Search

Chen-Yu Wei

Question

A **farmer** wants to get his **cabbage**, **goat**, and **wolf** across a river. He has a boat that only holds two. He cannot leave the cabbage and goat alone or the goat and wolf alone. How many river crossings does he need?

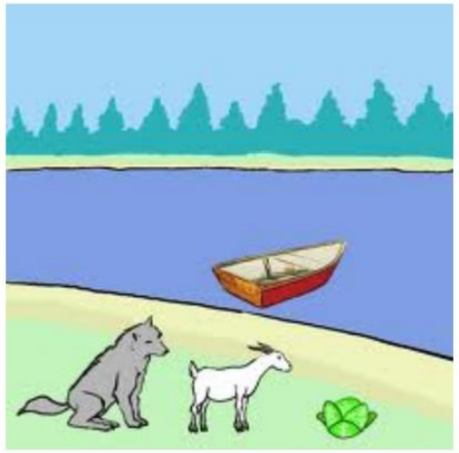
- 4
- 5
- 6
- 7
- no solution

https://stanford-cs221.github.io/autumn2023-extra/modules/search/modeling.pdf

Model the Problem

How many different “states”?

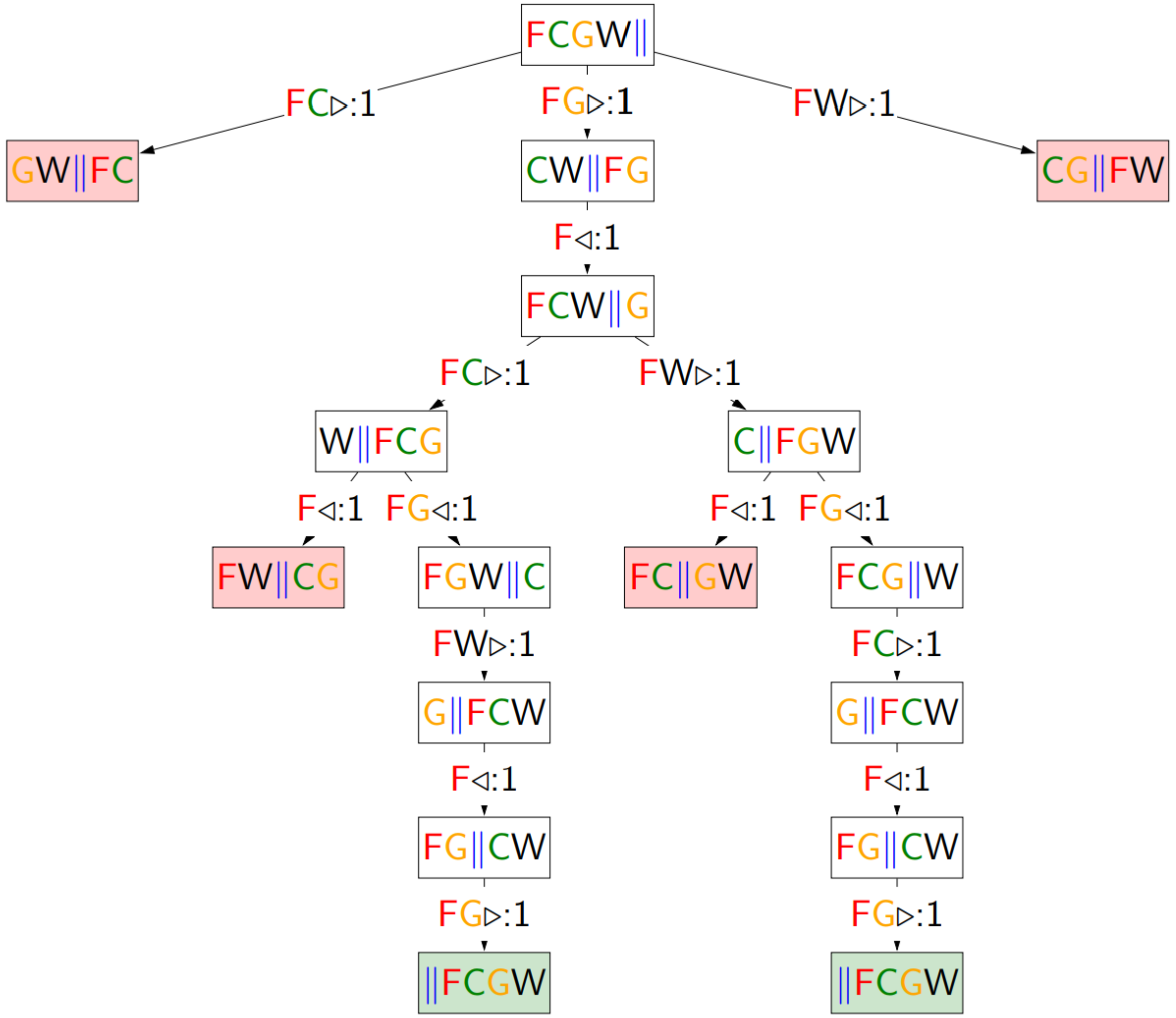
How many different “actions”?



Farmer Cabbage Goat Wolf

$F\triangleright$	$F\triangleleft$
$FC\triangleright$	$FC\triangleleft$
$FG\triangleright$	$FG\triangleleft$
$FW\triangleright$	$FW\triangleleft$

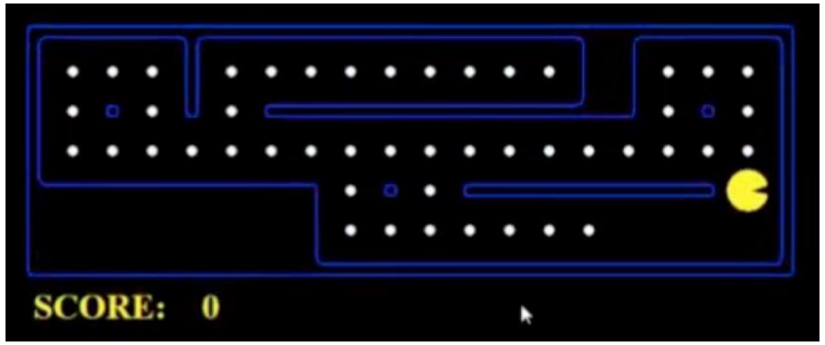
Build a Search Tree



Search Problem

- State space
- Initial state
- Goal test: Given a state, return whether the state is a goal
- Action
- Successor function: Given current state and action, return the new state
- (The cost of an action)

Example: PACMAN



Eat all dots

States: {(x,y), dot booleans}

Actions: NSEW

Successor: update location and possibly a dot boolean

Goal test: dots all false

Go to some destination

States: (x,y) location

Actions: NSEW

Successor: update location only

Goal test: is (x,y)=END

Example: SAINT (Slagle, 1961)

Symbolic Integrator

$$\int \frac{x^4}{(1-x^2)^{5/2}}\,dx \quad = \quad \frac{1}{3}\tan^3(\arcsin x) - \tan(\arcsin x) + \arcsin x$$

States: symbolic expression

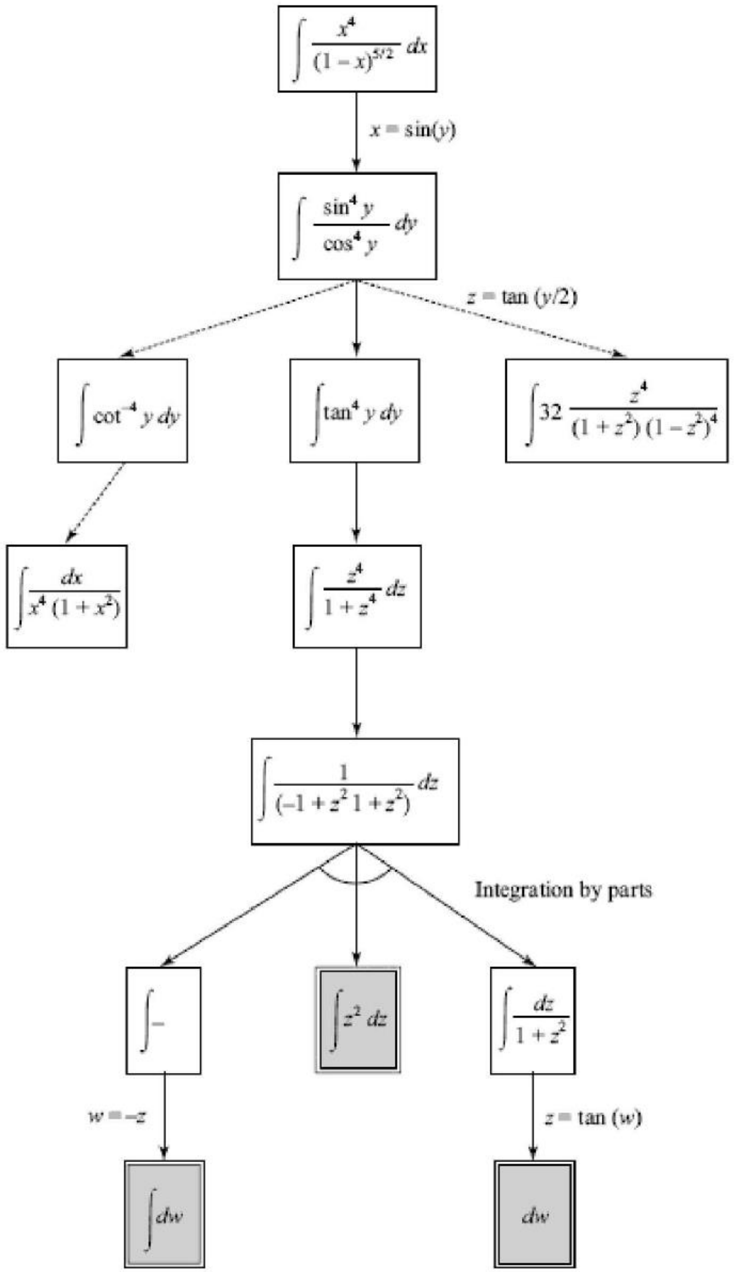
Actions: “common techniques”

Successor: the new expression after applying the technique

Goal test: whether the expression is in “standard form”

“common technique” examples:

- $\int c\, f(x)\mathrm{d}x = c \int f(x)\mathrm{d}x$
- $\int f(\tan x)\mathrm{d}x = \int \frac{f(y)}{1+y^2}\mathrm{d}y$
- If seeing $1-x^2$, then substitute $x = \sin y$


$\int \frac{x^4}{(1-x)^{5/2}}dx$
$x = \sin(y)$
$\int \frac{\sin^4 y}{\cos^4 y}dy$
$z = \tan(y/2)$
$\int \cot^{-4} y\, dy$
$\int \tan^4 y\, dy$
$\int 32 \frac{z^4}{(1+z^2)(1-z^2)^4}$
$\int \frac{dx}{x^4(1+x^2)}$
$\int \frac{z^4}{1+z^4}dz$
$\int \frac{1}{(-1+z^2 1+z^2)}dz$
Integration by parts
$\int -$
$\int z^2\, dz$
$\int \frac{dz}{1+z^2}$
$w = -z$
$z = \tan(w)$
$\int dw$
dw

Example: Machine translation

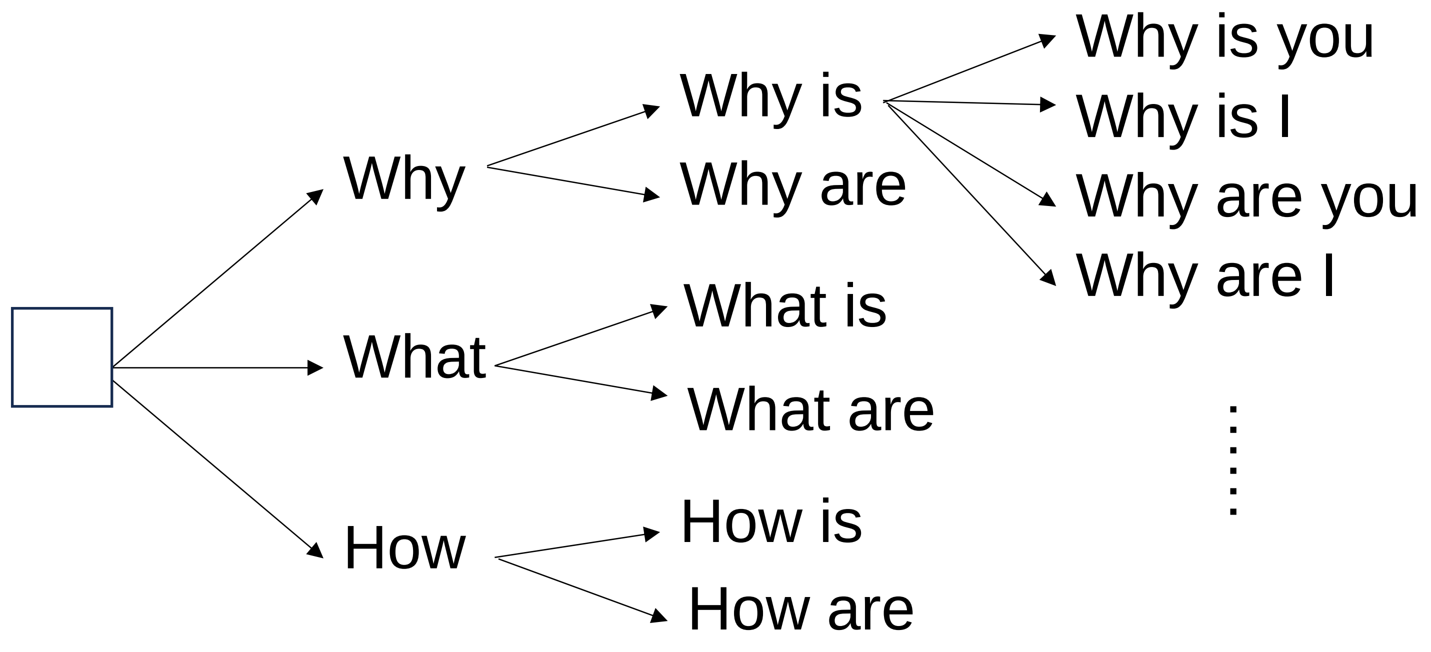
Translate “你好嗎” to English

States: current word sequence

Actions: the next word

Successor: the concatenation of current sequence and next word

Goal test: whether the current sequence means the same as 你好嗎

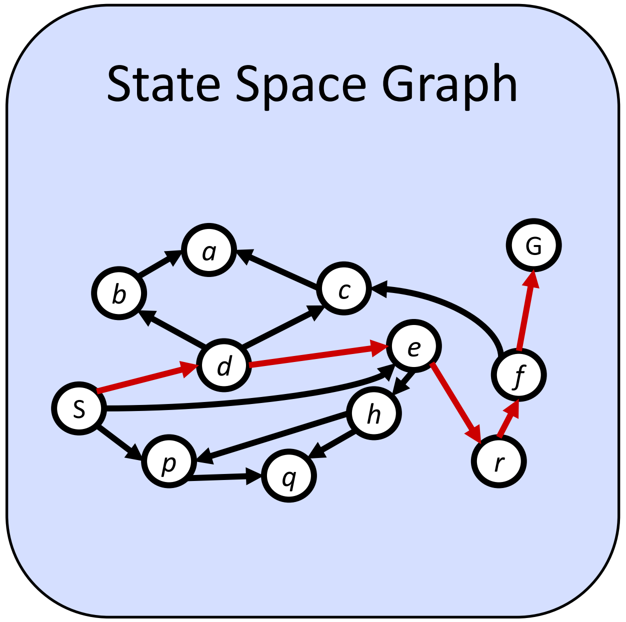


Topics

- BFS
- DFS
- UCS (Dijkstra Algorithm)
- Difference with DSA2:
  - The state space is exponentially large, and it’s unlikely we’ll store the whole state space in memory

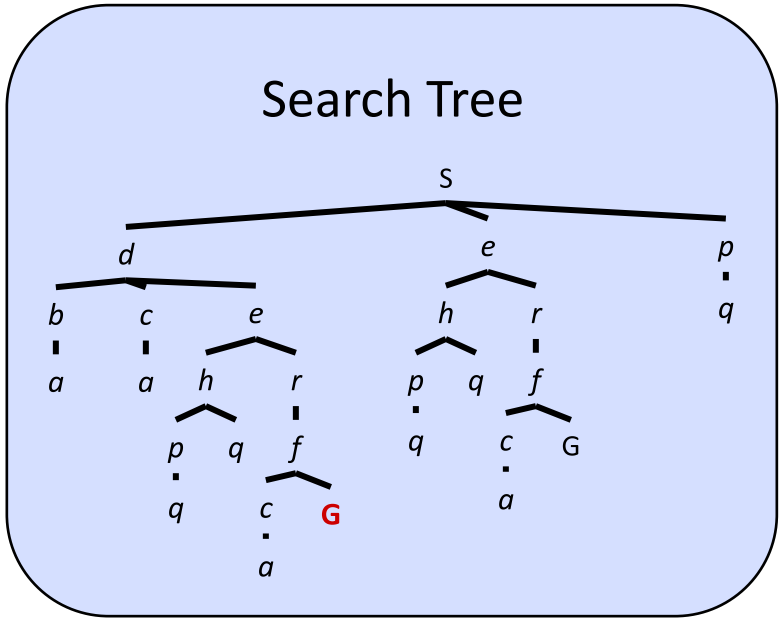
General Framework

State Space and Search Tree



Each NODE in in the search tree is an entire PATH in the state space graph.

We construct both on demand – and we construct as little as possible.



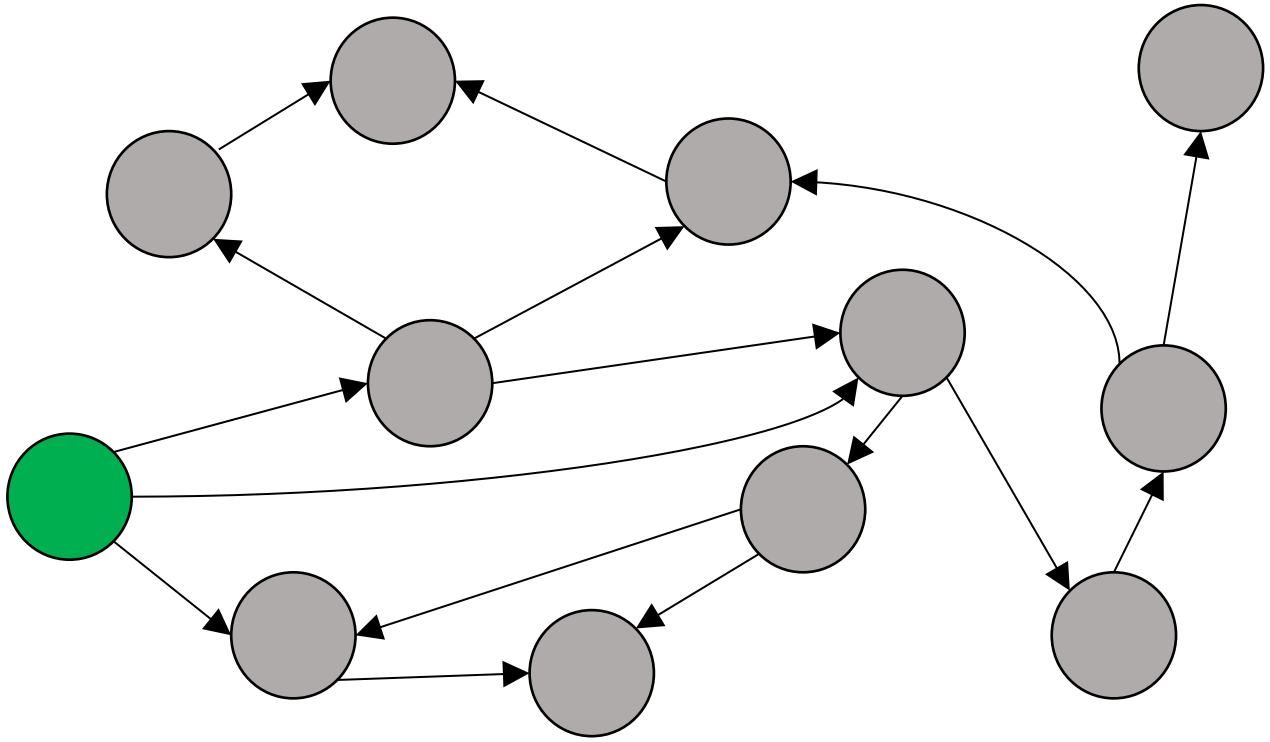
A General Framework

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier                  ①

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached      }
            Put $\mathrm{succ}(s, a)$ in Frontier          }  ②

    Move $s$ to Expanded                             ③
```

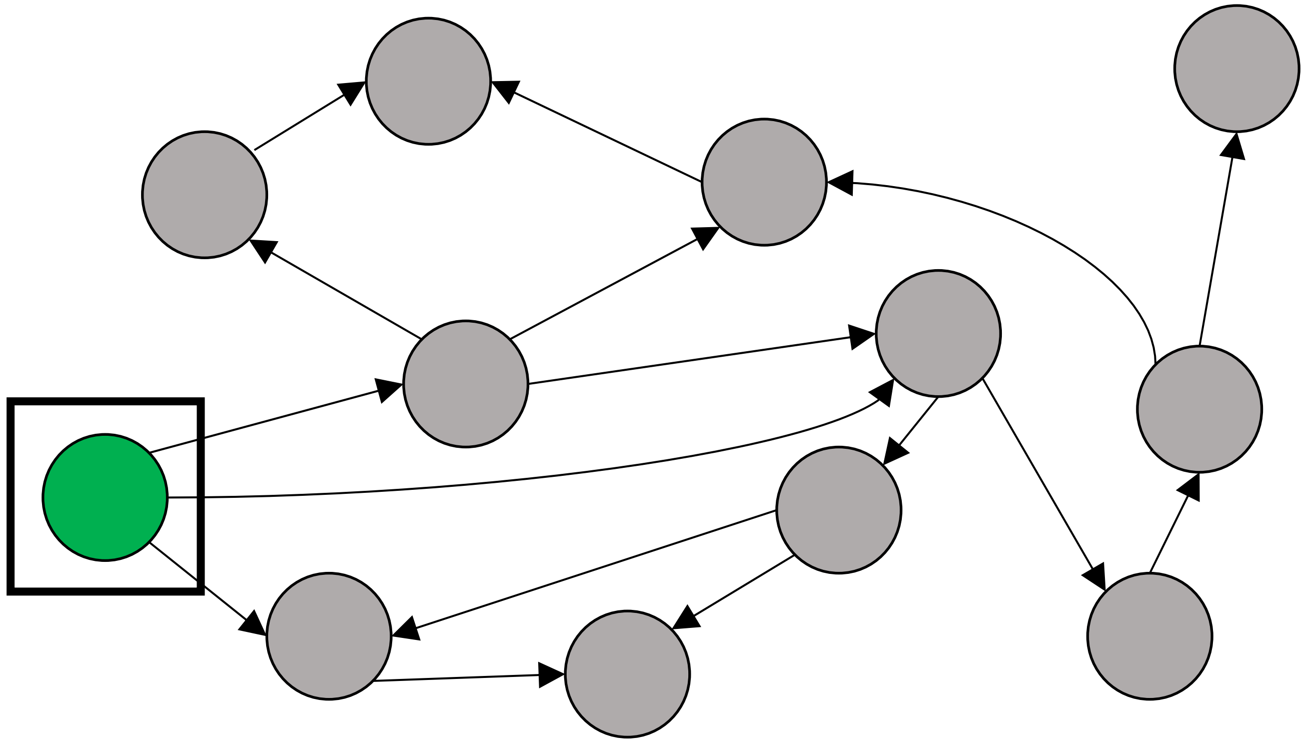
All nodes that are not in **Expanded** or **Frontier**

Nodes are divided into 3 groups: **Expanded**, **Frontier**, and **Unreached.**

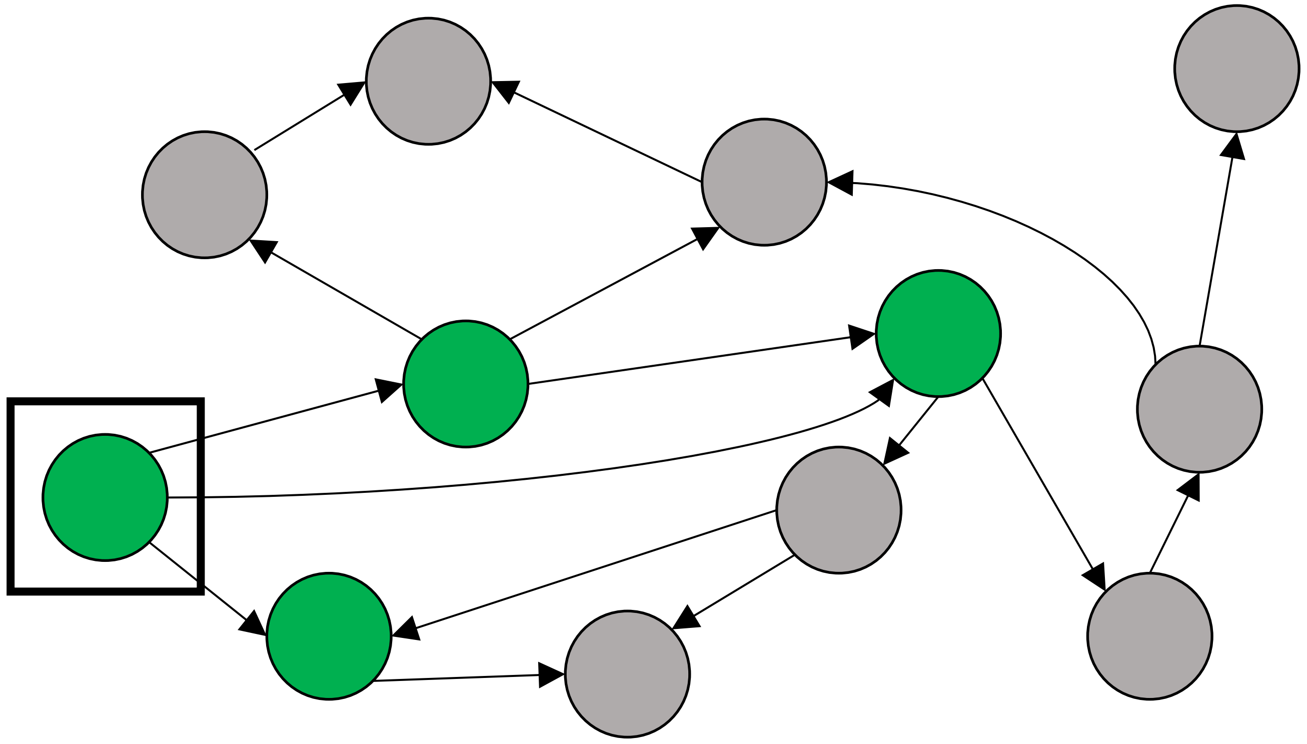
⓪ Place the initial state in **Frontier**



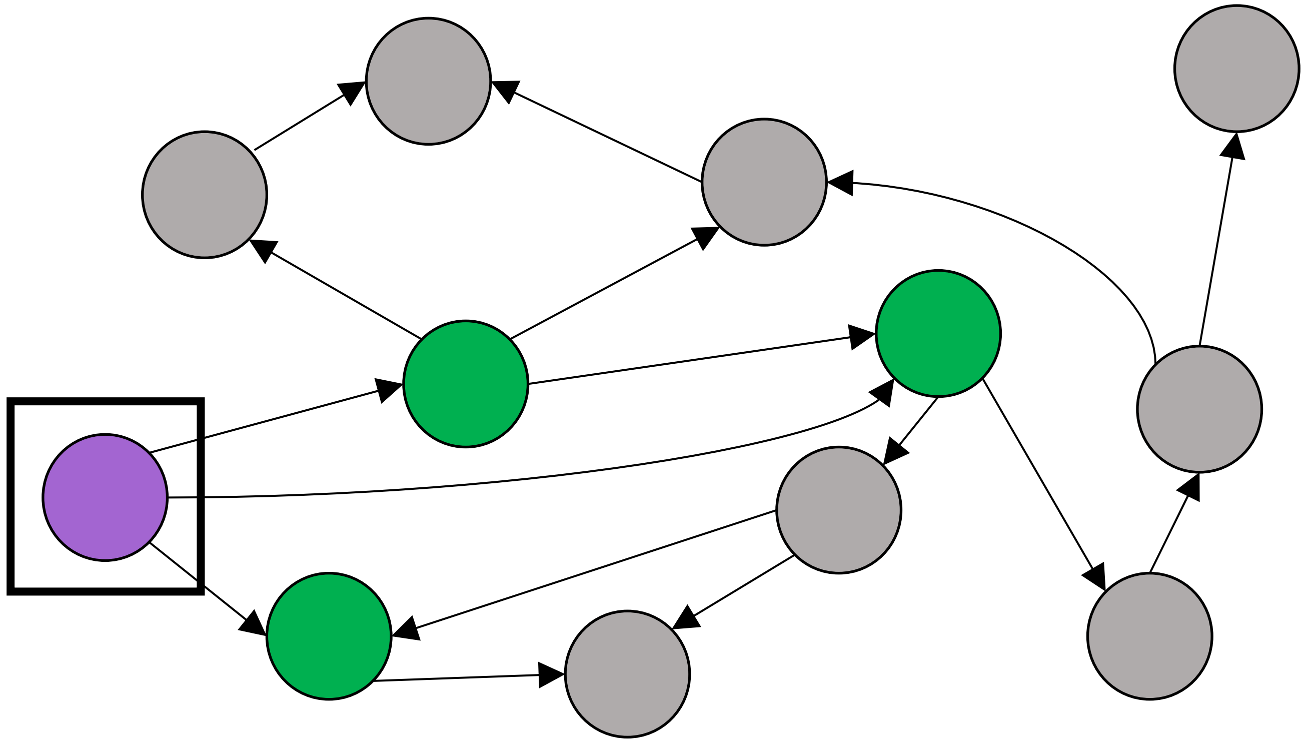
① Choose a node s from **Frontier**



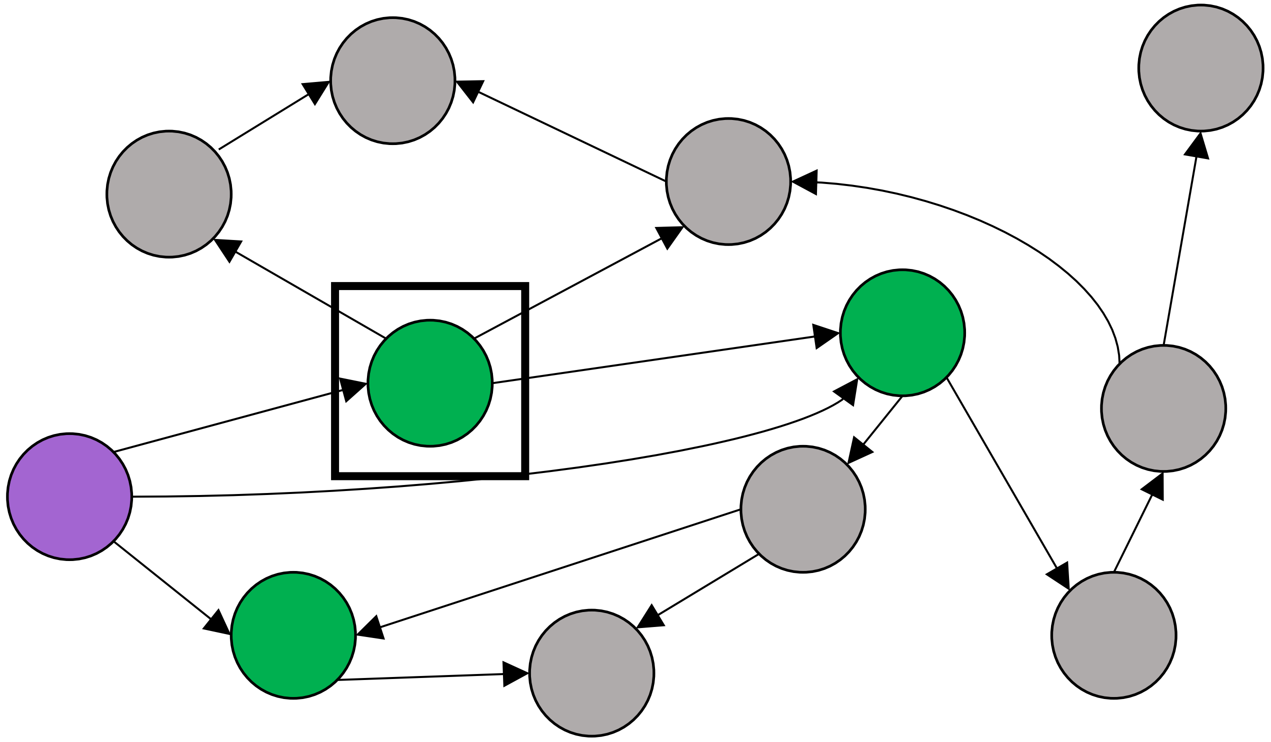
② Move all unreached successors of s to **Frontier**



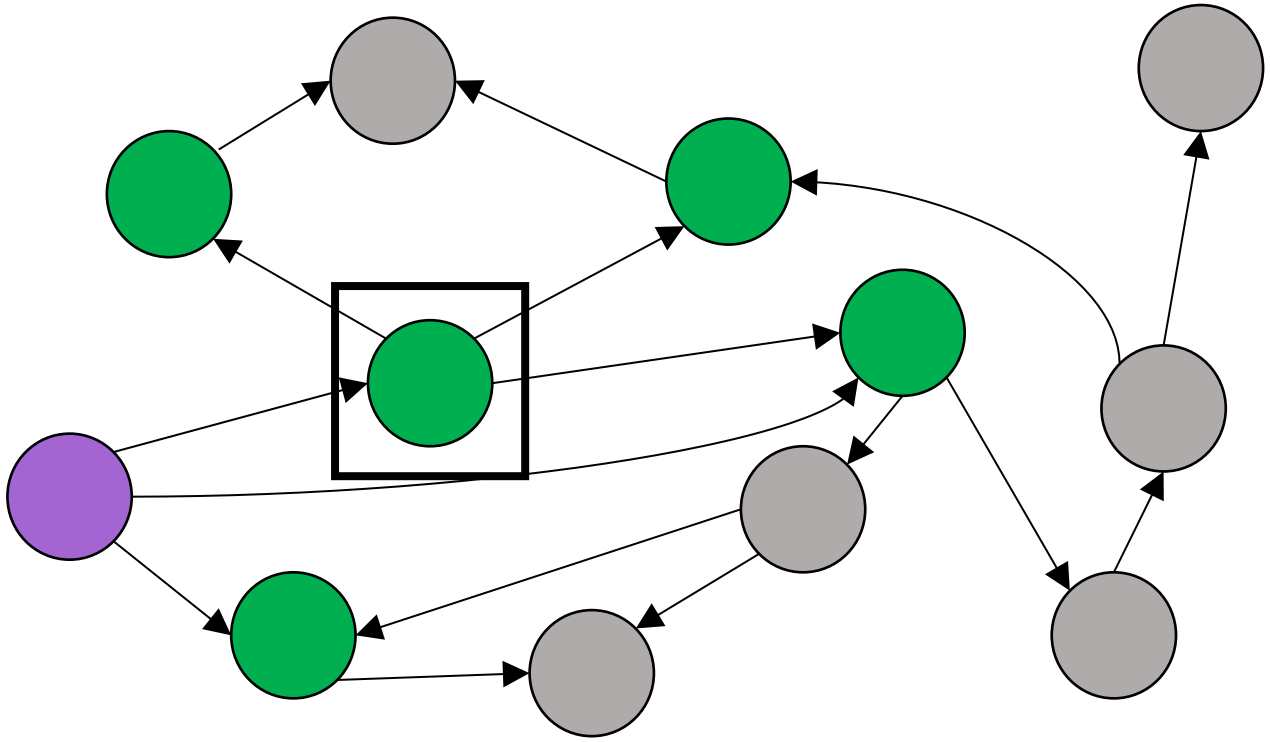
③ Move s to **Expanded**



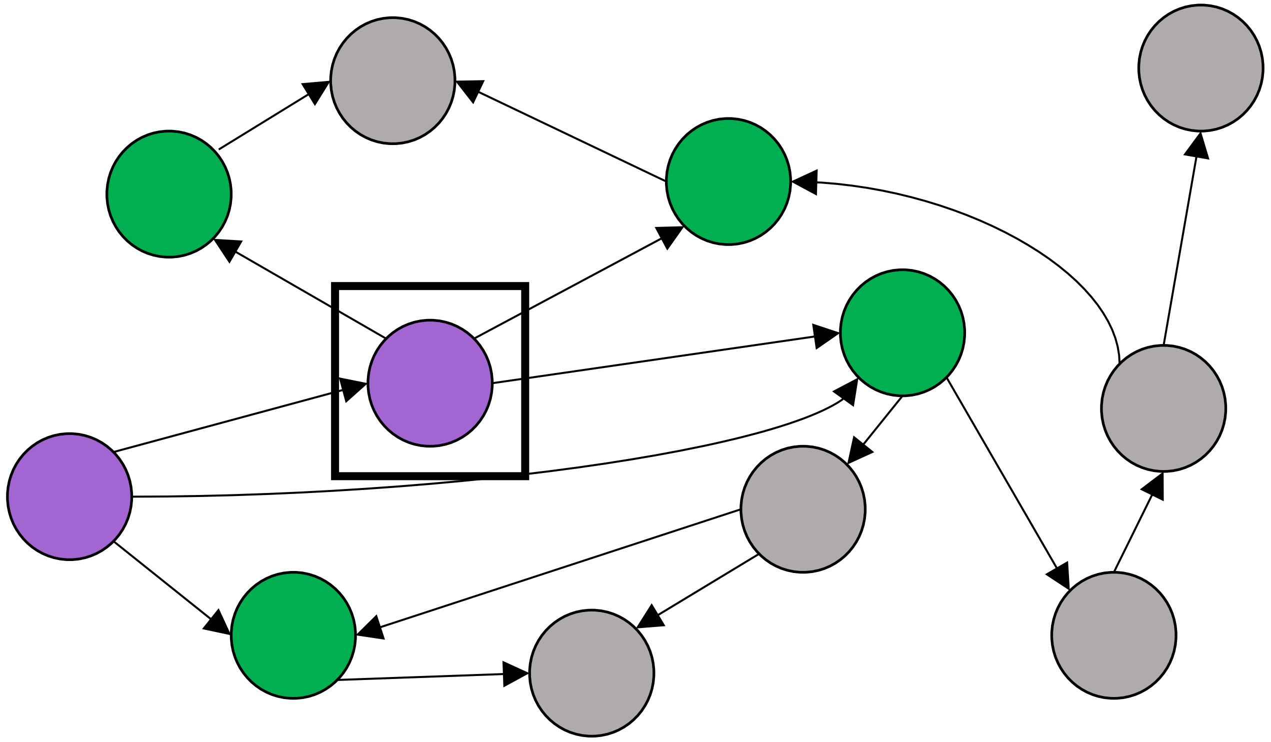
① Choose a node s from **Frontier**



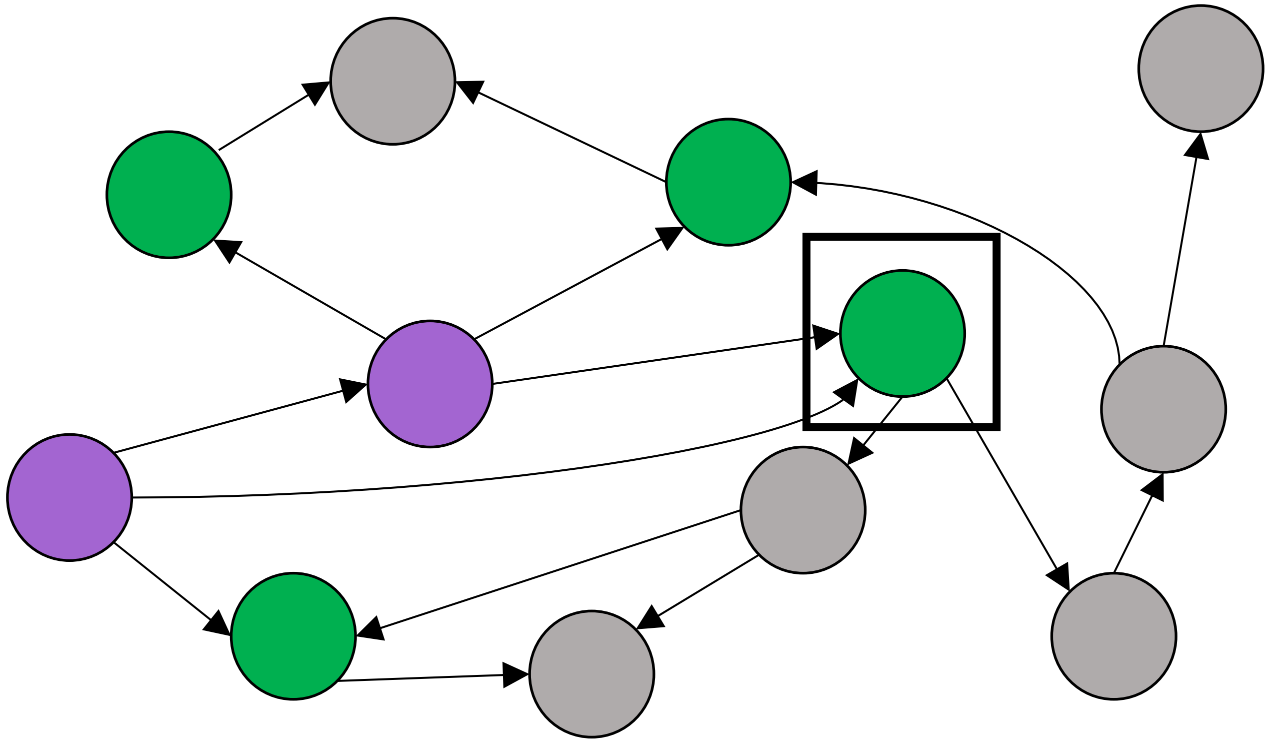
② Move all unreached successors of s to **Frontier**



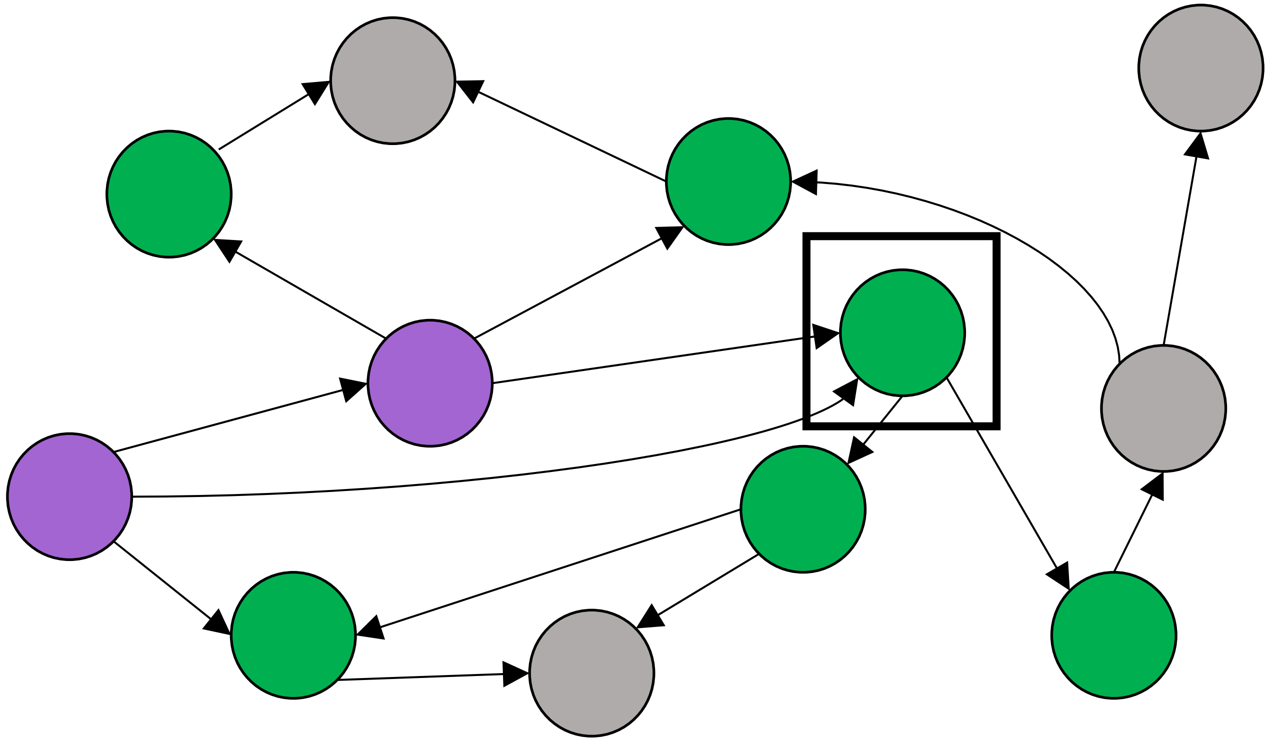
③ Move s to **Expanded**



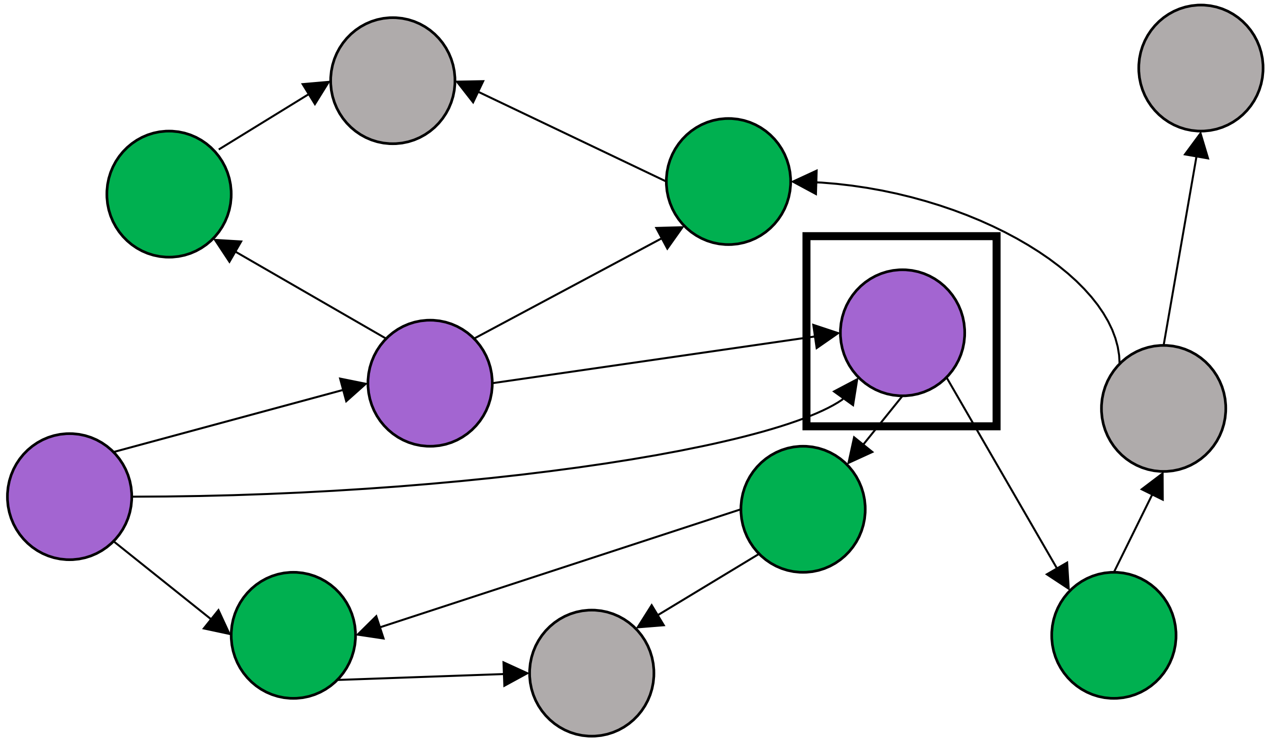
① Choose a node s from **Frontier**



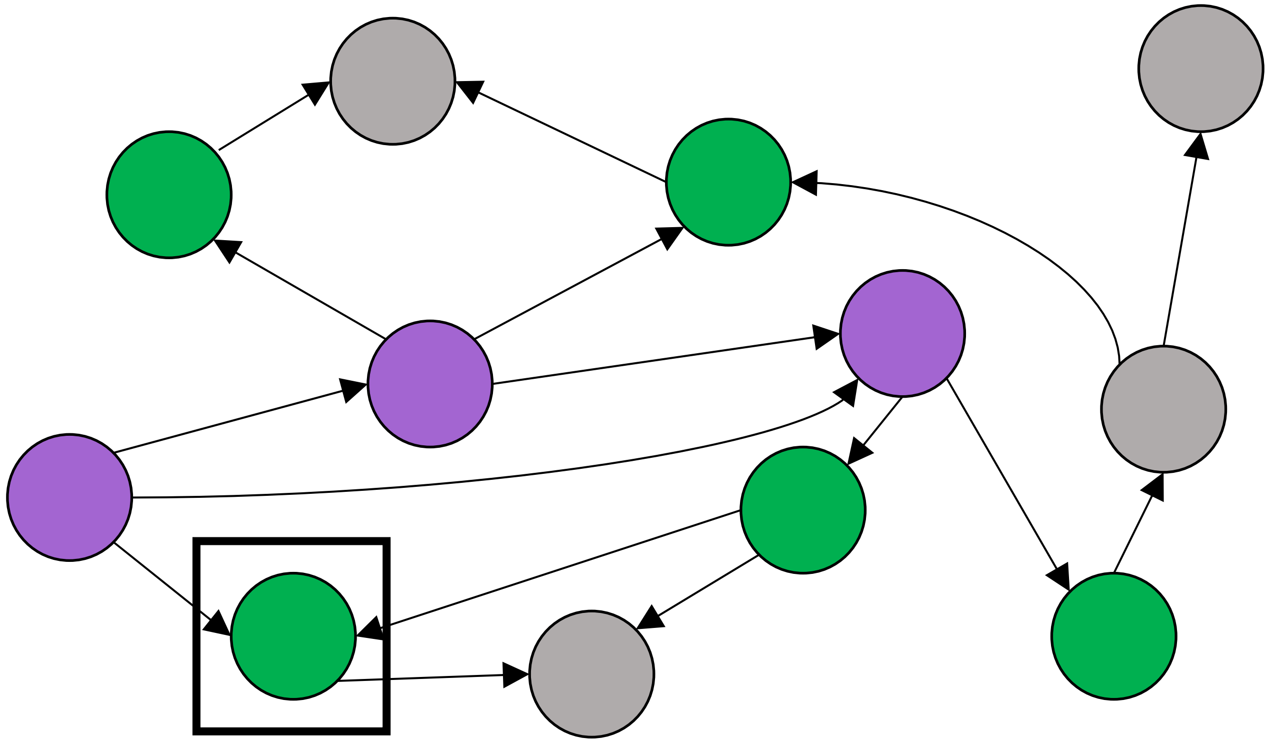
② Move all unreached successors of s to **Frontier**



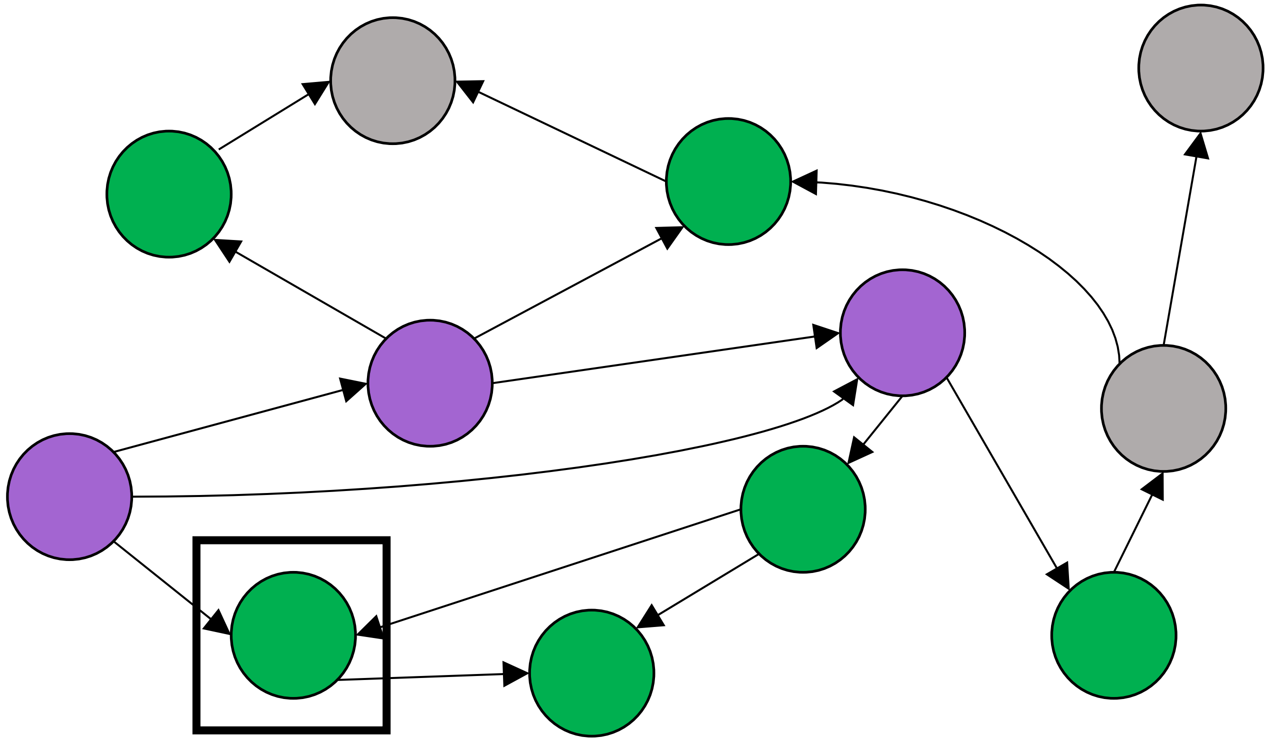
③ Move s to **Expanded**



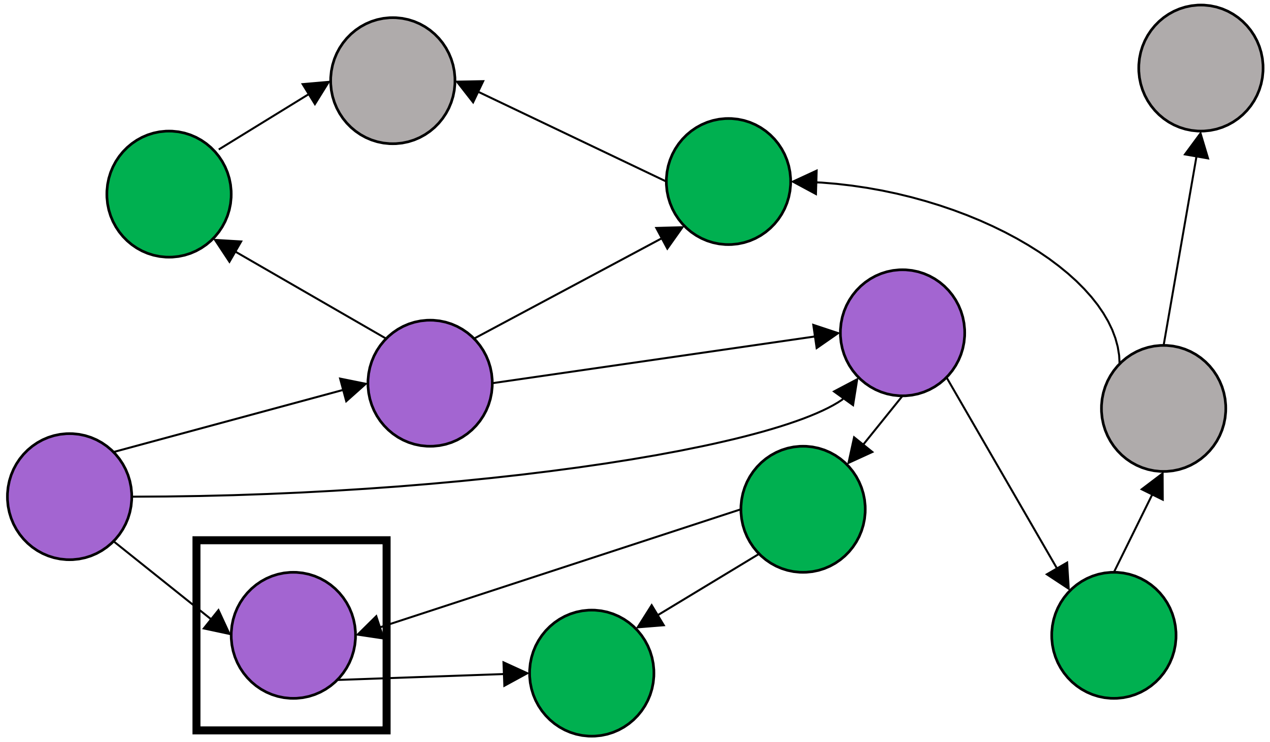
① Choose a node s from **Frontier**



② Move all unreached successors of s to **Frontier**



③ Move s to **Expanded**



When To Terminate?

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            Put $\mathrm{succ}(s, a)$ in Frontier

    Move $s$ to Expanded
```

- When a goal state is encountered
- If no goal state can be reached

Termination When Goal is Encountered

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier
    If $s$ is a goal state, then terminate

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            Put $\mathrm{succ}(s,a)$ in Frontier

    Move $s$ to Expanded
```

Late Goal Test

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            If $\mathrm{succ}(s,a)$ is a goal state, terminate
            Put $\mathrm{succ}(s,a)$ in Frontier

    Move $s$ to Expanded
```

Early Goal Test

Termination When Goal is Encountered

- Early Goal Test allows quicker termination when a goal is found.
  - Breadth First Search
  - Depth First Search

- However, when actions are associated with costs and we want to find a **minimum cost solution** (i.e., cost-sensitive), we may have to use the Late Goal Test.
  - Uniform Cost Search (Dijkstra Algorithm)

Termination If No Goal Can Be Reached

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier
    If $s$ is a goal state, then terminate

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            Put $\mathrm{succ}(s, a)$ in Frontier


    Move $s$ to Expanded
```

Late Goal Test

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            If $\mathrm{succ}(s, a)$ is a goal state, terminate
            Put $\mathrm{succ}(s, a)$ in Frontier


    Move $s$ to Expanded
```

Early Goal Test

Termination If No Goal Can Be Reached

```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a node $s$ from Frontier
    If $s$ is a goal state, then terminate

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            Put $\mathrm{succ}(s,a)$ in Frontier


    Move $s$ to Expanded
```

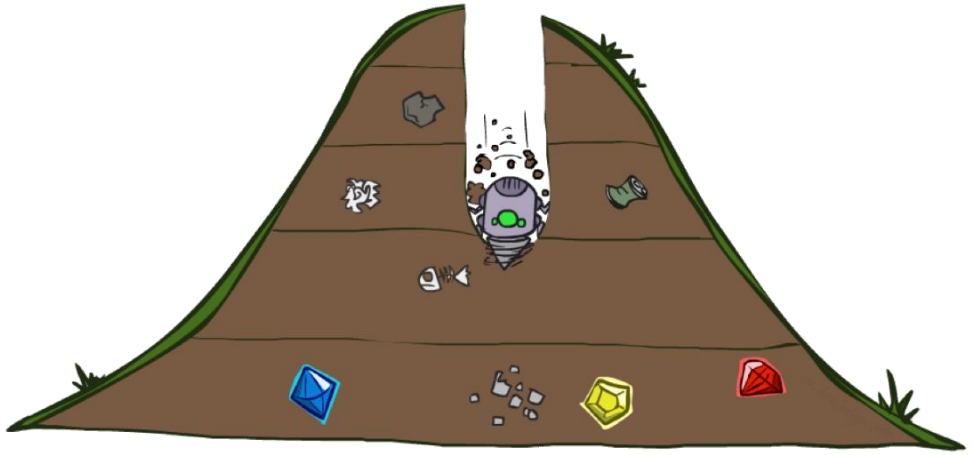
Late Goal Test

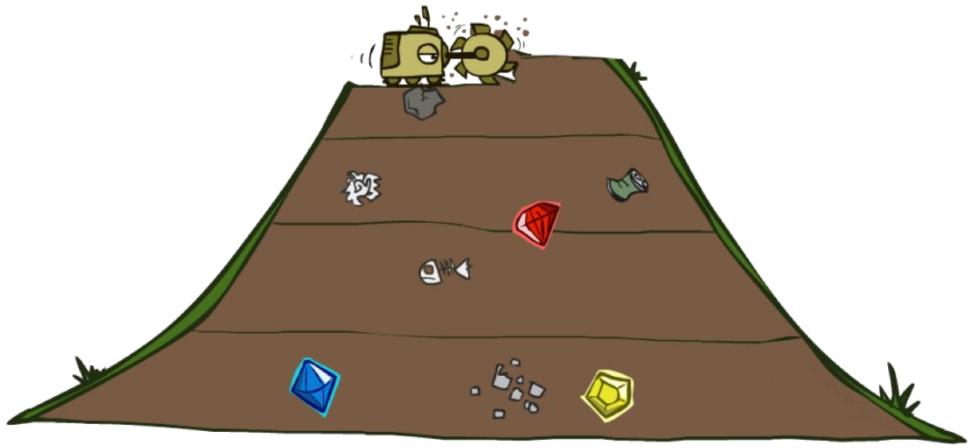
```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            If $\mathrm{succ}(s,a)$ is a goal state, terminate
            Put $\mathrm{succ}(s,a)$ in Frontier


    Move $s$ to Expanded
```

Early Goal Test

Uninformed Search

DFS and BFS


Depth First Search


Breadth First Search

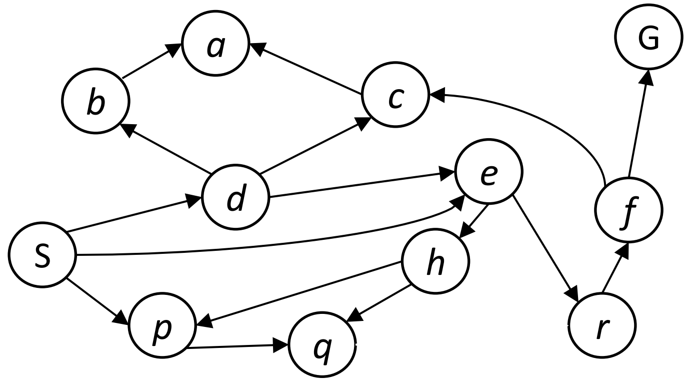
DFS and BFS

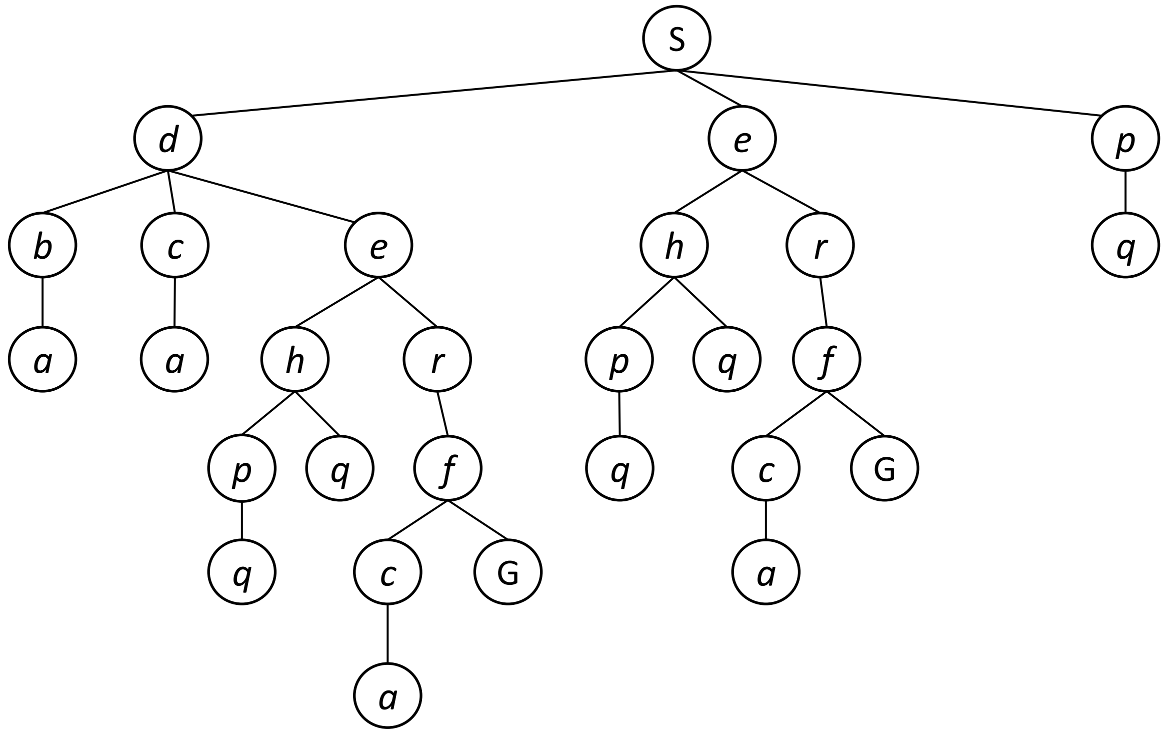
```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    $\boxed{\text{Choose a node } s \text{ from Frontier}}$   differ here

    For all action $a$:
        If $\text{succ}(s, a)$ has not been reached:
            If $\text{succ}(s, a)$ is a goal state, terminate
            Put $\text{succ}(s, a)$ in Frontier

    Move $s$ to Expanded
```

DFS

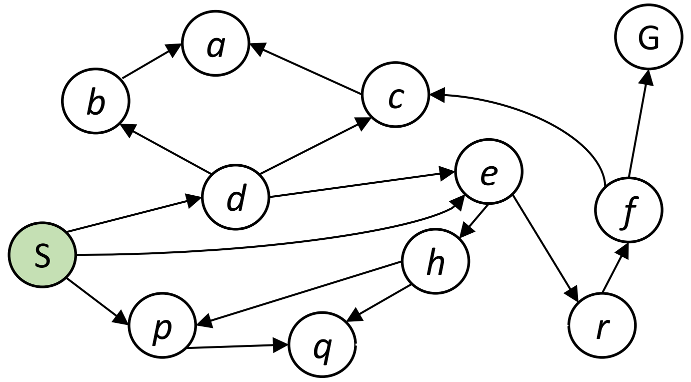
```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a newest node $s$ in Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            If $\mathrm{succ}(s, a)$ is a goal state, terminate
            Put $\mathrm{succ}(s, a)$ in Frontier

    Move $s$ to Expanded
```

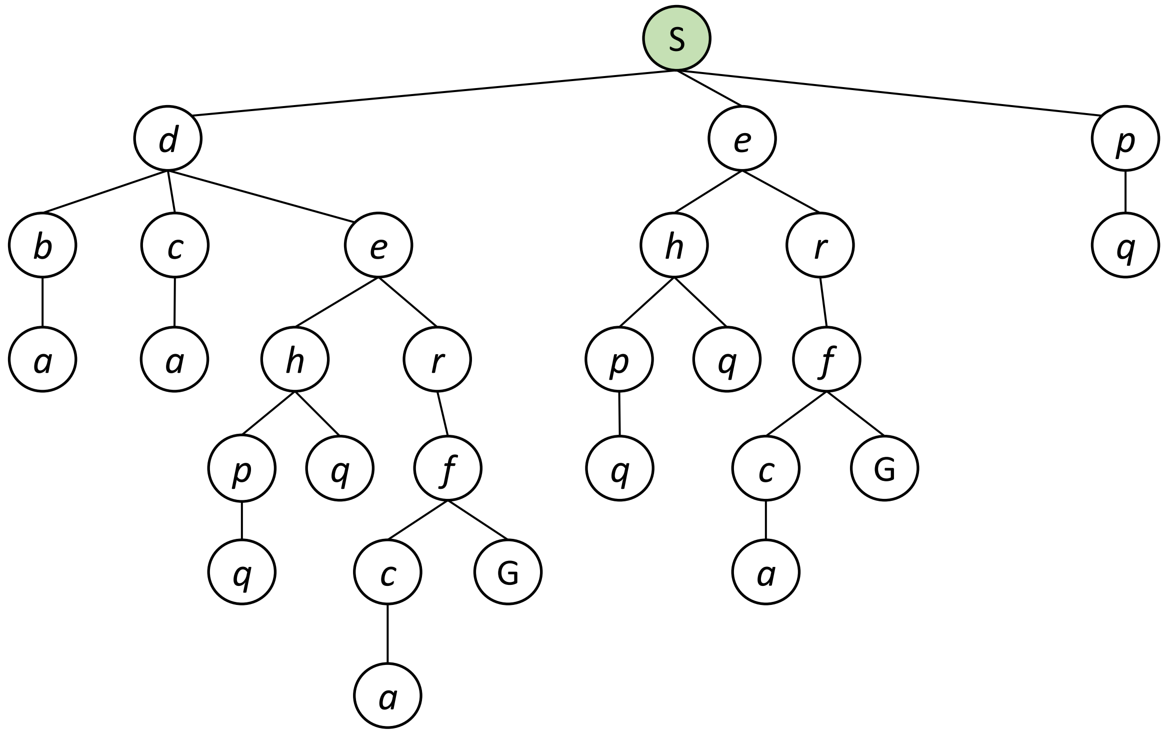
DFS



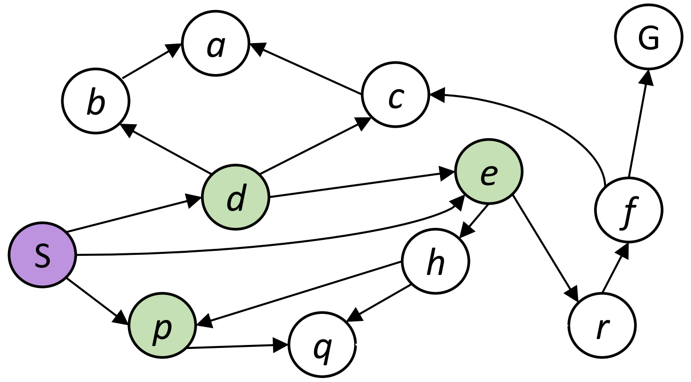


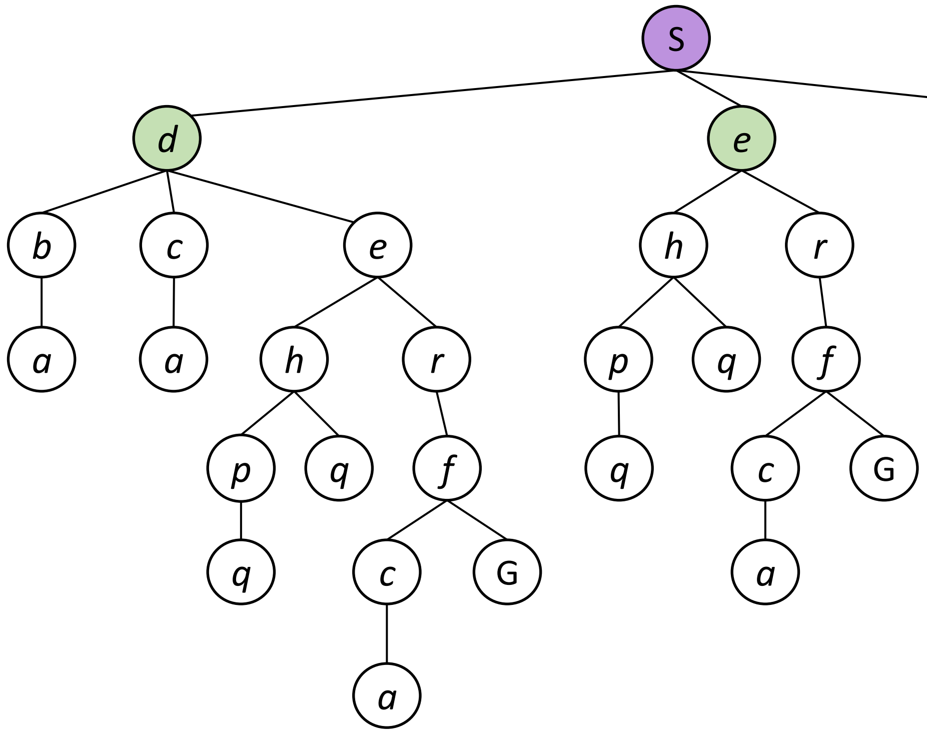
DFS



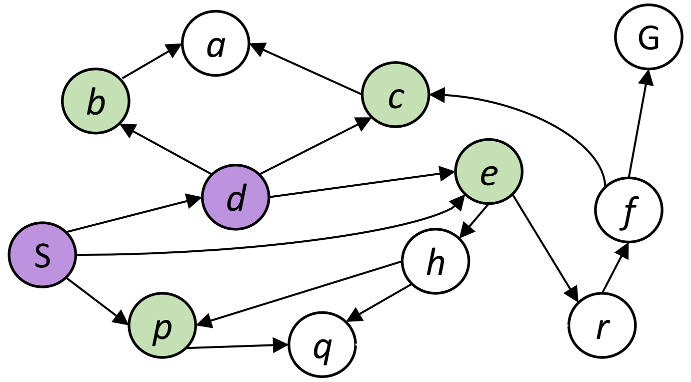


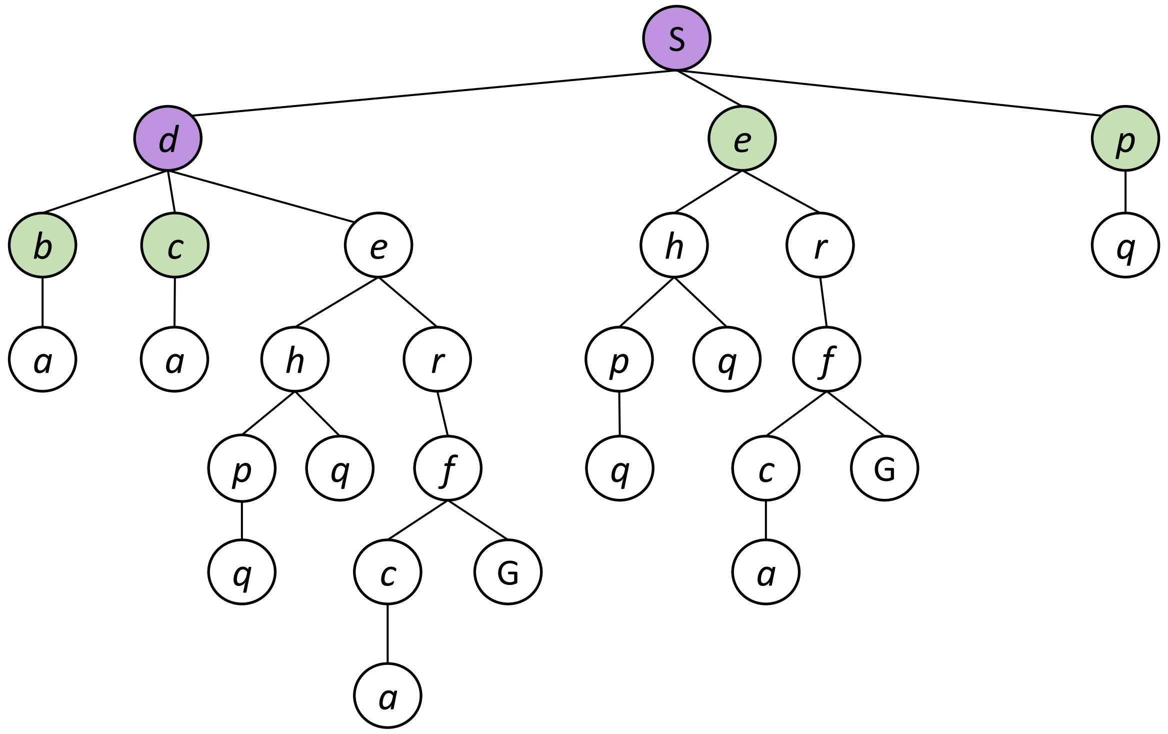
DFS



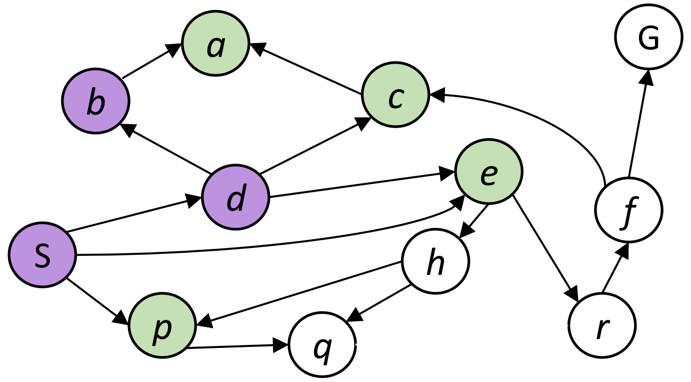


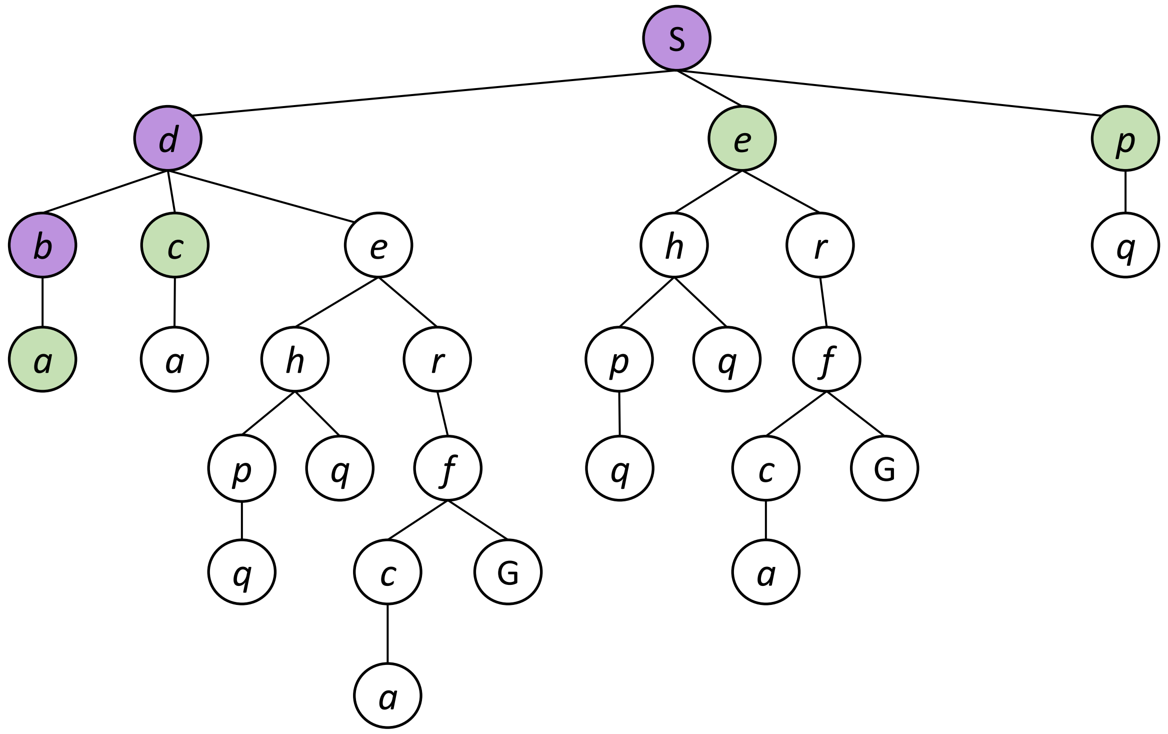
DFS



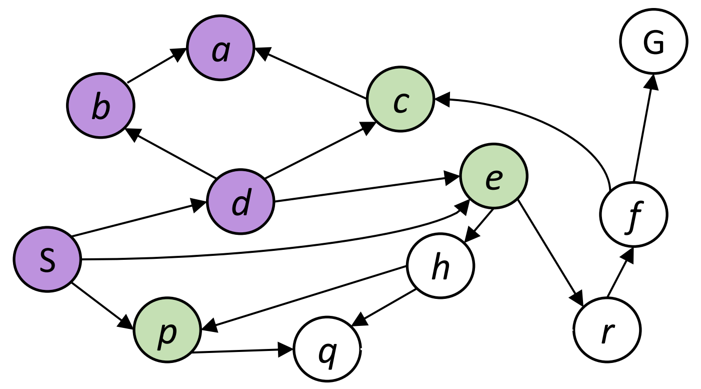


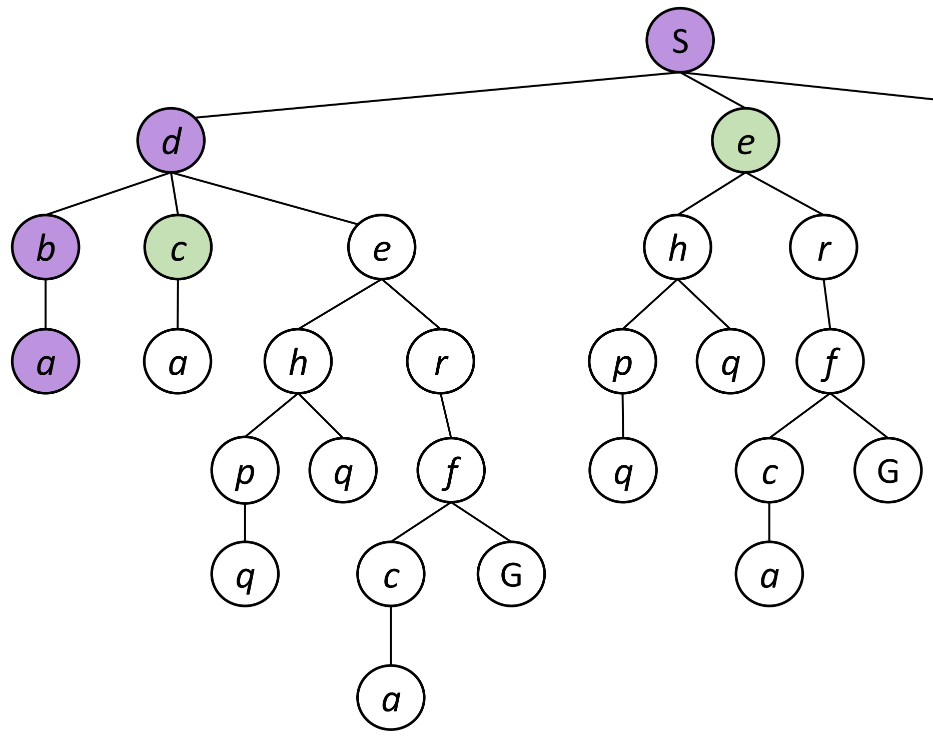
DFS



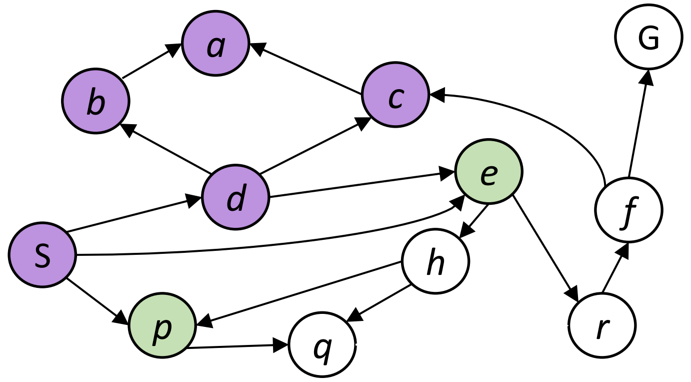


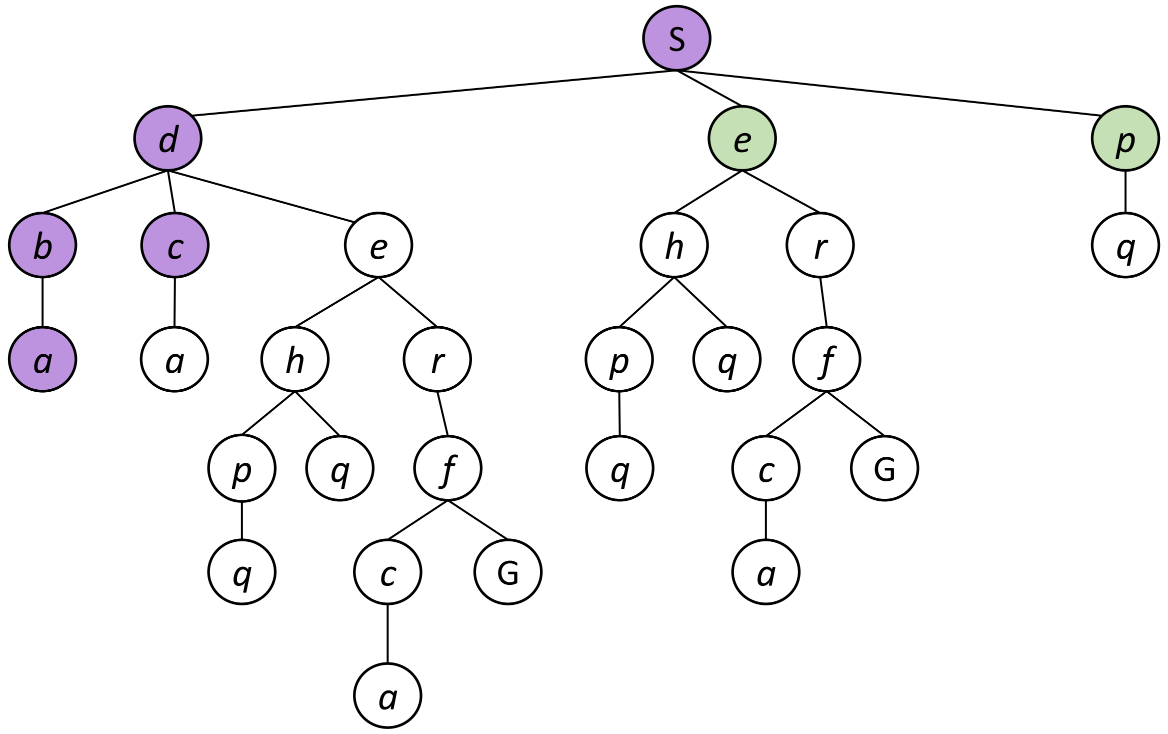
DFS



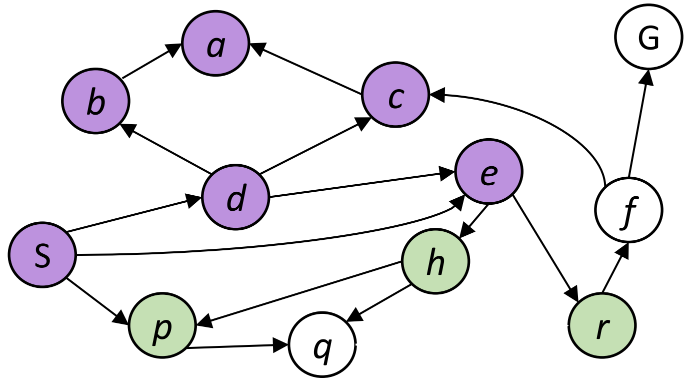


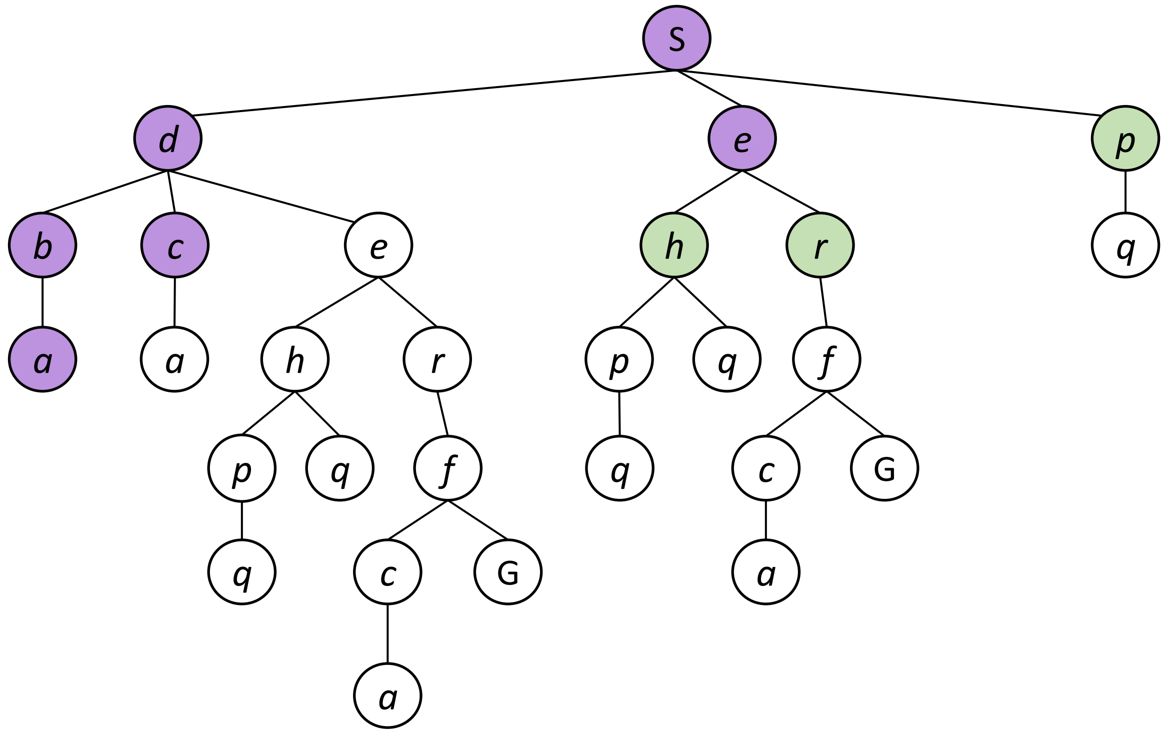
DFS



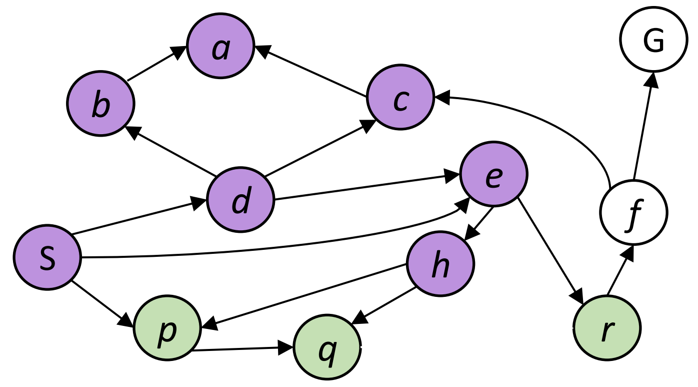


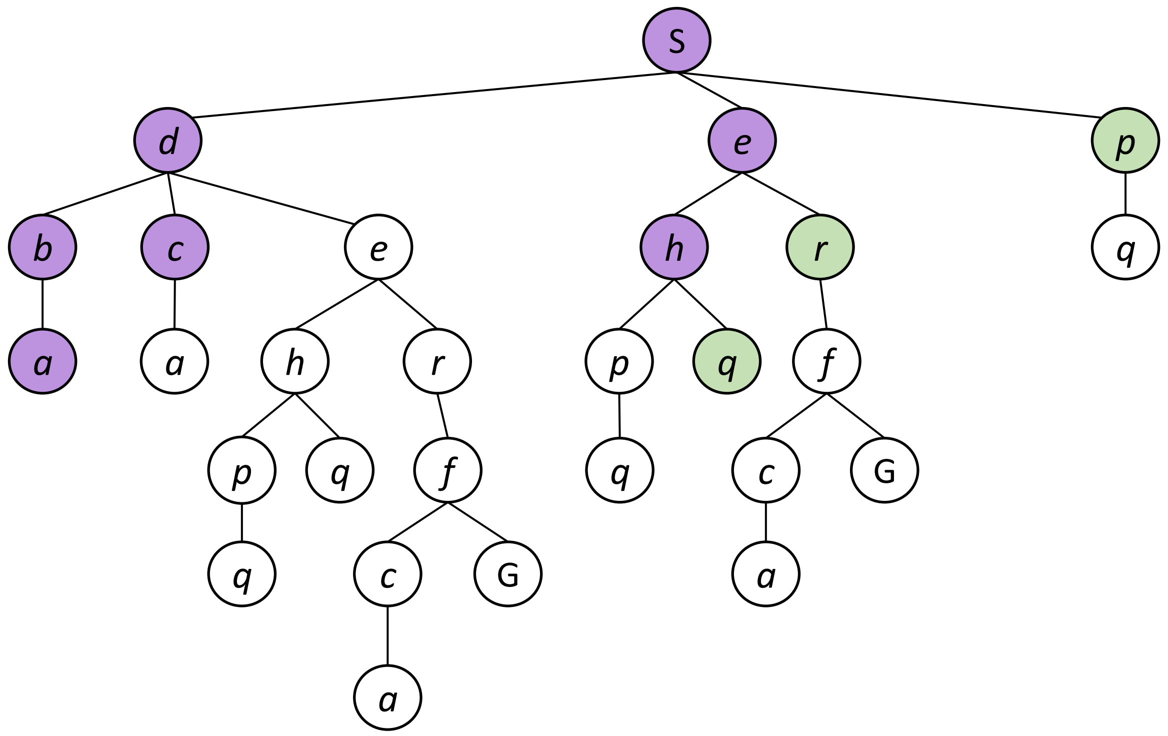
DFS



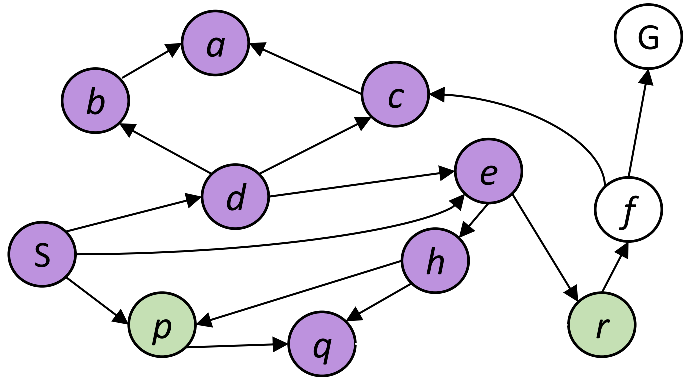


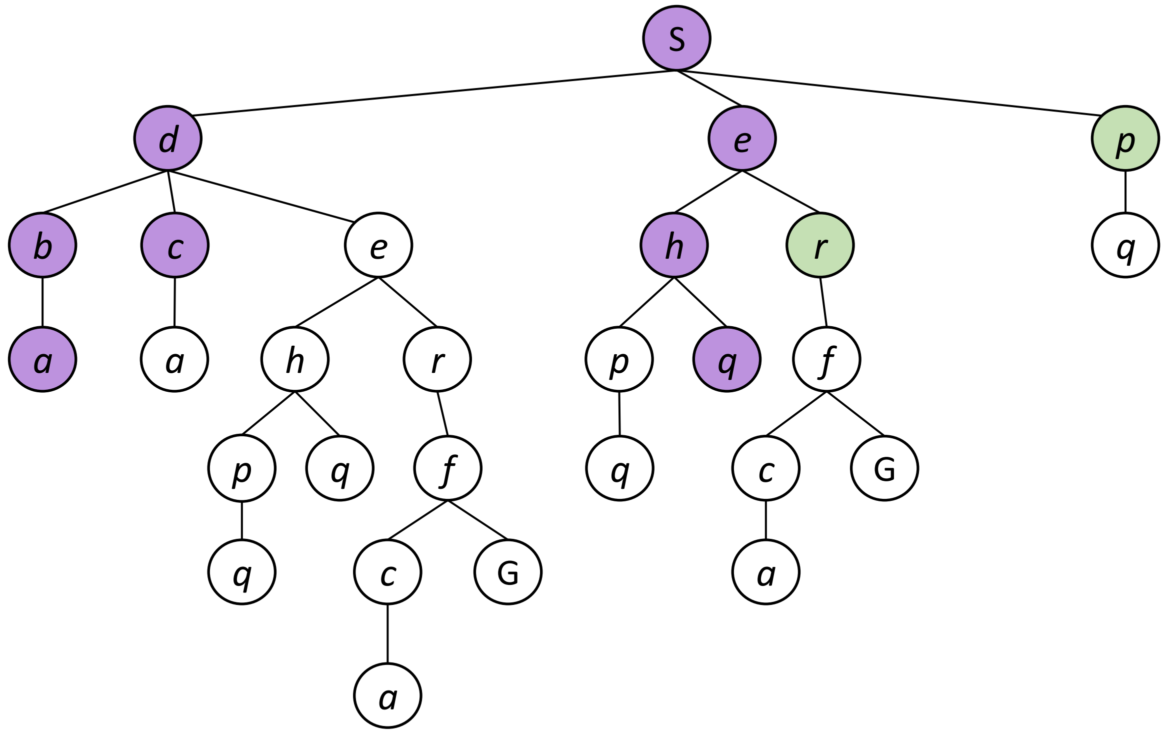
DFS



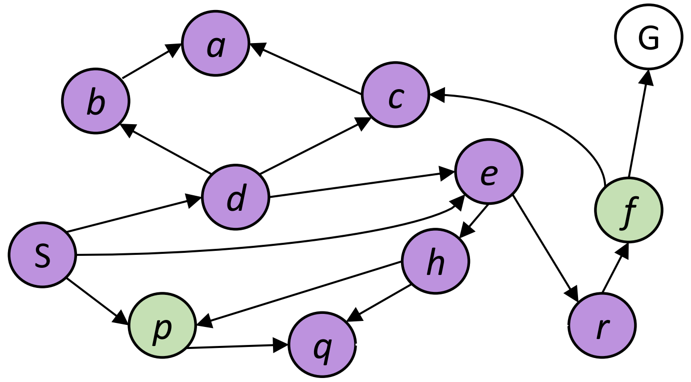


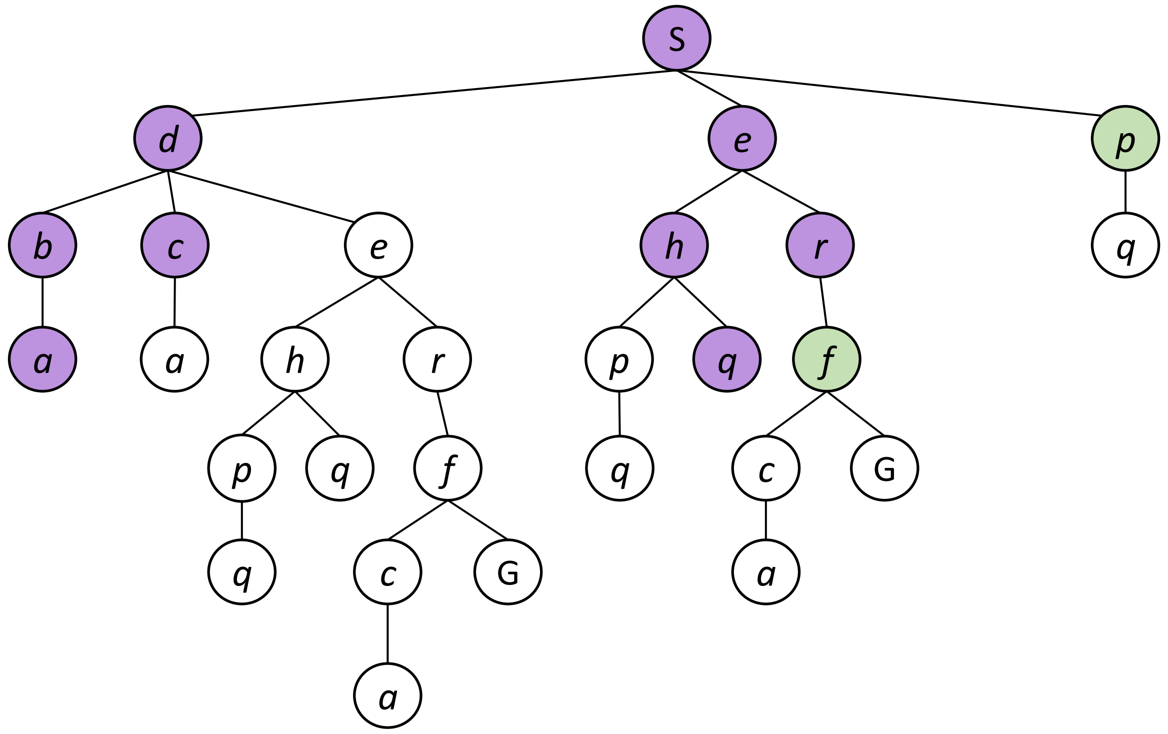
DFS



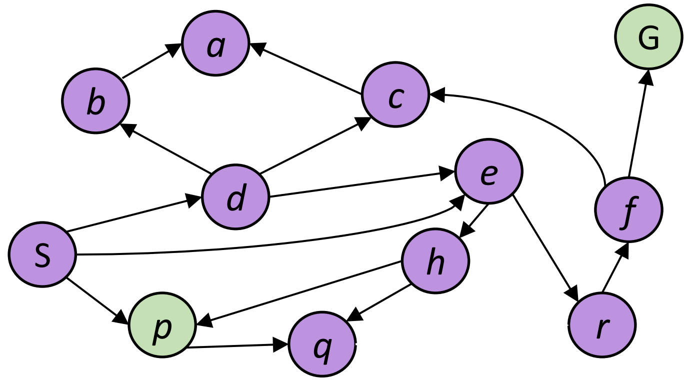


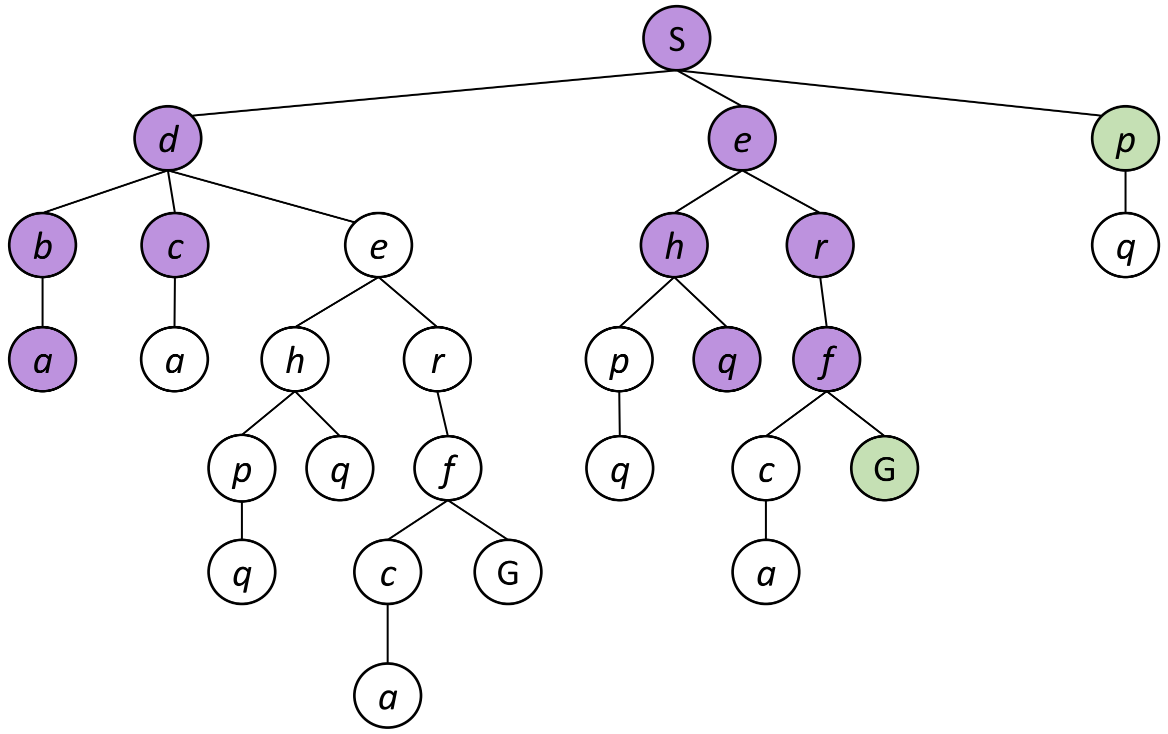
DFS





DFS

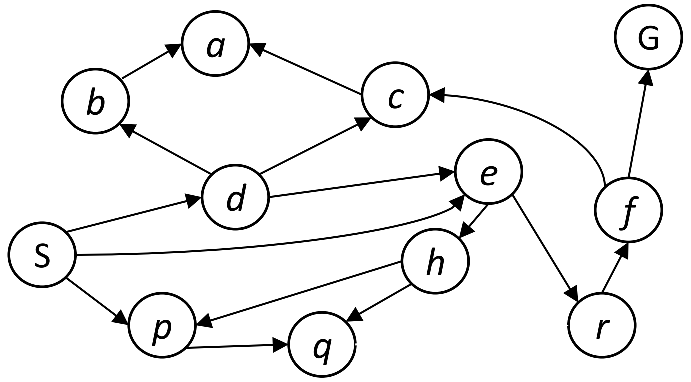


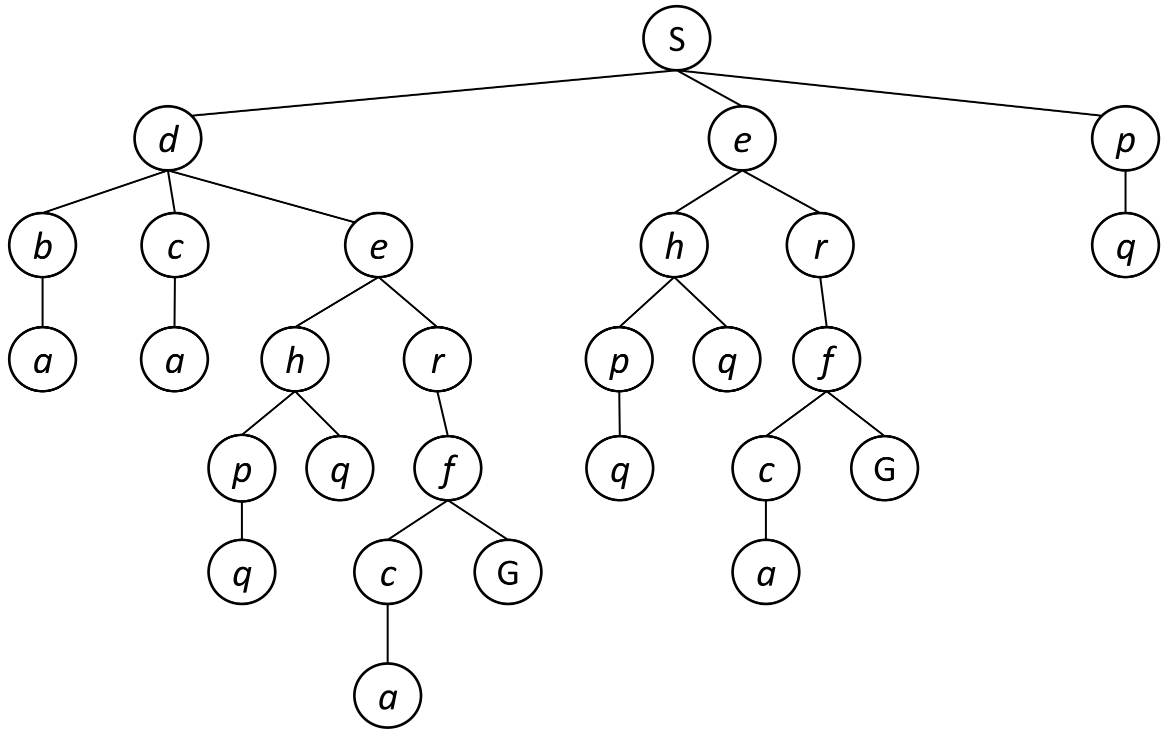


BFS

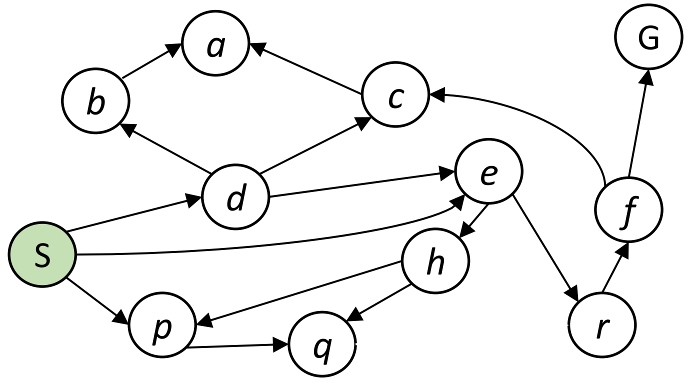
```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    Choose an oldest node $s$ in Frontier

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            If $\mathrm{succ}(s,a)$ is a goal state, terminate
            Put $\mathrm{succ}(s,a)$ in Frontier

    Move $s$ to Expanded
```

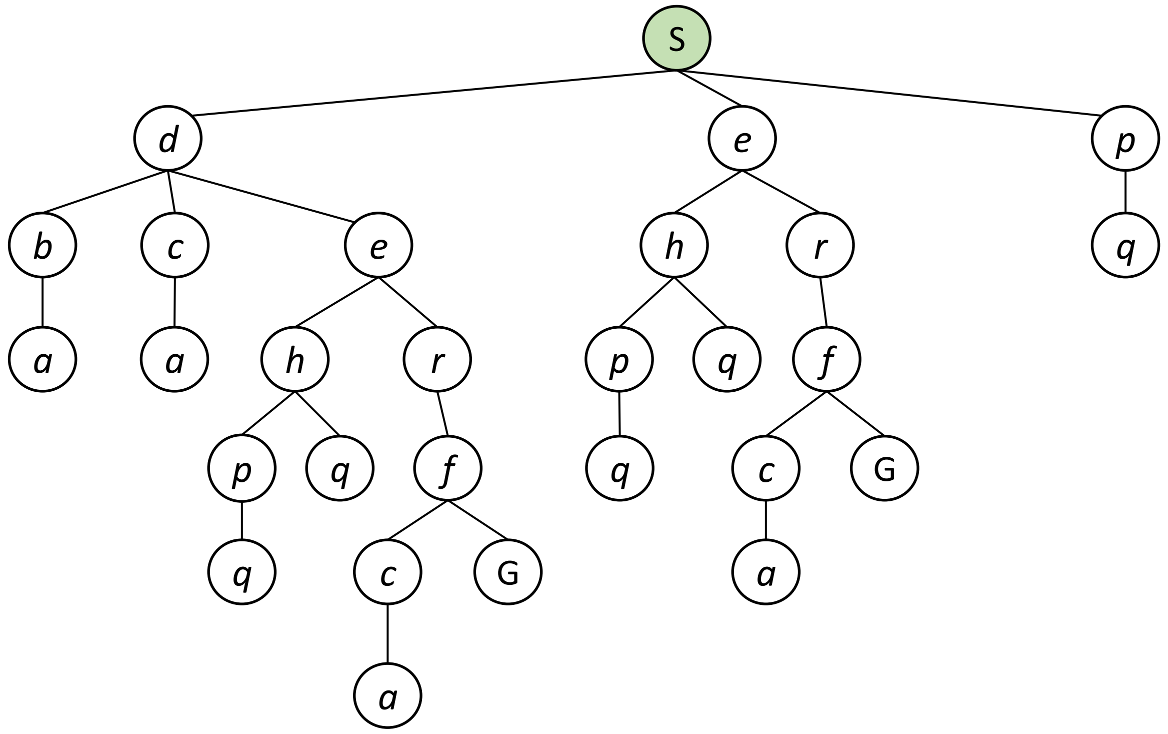
BFS



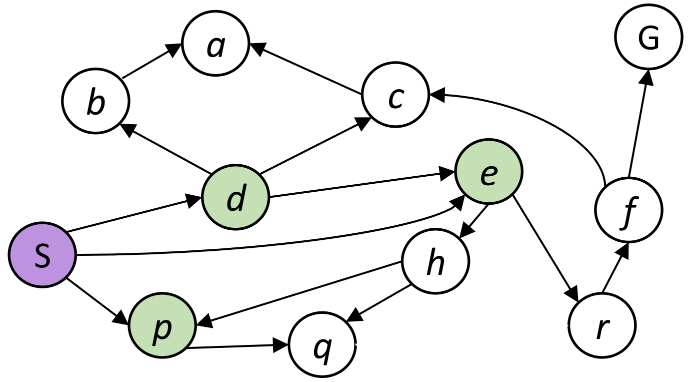


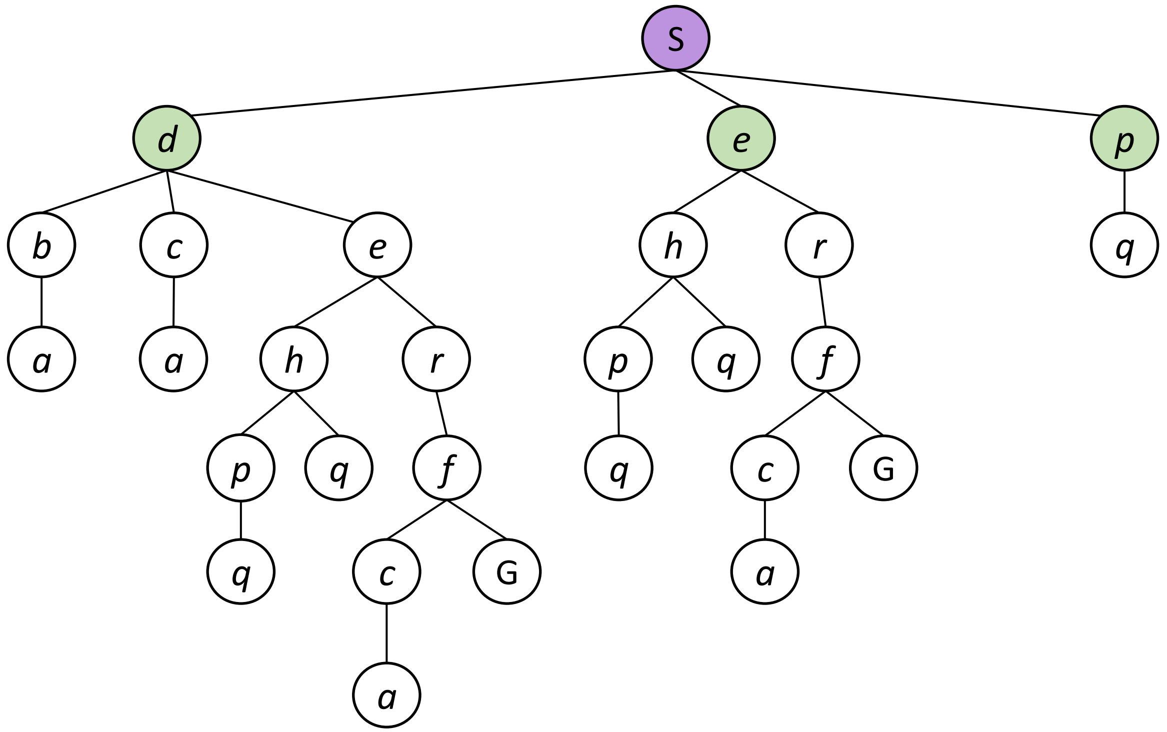
BFS



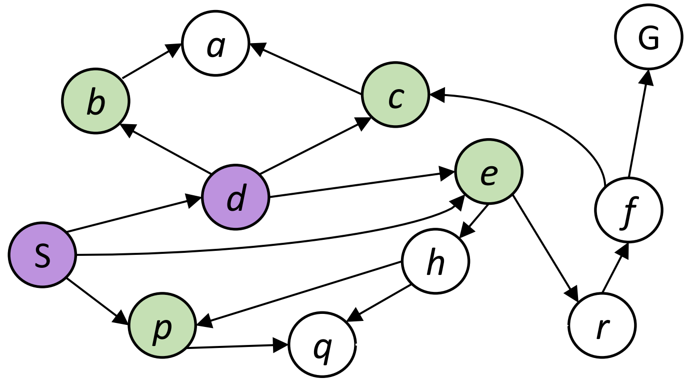


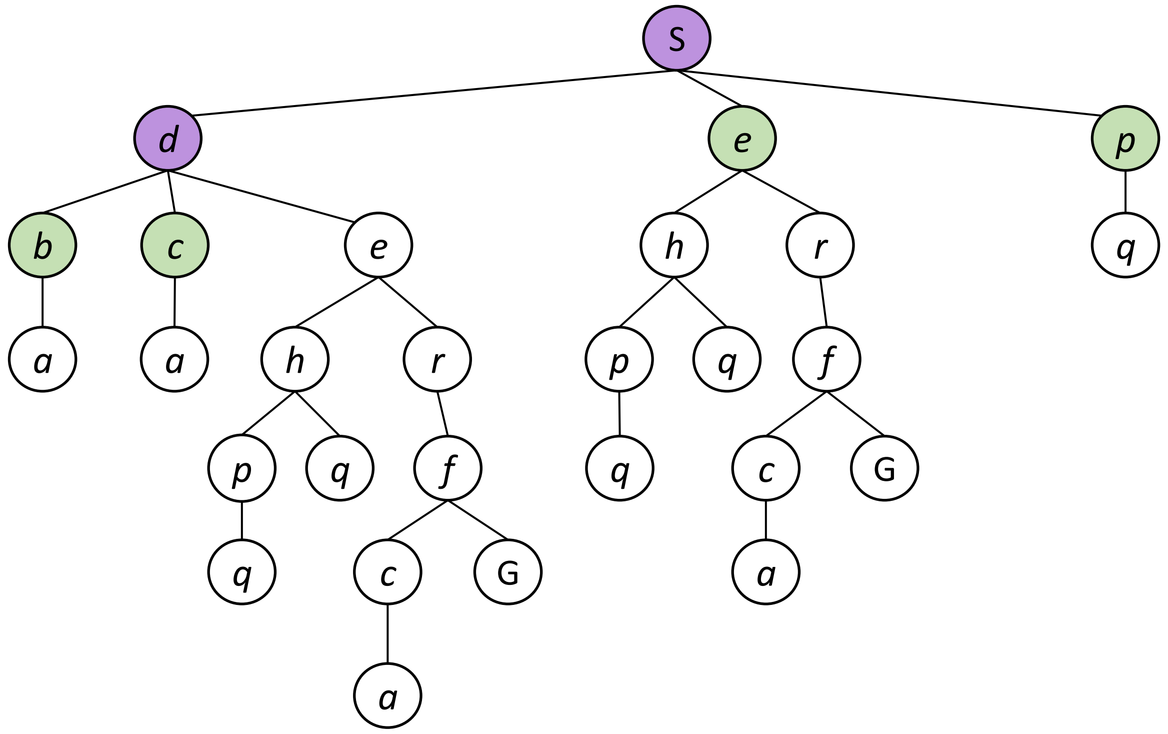
BFS



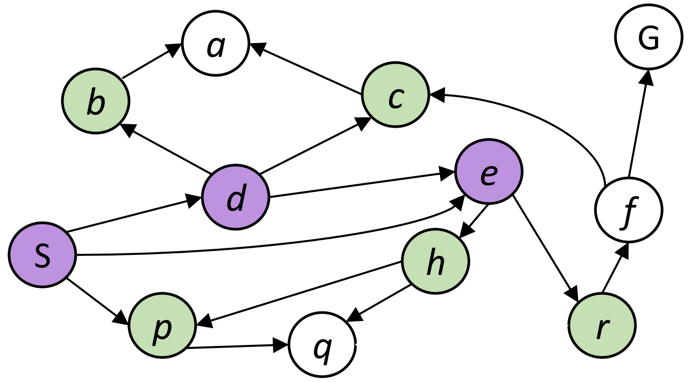


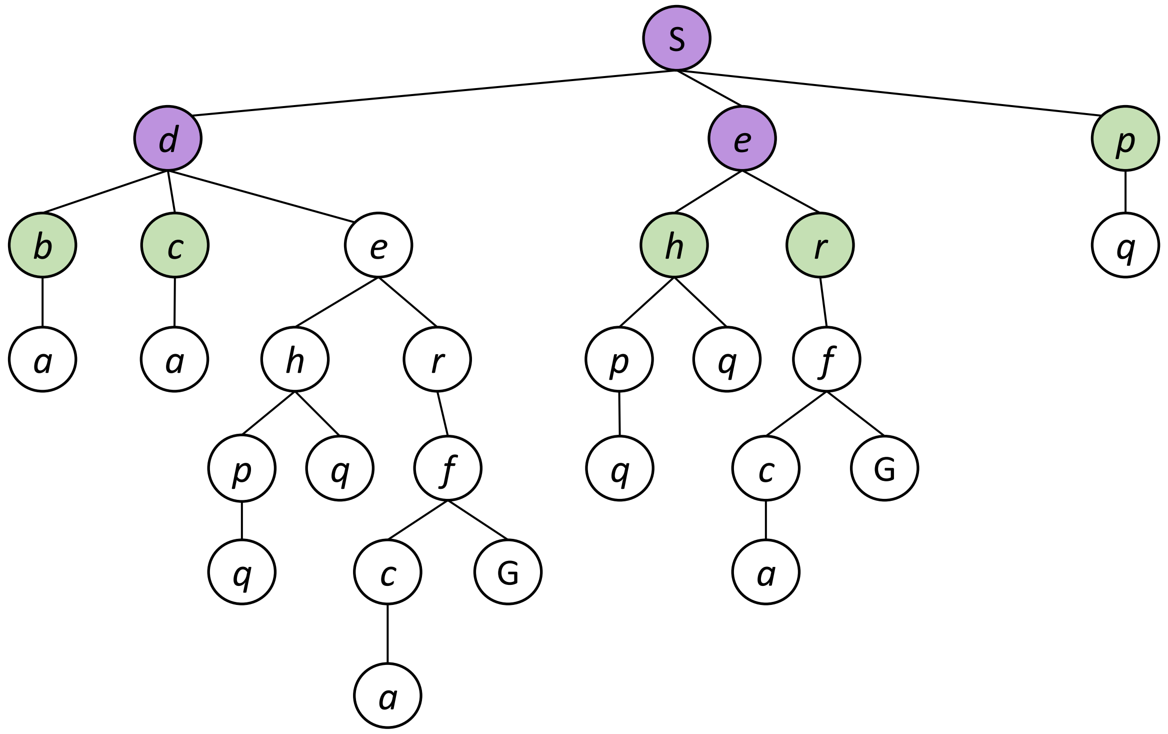
BFS



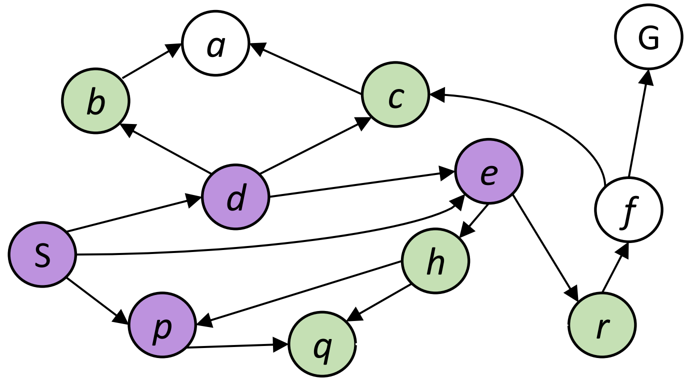


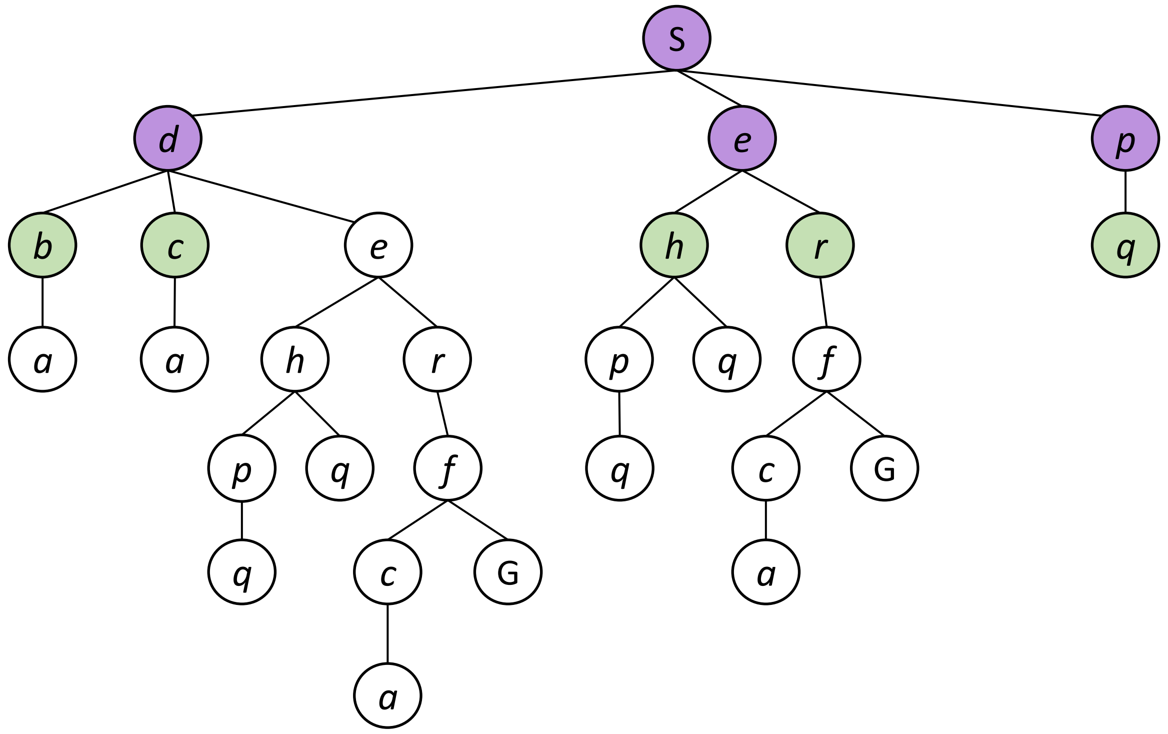
BFS



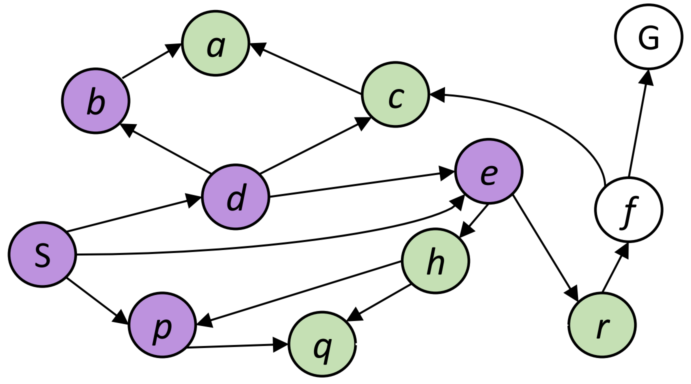


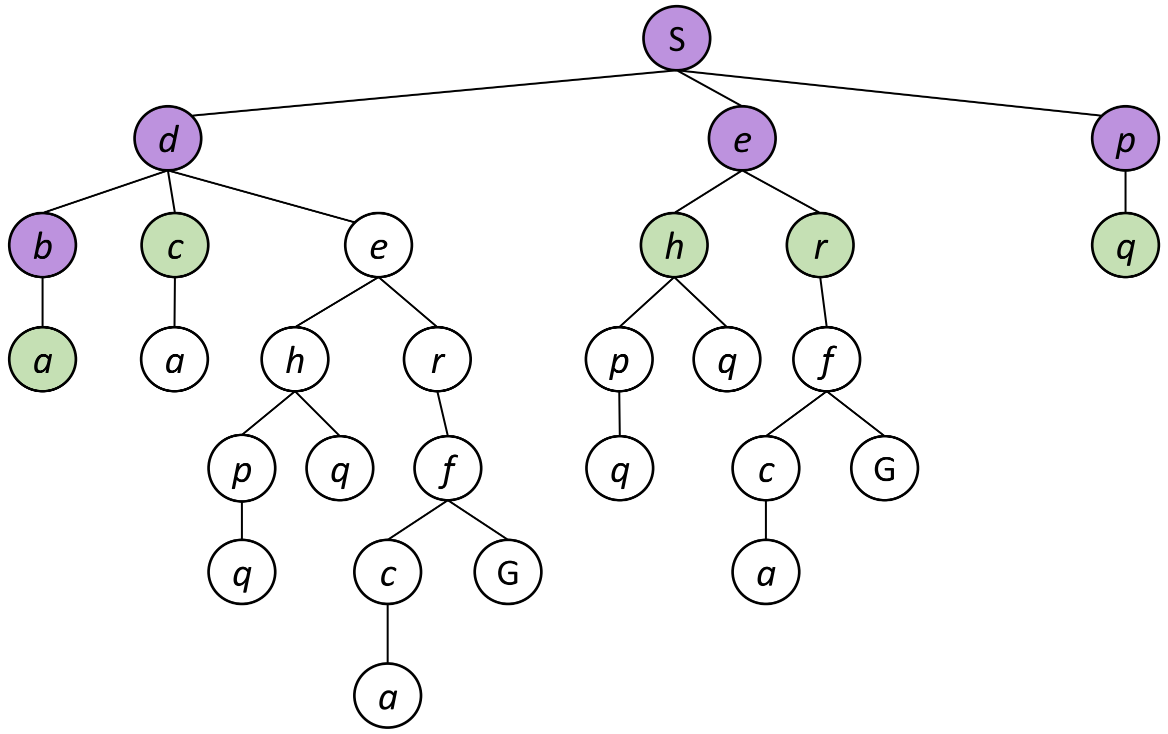
BFS



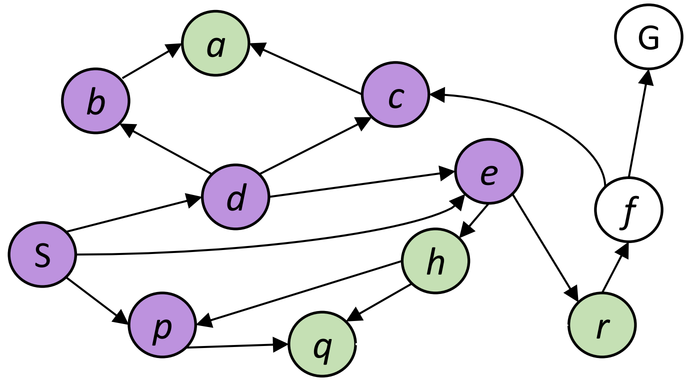


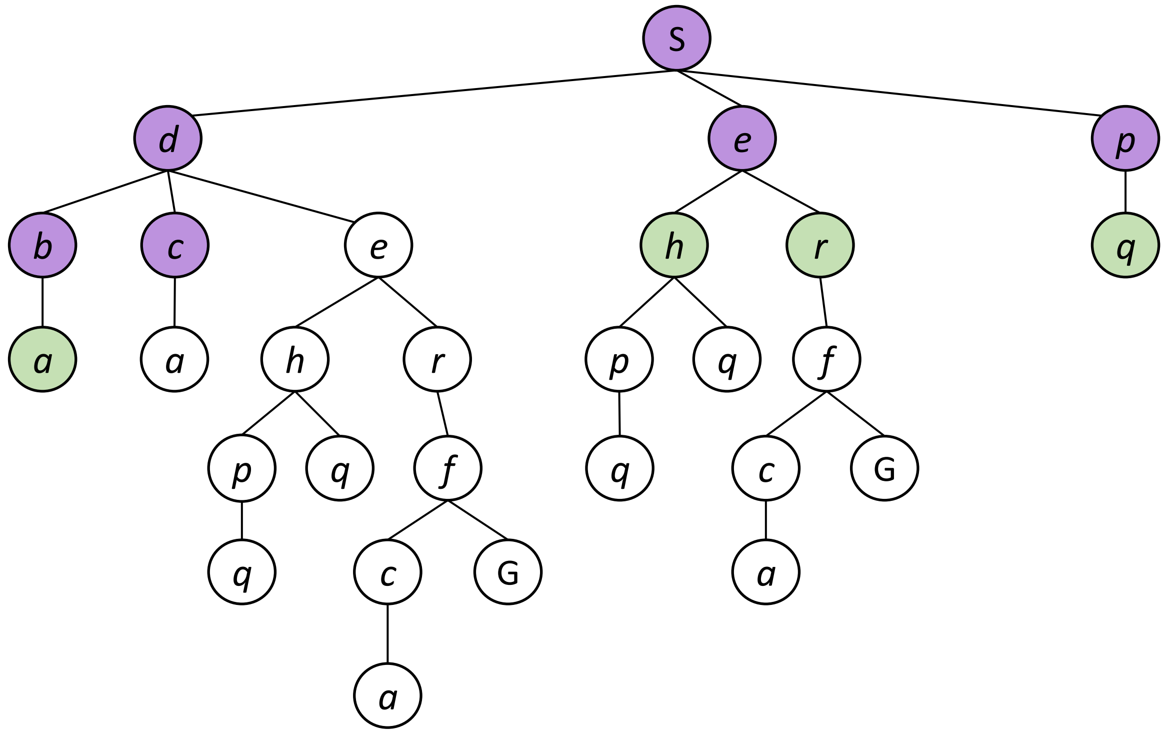
BFS



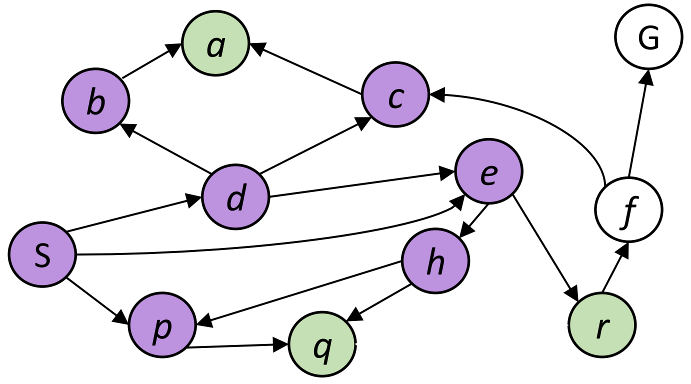


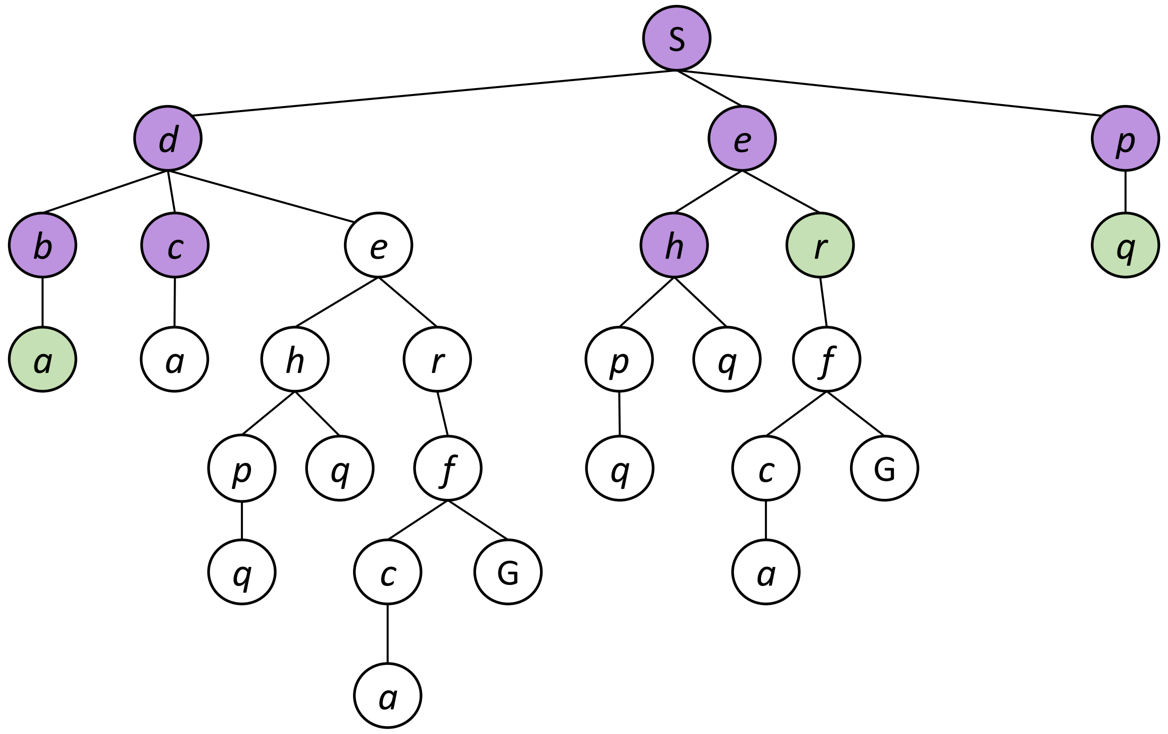
BFS



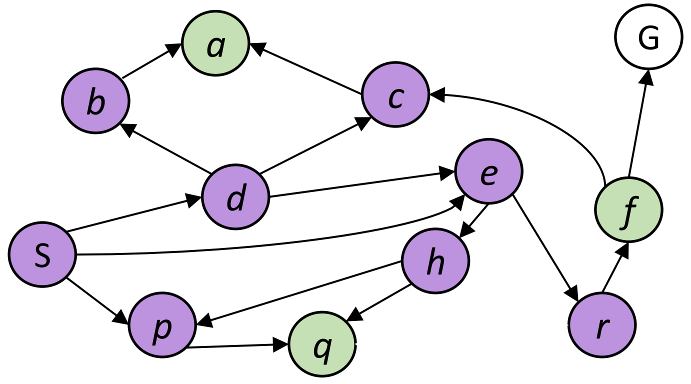


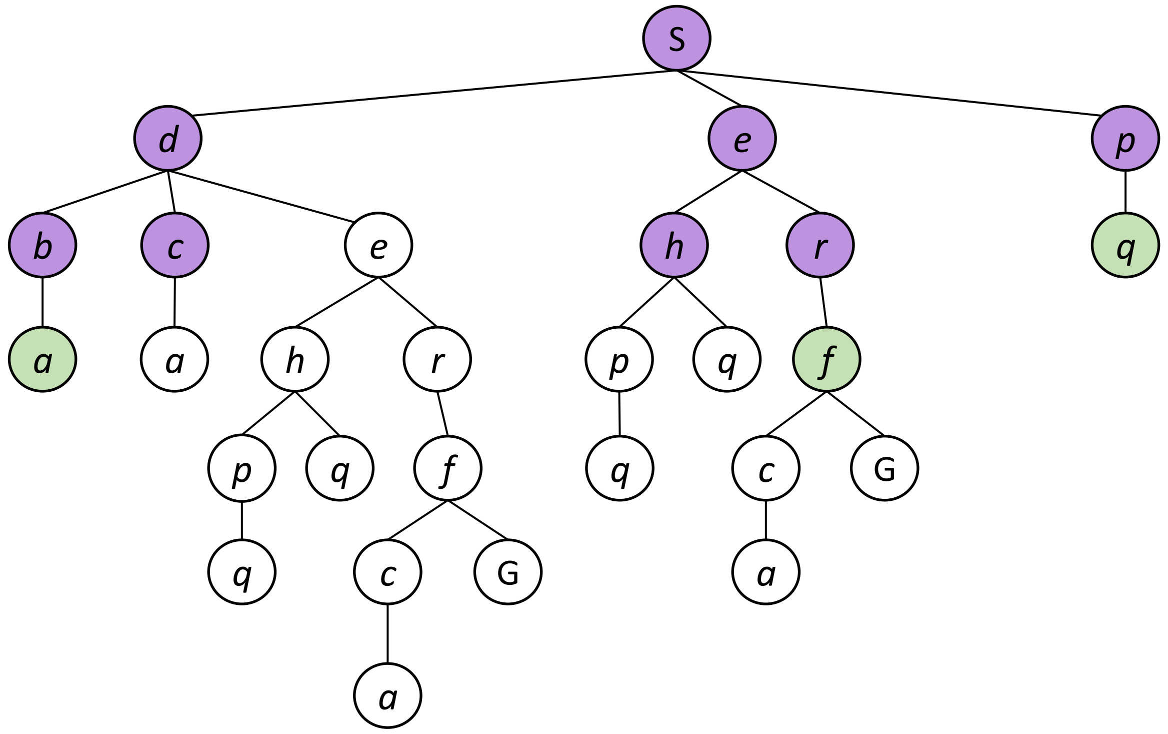
BFS



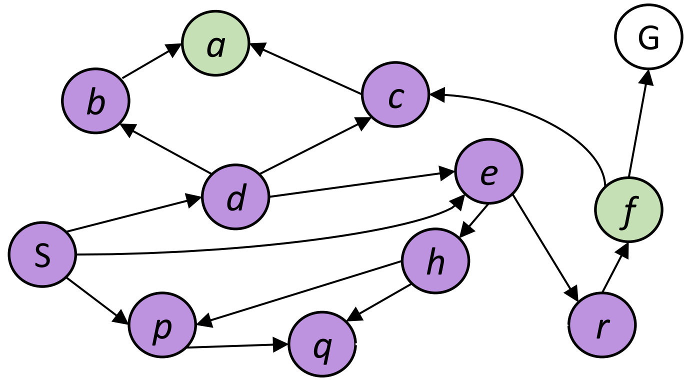


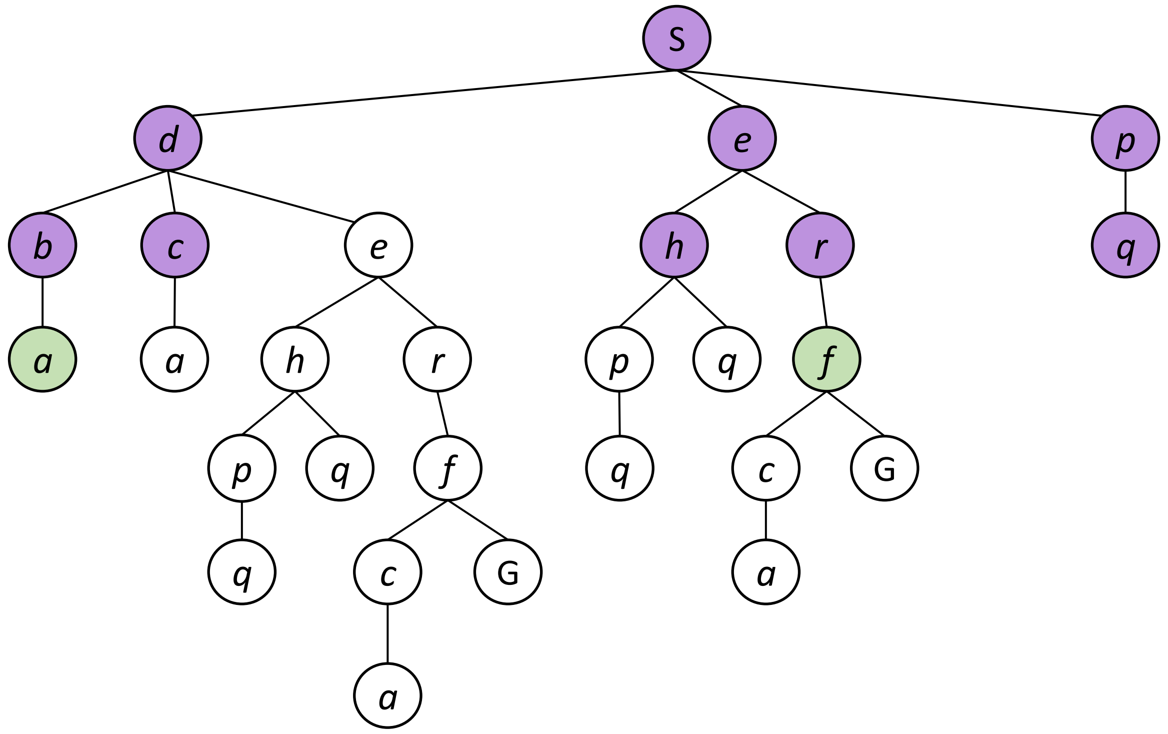
BFS



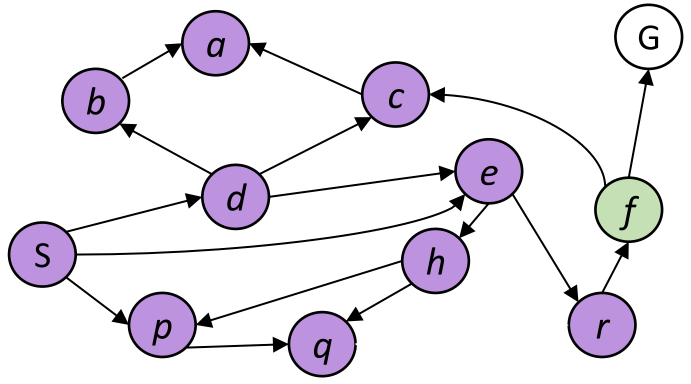


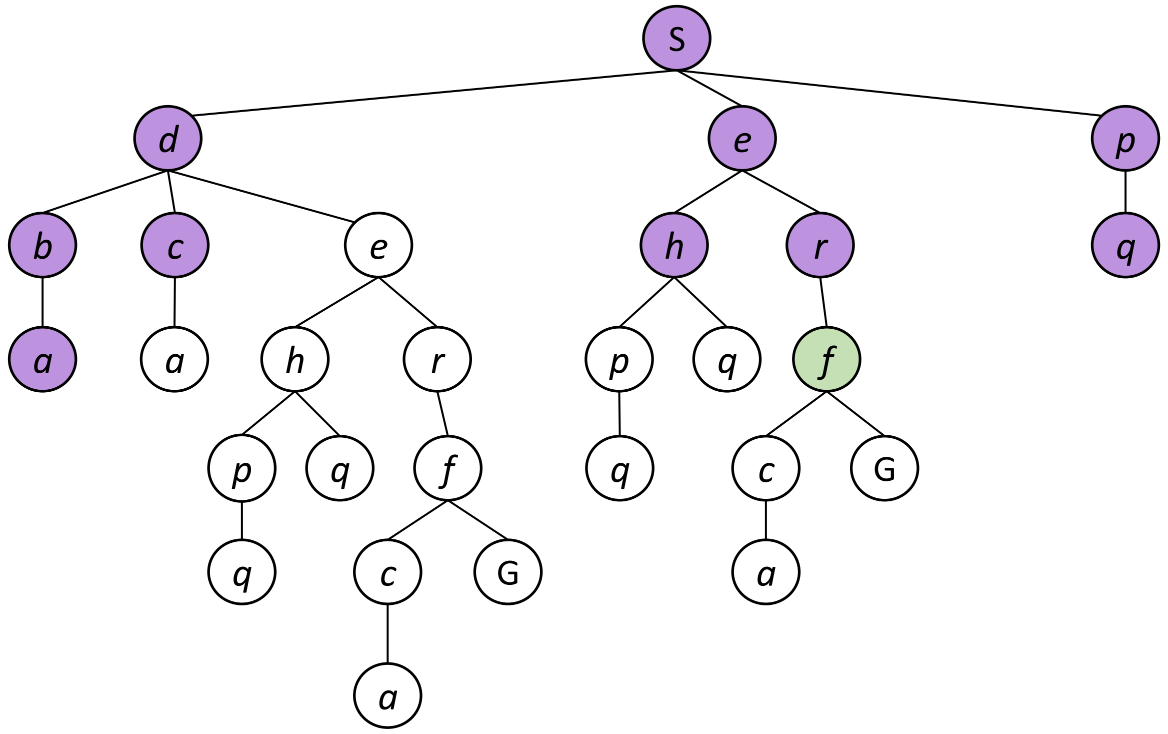
BFS



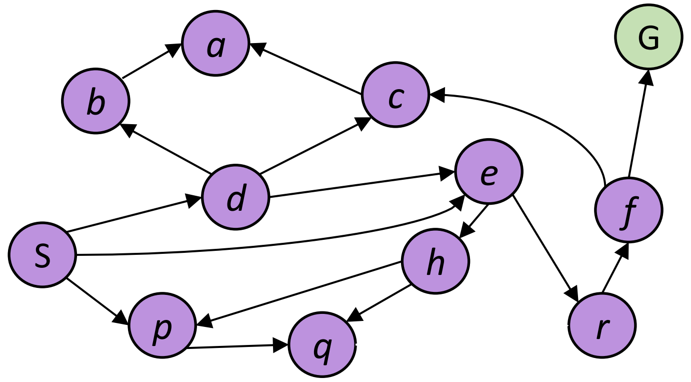


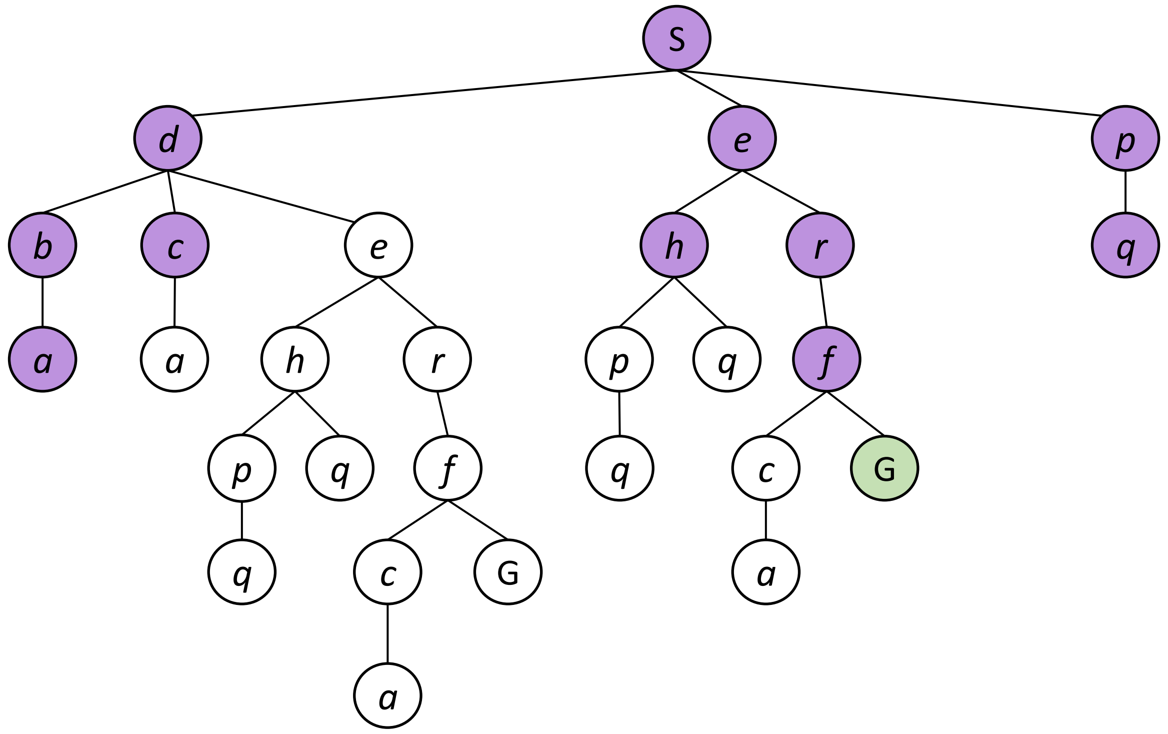
BFS





BFS



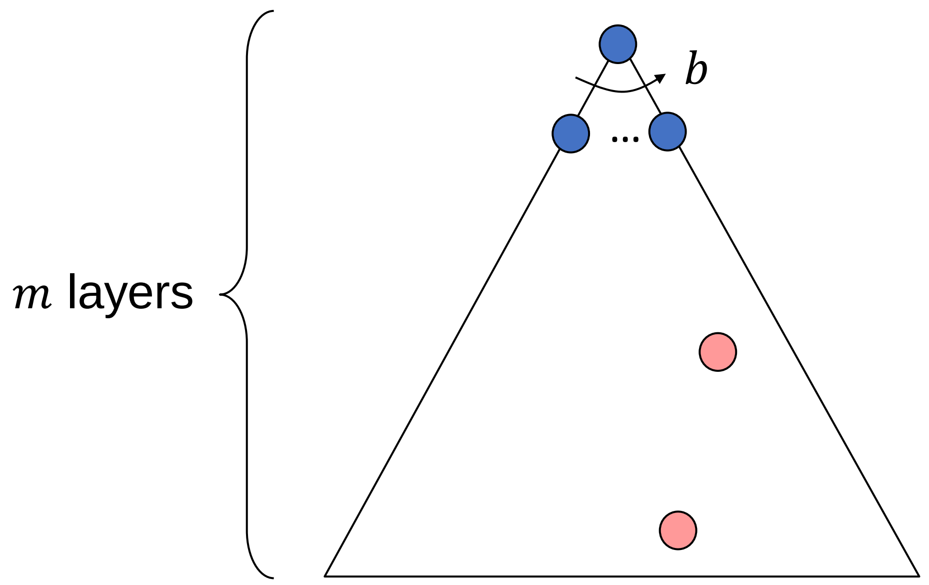


DFS vs. BFS

In what cases is DFS / BFS quicker to find the goal?

DFS vs. BFS

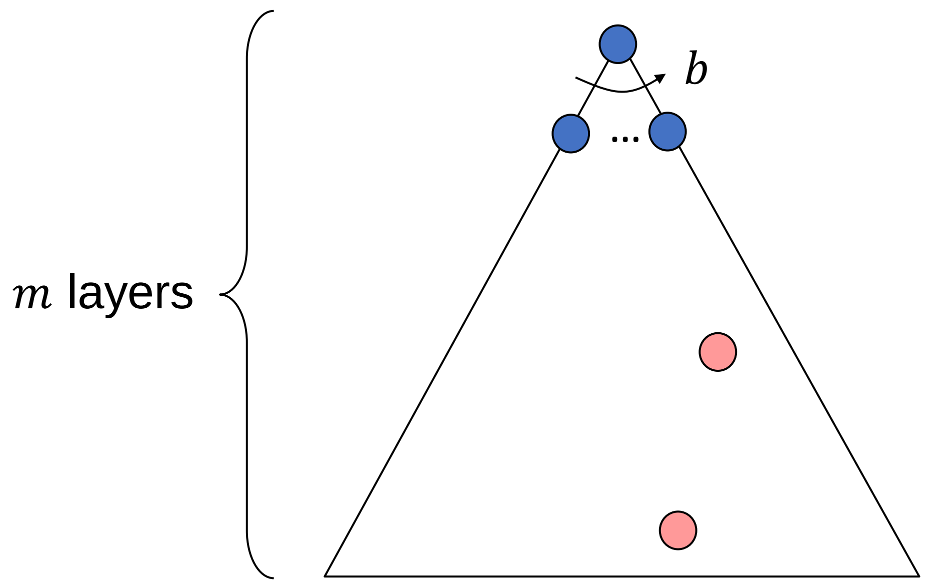
Does DFS / BFS find the goal with the smallest depth?



b: branching factor
m: maximum depth
Goals at various depths

DFS vs. BFS

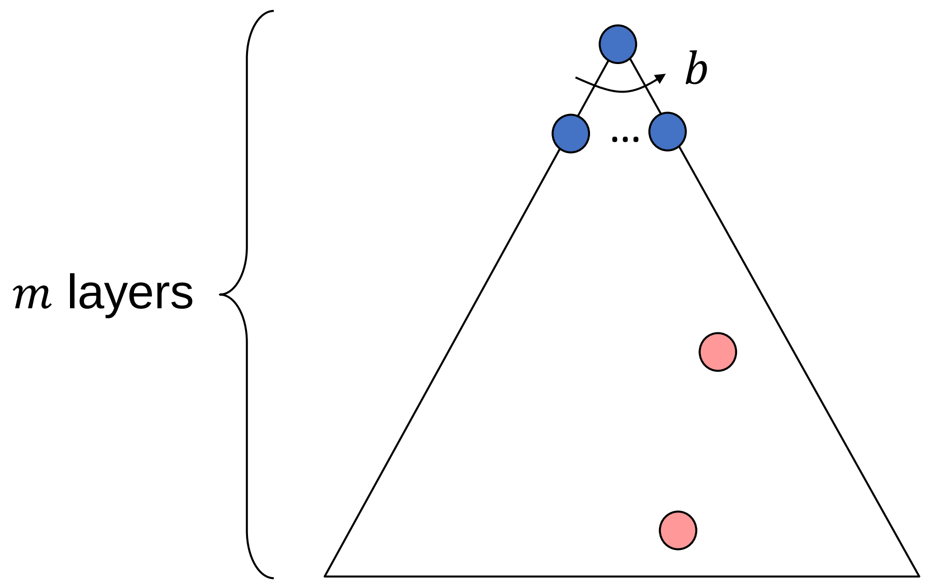
Suppose there exists a goal at layer $\leq d$. What is the time complexity for DFS / BFS to find a goal?



b: branching factor
m: maximum depth
Goals at various depths

DFS vs. BFS

What's the maximum possible size of **Frontier** in DFS / BFS?



b: branching factor
m: maximum depth
Goals at various depths

DFS vs. BFS

	Time	Frontier Size
DFS		
BFS		

So DFS can be more memory-efficient than BFS?

Yes … but not with our current implementation

DFS

```
Expanded ← { }
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a newest node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            If $\mathrm{succ}(s, a)$ is a goal state, terminate
            Put $\mathrm{succ}(s, a)$ in Frontier

    Move $s$ to Expanded
```

Memory Efficient DFS for Acyclic Graphs

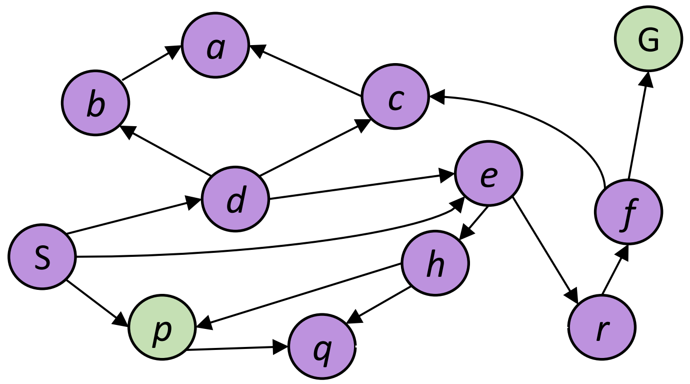
```
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a newest node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            If $\mathrm{succ}(s, a)$ is a goal state, terminate
            Put $\mathrm{succ}(s, a)$ in Frontier

    Remove $s$ from Frontier
```

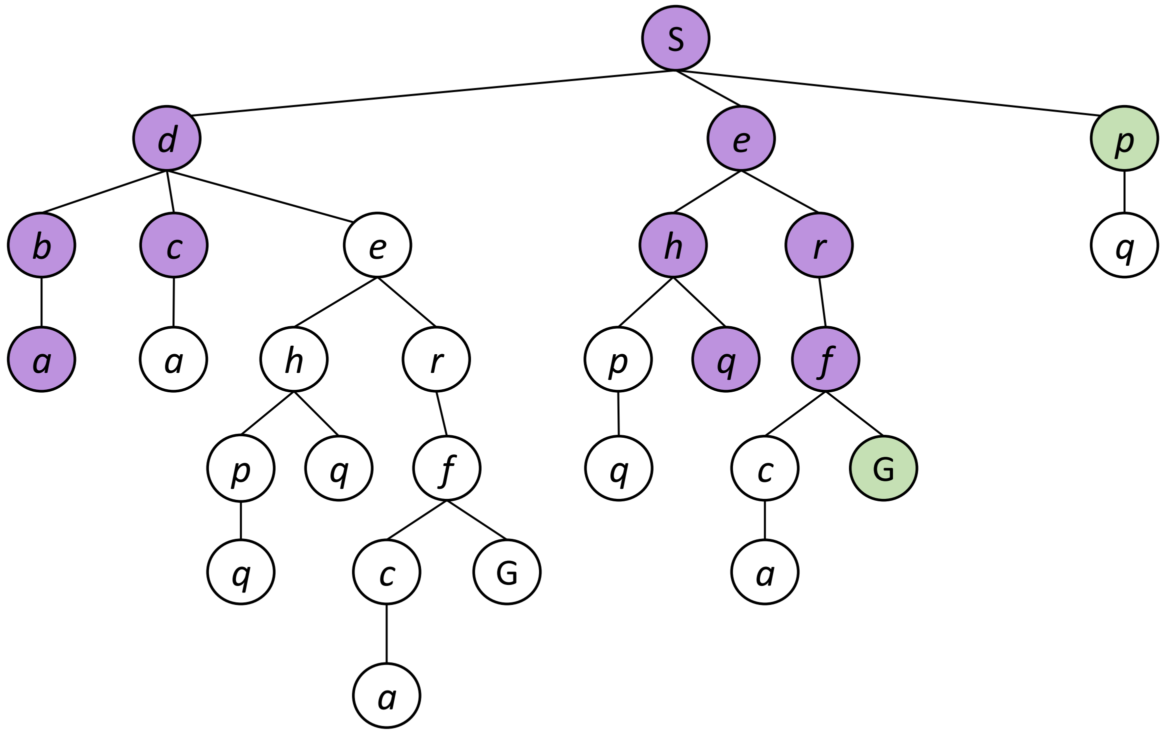
Memory Efficient DFS for Acyclic Graphs

```
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a newest node $s$ from Frontier

    For all action $a$:
        If $\mathrm{succ}(s, a)$ is a goal state, terminate
        Put $\mathrm{succ}(s, a)$ in Frontier


    Remove $s$ from Frontier
```

Because we omit the check, the algorithm may end up search in same sub-trees multiple times.

DFS (previous example)





Memory Efficient DFS for Cyclic Graphs

```
**Frontier** ← { initial_state }
While **Frontier** is not empty:
    Choose a newest node $s$ from **Frontier**

    For all action $a$:
        If $\mathrm{succ}(s, a)$ is a goal state, terminate
        Put $\mathrm{succ}(s, a)$ in **Frontier**



    Remove $s$ from **Frontier**
```

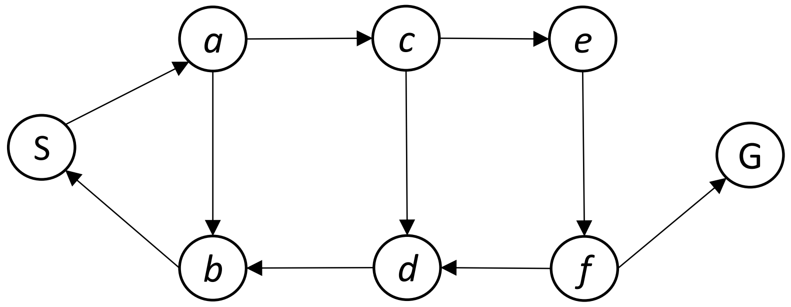
Memory Efficient DFS for Cyclic Graphs

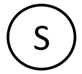
```
Frontier ← { initial_state }
While Frontier is not empty:
    Choose a newest node $s$ from Frontier

    For all action $a$:
        If $\text{succ}(s, a)$ is a goal state, terminate
        If $\text{succ}(s, a)$ is not an ancestor of $s$:
            Put $\text{succ}(s, a)$ in Frontier

    Remove $s$ from Frontier
```

Prevent cycle

Memory Efficient DFS

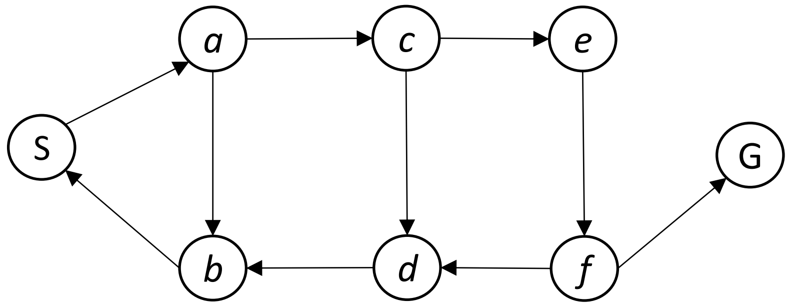
(handling cycles)





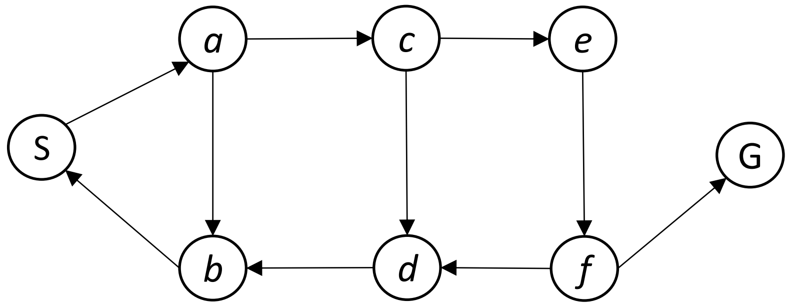
Memory Efficient DFS

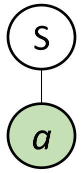
(handling cycles)



Memory Efficient DFS

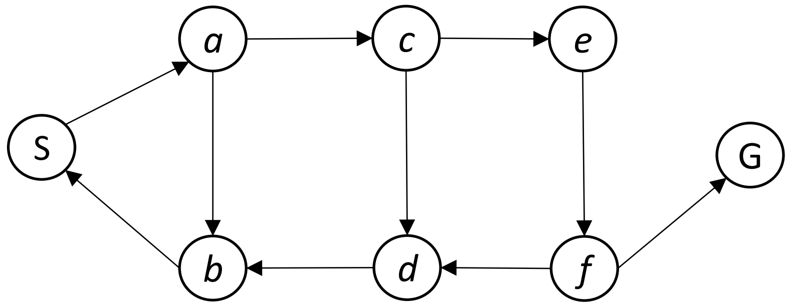
(handling cycles)

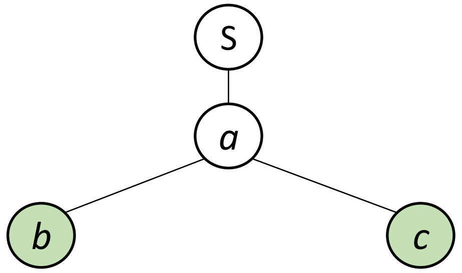




Memory Efficient DFS

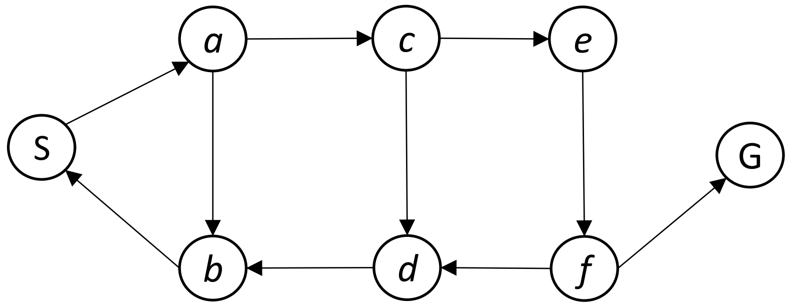
(handling cycles)

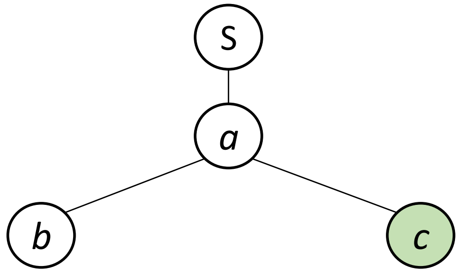




Memory Efficient DFS

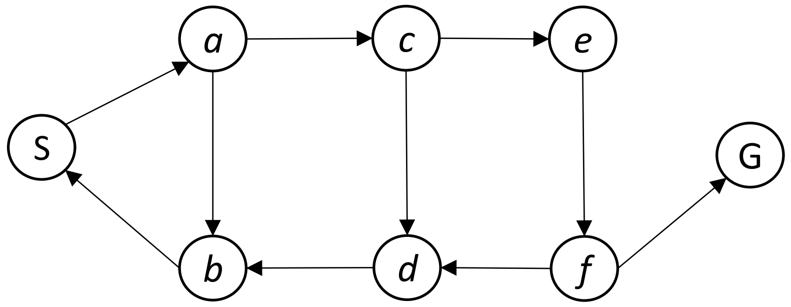
(handling cycles)

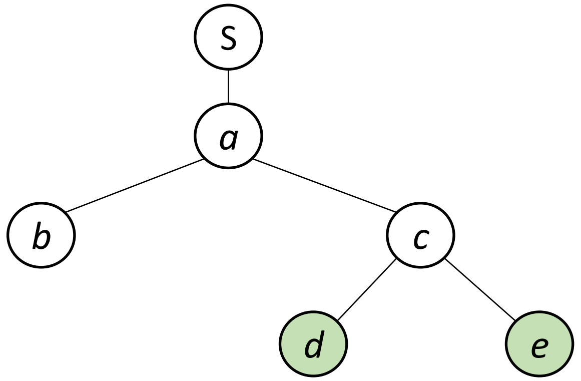




Memory Efficient DFS

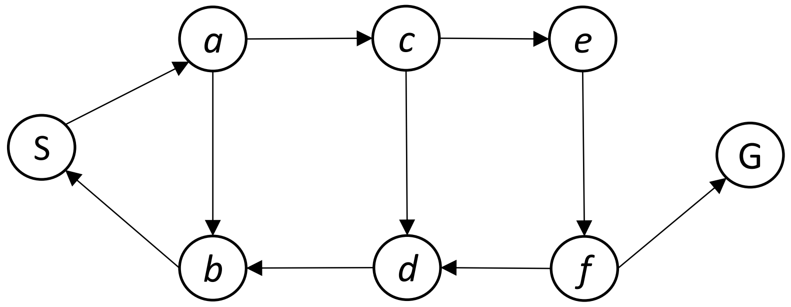
(handling cycles)

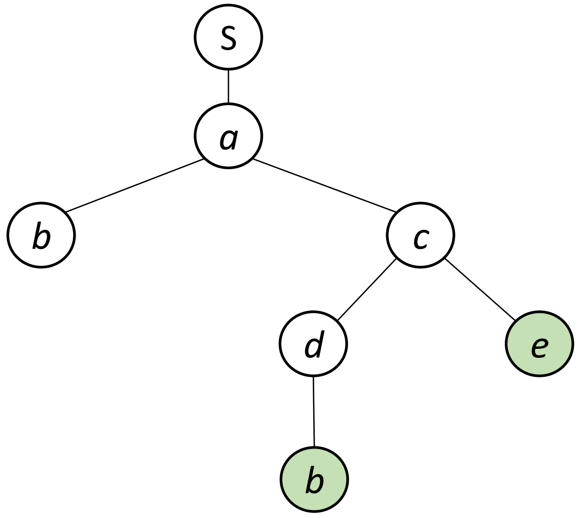




Memory Efficient DFS

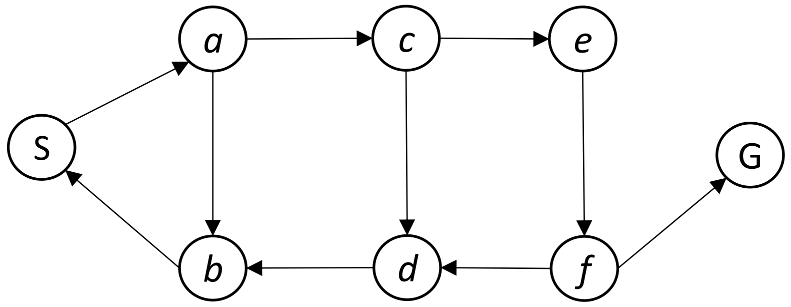
(handling cycles)

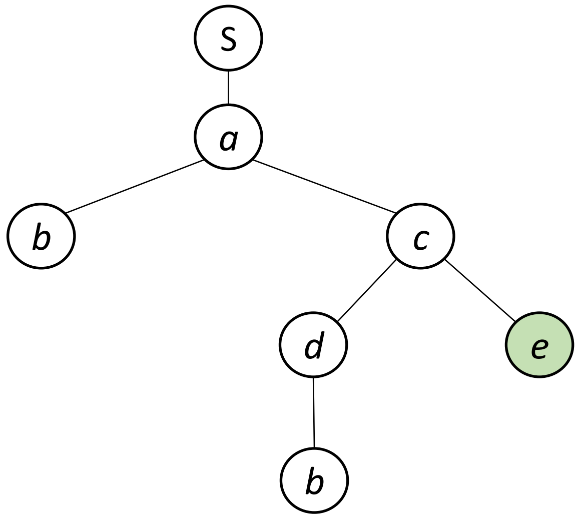




Memory Efficient DFS

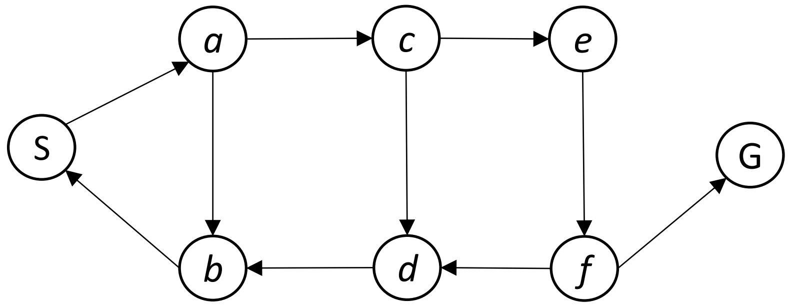
(handling cycles)

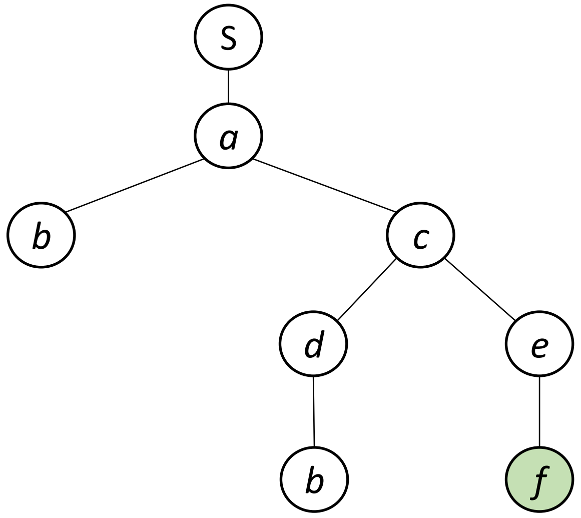




Memory Efficient DFS

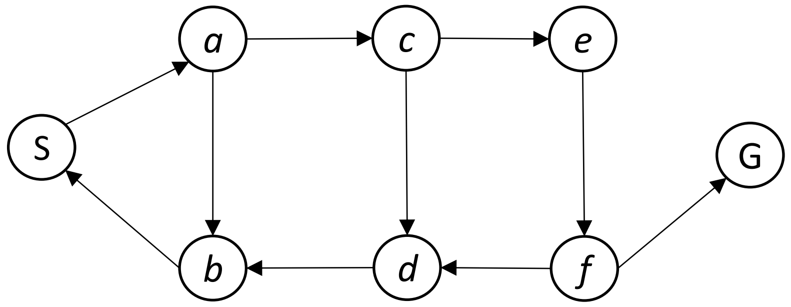
(handling cycles)

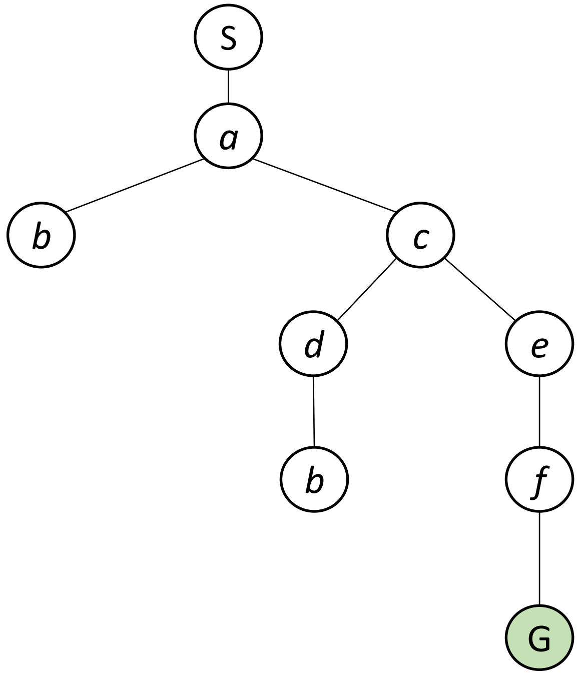




Memory Efficient DFS

(handling cycles)





DFS vs. BFS

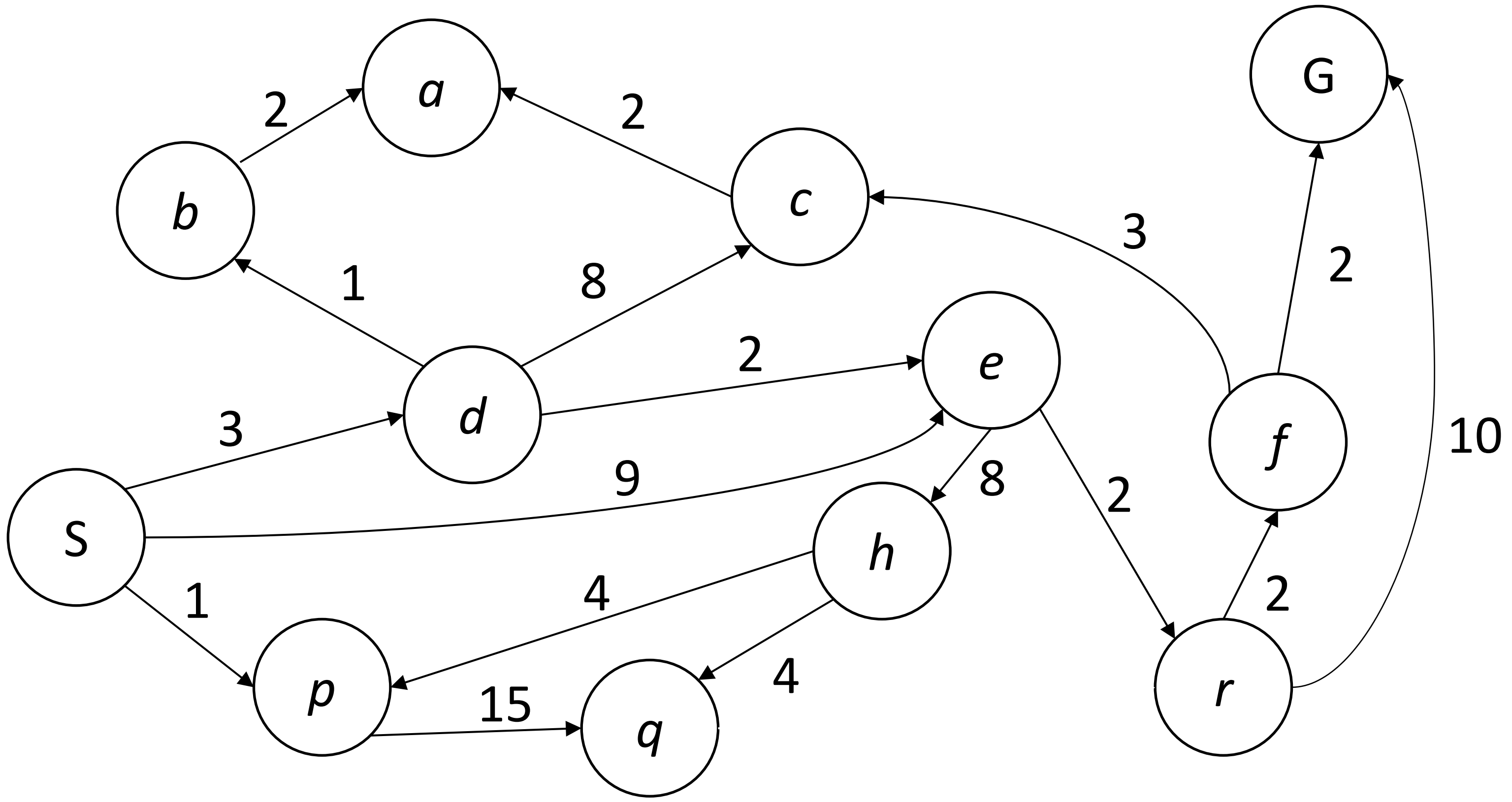
	Time	Space
(memory-efficient) DFS		
BFS		
IDS		

Iterative Deepening Search (IDS)

- Idea: get DFS’s space advantage with BFS’s time advantage
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn’t that wastefully redundant?
  - Generally most work happens in the last level
  - Branching factor 10, solution 5 deep:
    - BFS: 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110
    - IDS: 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450

Cost-Sensitive Search Problem



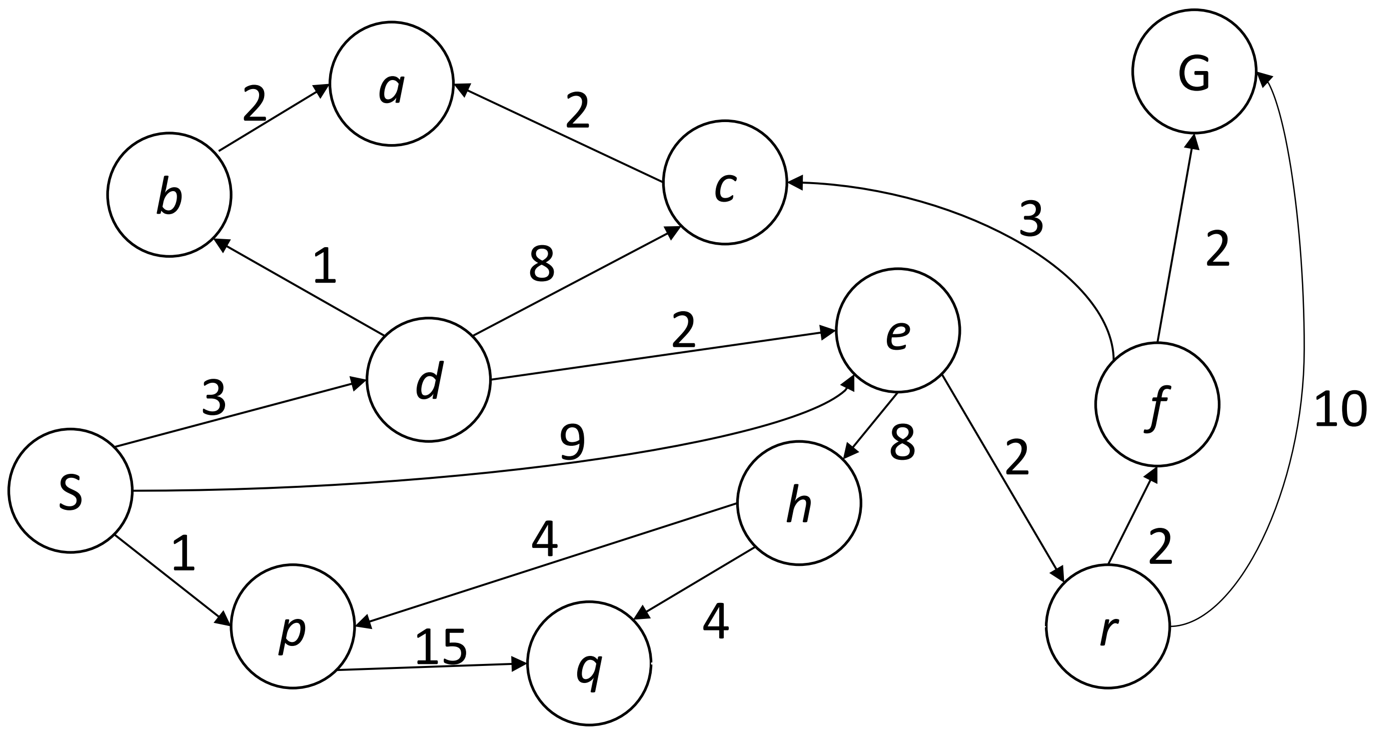
Recall the General Framework

```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier
    For all action $a$:
        If $\mathrm{succ}(s, a)$ has not been reached:
            Put $\mathrm{succ}(s, a)$ in Frontier
    Move $s$ to Expanded
```

Uniform Cost Search (Dijkstra)

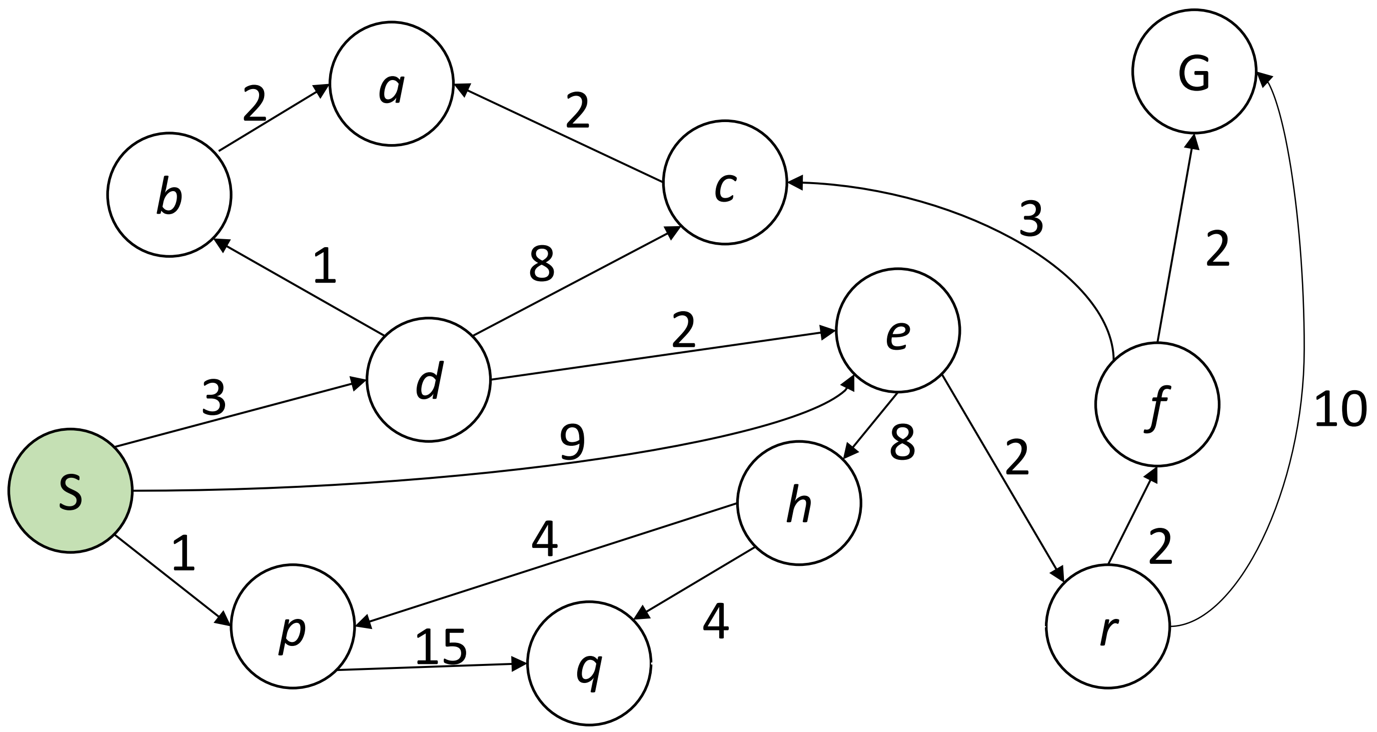
```
Expanded ← { }
Frontier ← { initial_state }
Loop:
    Choose a node $s$ from Frontier (Choose the one with smallest $g(s)$)
    If $s$ is a goal state, then terminate

    For all action $a$:
        If $\mathrm{succ}(s,a)$ has not been reached:
            Put $\mathrm{succ}(s,a)$ in Frontier
        $g\big(\mathrm{succ}(s,a)\big) \leftarrow \min\{\ g\big(\mathrm{succ}(s,a)\big),\quad g(s)+\mathrm{cost}(s,a)\ \}$

    Move $s$ to Expanded
```

UCS



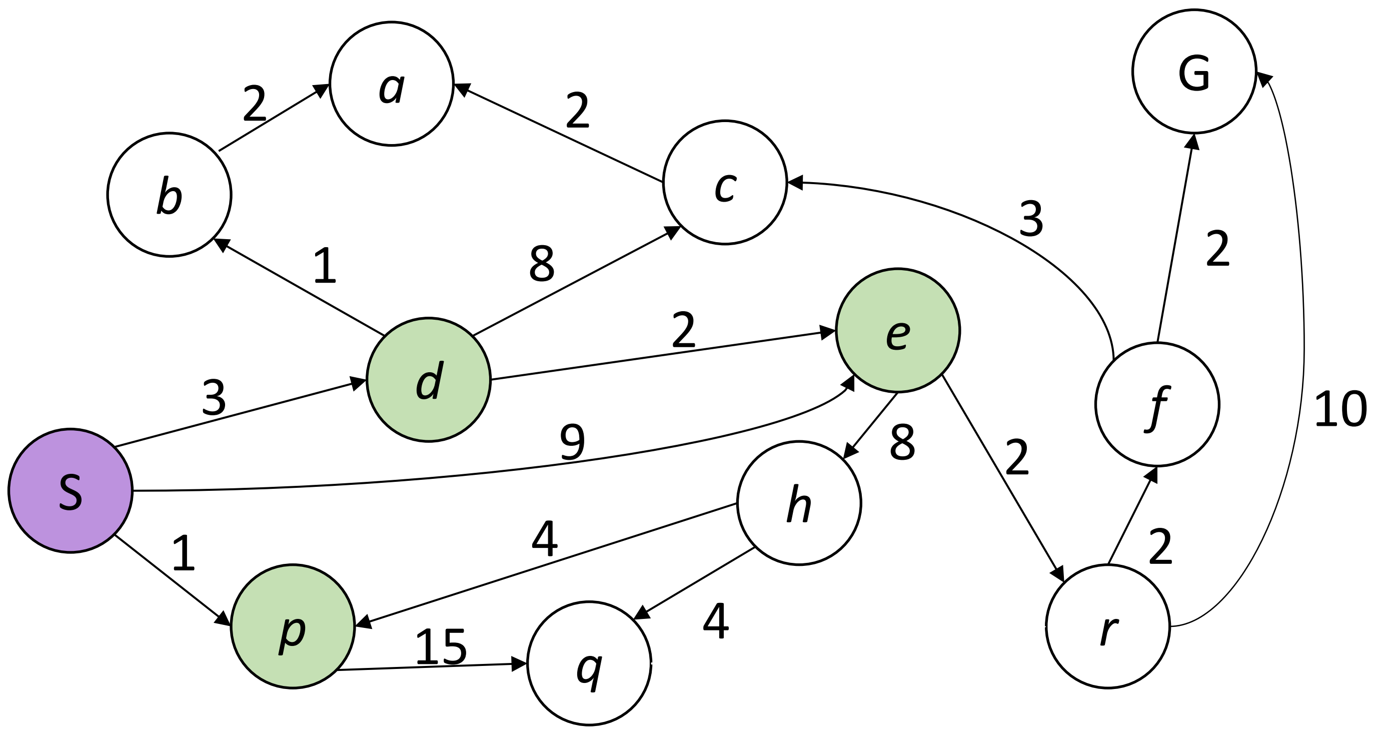
x	$g(x)$
S	
a	
b	
c	
d	
e	
f	
h	
p	
q	
r	
G	

UCS



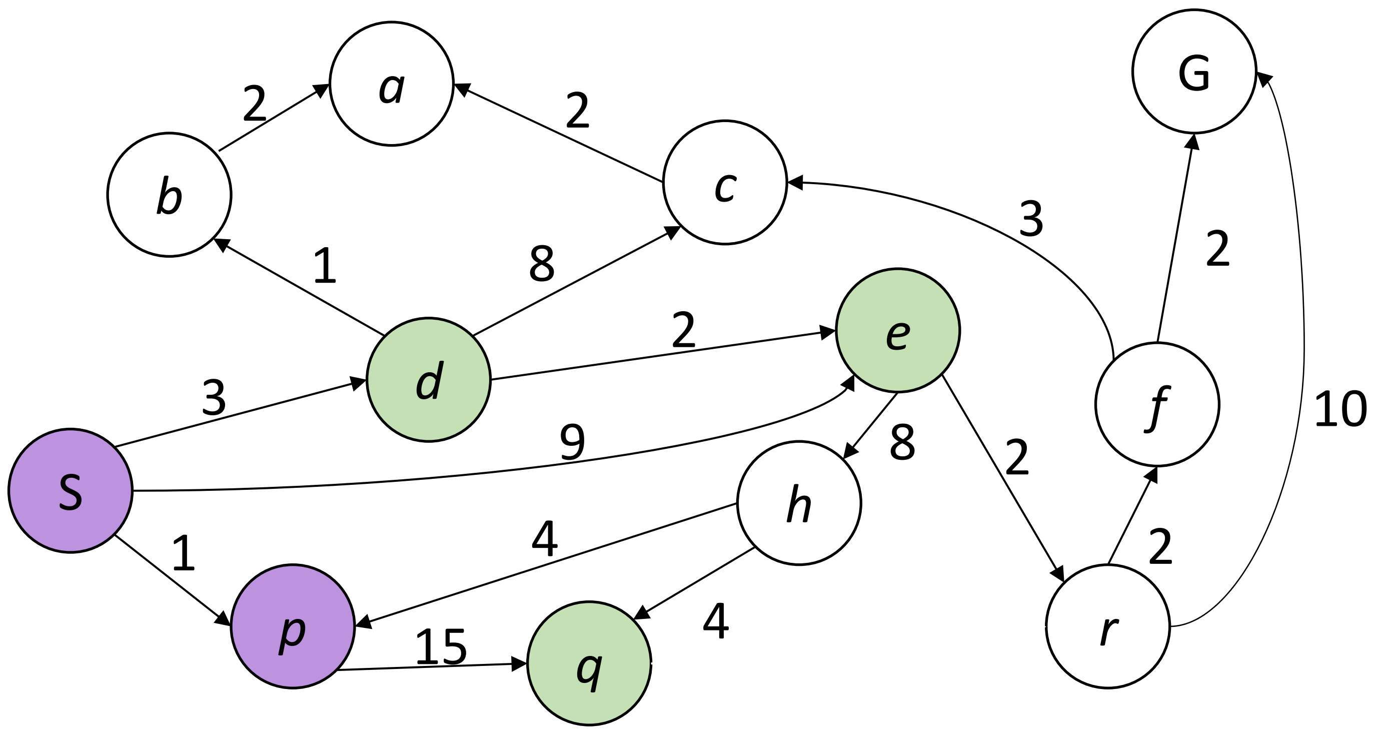
x	$g(x)$
S	0
a	∞
b	∞
c	∞
d	∞
e	∞
f	∞
h	∞
p	∞
q	∞
r	∞
G	∞

UCS



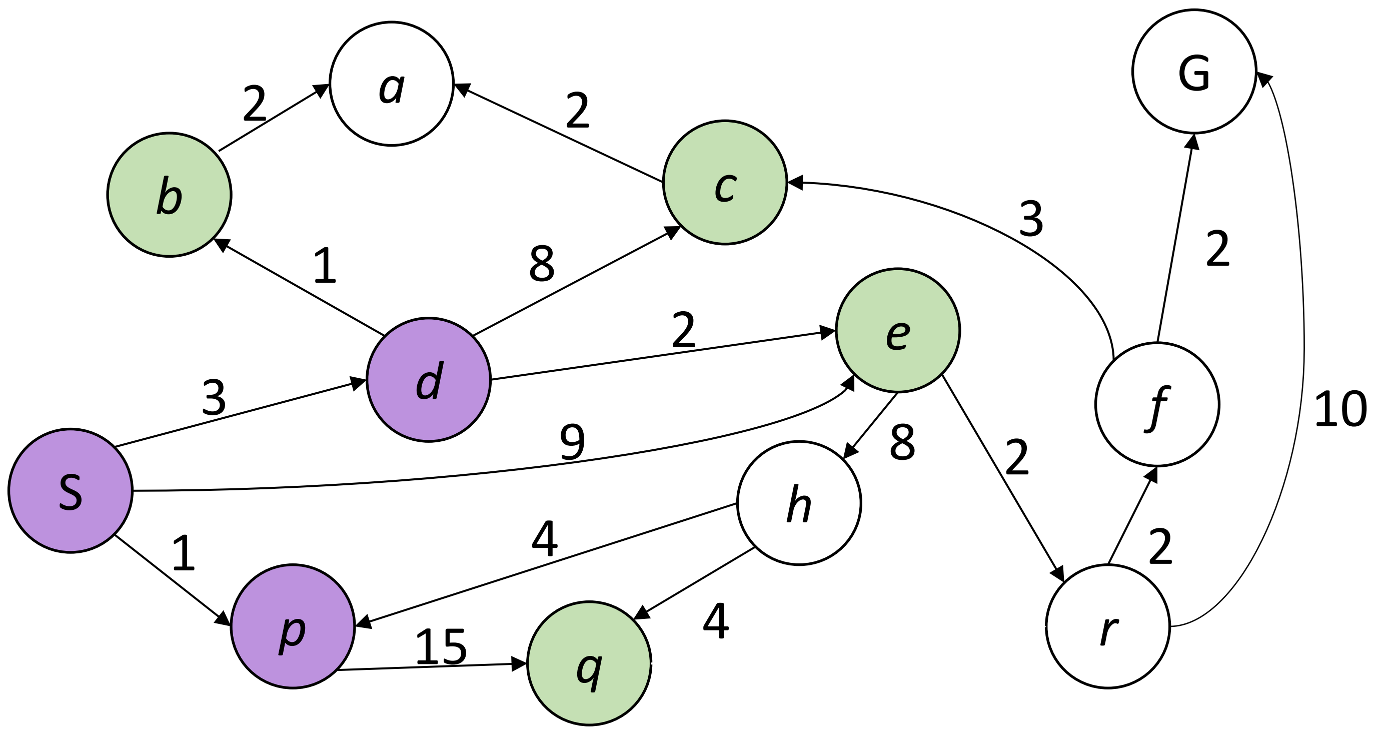
x	$g(x)$
S	0
a	∞
b	∞
c	∞
d	3
e	9
f	∞
h	∞
p	1
q	∞
r	∞
G	∞

UCS



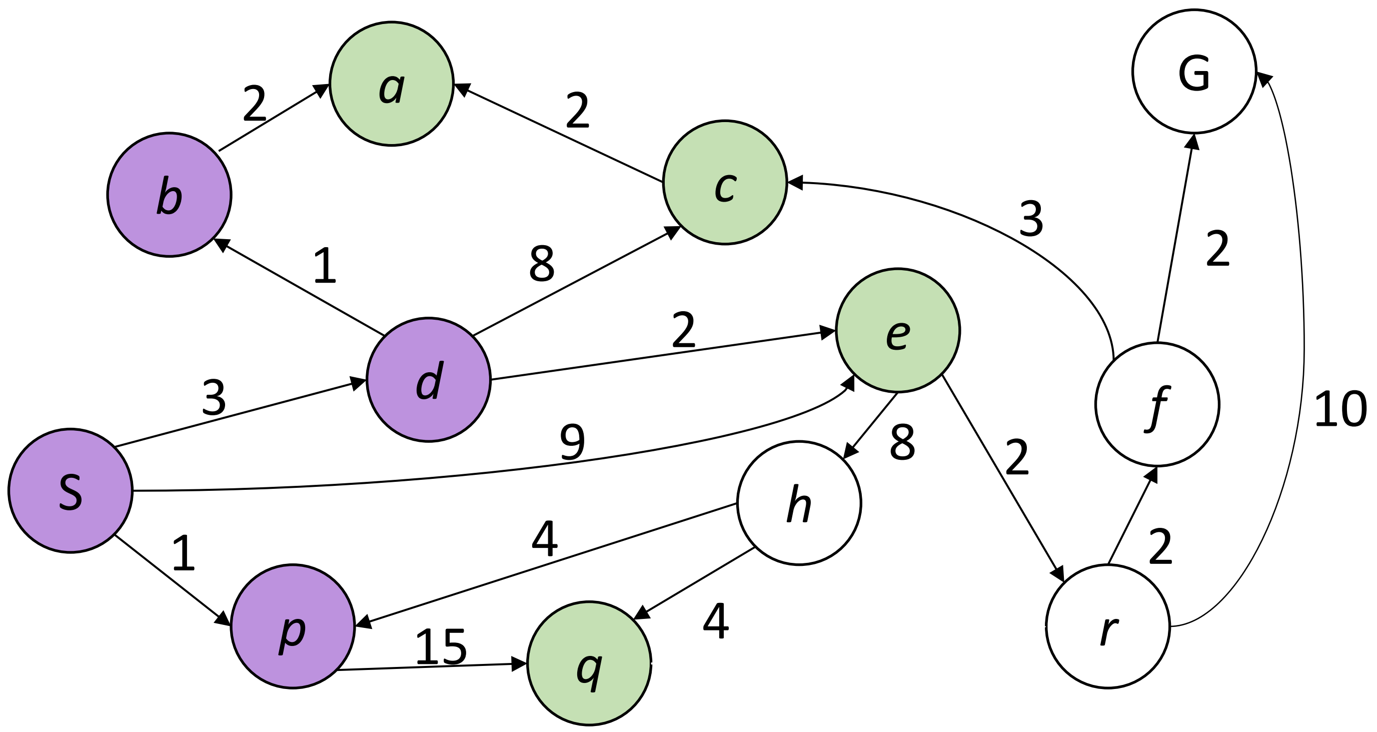
x	$g(x)$
S	0
a	∞
b	∞
c	∞
d	3
e	9
f	∞
h	∞
p	1
q	16
r	∞
G	∞

UCS



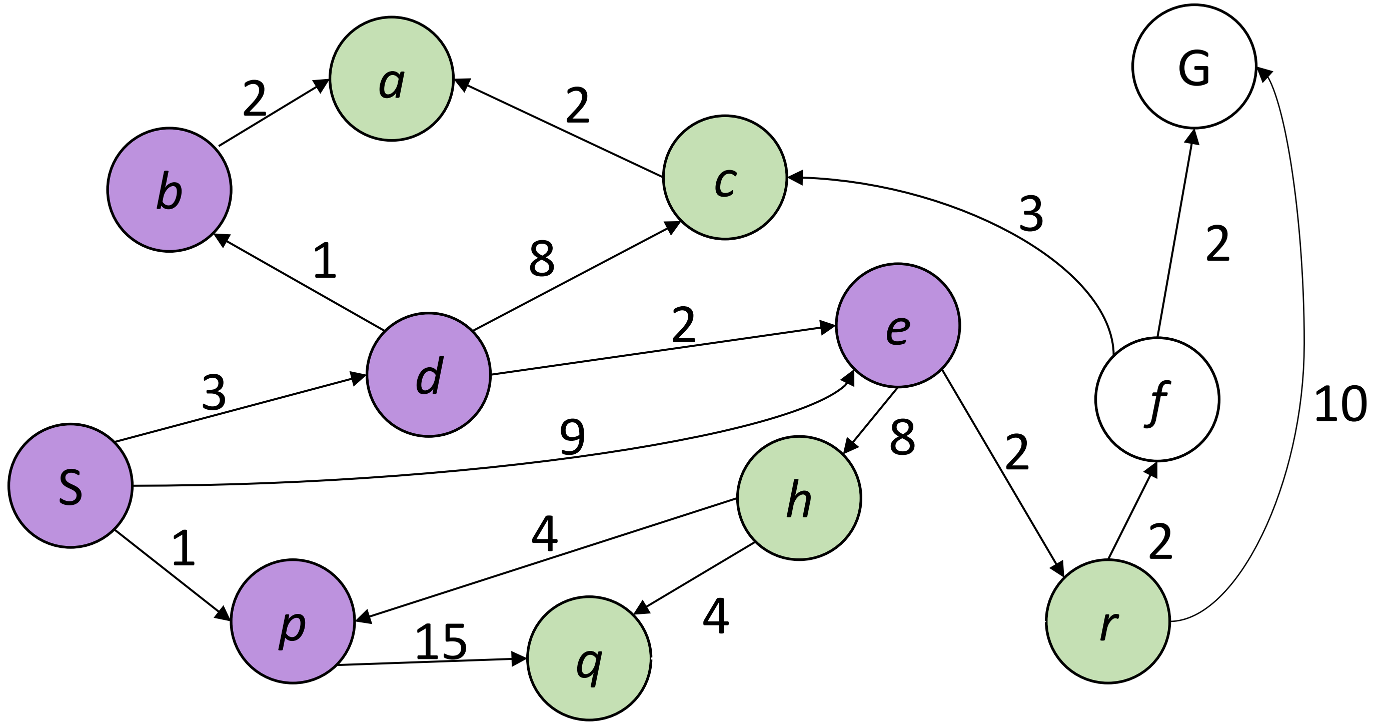
x	$g(x)$
S	0
a	∞
b	4
c	11
d	3
e	5
f	∞
h	∞
p	1
q	16
r	∞
G	∞

UCS



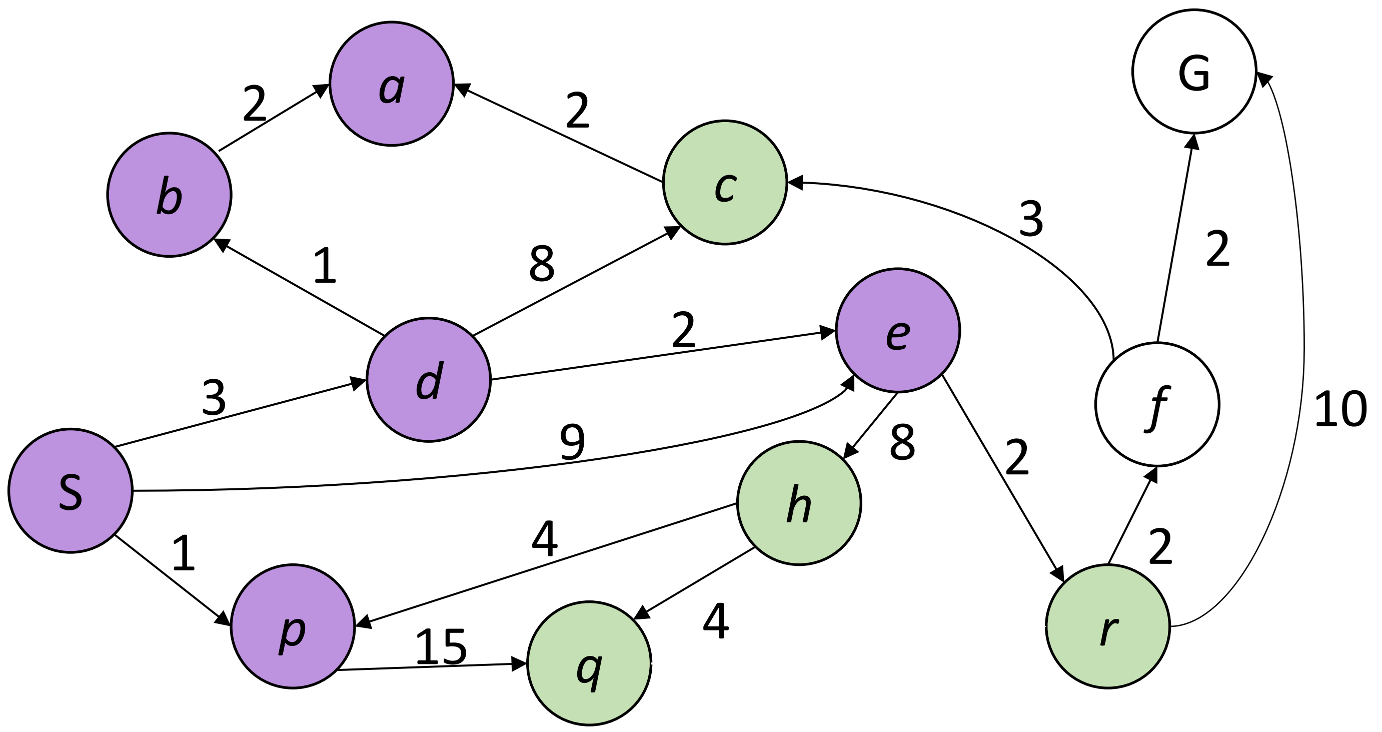
x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	∞
h	∞
p	1
q	16
r	∞
G	∞

UCS



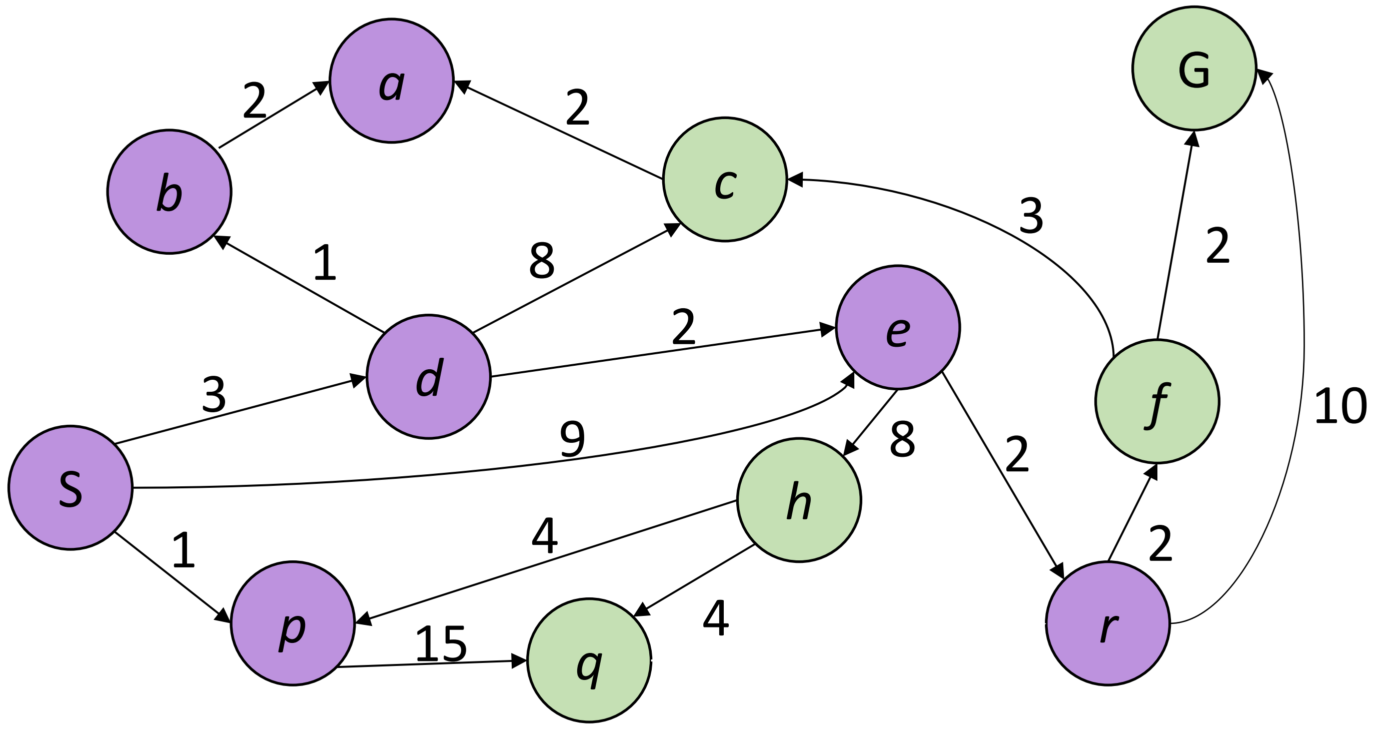
x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	∞
h	13
p	1
q	16
r	7
G	∞

UCS



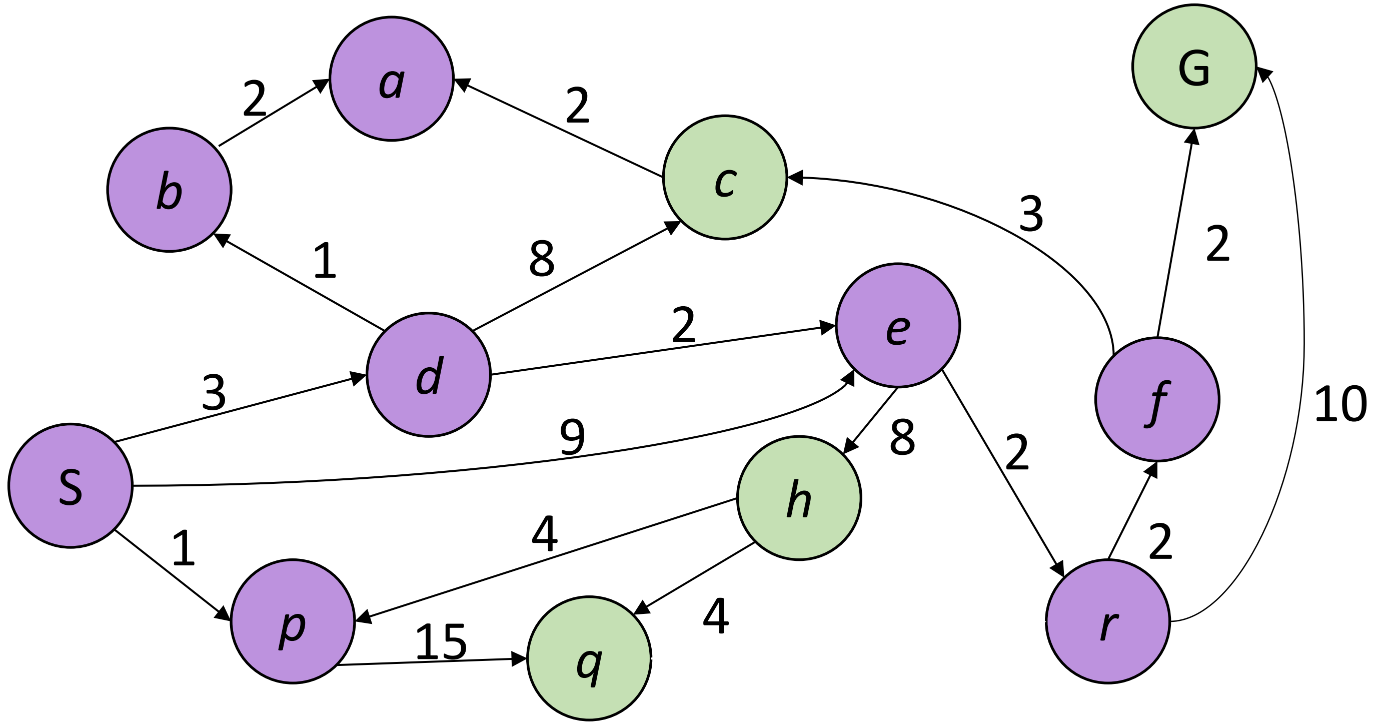
x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	∞
h	13
p	1
q	16
r	7
G	∞

UCS



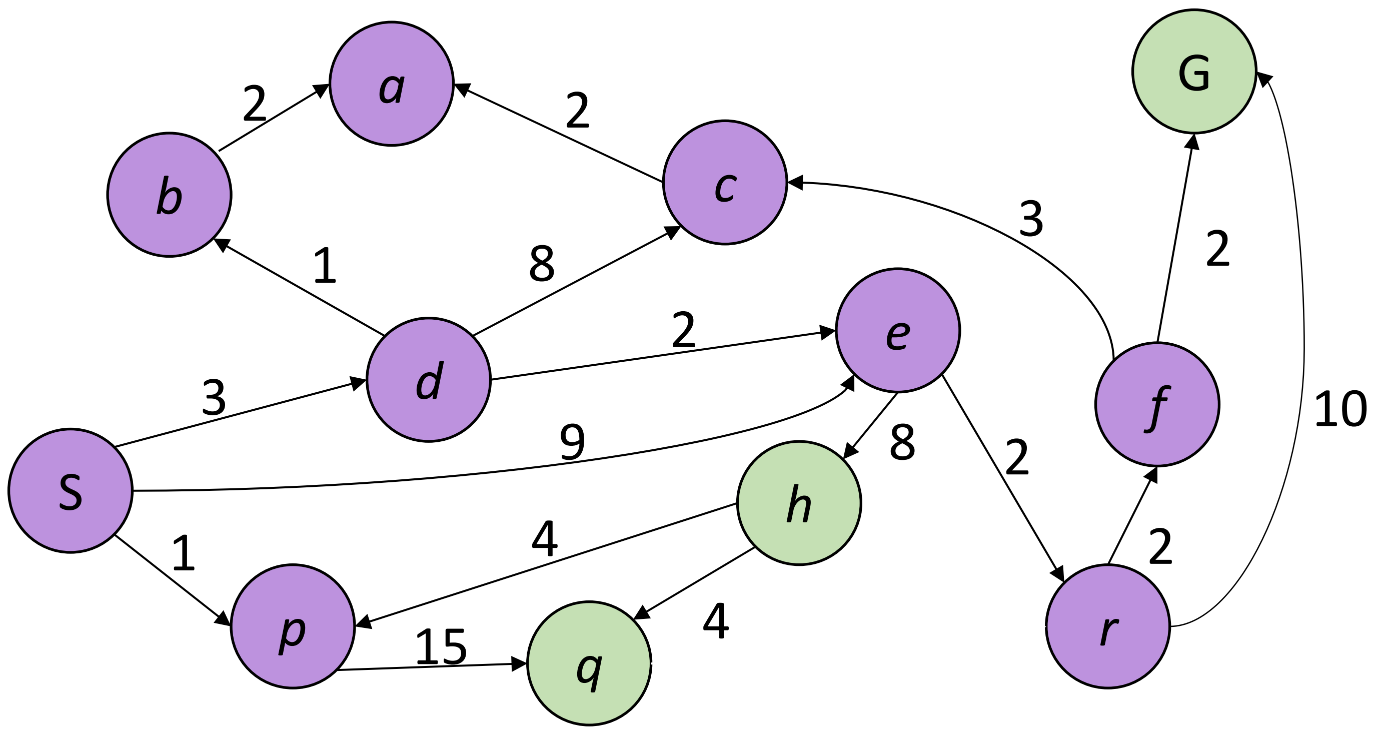
x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	9
h	13
p	1
q	16
r	7
G	17

UCS



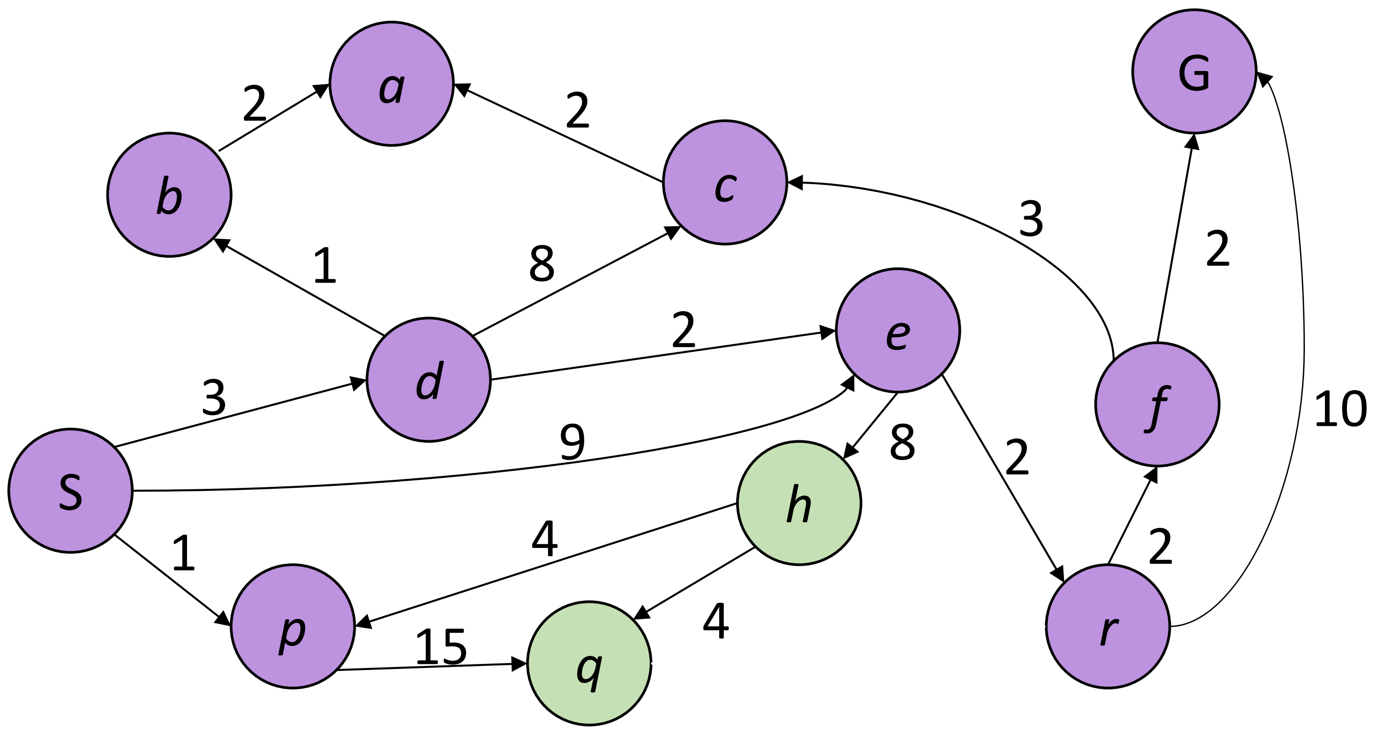
x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	9
h	13
p	1
q	16
r	7
G	11

UCS



x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	9
h	13
p	1
q	16
r	7
G	11

UCS



x	$g(x)$
S	0
a	6
b	4
c	11
d	3
e	5
f	9
h	13
p	1
q	16
r	7
G	11