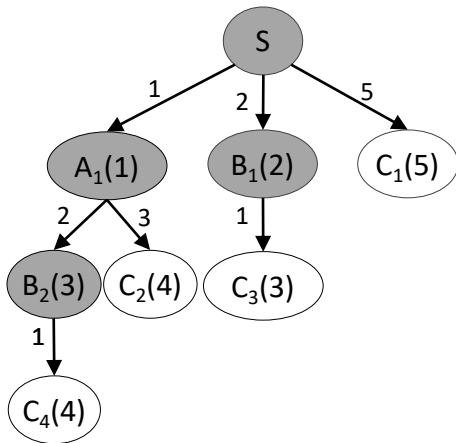
(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>





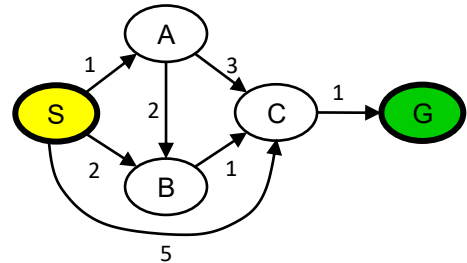
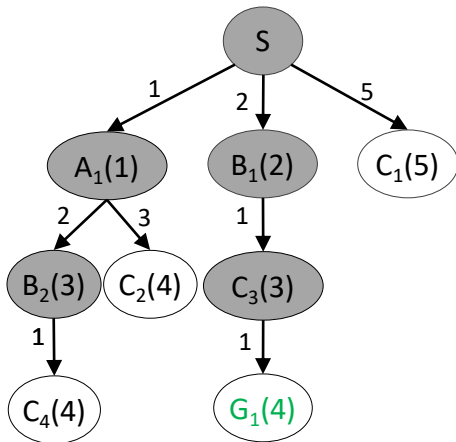
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)	B <sub>1</sub>



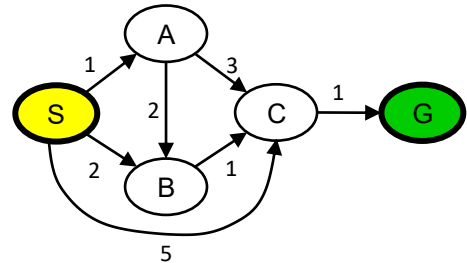
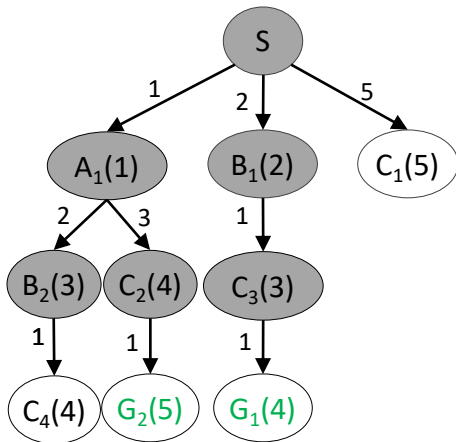
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)	B <sub>1</sub>
C <sub>3</sub> (3), C <sub>2</sub> (4), <b>C<sub>4</sub>(4)</b> , C <sub>1</sub> (5)	B <sub>2</sub>



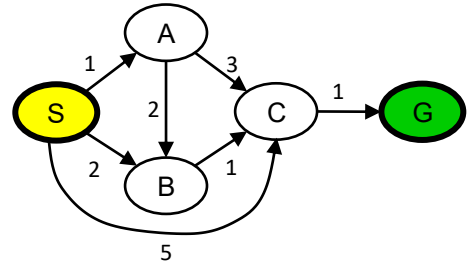
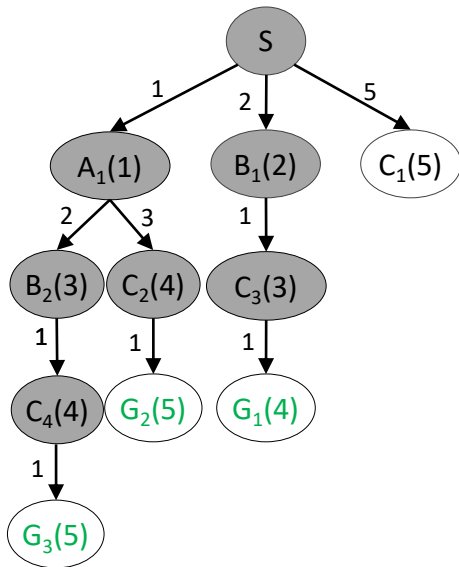
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)	B <sub>1</sub>
C <sub>3</sub> (3), C <sub>2</sub> (4), <b>C<sub>4</sub>(4)</b> , C <sub>1</sub> (5)	B <sub>2</sub>
C <sub>2</sub> (4), C <sub>4</sub> (4), <b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5)	C <sub>3</sub>



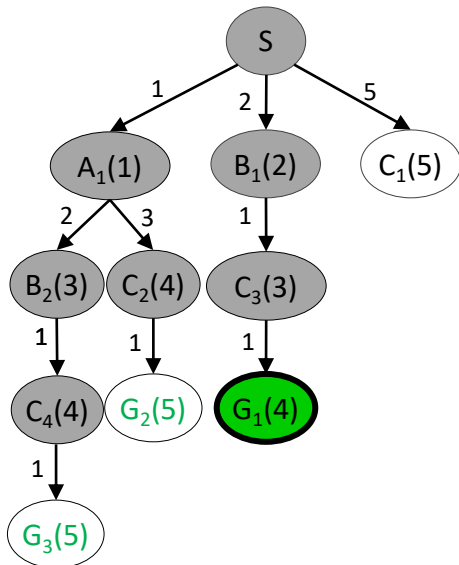
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)	B <sub>1</sub>
C <sub>3</sub> (3), C <sub>2</sub> (4), <b>C<sub>4</sub>(4)</b> , C <sub>1</sub> (5)	B <sub>2</sub>
C <sub>2</sub> (4), C <sub>4</sub> (4), <b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5)	C <sub>3</sub>
C <sub>4</sub> (4), G <sub>1</sub> (4), C <sub>1</sub> (5), <b>G<sub>2</sub>(5)</b>	C <sub>2</sub>



Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front) <b>Frontier</b> (back)	Selected Node
<b>S</b>	-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>	S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)	A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)	B <sub>1</sub>
C <sub>3</sub> (3), C <sub>2</sub> (4), <b>C<sub>4</sub>(4)</b> , C <sub>1</sub> (5)	B <sub>2</sub>
C <sub>2</sub> (4), C <sub>4</sub> (4), <b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5)	C <sub>3</sub>
C <sub>4</sub> (4), G <sub>1</sub> (4), C <sub>1</sub> (5), <b>G<sub>2</sub>(5)</b>	C <sub>2</sub>
G <sub>1</sub> (4), C <sub>1</sub> (5), G <sub>2</sub> (5), <b>G<sub>3</sub>(5)</b>	C <sub>4</sub>



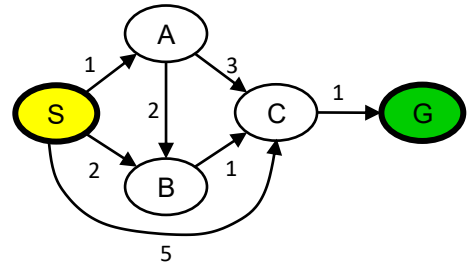
Solution Path: **S → B → C → G**

Cost: **4**

Optimal: **Yes**

Nodes explored: **8**

Memory required (frontier): **4**



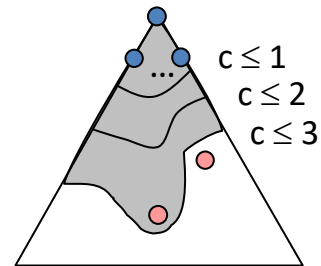
Notes: Optimal path for this problem is  $S \rightarrow B \rightarrow C \rightarrow G$

(front)	Frontier	(back)	Selected Node
<b>S</b>			-
<b>A<sub>1</sub>(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>			S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)			A <sub>1</sub>
B <sub>2</sub> (3), <b>C<sub>3</sub>(3)</b> , C <sub>2</sub> (4), C <sub>1</sub> (5)			B <sub>1</sub>
C <sub>3</sub> (3), C <sub>2</sub> (4), <b>C<sub>4</sub>(4)</b> , C <sub>1</sub> (5)			B <sub>2</sub>
C <sub>2</sub> (4), C <sub>4</sub> (4), <b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5)			C <sub>3</sub>
C <sub>4</sub> (4), <b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5), <b>G<sub>2</sub>(5)</b>			C <sub>2</sub>
<b>G<sub>1</sub>(4)</b> , C <sub>1</sub> (5), G <sub>2</sub> (5), <b>G<sub>3</sub>(5)</b>			C <sub>4</sub>
C <sub>1</sub> (5), G <sub>2</sub> (5), G <sub>3</sub> (5)			G <sub>1</sub>

# Uniform Cost Issues

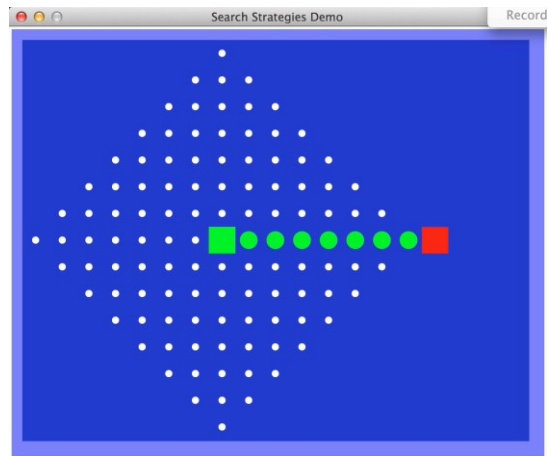
## Strength:

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal



## Weakness:

- Explores options in every “direction”
- No information about goal location, not efficient
- We’ll fix that next!





# Search Algorithms

---

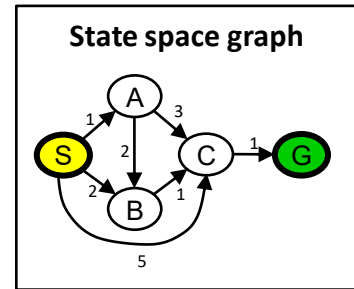
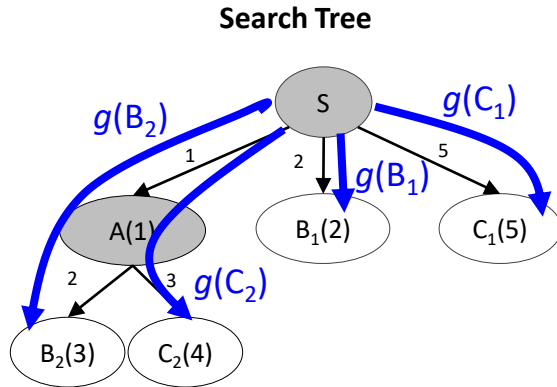
- General Search Framework
  - General Tree Search
  - General Graph Search
- Uninformed Search Algorithms
  - Breadth-First Search (BFS)
  - Depth first search (DFS)
  - Uniform cost search (UCS)
- **Informed Search**
  - Informed Search strategy
  - A\* Search



# Issues with Uniform Cost Search (UCS)

- UCS is **backward looking** - the nodes in *frontier* is processed based on  $g(n)$  i.e., its distance from the start node

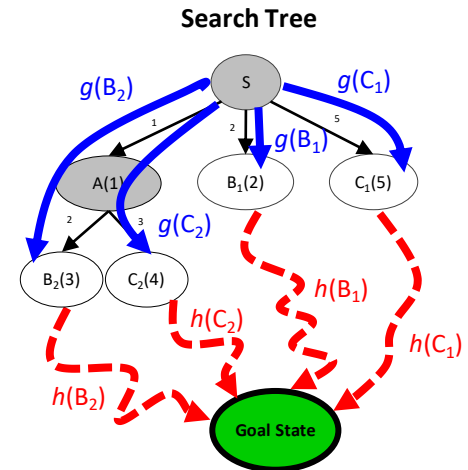
$$f(n) = g(n)$$



(front)	Frontier	(back)	Explored	Selected Node
S				-
<b>A(1), B<sub>1</sub>(2), C<sub>1</sub>(5)</b>		S		S
B <sub>1</sub> (2), <b>B<sub>2</sub>(3), C<sub>2</sub>(4)</b> , C <sub>1</sub> (5)		S, A		A
...		...		...

# Informed Search strategy

- **Informed search** algorithms uses a **heuristic function** to choose a most likely good node
- A heuristic function  $h(n)$ :
  - is a function that **estimates** how close a state is to the goal
  - makes the search algorithm **forward looking**
  - is designed for a particular search problem
  - $h(n)$  does not have to be perfect!

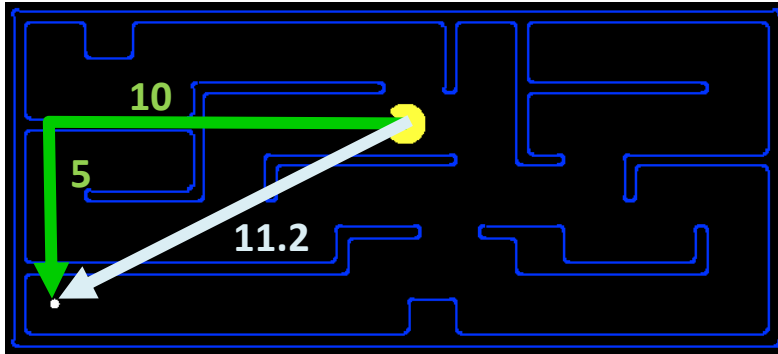


- Types of **informed search** algorithms:

**Greedy Best First Search (GBFS)** considers only  $h(n)$        $f(n) = h(n)$

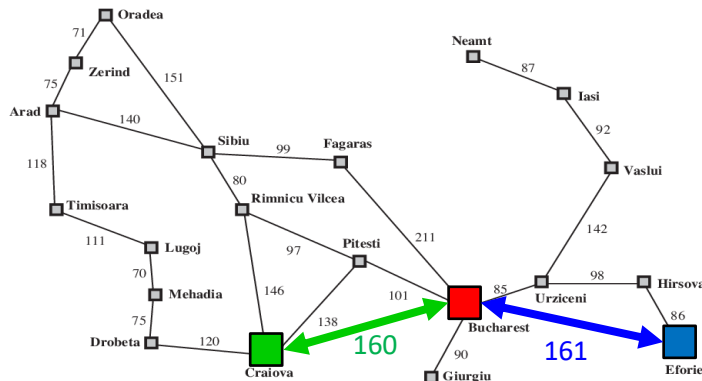
**A\* Search** considers both  $g(n)$  and  $h(n)$        $f(n) = g(n) + h(n)$

# Example heuristics



## Pacman:

Manhattan distance = 15  
Euclidean distance = 11.2



## Map:

Straight line distance to destination

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

# A\* search

- A\* Search:
  - Use a new evaluation function,  $f(n)$ , that combines the actual performance of the various path explored so far, and the estimated cost to the goal:

$$f(n) = g(n) + h(n)$$

where  $g(n)$  : Gives the real cost of the path traveled so far .

$h(n)$  : Gives the estimated cost to the goal.

- $f(n)$  is the estimate cost of the solution through node  $n$



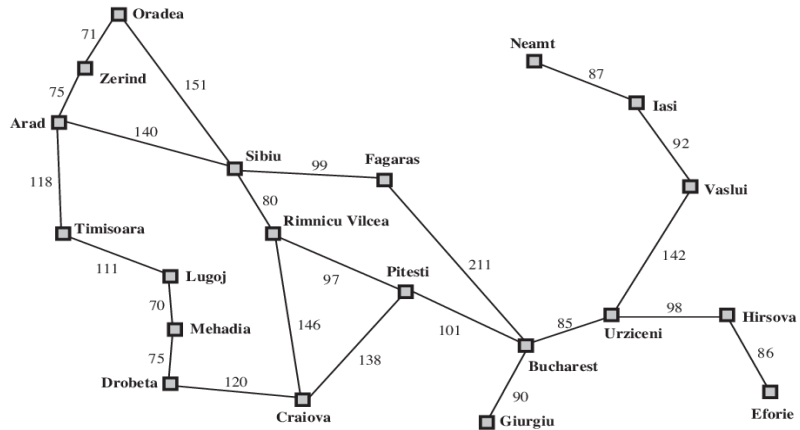
# A\* Tree Search: Example

**Objective:** Find a path from *Arad* to *Bucharest* using A\* Tree Search

**Heuristics:** Straight line distance heuristic  $h_{SLD}$  distances to goal

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure: Road map of Romania

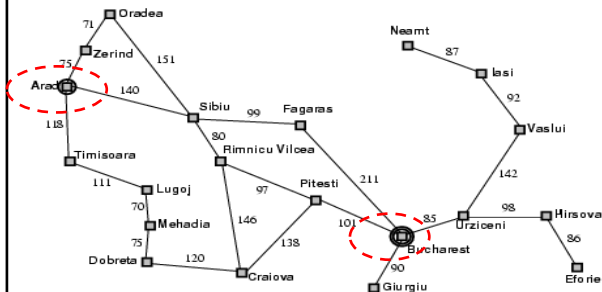


1. Find the solution path and its cost
2. Is the solution optimal?
3. How many nodes are explored?
4. What is the memory requirement (in terms of number of nodes) ?

**A**  
 $0+366 = 366$

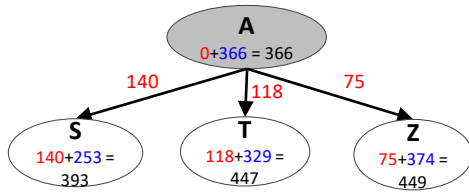
$$\begin{aligned} f(n) &= g(n) + h(n) \\ &= 0 + 366 \\ &= 366 \end{aligned}$$

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

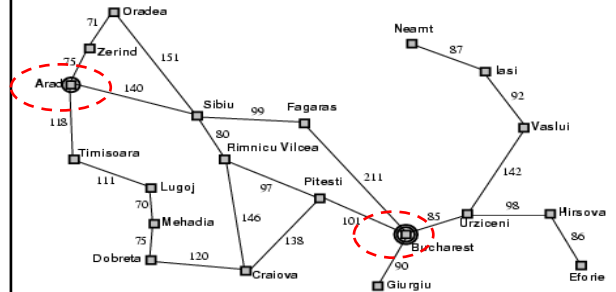


Notes: Optimal path for this problem is  $A \rightarrow S \rightarrow R \rightarrow P \rightarrow B$

[illegible]

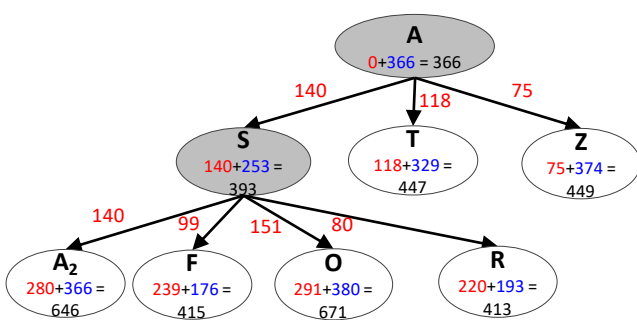


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

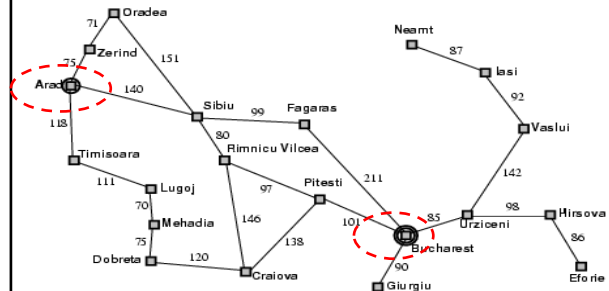


Notes: Optimal path for this problem is A → S → R → P → B

(Front)	Frontier	(Back)	Selected Node
	A(366)		-
	S(393), T(447), Z(449)		A
			87



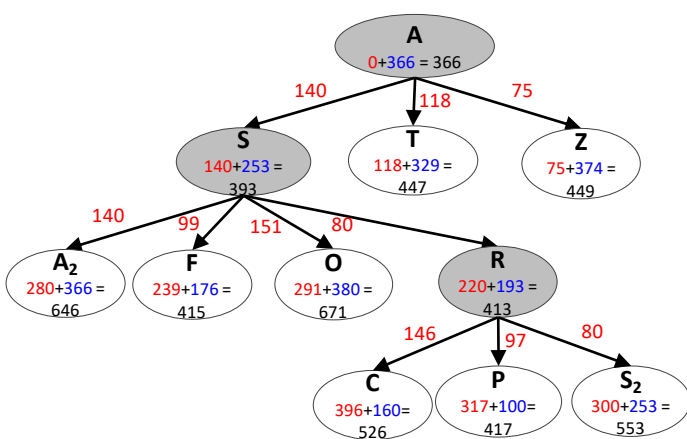
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



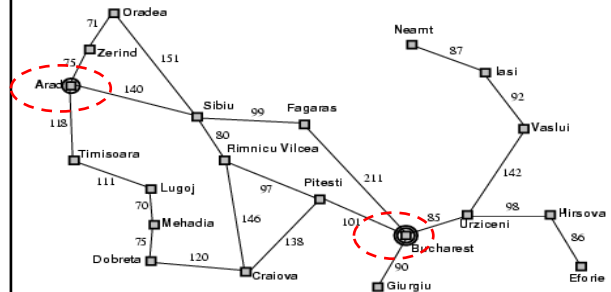
Notes: Optimal path for this problem is A → S → R → P → B

(Front)	Frontier	(Back)	Selected Node
	<b>A(366)</b>		-
	<b>S(393), T(447), Z(449)</b>		A
	<b>R(413), F(415), T(447), Z(449), A<sub>2</sub>(646), O(671),</b>		S
			88



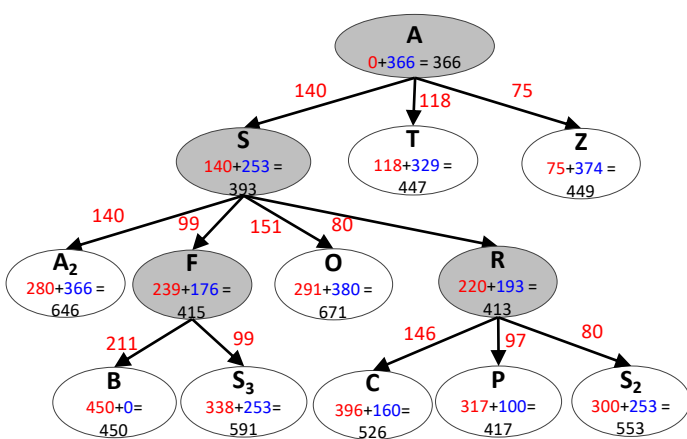


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

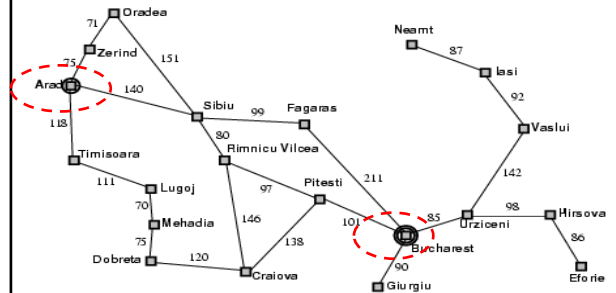


Notes: Optimal path for this problem is A → S → R → P → B

(Front)	Frontier	(Back)	Selected Node
	<b>A(366)</b>		-
	<b>S(393), T(447), Z(449)</b>		A
	<b>R(413), F(415), T(447), Z(449), A<sub>2</sub>(646), O(671),</b>		S
	<b>F(415), P(417), T(447), Z(449), C(526), S<sub>2</sub>(553), A<sub>2</sub>(646), O(671)</b>		R
			89

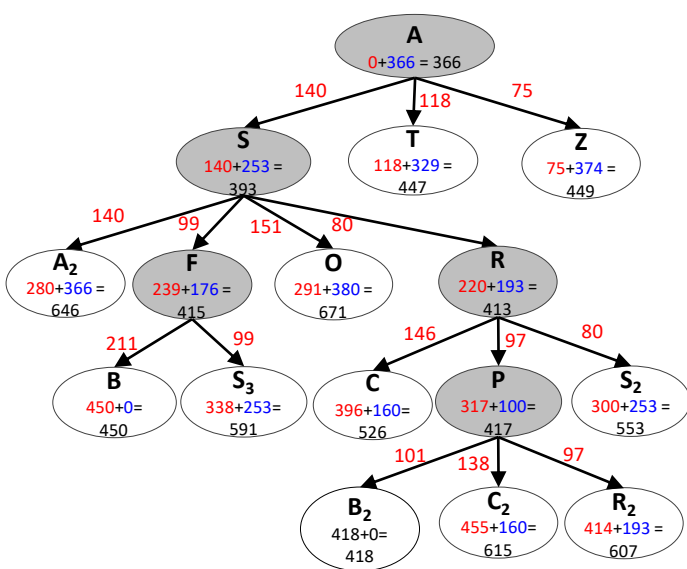


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

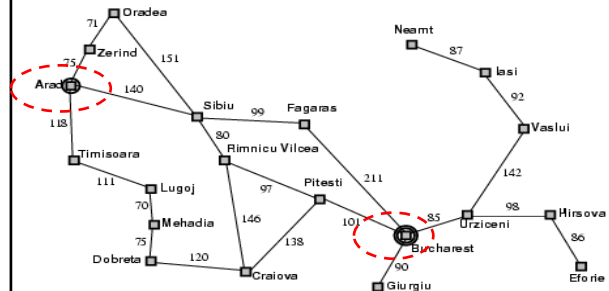


Notes: Optimal path for this problem is A → S → R → P → B

(Front)	Frontier	(Back)	Selected Node
	<b>A(366)</b>		-
	<b>S(393), T(447), Z(449)</b>		A
	<b>R(413), F(415), T(447), Z(449), A<sub>2</sub>(646), O(671),</b>		S
	<b>F(415), P(417), T(447), Z(449), C(526), S<sub>2</sub>(553), A<sub>2</sub>(646), O(671)</b>		R
	<b>P(417), T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), A<sub>2</sub>(646), O(671)</b>		F
			90

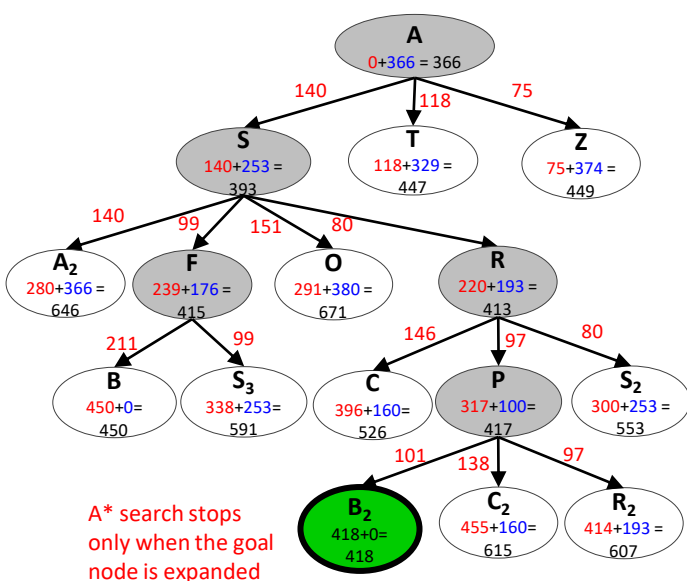


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

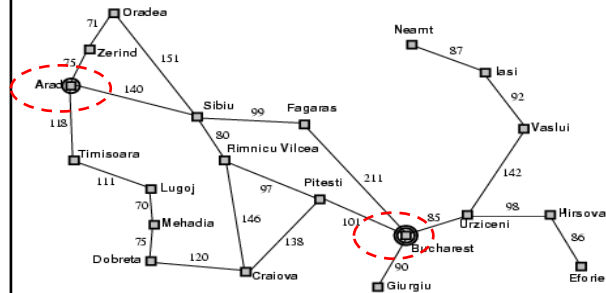


Notes: Optimal path for this problem is A → S → R → P → B

(Front)	Frontier	(Back)	Selected Node
	<b>A(366)</b>		-
	<b>S(393), T(447), Z(449)</b>		A
	<b>R(413), F(415), T(447), Z(449), A<sub>2</sub>(646), O(671),</b>		S
	<b>F(415), P(417), T(447), Z(449), C(526), S<sub>2</sub>(553), A<sub>2</sub>(646), O(671)</b>		R
	<b>P(417), T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), A<sub>2</sub>(646), O(671)</b>		F
	<b>B<sub>2</sub>(418), T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), R<sub>2</sub>(607), C<sub>2</sub>(615), A<sub>2</sub>(646), O(671)</b>		P
			91



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Notes: Optimal path for this problem is A → S → R → P → B

Solution Path: **A → S → R → P → B**

Cost: **418**

Optimal: Yes

#Nodes explored: **6**

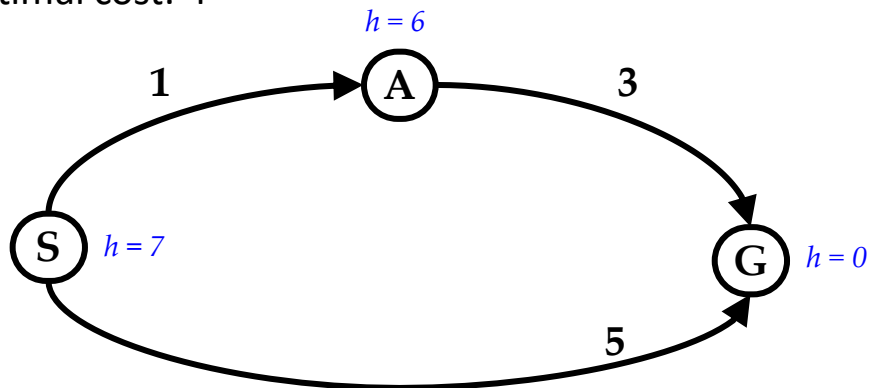
#memory required: **11**

(Front)	Frontier	(Back)	Selected Node
	<b>A(366)</b>		-
	<b>S(393), T(447), Z(449)</b>		A
	<b>R(413), F(415), T(447), Z(449), A<sub>2</sub>(646), O(671),</b>		S
	<b>F(415), P(417), T(447), Z(449), C(526), S<sub>2</sub>(553), A<sub>2</sub>(646), O(671)</b>		R
	<b>P(417), T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), A<sub>2</sub>(646), O(671)</b>		F
	<b>B<sub>2</sub>(418), T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), R<sub>2</sub>(607), C<sub>2</sub>(615), A<sub>2</sub>(646), O(671)</b>		P
	<b>T(447), Z(449), B(450), C(526), S<sub>2</sub>(553), S<sub>3</sub>(591), R<sub>2</sub>(607), C<sub>2</sub>(615), A<sub>2</sub>(646), O(671)</b>		B

## Constrain on heuristic function $h$

Optimal path: S  $\rightarrow$  A  $\rightarrow$  G

Optimal cost: 4



$g + h = f$

<del>S</del>	<del>0 + 7 = 7</del>
S $\rightarrow$ A	1 + 6 = 7
S $\rightarrow$ G	5 + 0 = 5

- What went wrong?
- Problem: Actual cost ( $d(A,G)=3$ ) < estimated cost ( $h(A)=6$ ) *overestimated!*
- We need estimates to be *less than* actual costs!
- **For A\* Tree Search to be optimal, the estimates must be less than the actual costs!** (known as the **admissibility**)

# Admissibility Requirement for A\* Tree Search

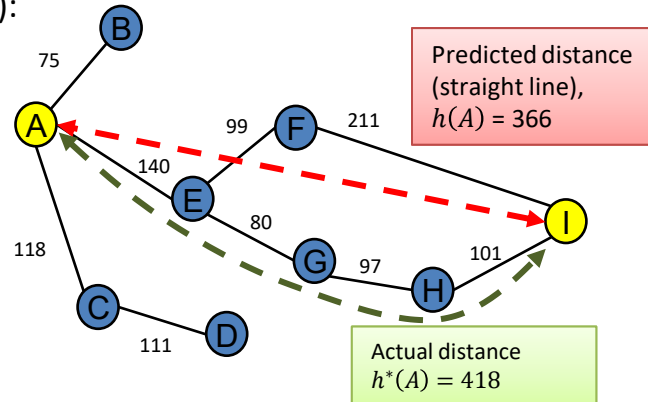
- For **tree search**, A\* is optimal if  $h(n)$  used is an **admissible** heuristic, i.e.,  $h(n)$  never overestimate the cost to reach the goal from  $n$

$$0 \leq h(n) \leq h^*(n)$$

where  $h^*(n)$  is the true cost to the nearest goal for node  $n$

- An admissible heuristic is by nature optimistic, since they think the cost of solving the problem is less than it actually is.
- Example of a heuristic distance (straight line):

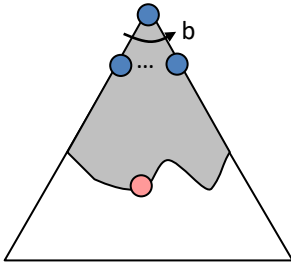
State	Actual Distance $h^*(n)$	Predicted Distance $h(n)$
A	418	366
B	493	374
C	536	329
D	647	244
E	278	253
F	211	178
G	198	193
H	101	98
I (final)	0	0



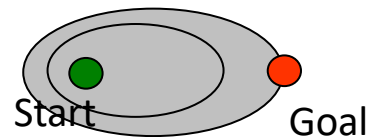
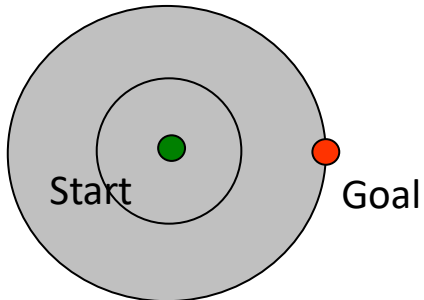
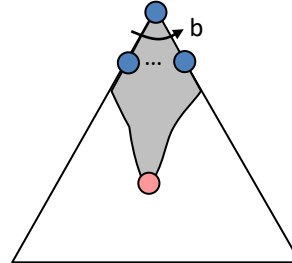
The heuristic using the straight line distance between any two nodes are always smaller than their actual costs. Therefore, the heuristic  $h(n)$  is admissible

# Properties of A\* Search

Uniform-Cost



A\*





# Outline

---

- Search Problems
  - Problem formulation
- Search Algorithms
  - Uninformed search
  - Informed search
- **Adversarial Search**
  - **Minimax**
  - **Alpha-Beta Pruning**



# Adversarial Search (Game)

- Search algorithms try to find a solution to a question
- In **adversarial search**, the algorithm also faces an **opponent** that tries to achieve the **opposite goal**.
- Often, AI that uses adversarial search is encountered in games, such as *tic tac toe*, *chess*, or *go*.

O	?	
X	X	O
O		X

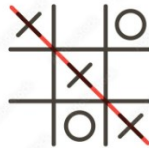
tic tac toe



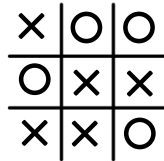
Go game

# Minimax

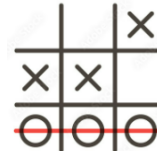
- **Minimax** is an adversarial search algorithm which assigns scores (**utility** value) to all possible **terminal** states: *winning, losing, and tie*.
- Tic tac toe example: (Assuming o is the opponent)



+1 (win)



0 (tie)

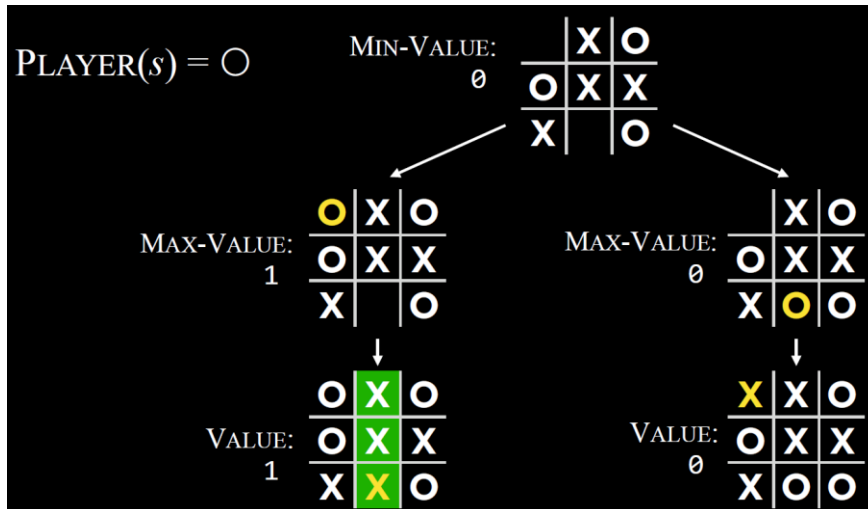


-1 (lose)

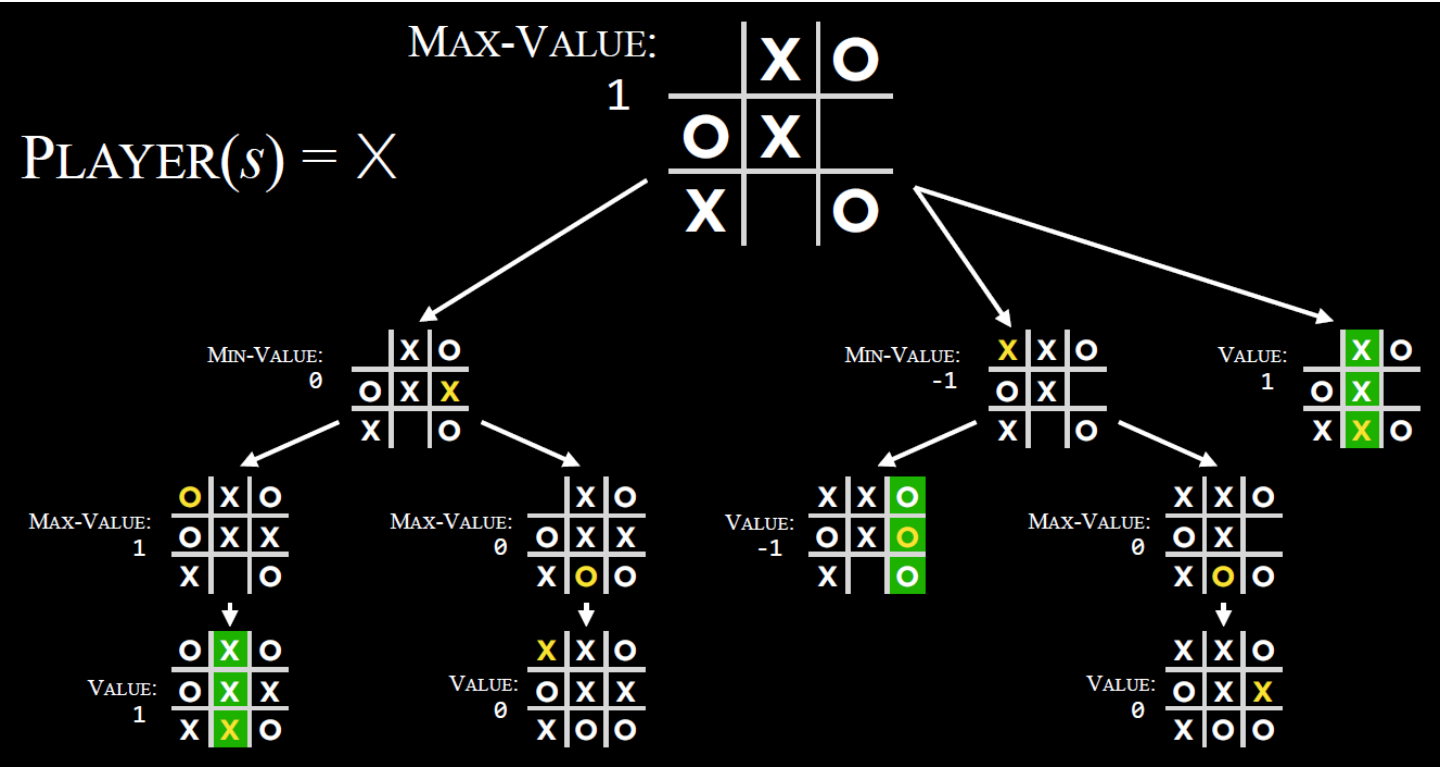
- Depending on the player's turn, **Minimax** aims to choose a play action that:
  - Maximize the score for the main player, e.g. MAX(x)
  - Minimize the score for the opponent player, e.g. MIN(o)

# Utility value for intermediate states

- The utility value for an intermediate state can be estimated by recursively simulating all possible actions that can take place from the current state until a terminal state is reached, assuming each player plays **optimally**.



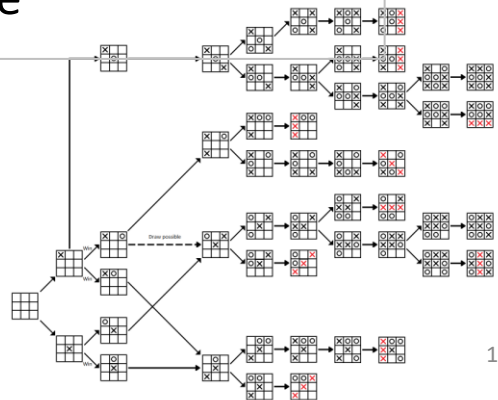
---



What is the maximum depth of the game tree for a 3x3 tic tac toe game?

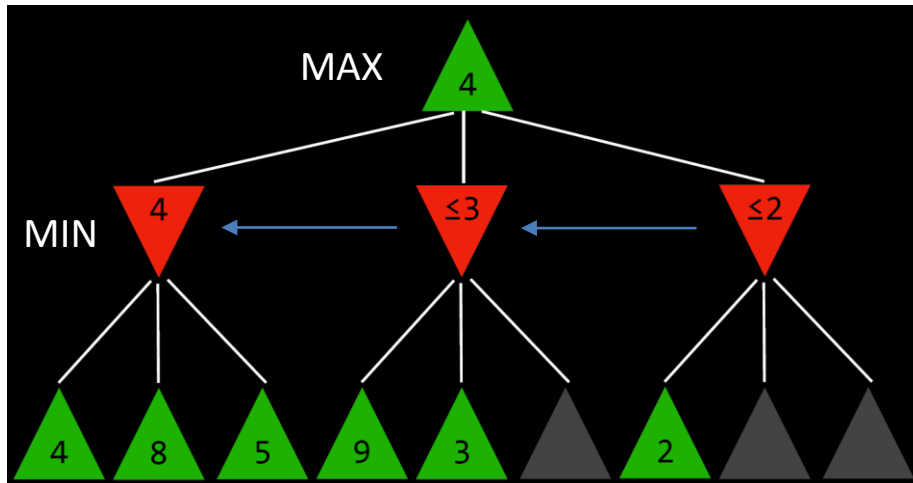
# Minimax Algorithm

```
if player is the main player (x):  
    for all possible actions:  
        compute utility value (score) for the state  
    choose action with maximum value  
else (o):  
    for all possible actions:  
        compute utility value (score) for the state  
    choose action with minimum value
```



# Alpha-Beta Pruning

- For complex game, there might be many possible branches in a game tree which require significant amount of time to explore.
- **Alpha-Beta Pruning** skips some of the branches that are not worthy to explore further.



# Issue with Minimax

- In realistic games, it might not be possible to get to the *terminal* states to obtain the utilities (scores).
  - Tic-tac-toe: 255,168 possible games
  - Chess:
    - 288,000,000,000 possible game after 4 moves
    - $10^{29000}$  possible games (lower bound)

# Depth-Limited Minimax

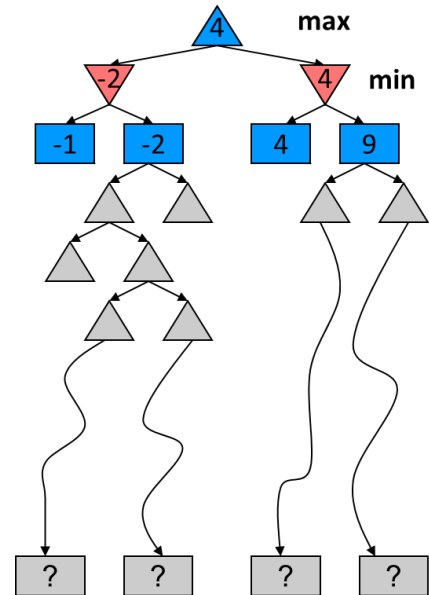
## ■ Depth-Limited Minimax:

- Search only to a limited depth in the tree.
- Estimate terminal utilities with an **evaluation function** for non-terminal positions.

– For example, for chess:

$Eval(s) = f(\text{pieces each player has and their positions})$

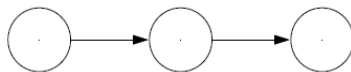
- The better the evaluation function, the better the Minimax result.
- No guarantee of optimal play.



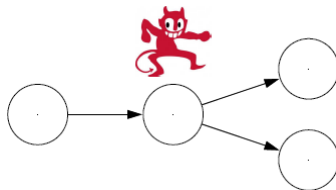


# Decision Making Models

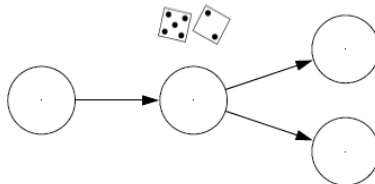
- **Search Problem** – you control everything



- **Adversarial Games** – against opponent (e.g., chess, go)




- **Markov Decision Processes** – against nature/chance (e.g., Blackjack), a form of *non-deterministic* search



# Decision Making Models

- **Constraint Satisfaction Problems** – to achieve certain goal under some constraints, **action order is not important** (e.g., Sudoku)

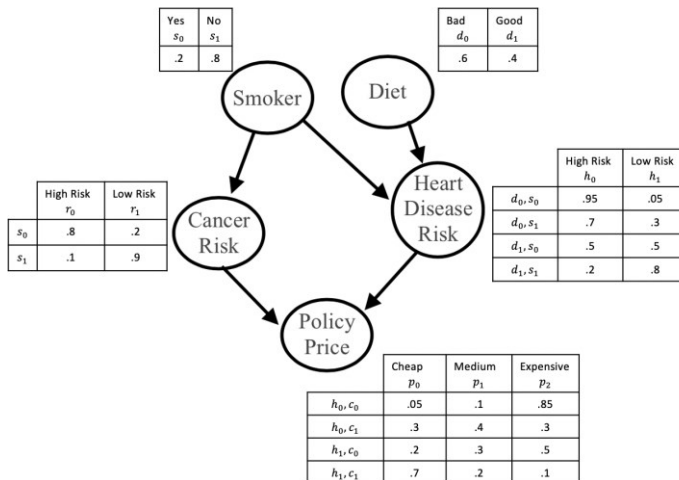


5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

# Decision Making Models

- Bayesian Networks** — A Bayesian network (BN) is a probabilistic graphical model for representing knowledge about an uncertain domain where each node corresponds to a random variable and each edge represents the conditional probability for the corresponding random variables (will be covered next)





Next:

**Uncertainty**