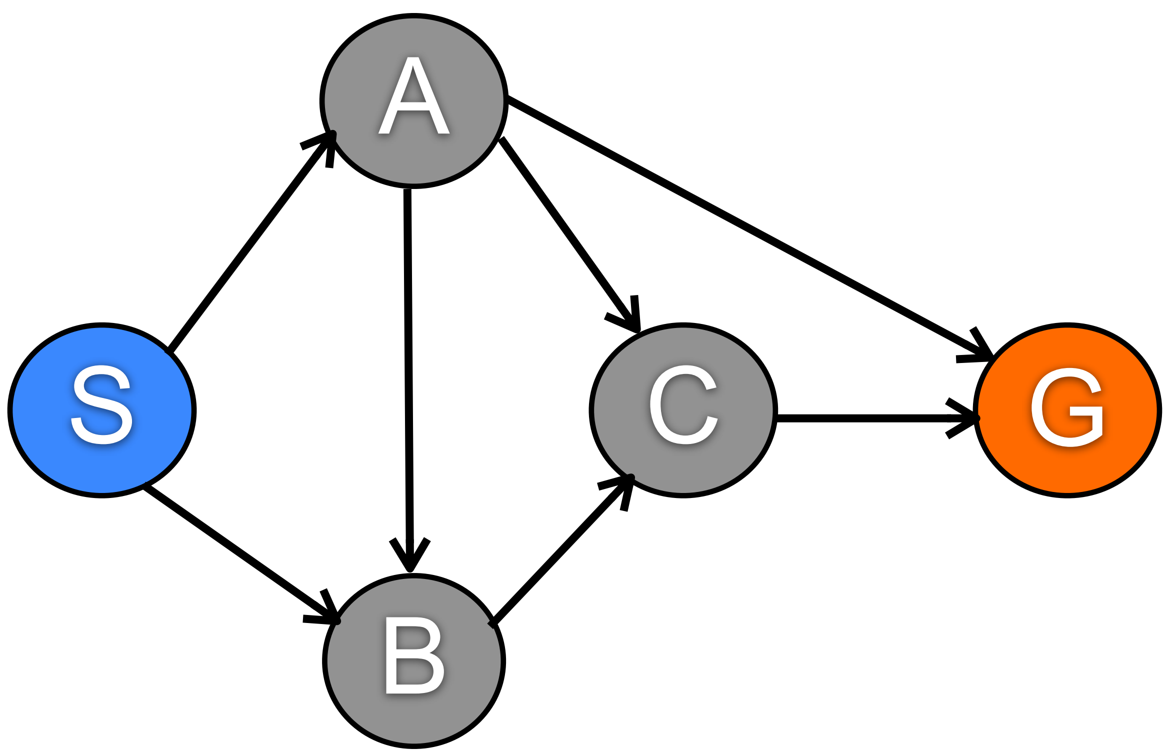
Exercises
6.1. Search & Minimax

Run BFS (Graph Search) From S to G

```
for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```



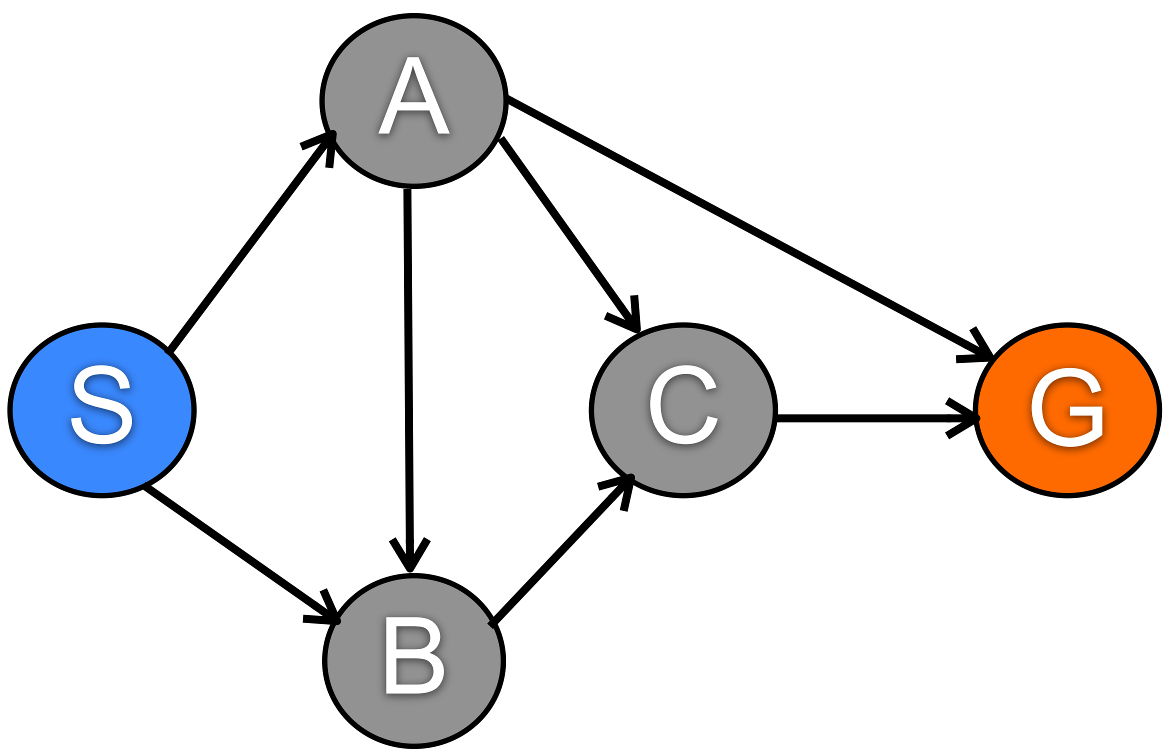
Frontier	Explored
1. [S]	[ ]
2. [S-A, S-B]	[S]
3. [S-A-C, S-B]	[S, A]

S-A-G passes goal test, return as SOLUTION

Try selecting S-B at step 2

Run BFS (Graph Search) From S to G

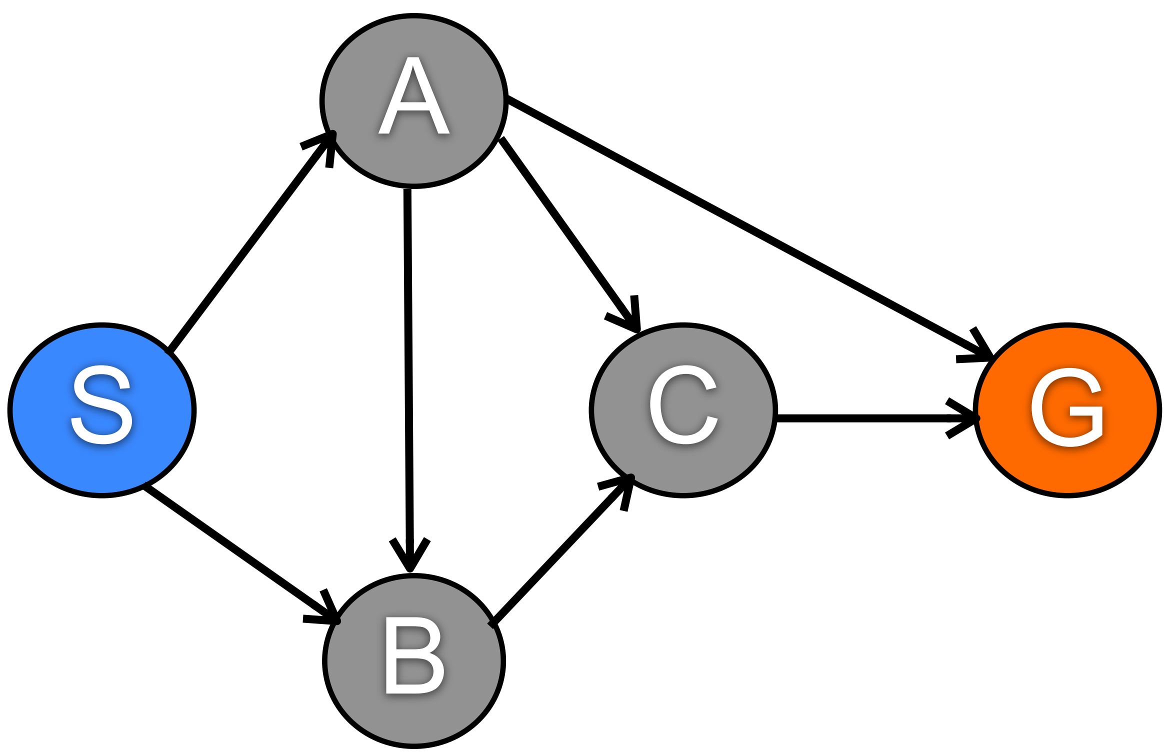
for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
        if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*child*)
        frontier ← INSERT(*child*, *frontier*)

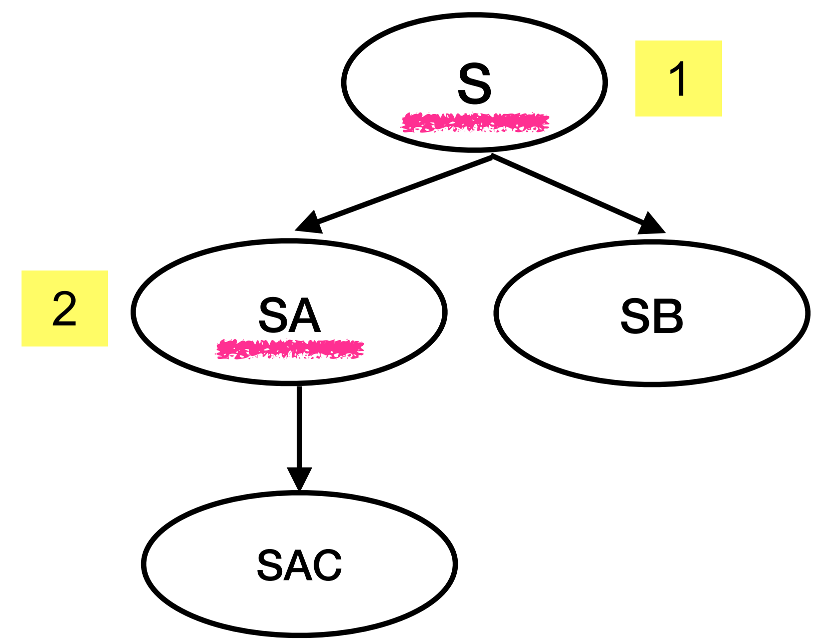


Frontier	**Explored**
1. [S]	[ ]
2. [S-A, S-B]	[S]
3. [S-A, S-B-C]	[S, B]
4. [S-B-C]	[S, B, A]

S-A-G passes goal test, return as SOLUTION

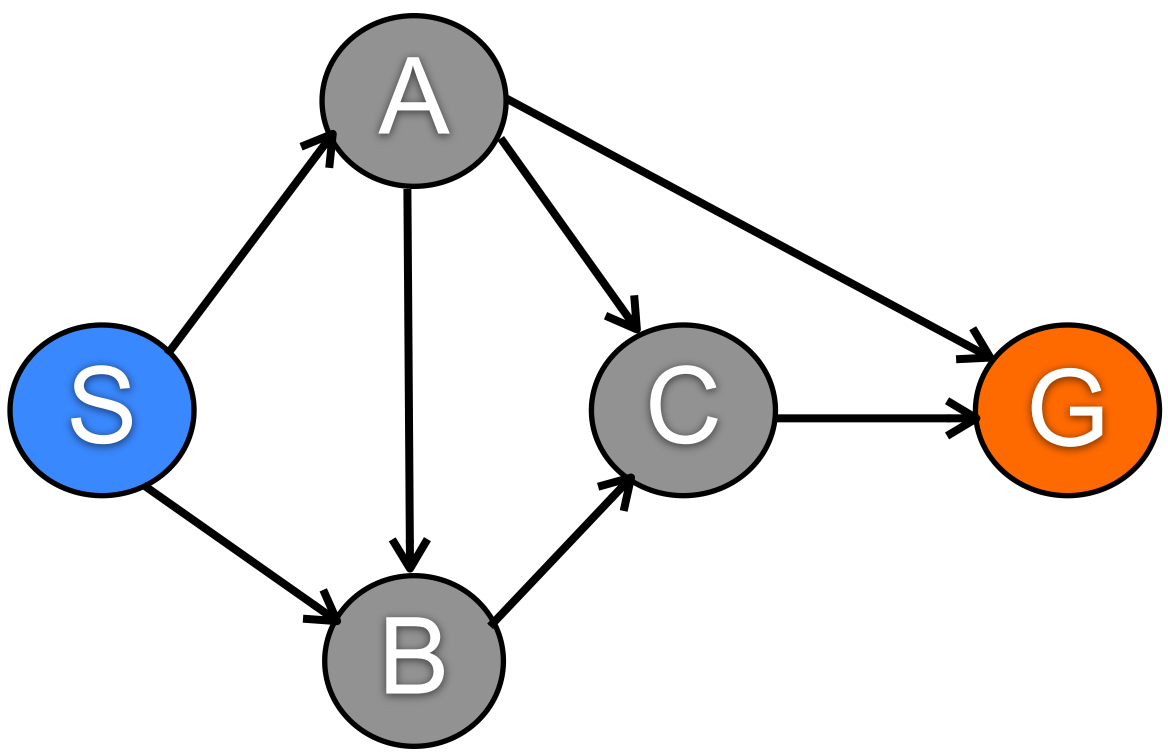
Run BFS (Graph Search) From S to G

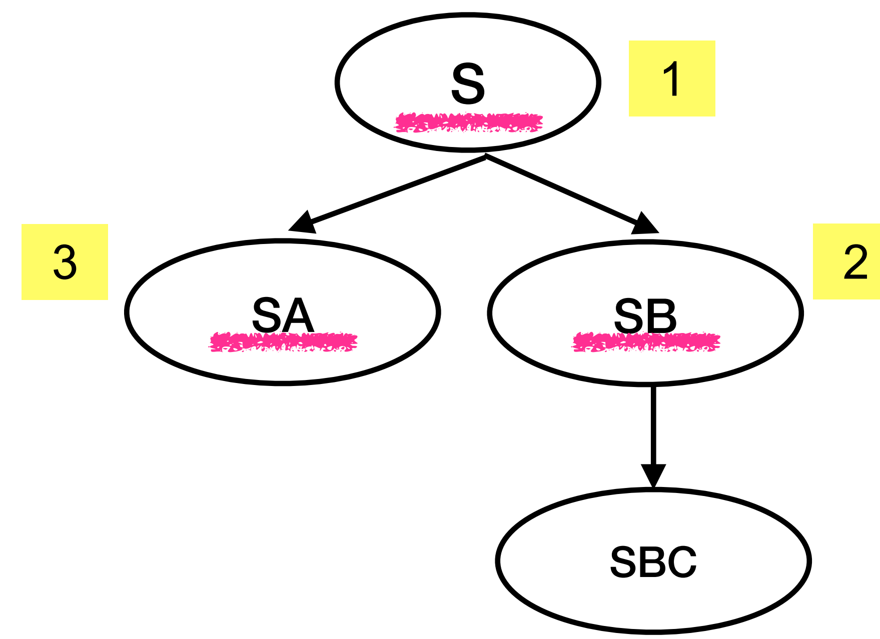




S-A-G passes goal test, return as SOLUTION

Run BFS (Graph Search) From S to G

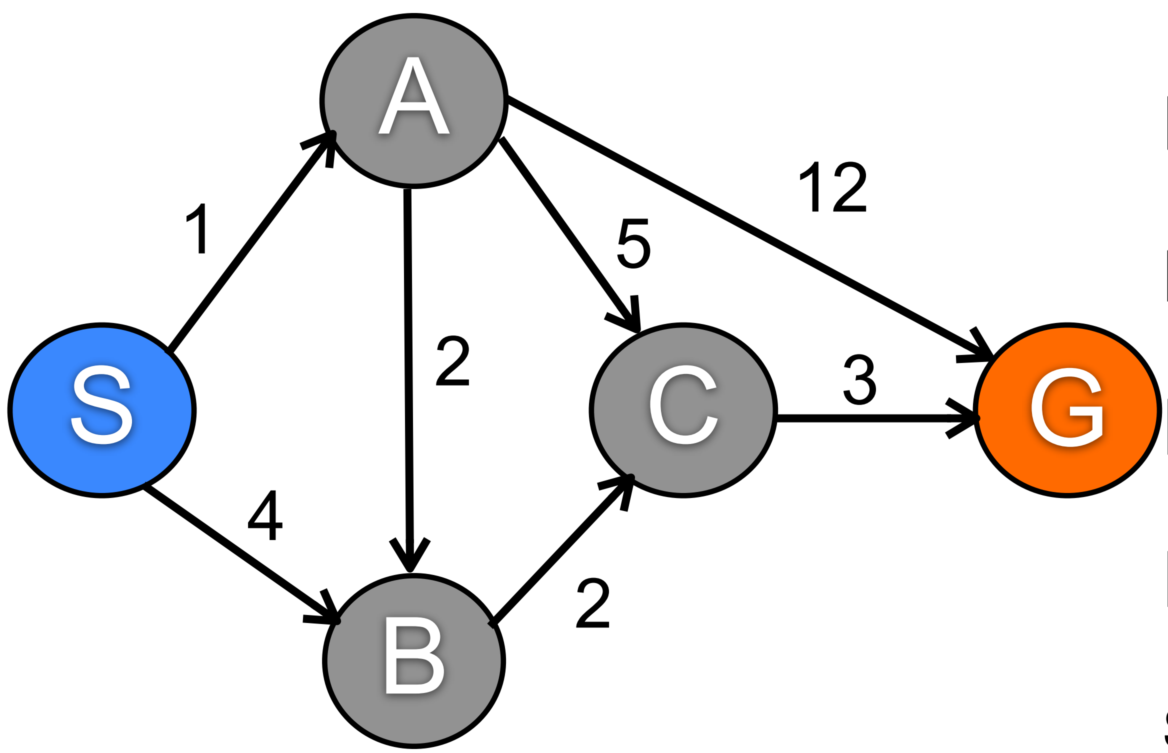




S-A-G passes goal test, return as SOLUTION

UCS S to G – List

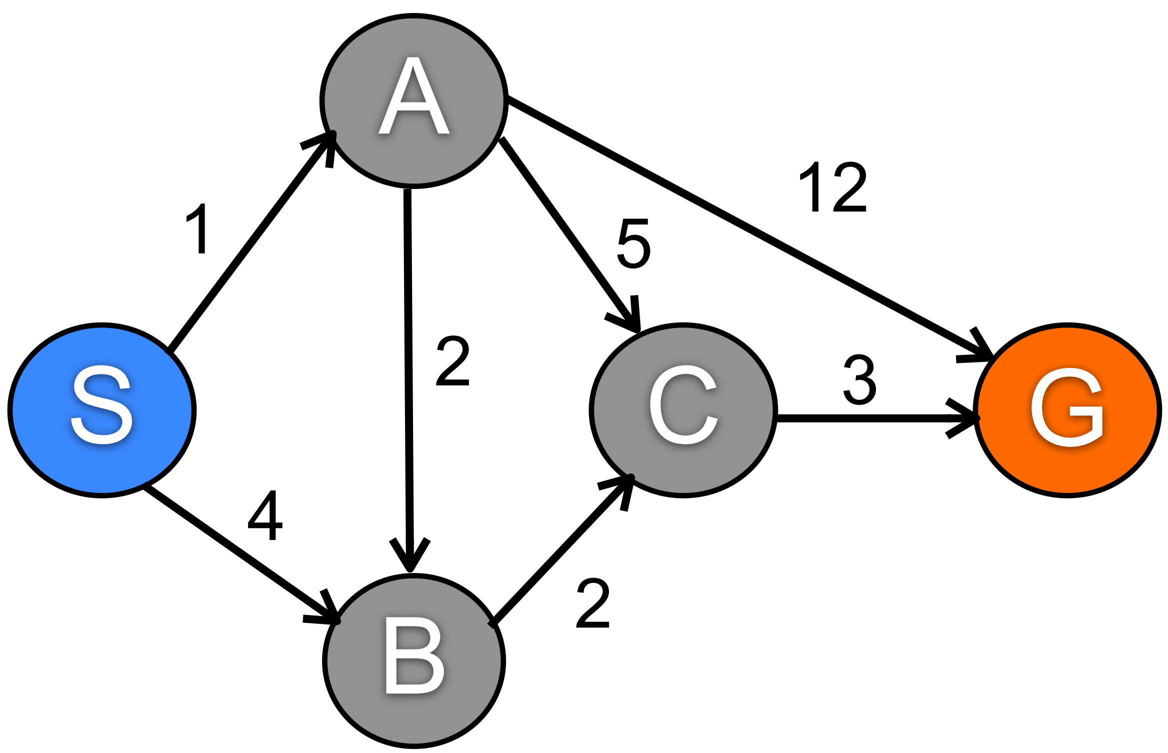
```
node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
add node.STATE to explored
for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
    else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

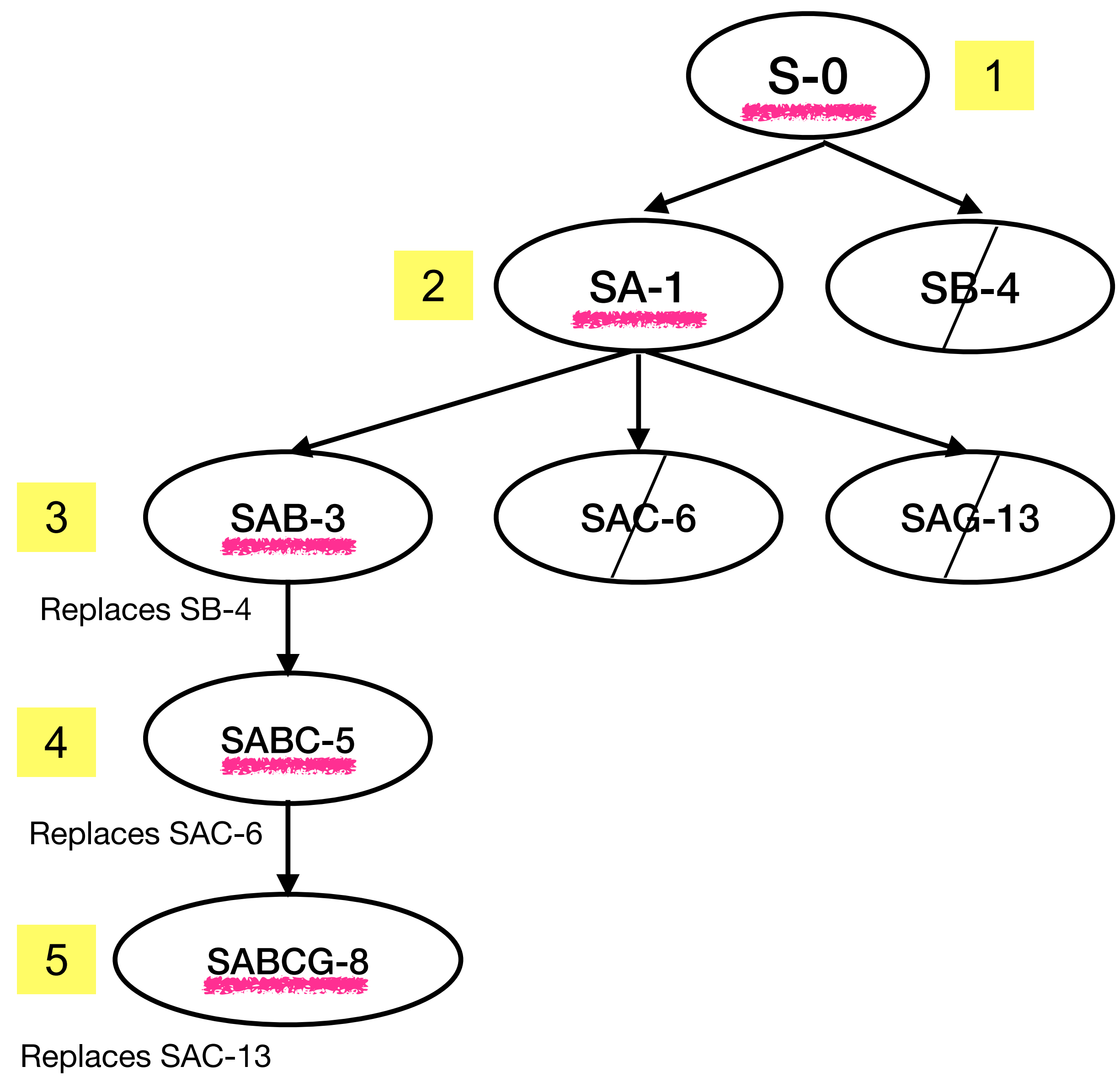


Frontier	**Explored**	
[S-0]	[ ]	
[SA-1, SB-4]	[S]	
[SAB-3, SAC-6, SAG-13]	[S, A]	SAB-3 replaces SB-4
[SABC-5, SAG-13]	[S, A, B]	SABC-5 replaces SAC-6
[SABCG-8]	[S, A, B, C]	
[ ]	[S, A, B, C]	

SABCG-8 passes goal test, return as SOLUTION

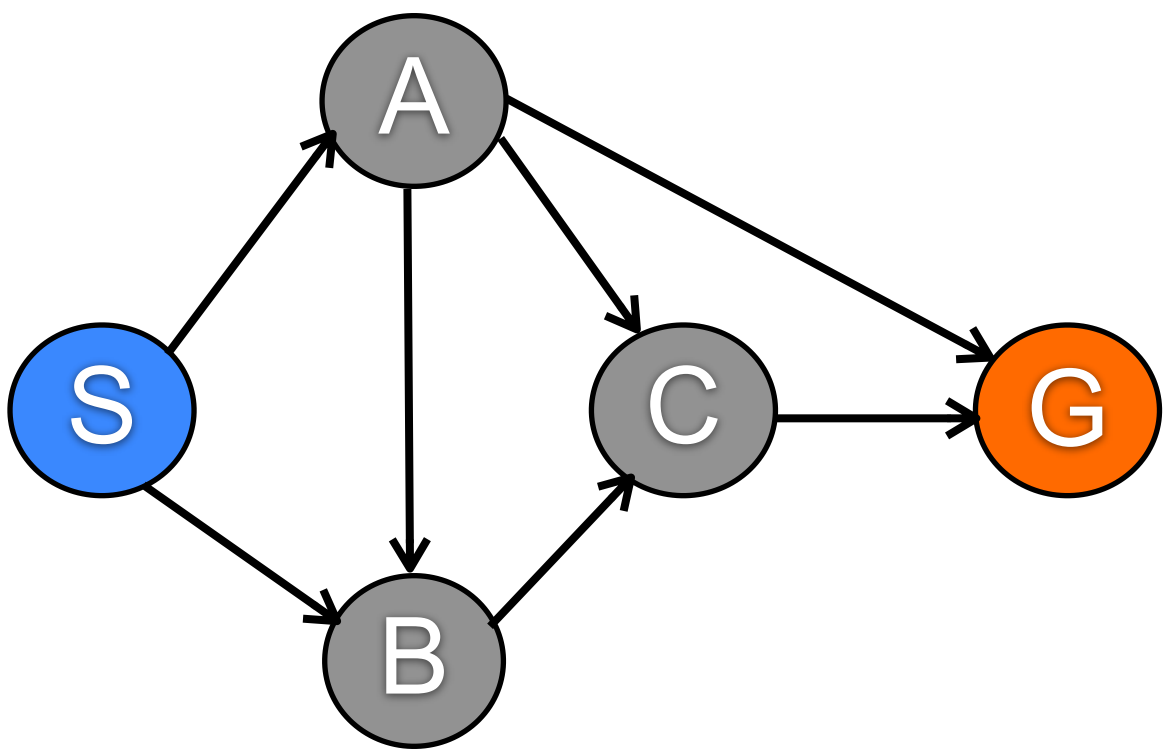
UCS S to G – Tree

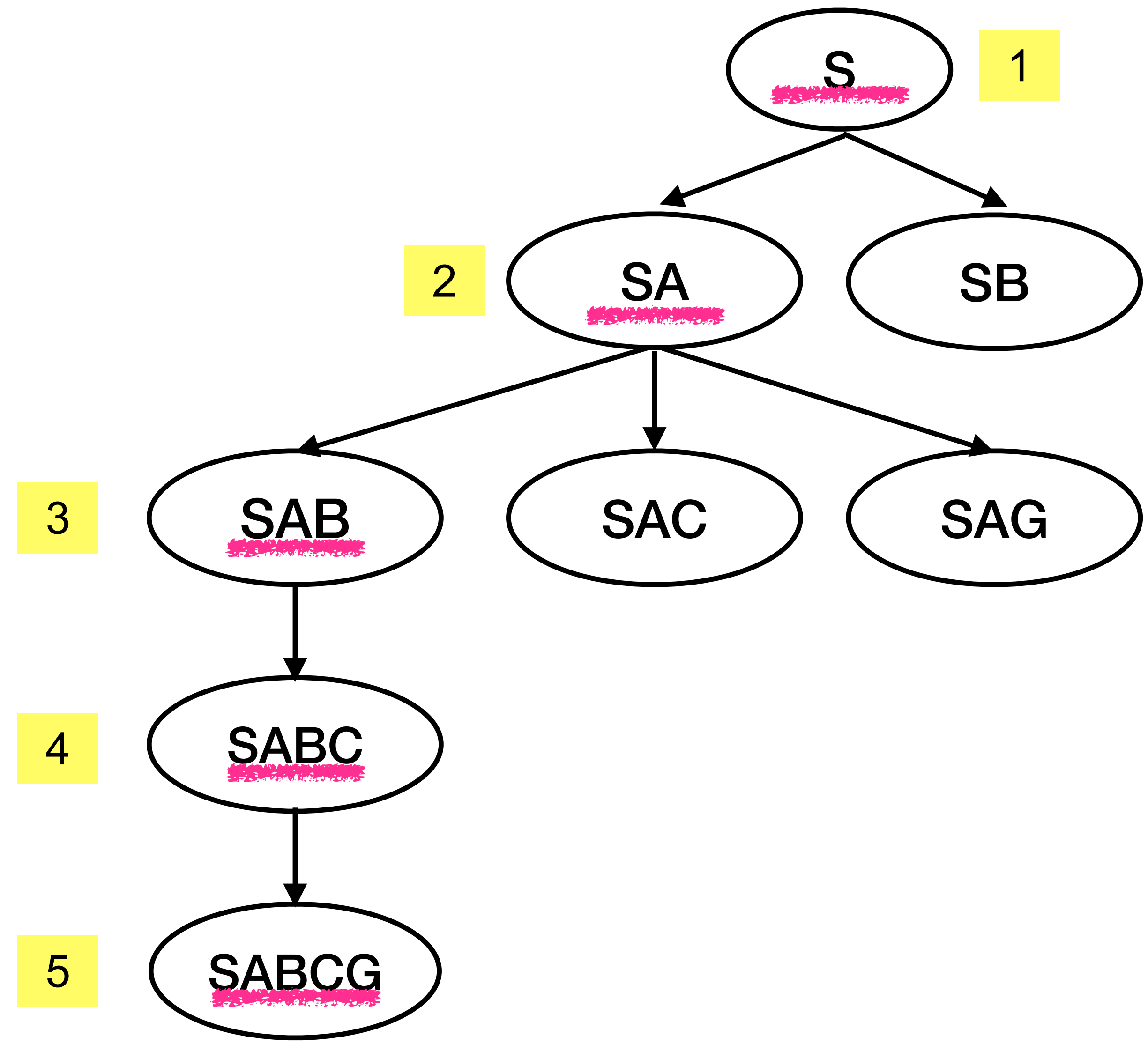




SABCG-8 passes the goal test, return as SOLUTION

DFS S to G – Tree Search

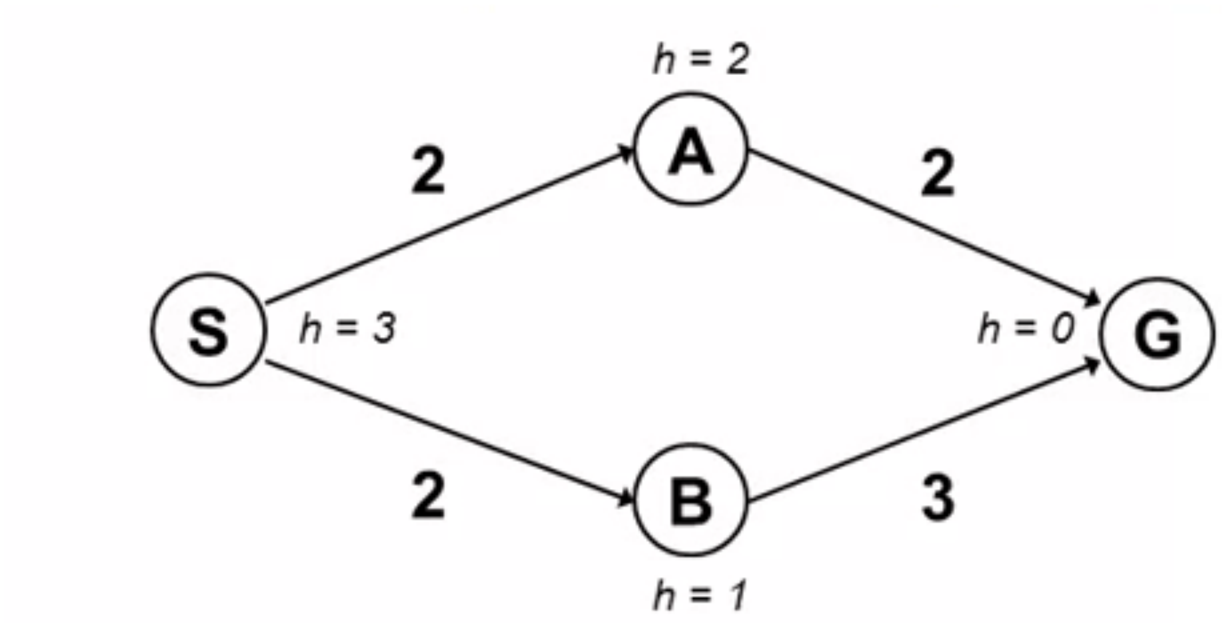




SABCG passes the goal test, return as SOLUTION
***Note: DFS could also return SACG, SBCG and SAG**

Greedy Search Exercise

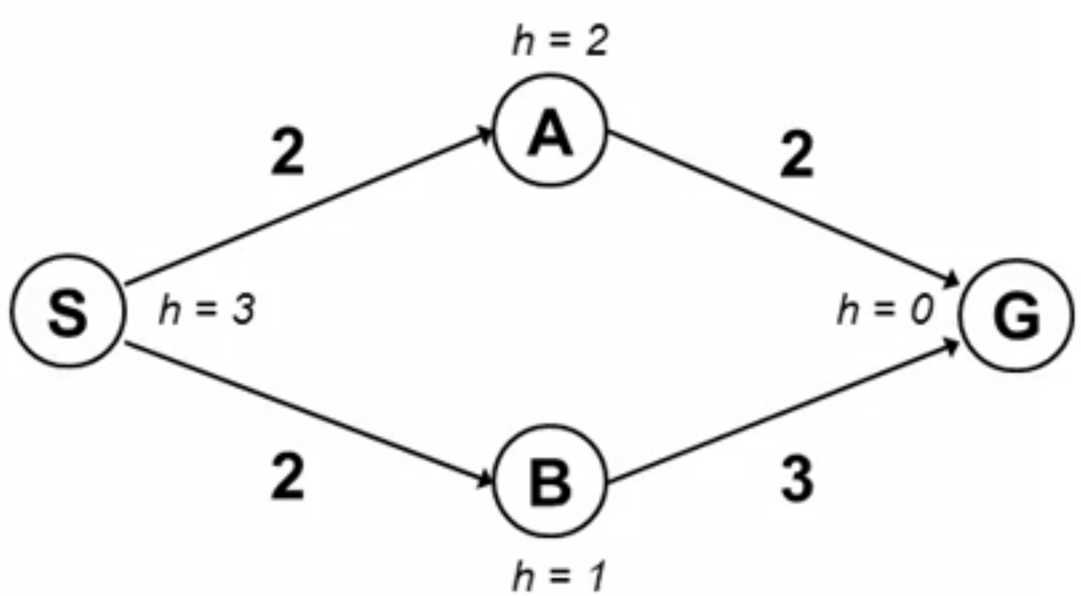
$$f(n) = h(n)$$



- Perform greedy search from S to G by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

Greedy Search – List

$$f(n) = h(n)$$



Frontier	f calculation (Optional)
[S-3]	f(S) = h(S) = 3
[SA-2, SB-1]	f(SA)=h(A)=2, f(SB)=h(B)=1
[SBG-0, SA-2]	f(SBG) = h(G) = 0

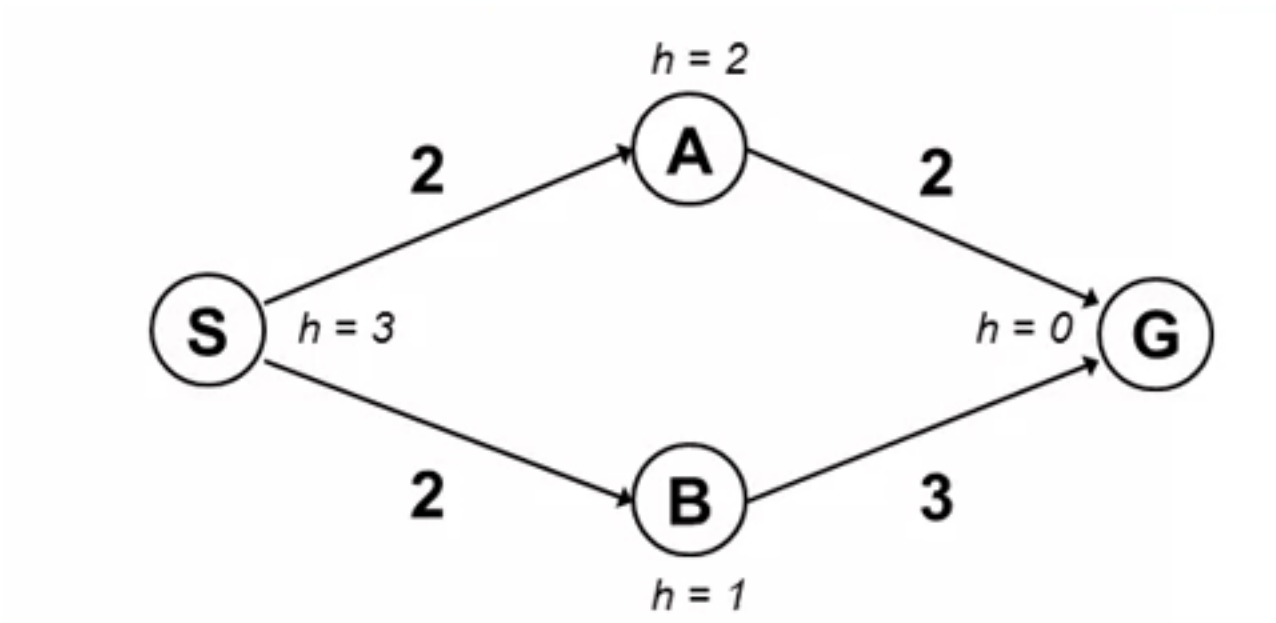
SBG-0 passes goal test

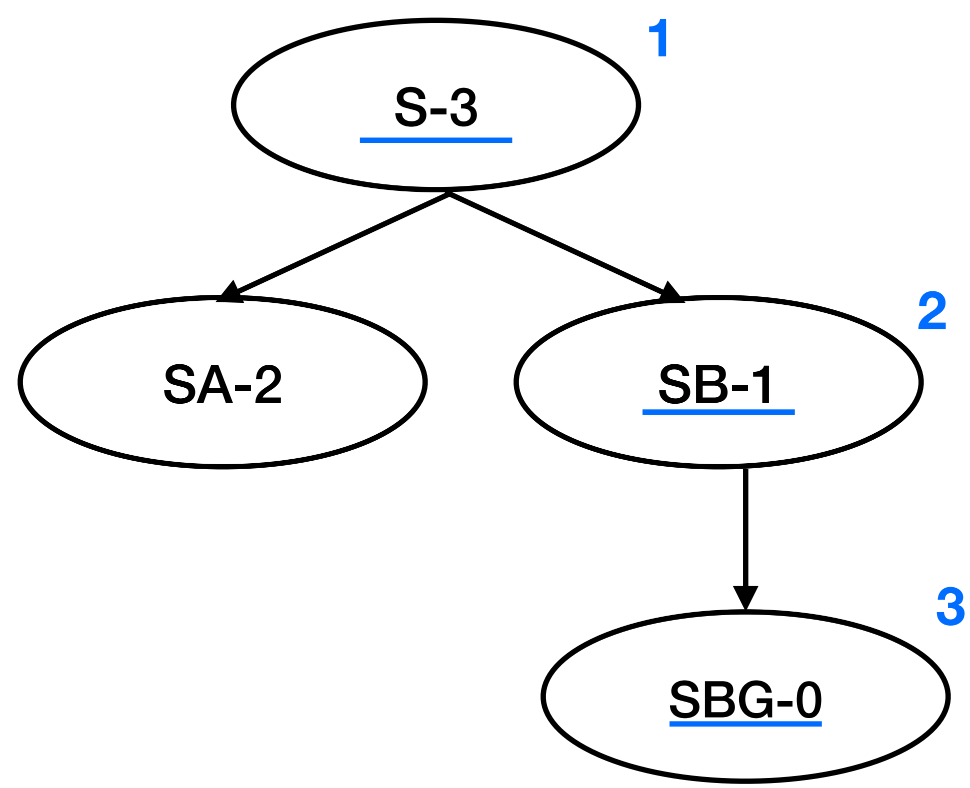
Solution: SBG-0

- Perform greedy search from S to G by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

Greedy Search – Tree

$$f(n) = h(n)$$



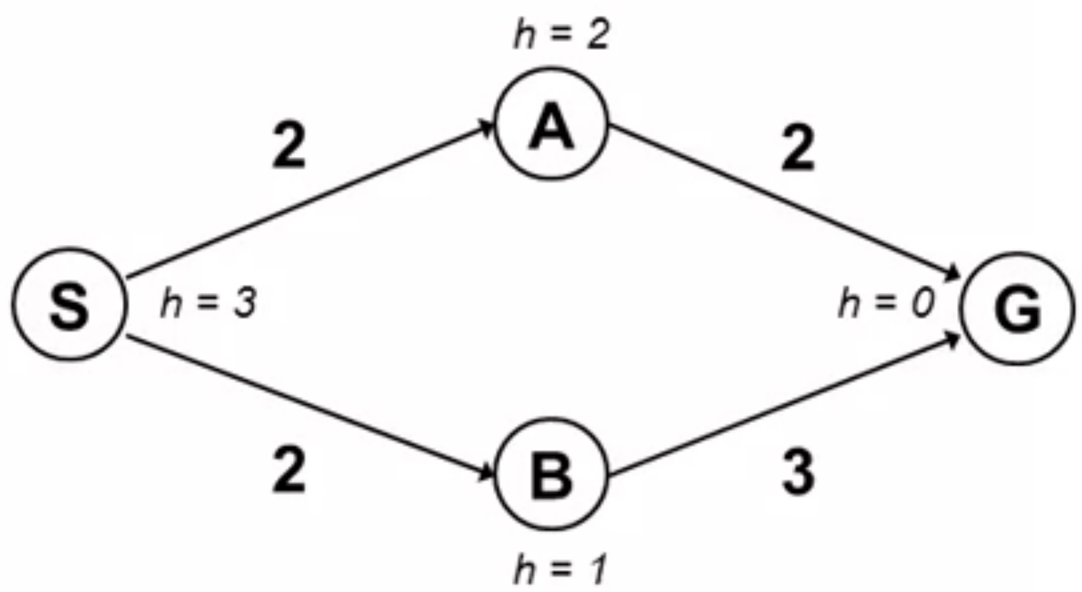


SBG-0 passes goal test, return as solution

- Perform greedy search from S to G by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

A* Search Exercise

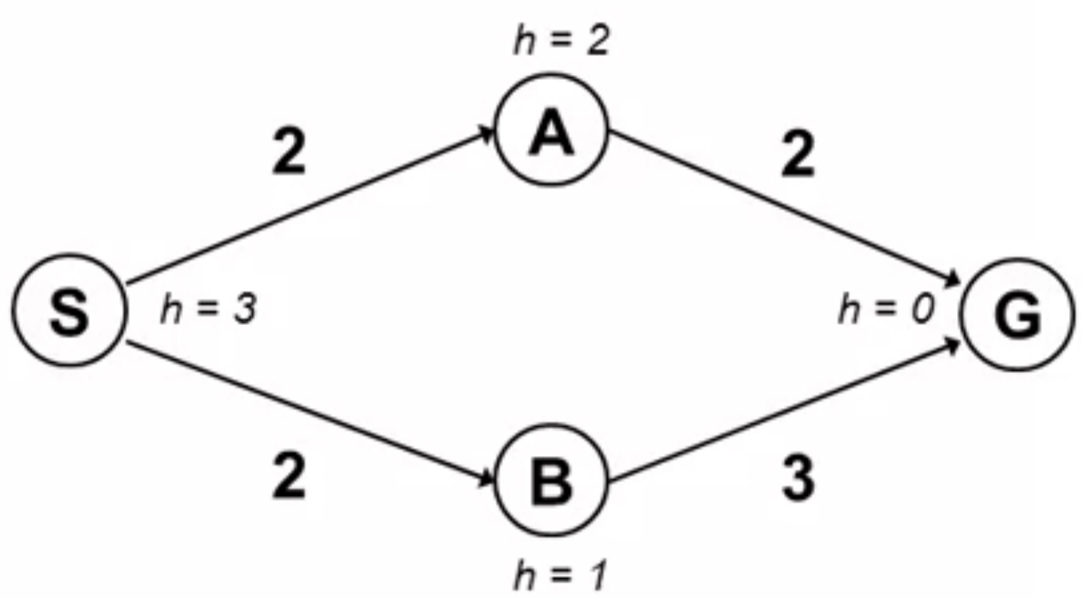
$$f(n) = g(n) + h(n)$$



- Perform A* tree search by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

A* Search – List

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



Frontier	**f (Optional)**
[S-3]	**f(S)=0+3**
[SA-4, SB-3]	**f(SA)=2+2, f(SB)=2+1**
[SA-4, SBG-5]	**f(SBG) = 2+3+0**
[SBG-5, SAG-4]	**f(SAG)=4+0**

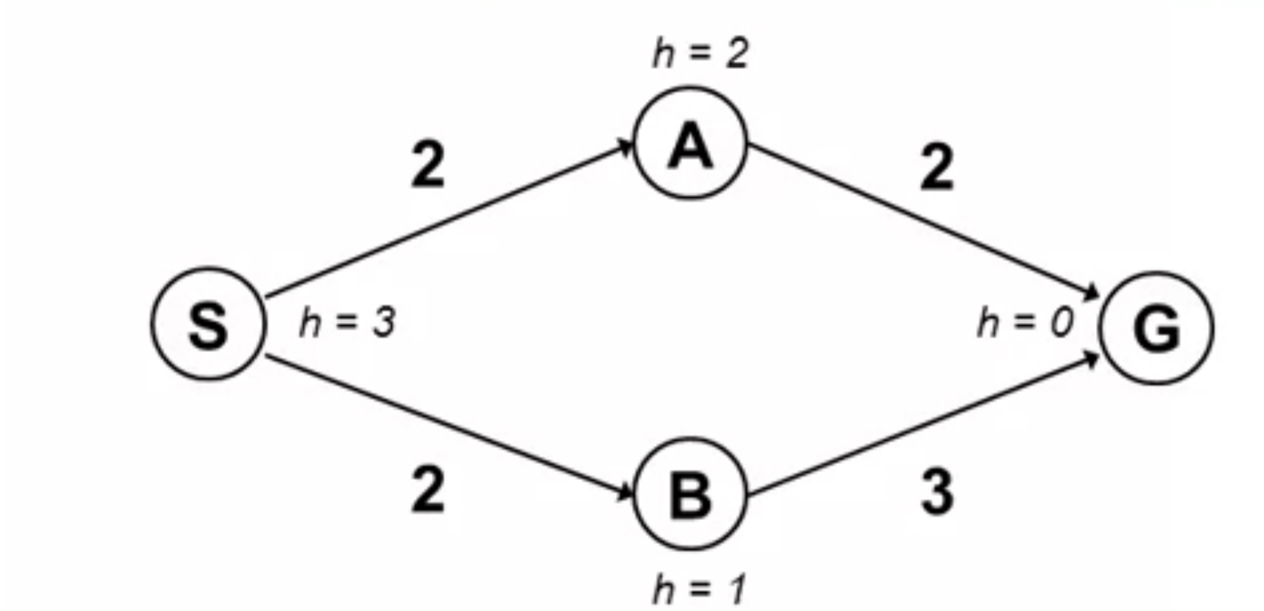
SAG-4 passes goal test

Solution: SAG-4

- Perform A* tree search by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

A* Search – List 2

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

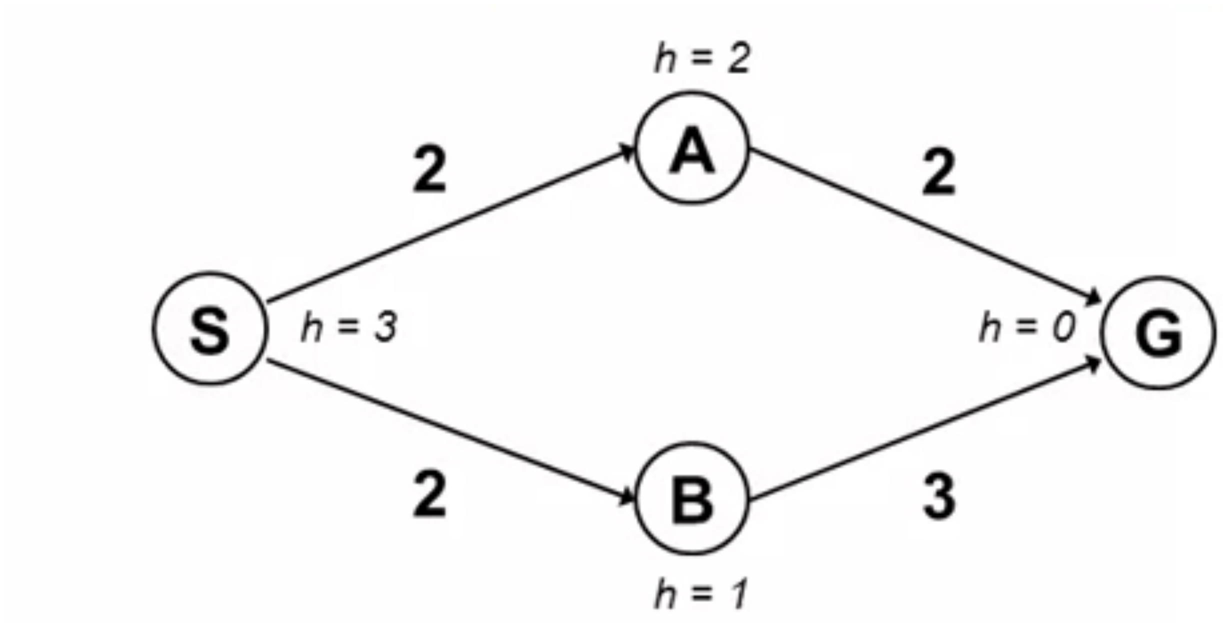


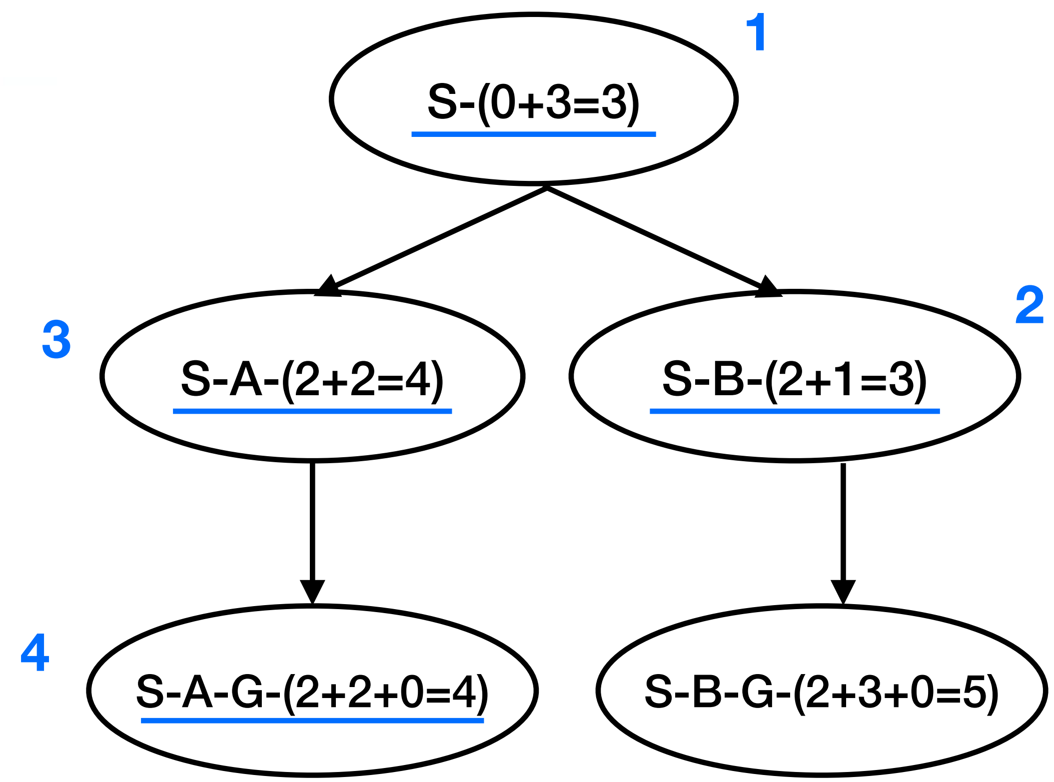
	frontier	g	+ h	= f
1	~~[S]~~	~~0~~	~~3~~	~~3~~
3	~~[SA]~~	~~2~~	~~2~~	~~4~~
2	~~[SB]~~	~~2~~	~~1~~	~~3~~
	[SBG]	5	0	**5**
4	~~[SAG]~~	~~4~~	~~0~~	~~**4**~~

SAG passes goal test –
return as solution

A* Search – Tree

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





SAG passes goal test, return as solution

- Perform A* tree search by providing the frontier at each step and selected node with the evaluation function OR a search tree. What is the solution and is it optimal?

Greedy, A* Search Points

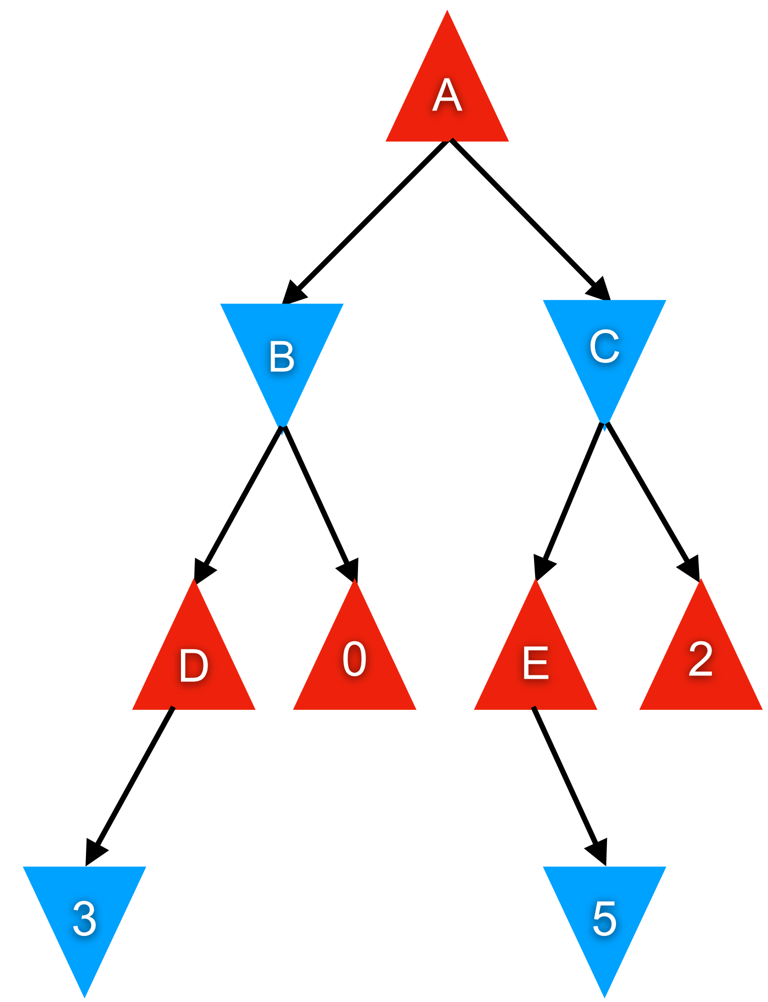
- Remember to perform them in tree search manner
  - Do NOT have to keep track of visited nodes
- Greedy search: $f(n) = h(n)$
- A* search: $f(n) = g(n) + h(n)$
- General tip: if you provide trees, make sure the nodes that get replaced are still readable

Recap – Local Search

- **Hill-climbing/gradient ascent/descent (HA)** – only chooses closes neighbour that improves current object function, can get stuck in local optimum (maximum/minimum)
- **Simulated annealing (SA)** – allows random walks (exploration) or "worse moves" with higher probability (temperature) at the beginning in order to escape local maximum/minimum, over time this probability decreases and finally it behaves like HC
- **Genetic algorithms (GA)** – mimics human evolution where fittest members of a random population are selected, paired and the pairs cross-over information. A small probability of mutation allows exploration of space and thus avoids local maximum/minimum. The process repeats on the resulting offsprings (new set of population) until convergence

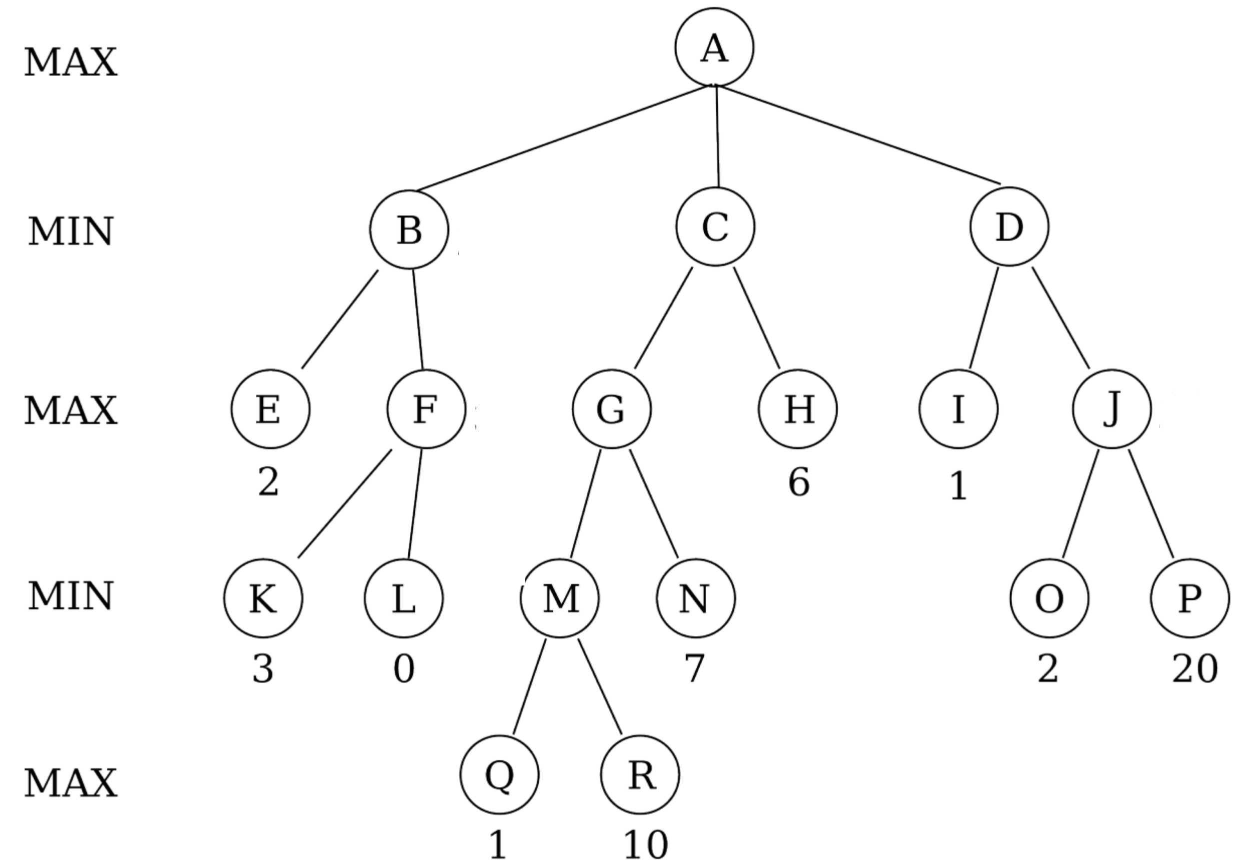
2-player Adversarial Game

- Max (red) and Min (blue) take turns to play
- Competing with each other, both have opposing intents as far as the search is concerned
- Leaf nodes represent the utility values for Max if that path is taken



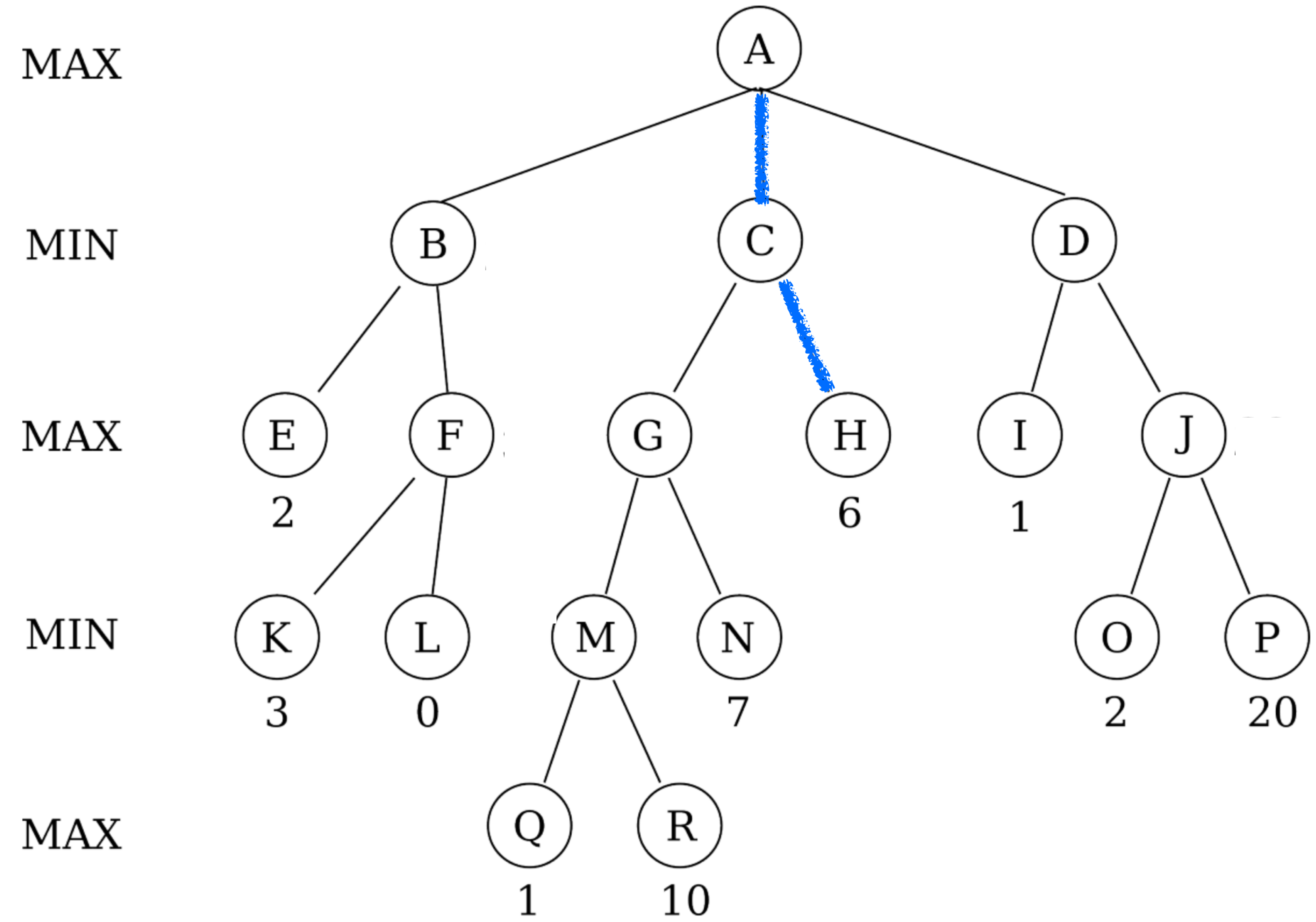
Exercise – Minimax

```
function MINIMAX-DECISION(state) returns an action
   return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for each a in ACTIONS(state) do
      v ← MAX(v, MIN-VALUE(RESULT(s, a)))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for each a in ACTIONS(state) do
      v ← MIN(v, MAX-VALUE(RESULT(s, a)))
   return v
```



- Order the evaluations for **non-leaf nodes** and minimax values. Use <= for Min nodes and >= for Max nodes and = when node values are known
- First two steps:
  1. B <= 2
  2. F >= 3

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$



Terminal node evaluation is optional

1. E = 2
2. B <= 2
3. K = 3
4. F >= 3
5. L = 0
6. F = 3
7. B = 2
8. A >= 2
9. Q = 1
10. M <= 1
11. R = 10
12. M = 1
13. G >= 1
14. N = 7
15. G = 7
16. C <= 7
17. H = 6
18. C = 6
19. A >= 6
20. I = 1
21. D <= 1
22. O = 2
23. J >= 2
24. P = 20
25. J = 20
26. D = 1
27. A = 6

Q. Which move should Max take? C

Q. Which move should Min take after that? H

Quiz – Wednesday

- Around 1 hour during class (3.00pm – 4.15pm)
- Closed-book exam – no alternate online quiz will be given to those who have a valid excuse for absence (score for quiz will be based on your final exam)
- Cover all topics until minimax
- NO notes, calculators, mobile phones, headphones, scrap paper – you will be given a quiz booklet
- Use pen or pencil+eraser (refrain from using red ink)
- You will be given a sheet containing all the main algorithms
- Remember to review agents and rationality too