JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

# Classical Search

601.464
Artificial Intelligence
TR 10.30AM—11.45PM

# Material

- Definitions
- Uninformed Search
    - Depth-First Search
    - Breadth-First Search
- Informed Search
    - Hill Climbing
    - Beam Search
- Branch and Bound
- Branch and Bound + Extended List
- A*
- Heuristics
- Sample Problem #2
- Maze

# goal test

way to determine whether a given state is a goal state

# path cost

numerical cost associated with a given path

# solution

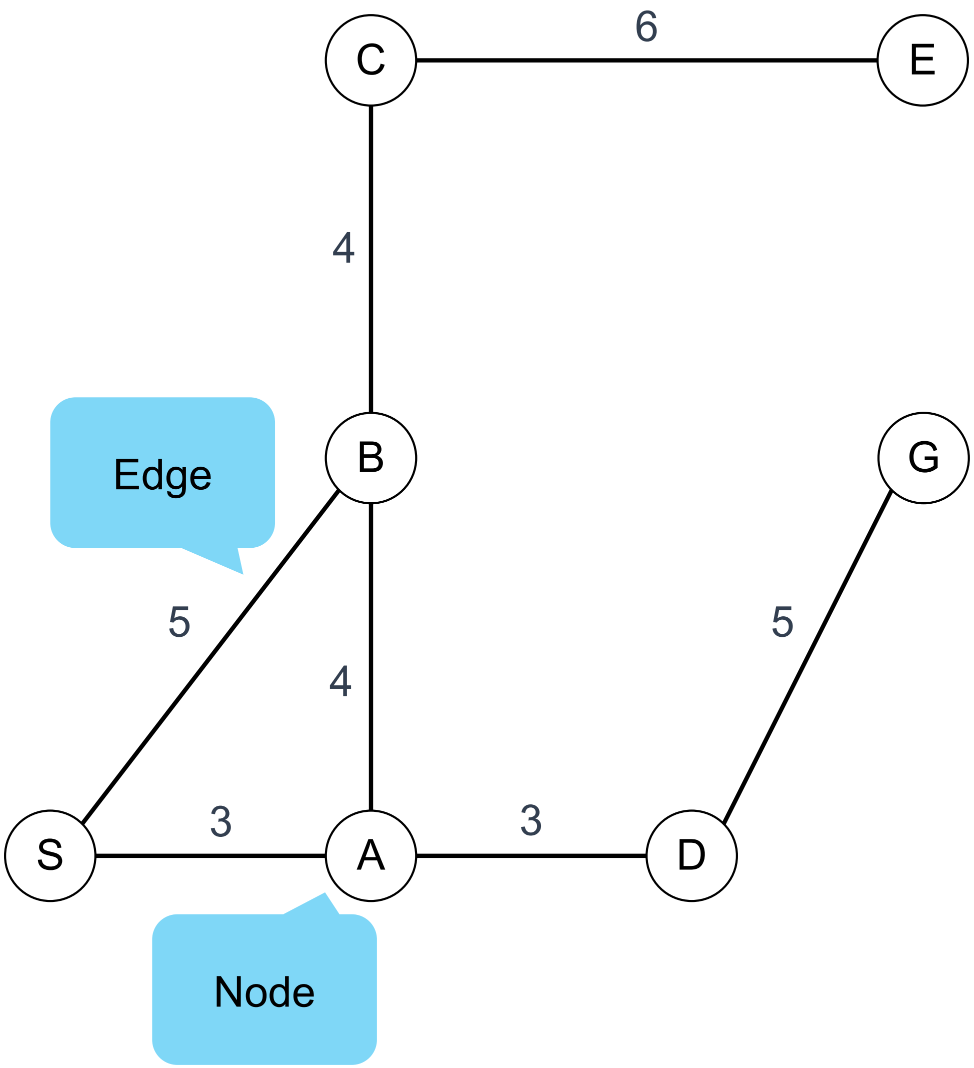
a sequence of actions that leads us from the initial state to the goal state

# optimal solution

a solution that has the lowest path cost among all the solutions

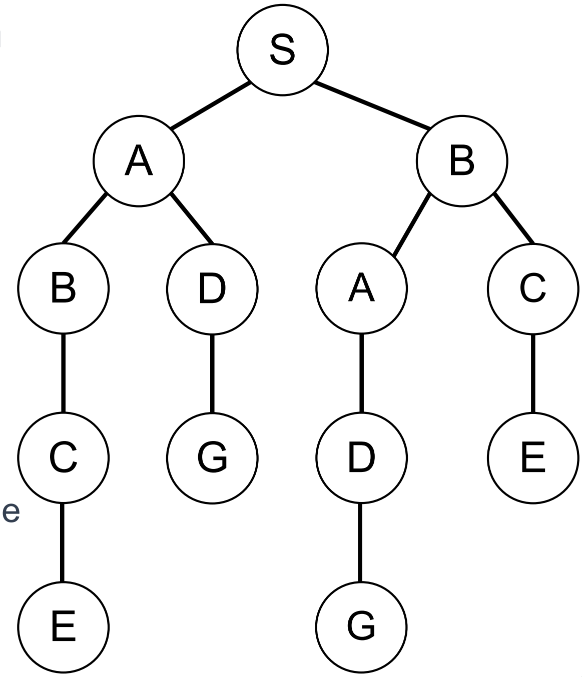
# The Problem

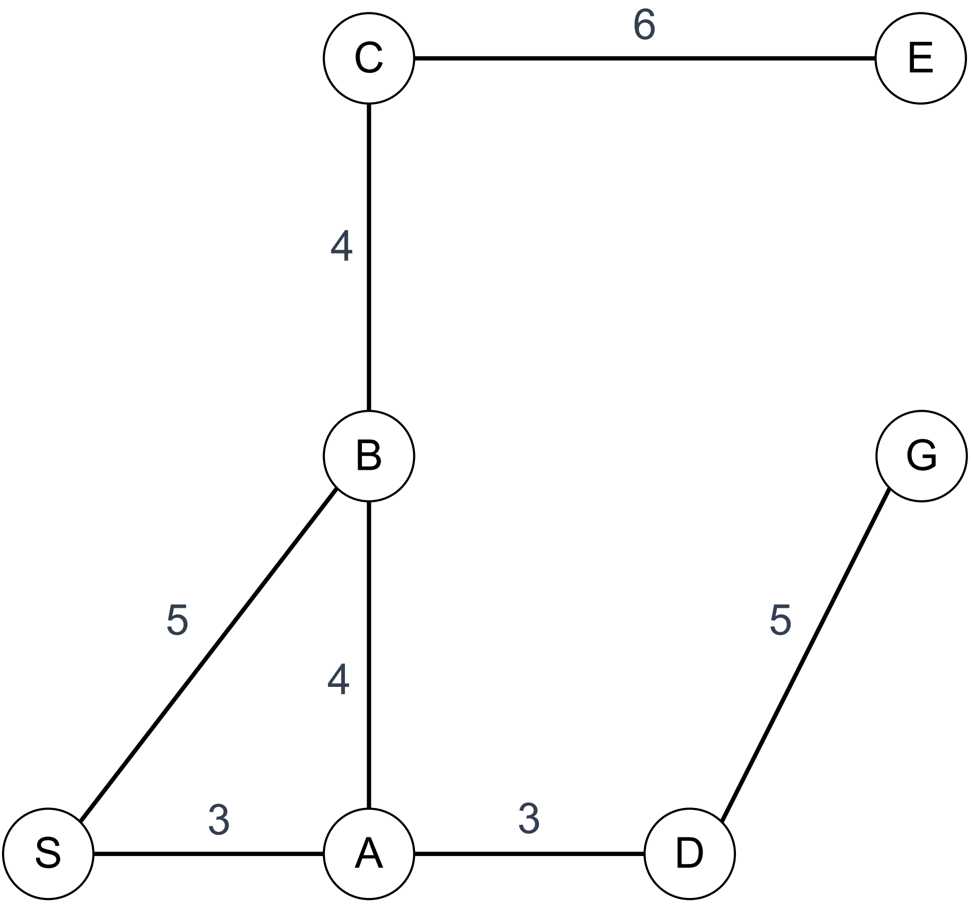
- Find a *path* to goal (G)
  - A path (in this case) is a sequence of node(s) starting from *S* that result from travelling via the edges (links) without revisiting a node



# Representation

- British Museum
  - Tree
  - List all nodes
  - Lexical order (left to right)
  - Parent can’t reappear as along the same branch





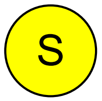
# SEARCH != MAPS

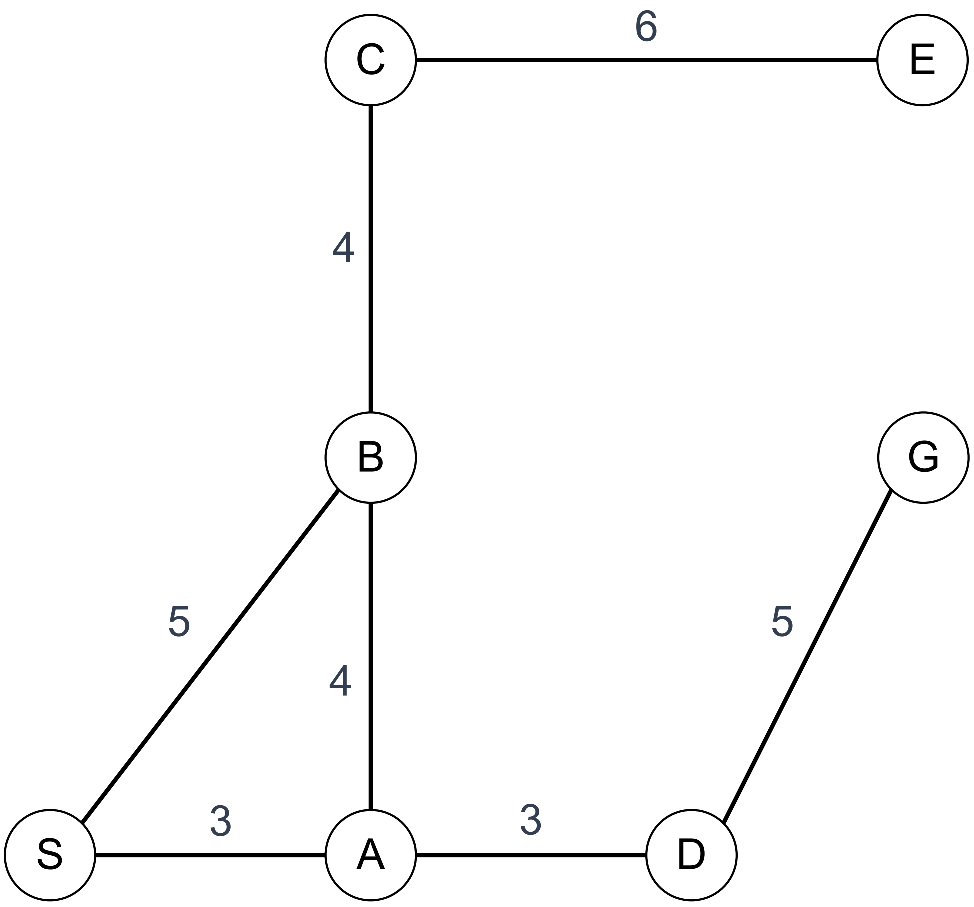
- Search is about ***CHOICES***

- Maps are great for illustrative purposes, but we shouldn't automatically equate search to maps
  - A node could represent a chess board configuration, for instance

# Depth-First Search

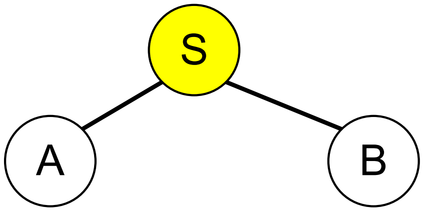
- When there's a choice, pick left-most node and keep barreling down

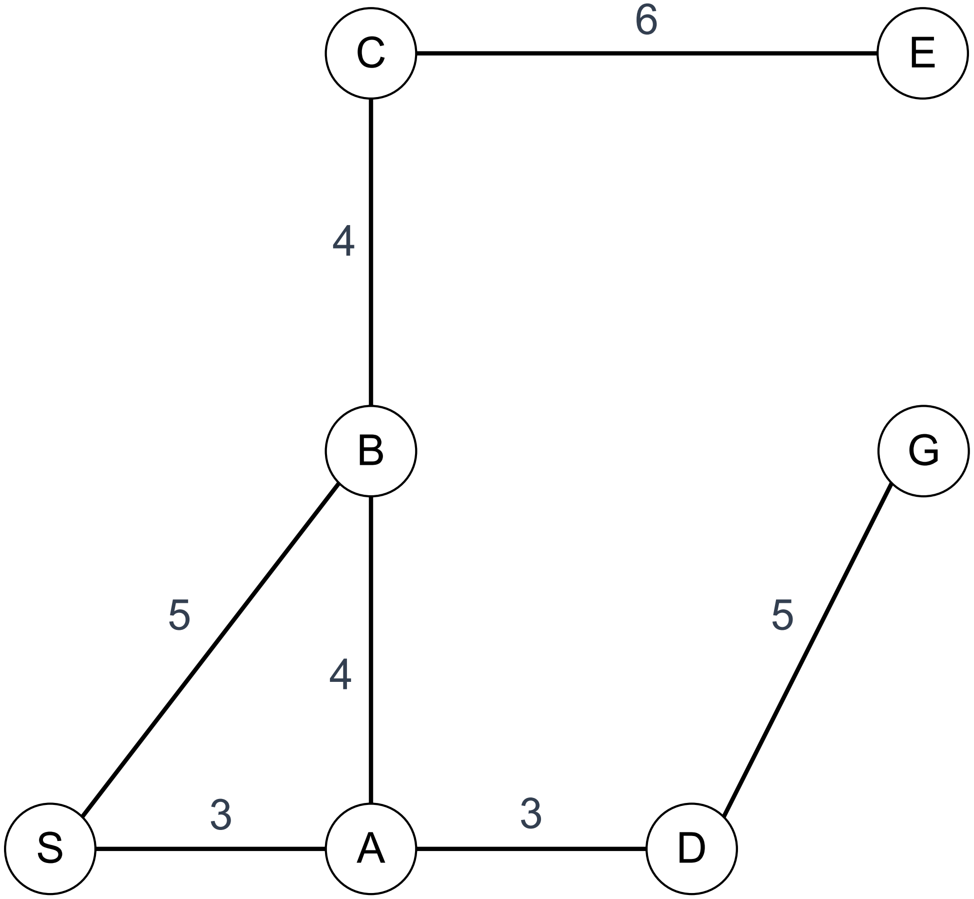




# Depth-First Search

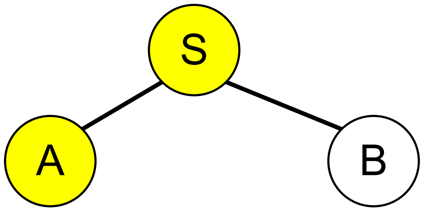
- When there's a choice, pick left-most node and keep barreling down

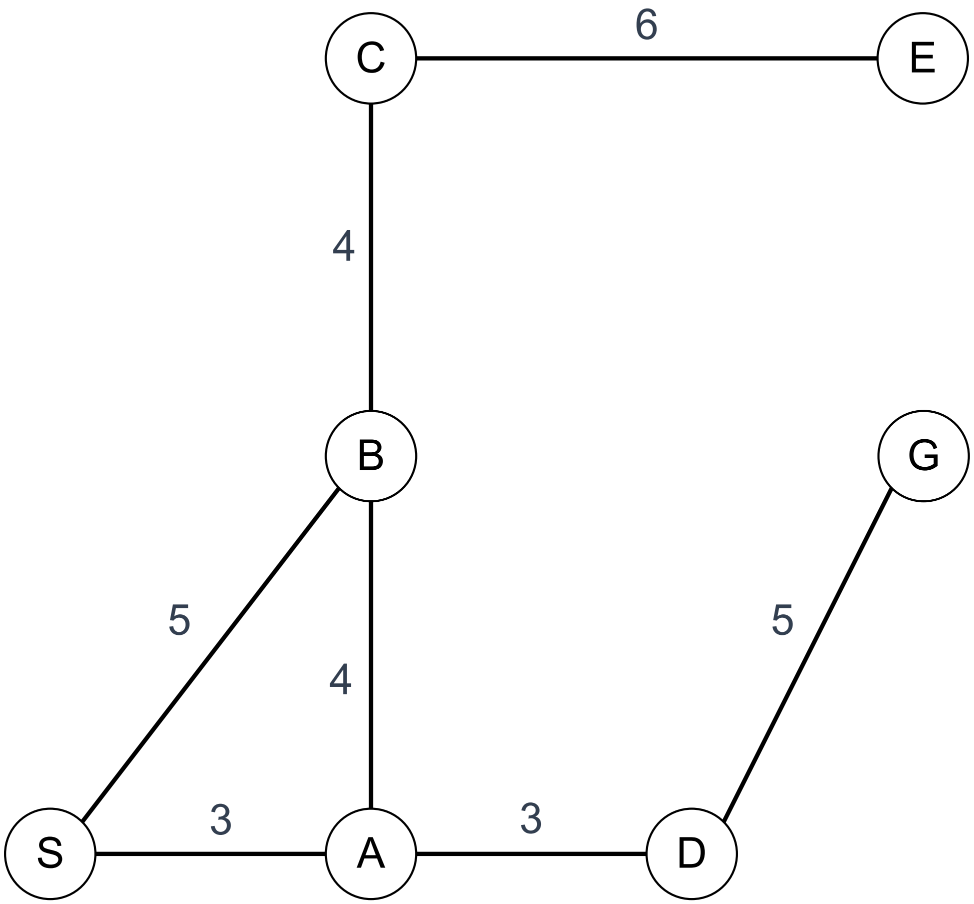




# Depth-First Search

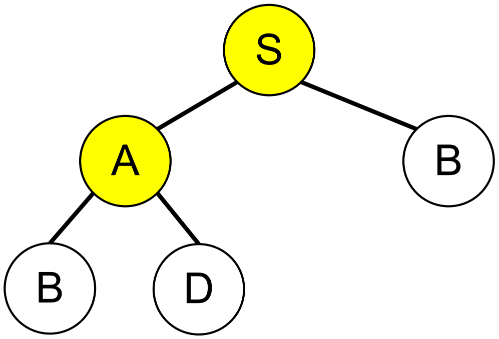
- When there’s a choice, pick left-most node and keep barreling down

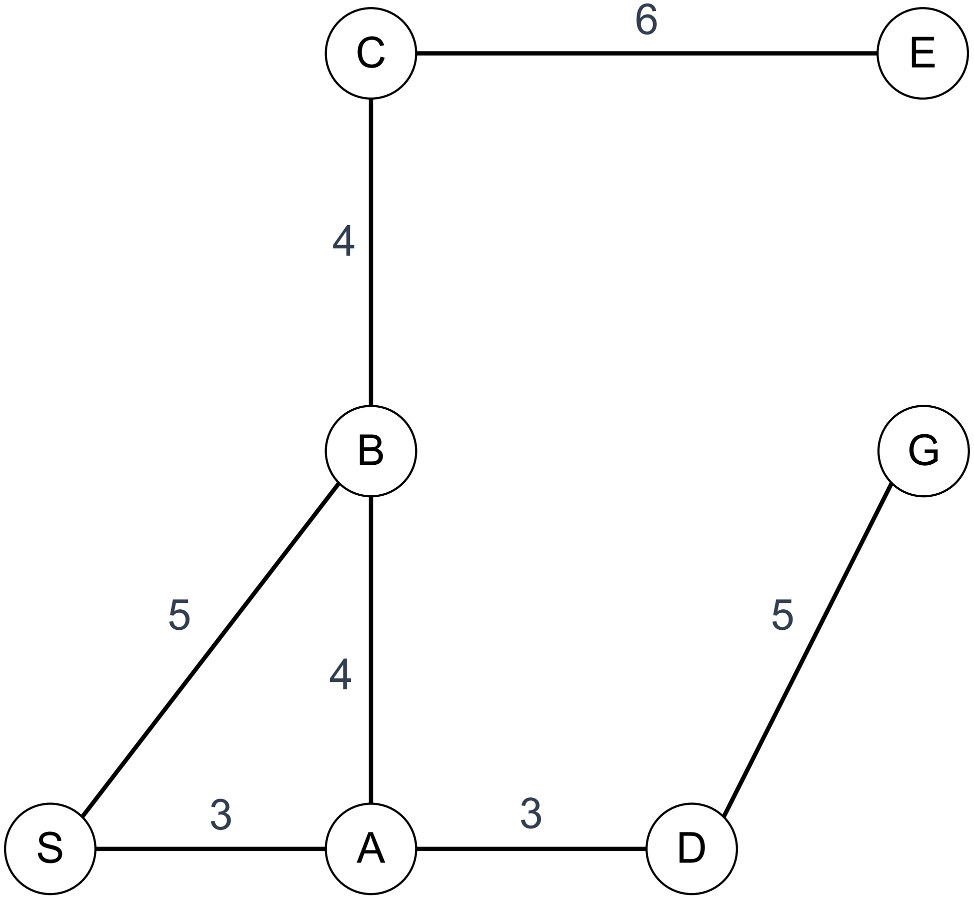




# Depth-First Search

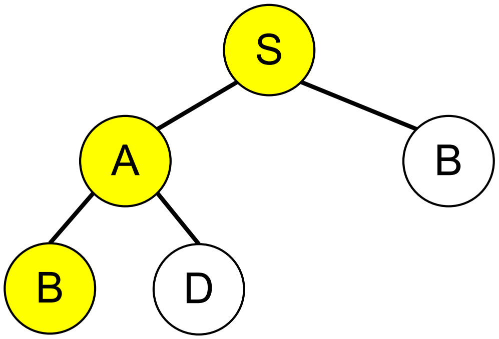
- When there's a choice, pick left-most node and keep barreling down

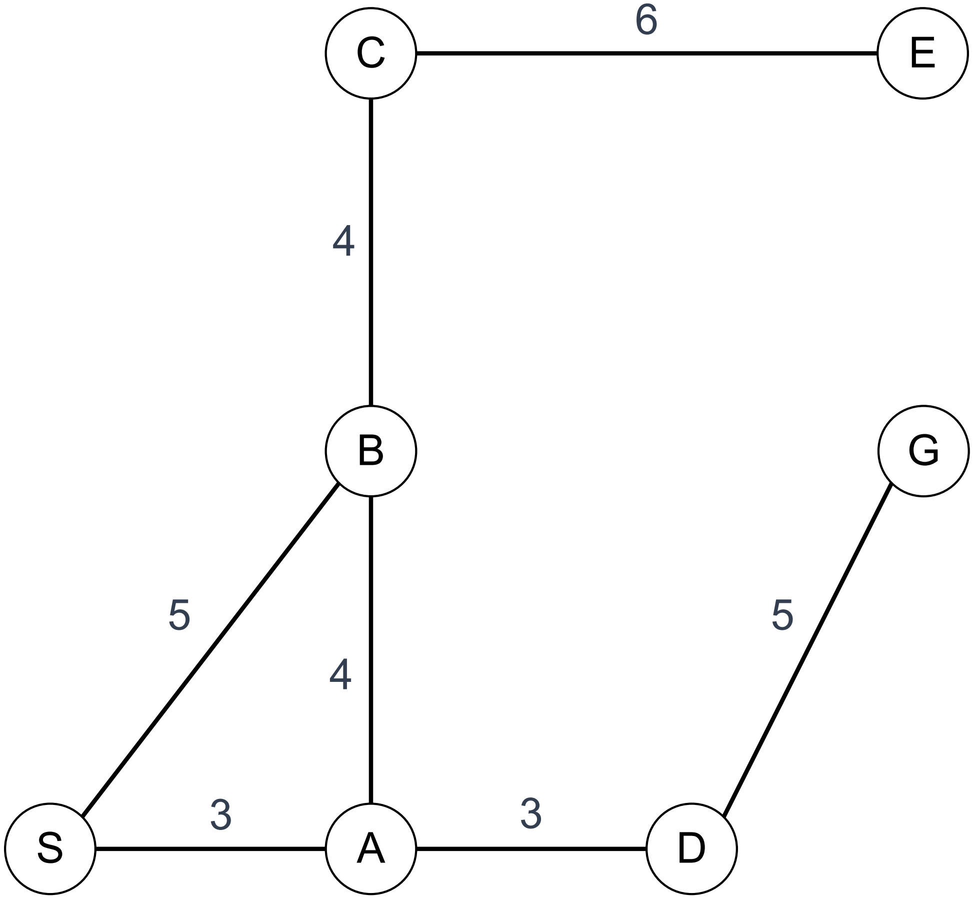




# Depth-First Search

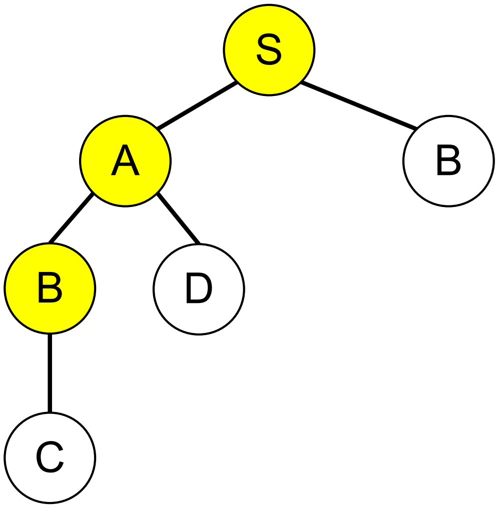
- When there's a choice, pick left-most node and keep barreling down

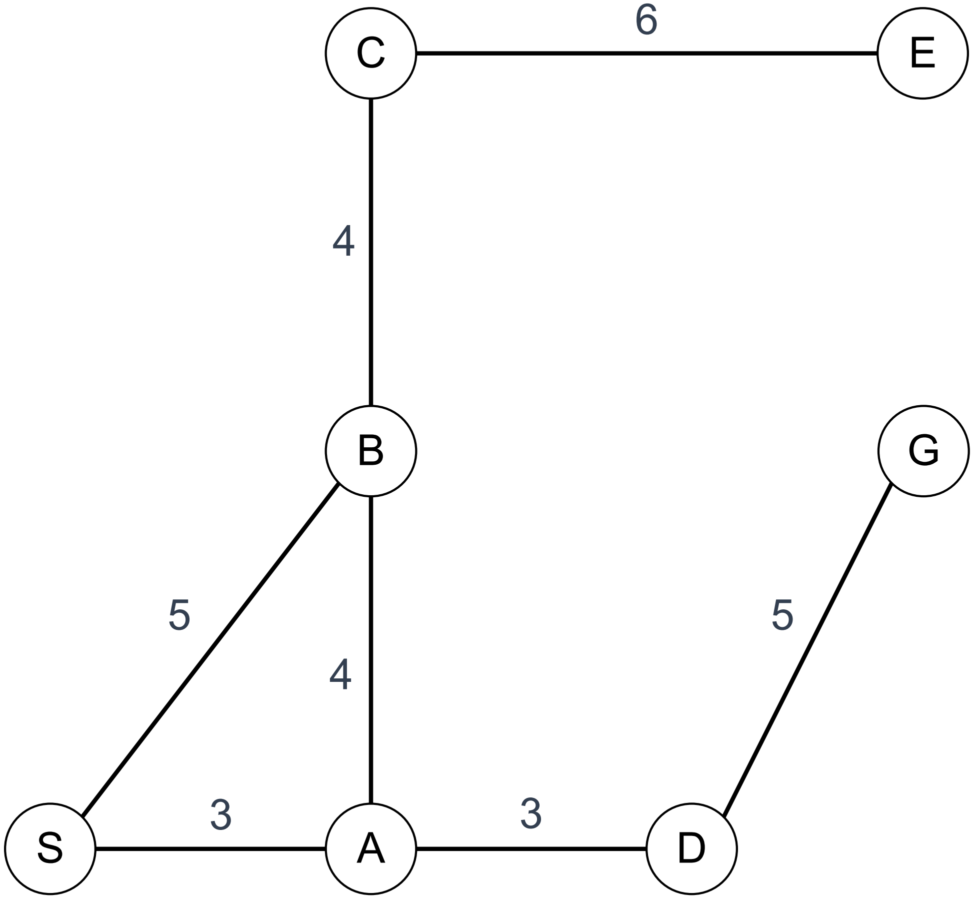




# Depth-First Search

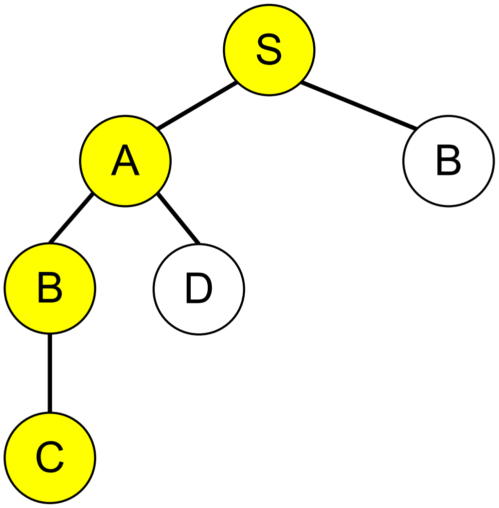
- When there's a choice, pick left-most node and keep barreling down

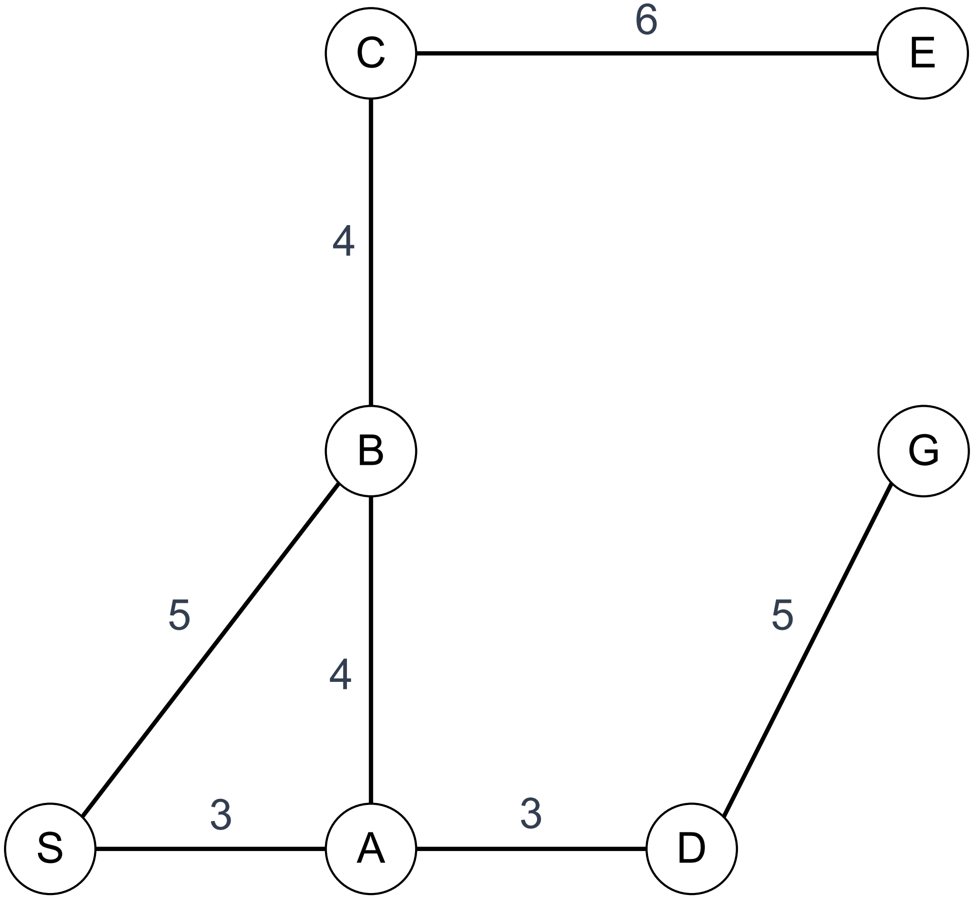




# Depth-First Search

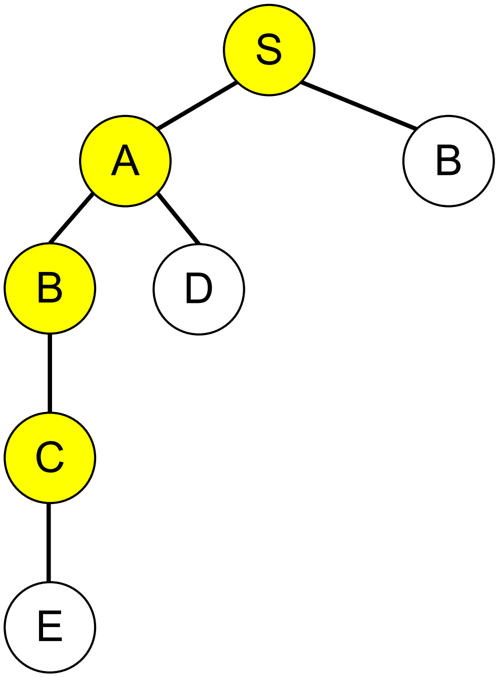
- When there's a choice, pick left-most node and keep barreling down

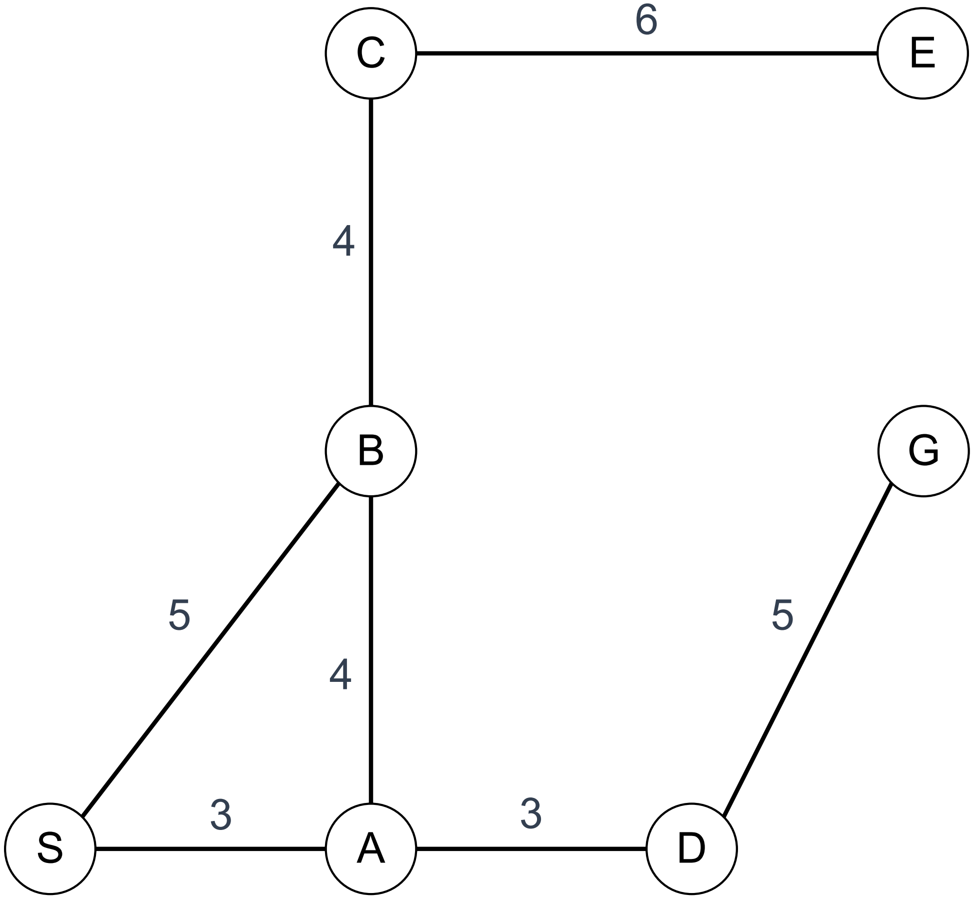




# Depth-First Search

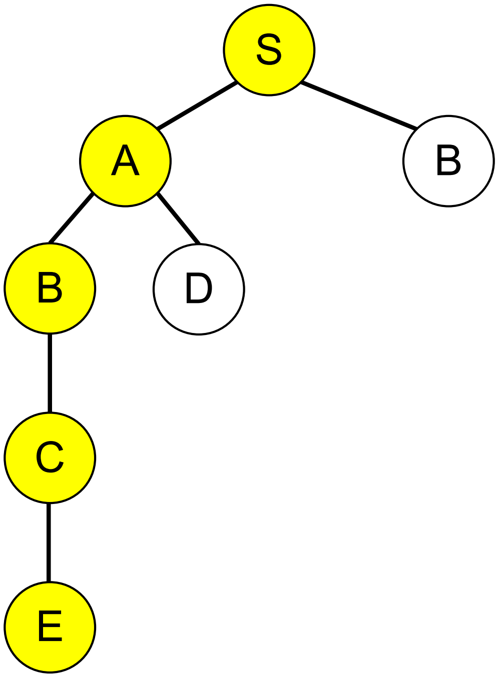
- When there's a choice, pick left-most node and keep barreling down

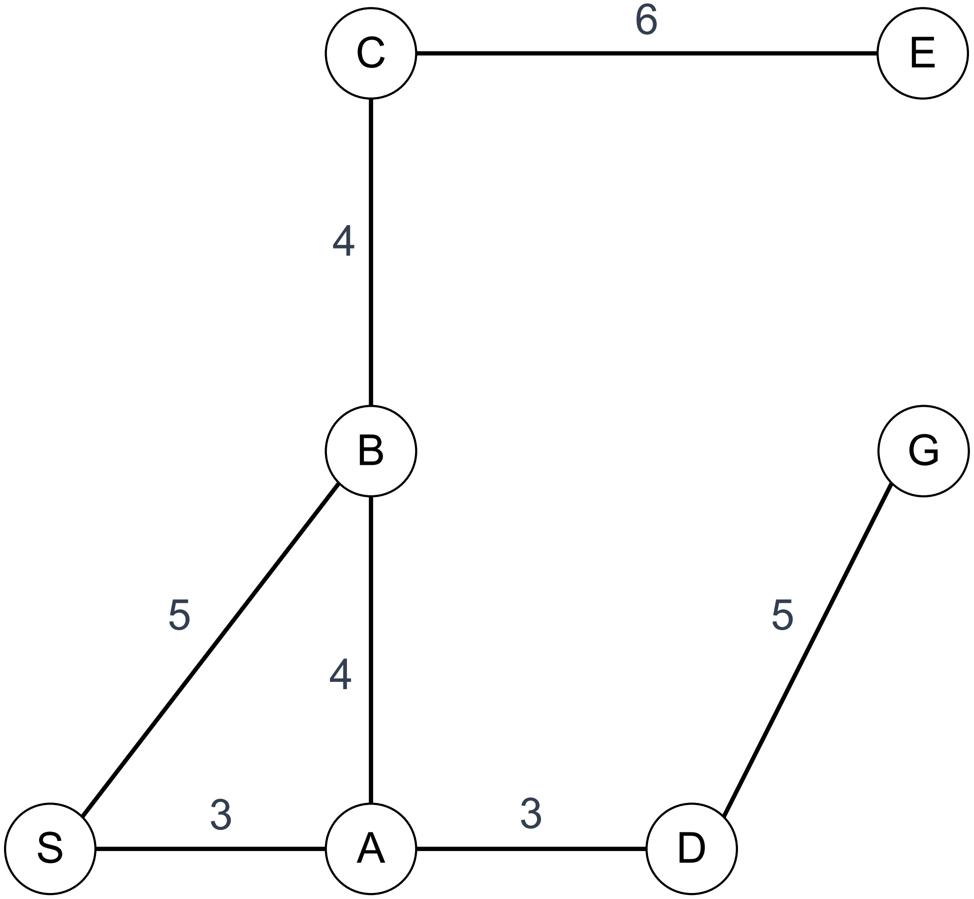




# Depth-First Search

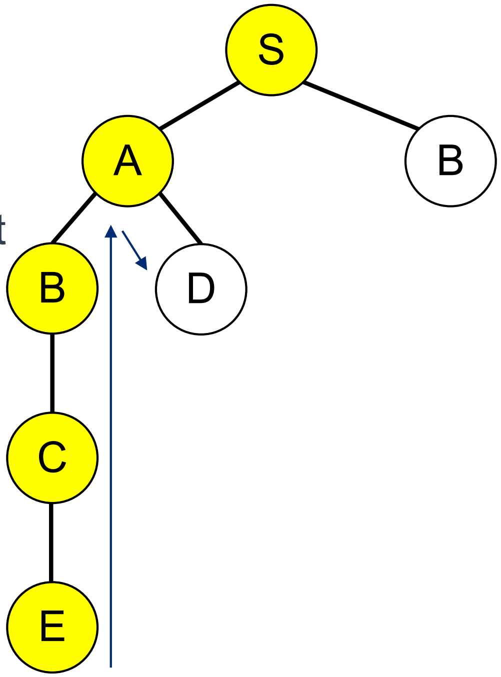
- When there's a choice, pick left-most node and keep barreling down

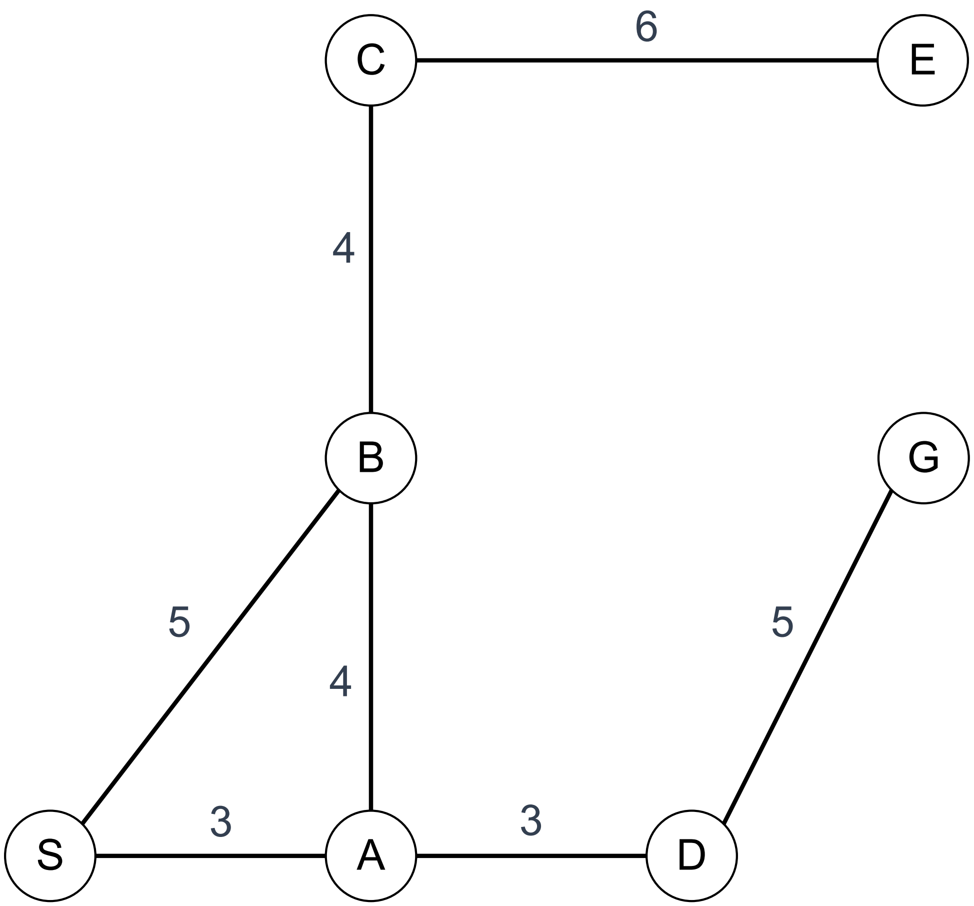




# Depth-First Search

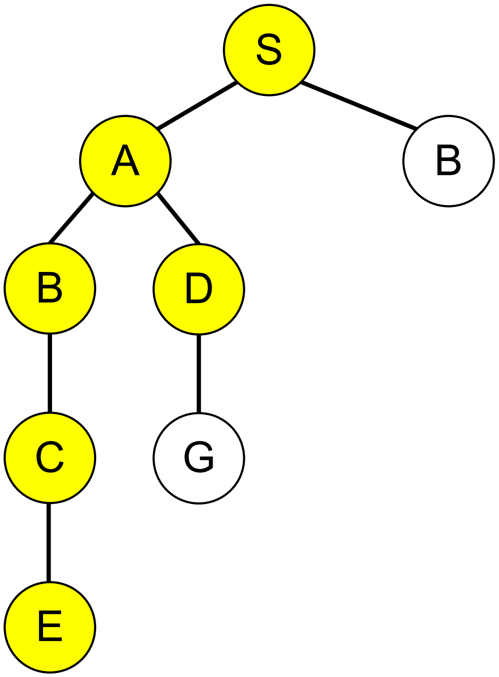
- Cant' expand anymore
- The path did not lead to goal
- Backtrack!
  - Go back to the last position a decision was made. (SAB vs SAD)
  - This is a feature that can be added (or not)

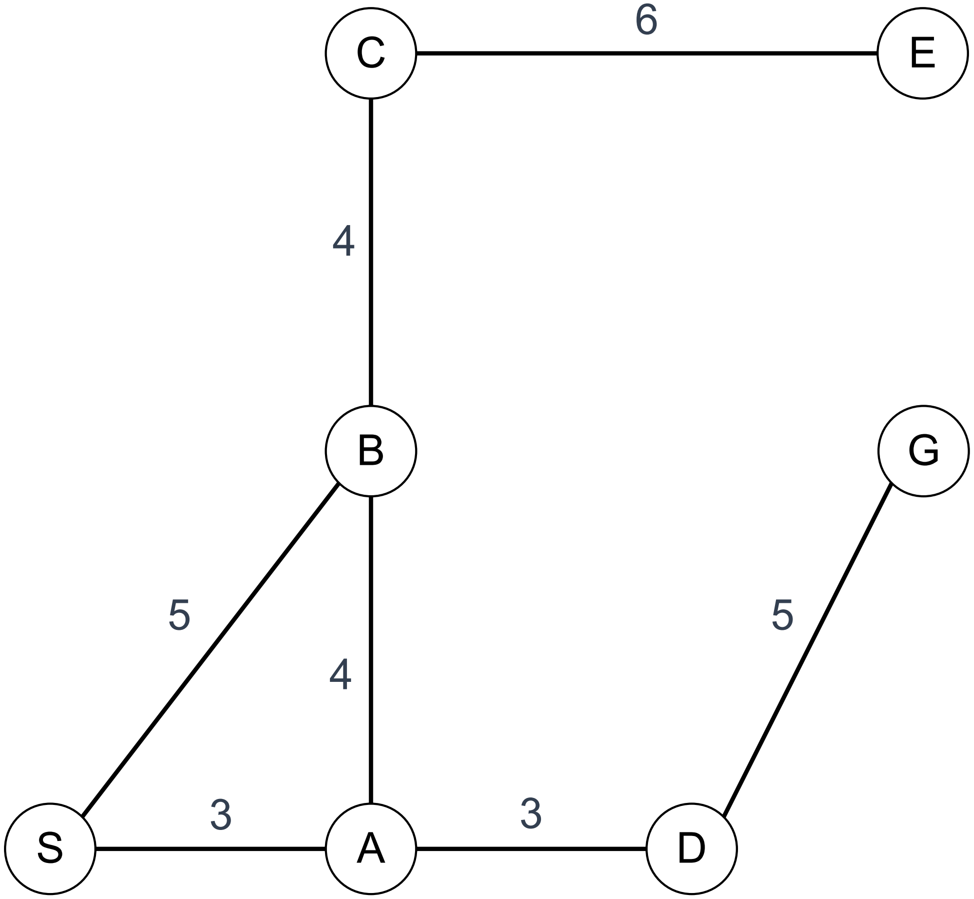




# Depth-First Search

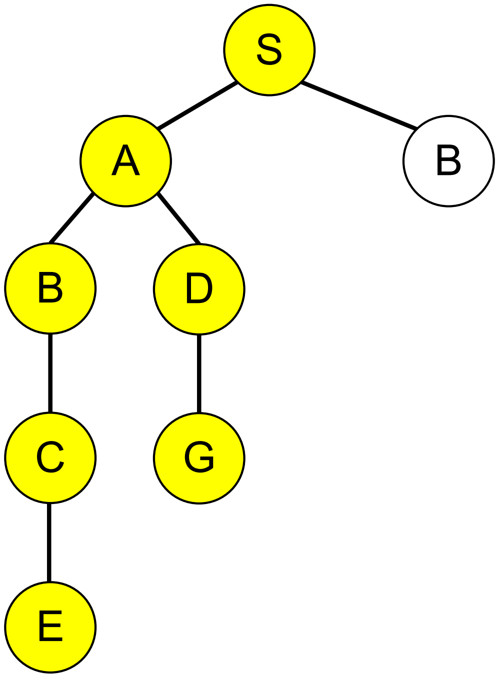
- When there's a choice, pick left-most node and keep barreling down

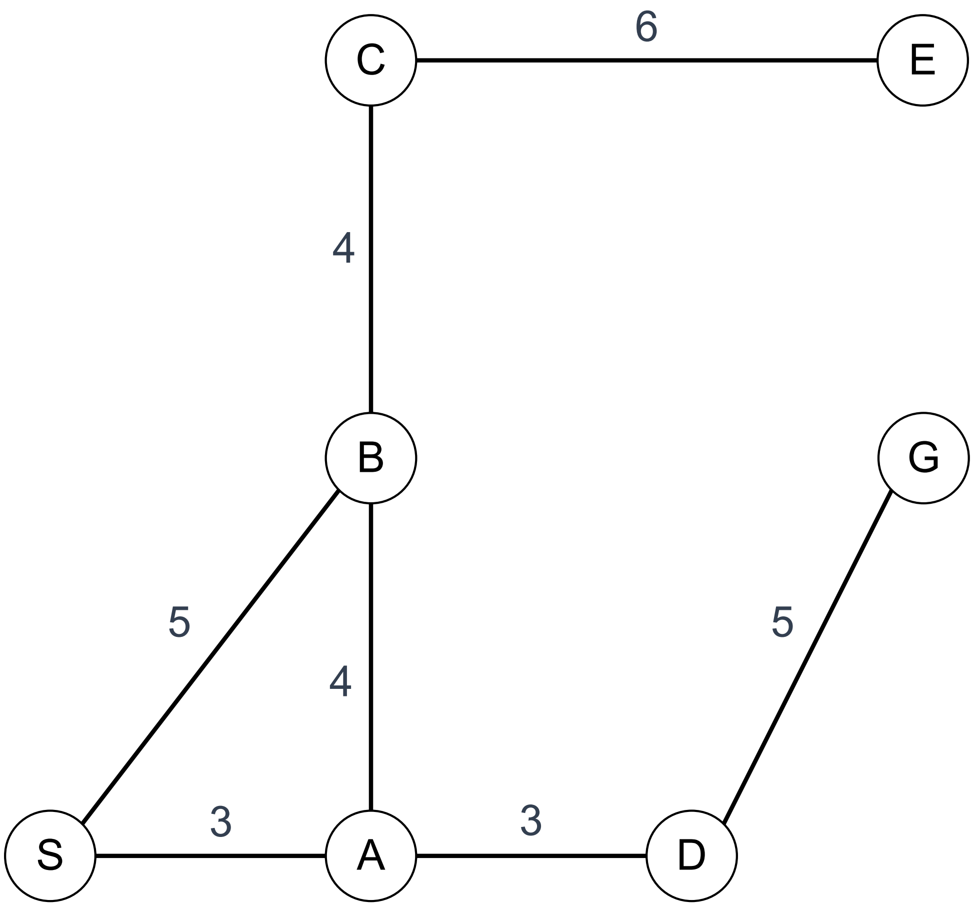




# Depth-First Search

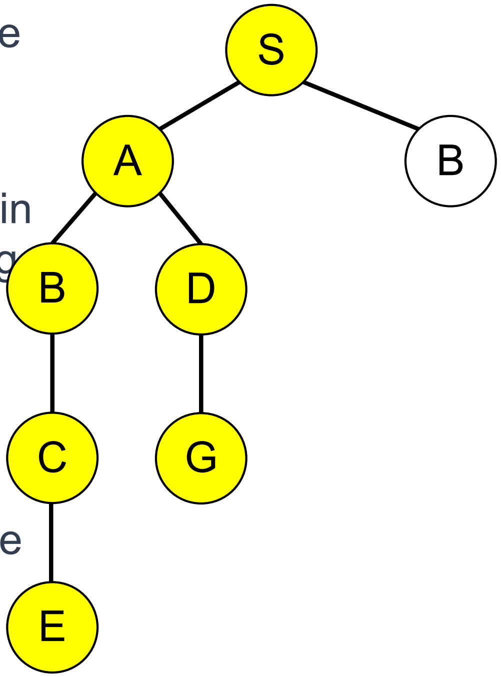
- Path contains goal, DONE

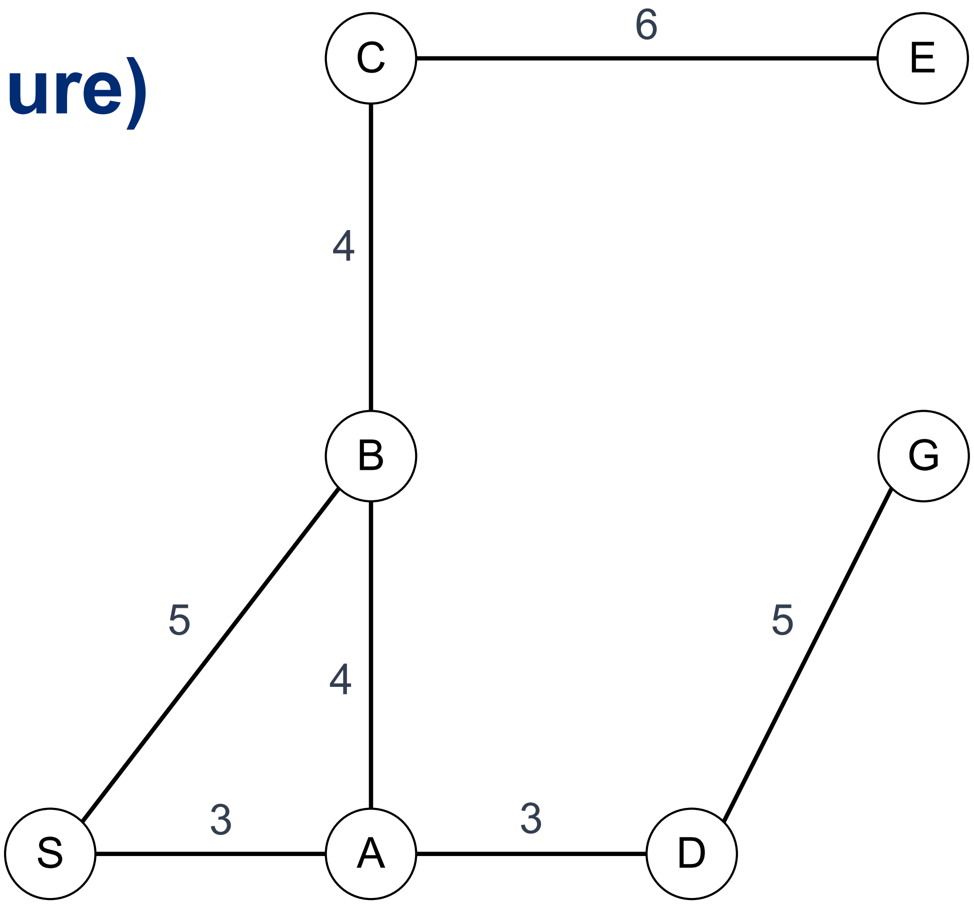




# Depth-First Search (Procedure)

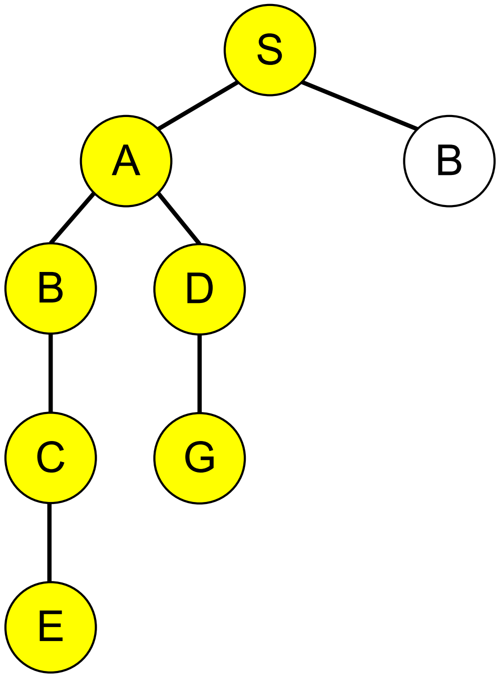
1. Initialize queue
2. Extend first path in queue (check if goal in that path being extended)
3. Enqueue: Place extensions in the ***front*** of the queue





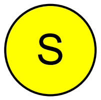
# Depth-First Search (Procedure)

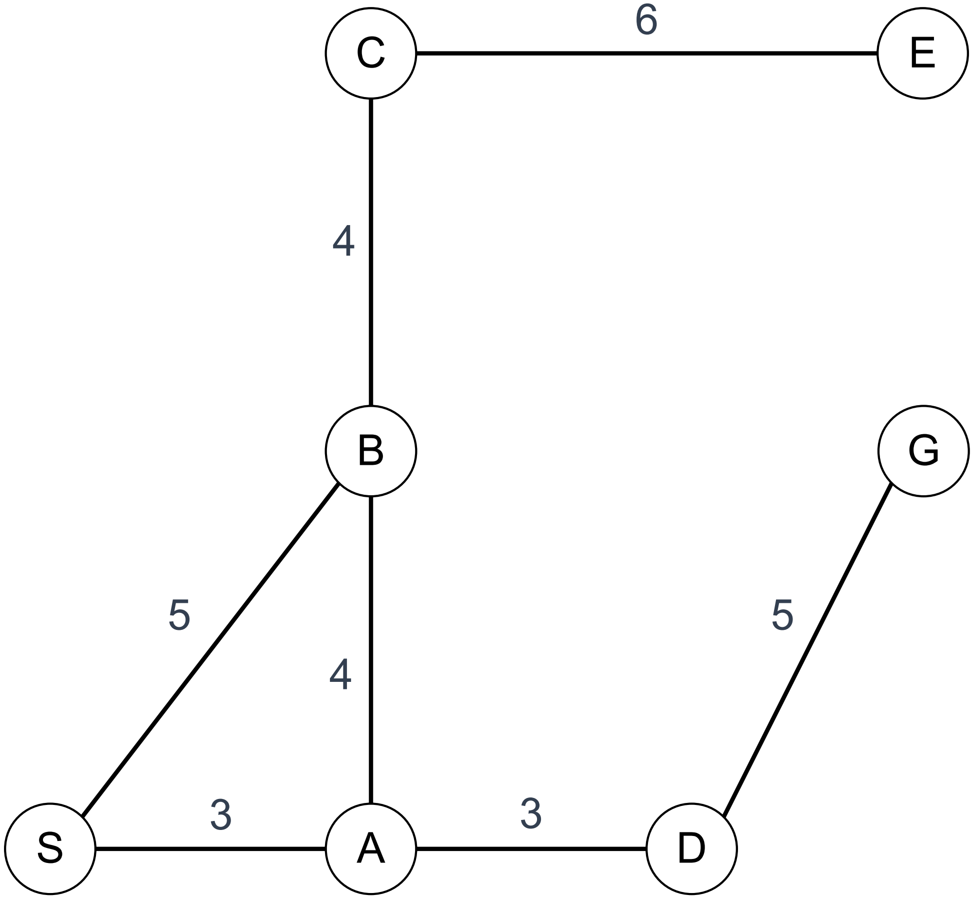
1. (S)
2. (SA)(SB)
3. (SAB)(SAD)(SB)
4. (SABC)(SAD)(SB)
5. (SABCE)(SAD)(SB)
6. (SAD)(SB)
7. (SADG)(SB)
8. DONE



# Breadth-First Search

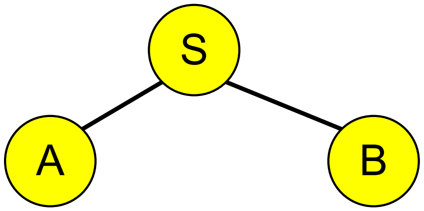
- Search the tree level by level

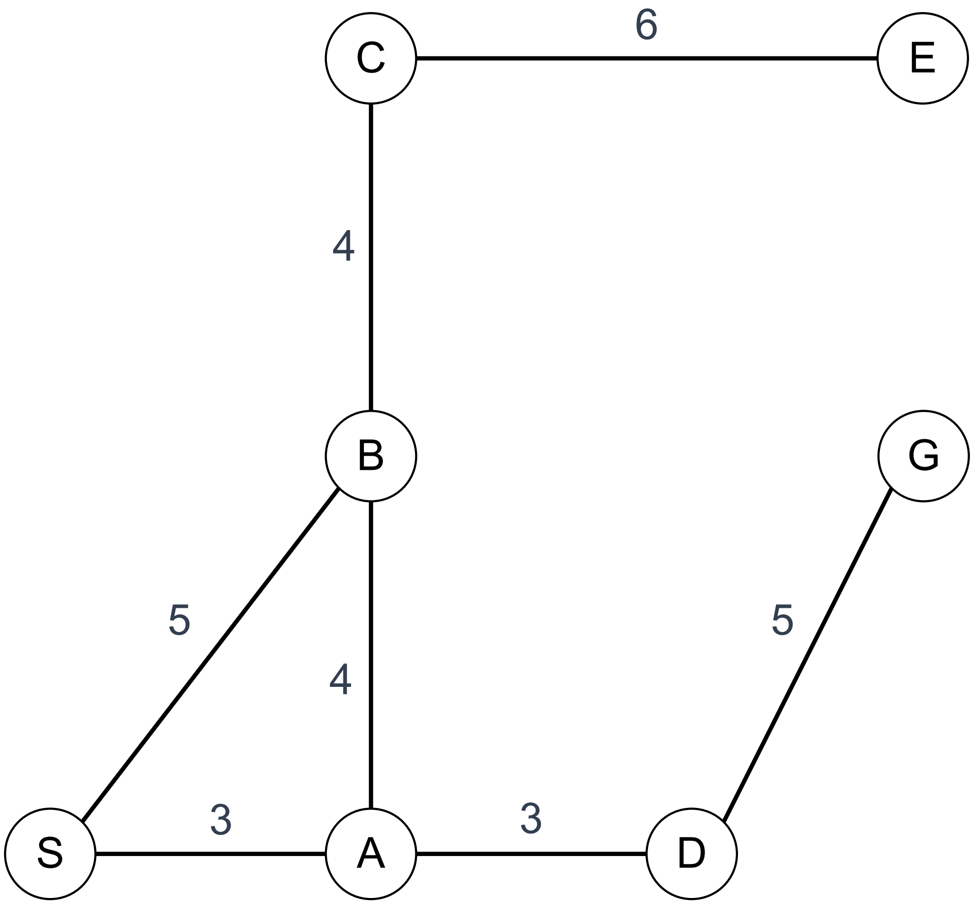




# Breadth-First Search

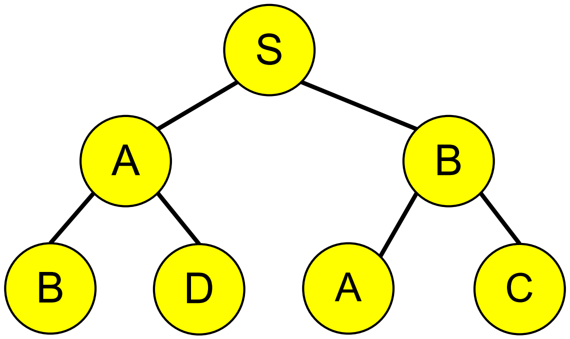
- Search the tree level by level

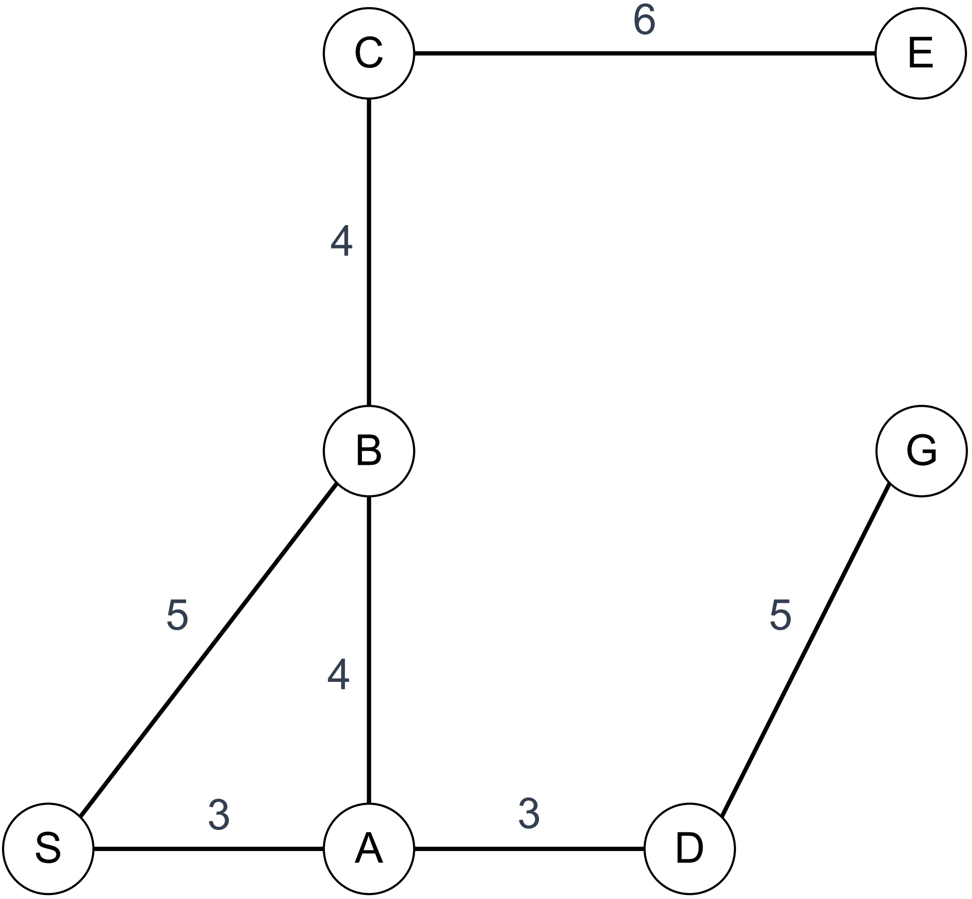




# Breadth-First Search

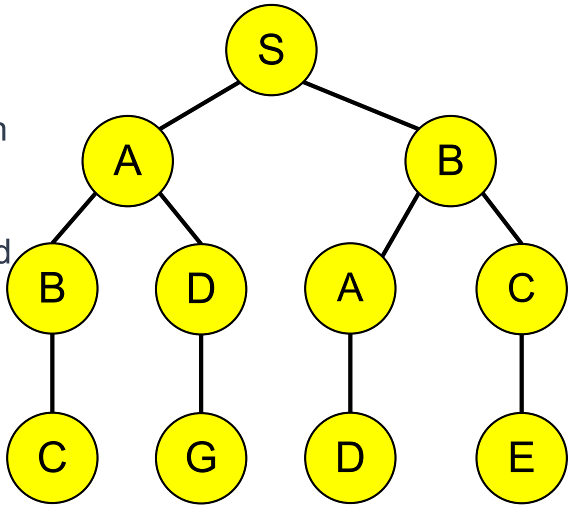
- Search the tree level by level

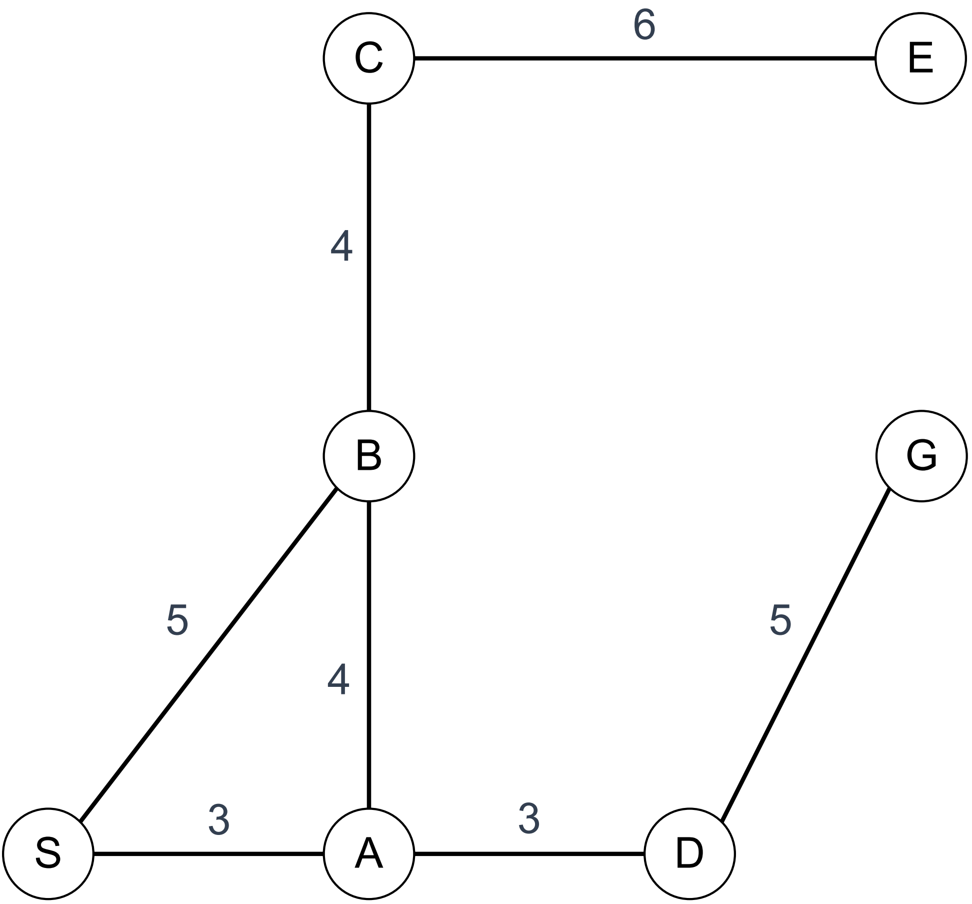




# Breadth-First Search

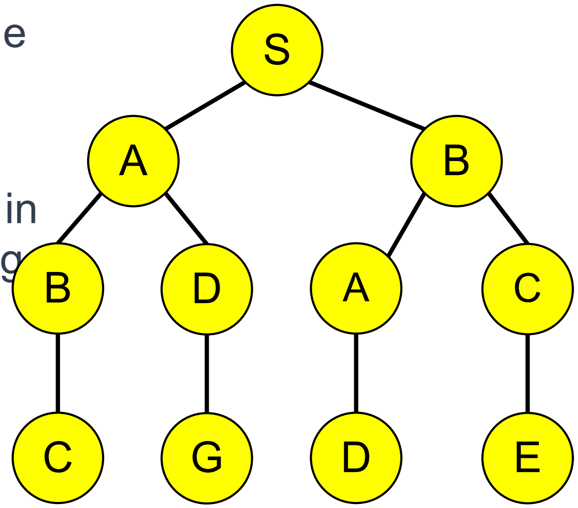
- DONE
  - Typical implementation detail: each level was further checked node by node

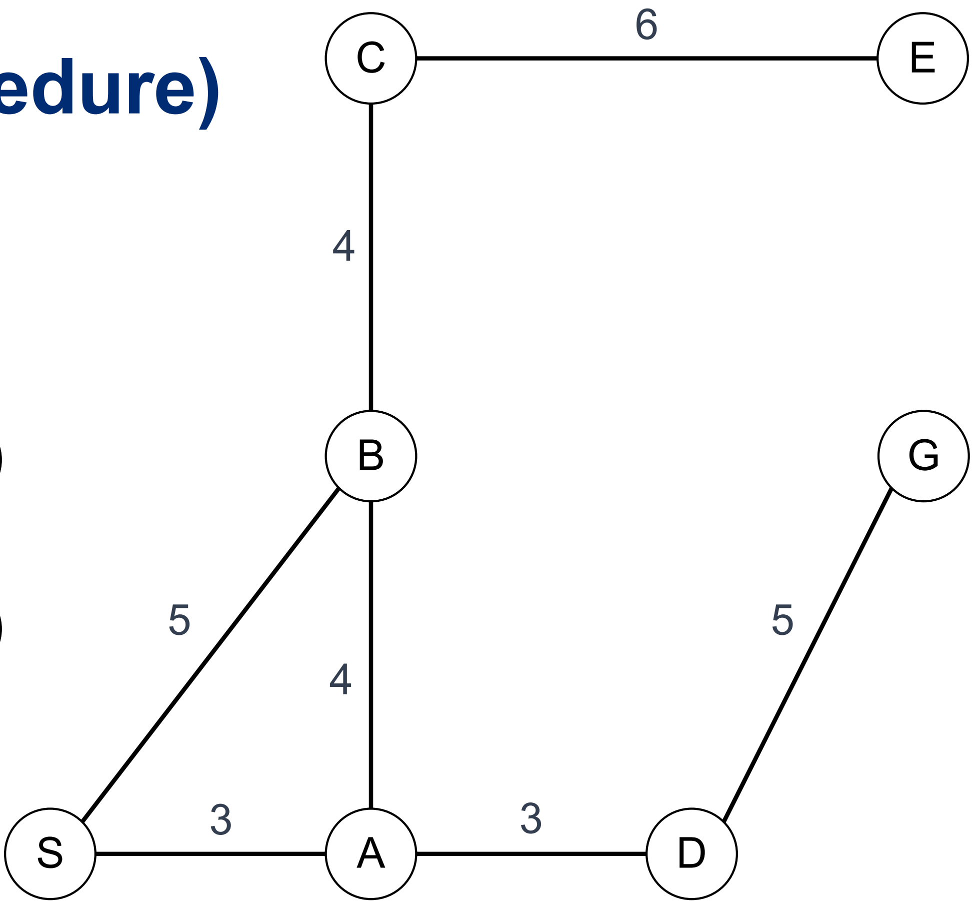




# Breadth-First Search (Procedure)

1. Initialize queue
2. Extend first path in queue (check if goal in that path being extended)
3. Enqueue: Place extensions in the ***back*** of the queue

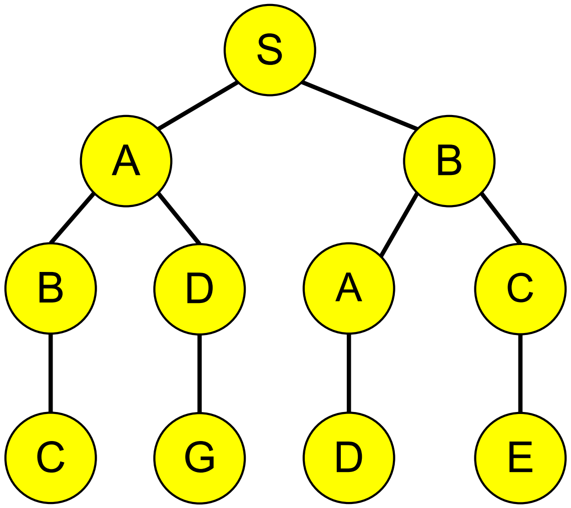




# Breadth-First Search (Procedure)

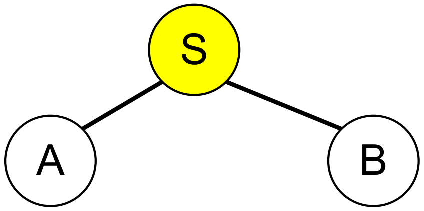
1. (S)
2. (SA)(SB)
3. (SB)(SAB)(SAD)
4. (SAB)(SAD)(SBA)(SBC)
5. (SAD)(SBA)(SBC)(SABC)
6. (SBA)(SBC)(SABC)(SADG)
7. (SBC)(SABC)(SADG)(SBAD)
8. (SABC)(SADG)(SBAD)(SBCE)
9. (SADG)(SBAD)(SBCE)(SABCE)

Don’t stop here

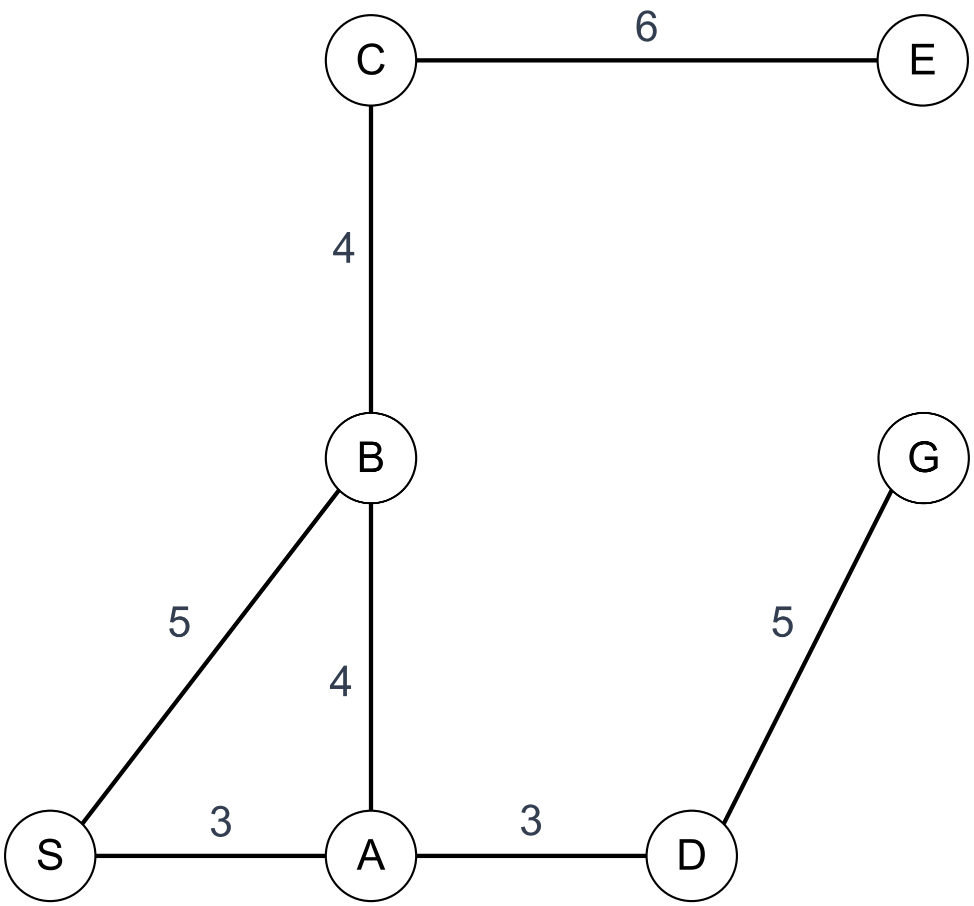


# Hill Climbing Search

- Informed version of depth-first search

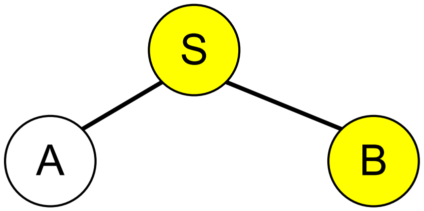


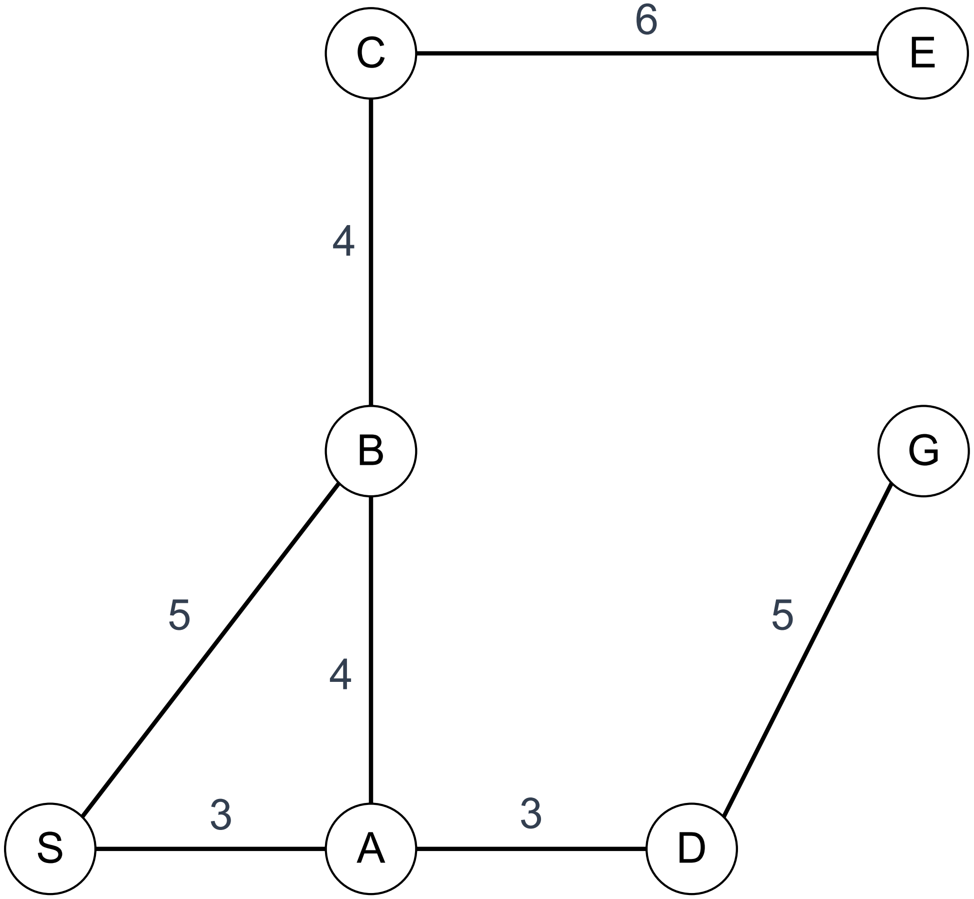
- Make use of “helpful” information
- For instance, if ***straight line distance (SLD)*** from a node to goal is provided, the idea is to use it, rather than ignore this help



# Hill Climbing Search

- Informed version of depth-first search
- For instance, if straight line distance (SLD) from a node to goal is provided, the idea is to use it, rather than ignore this help

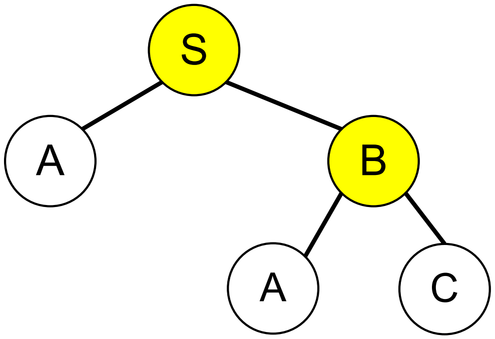


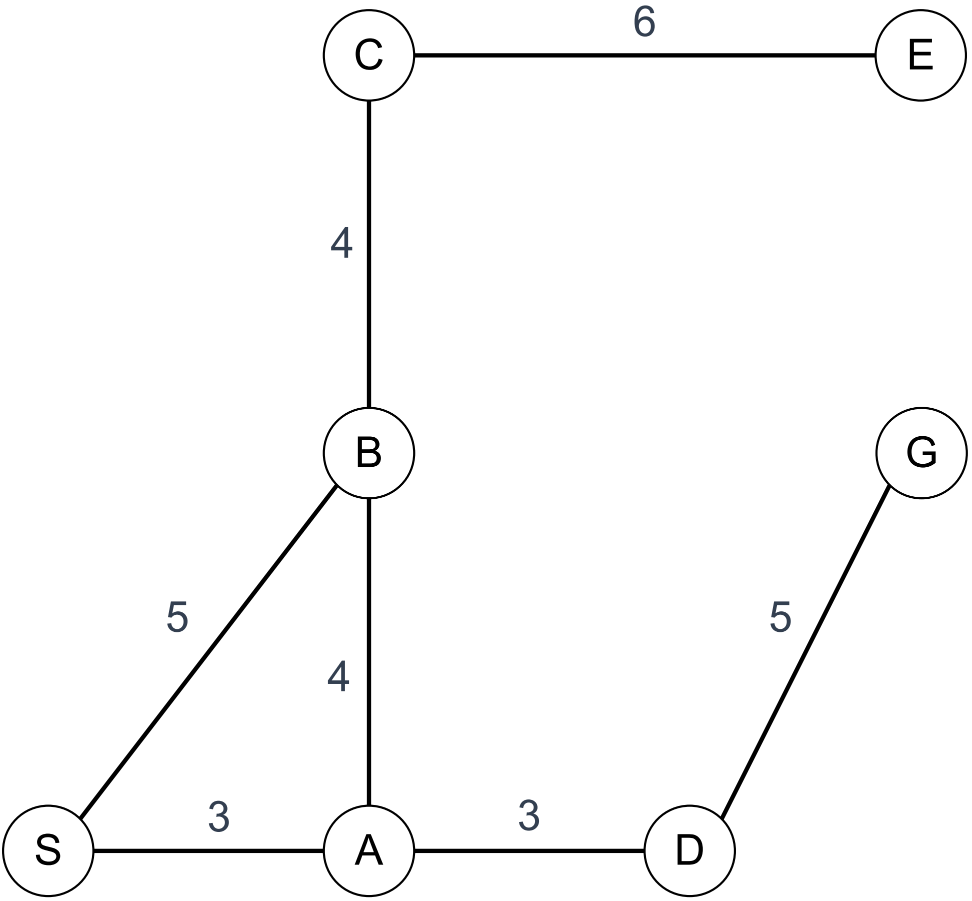


- SLD(A,G) > SLD(B,G)
  - B closer to goal

# Hill Climbing Search

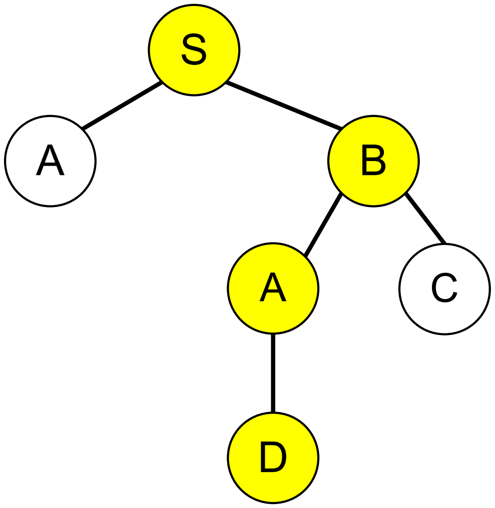
- Informed version of depth-first search
- For instance, if straight line distance (SLD) from a node to goal is provided, the idea is to use it, rather than ignore this help
- SLD(A,G) = SLD(C,G)
  - Lexical ordering wins

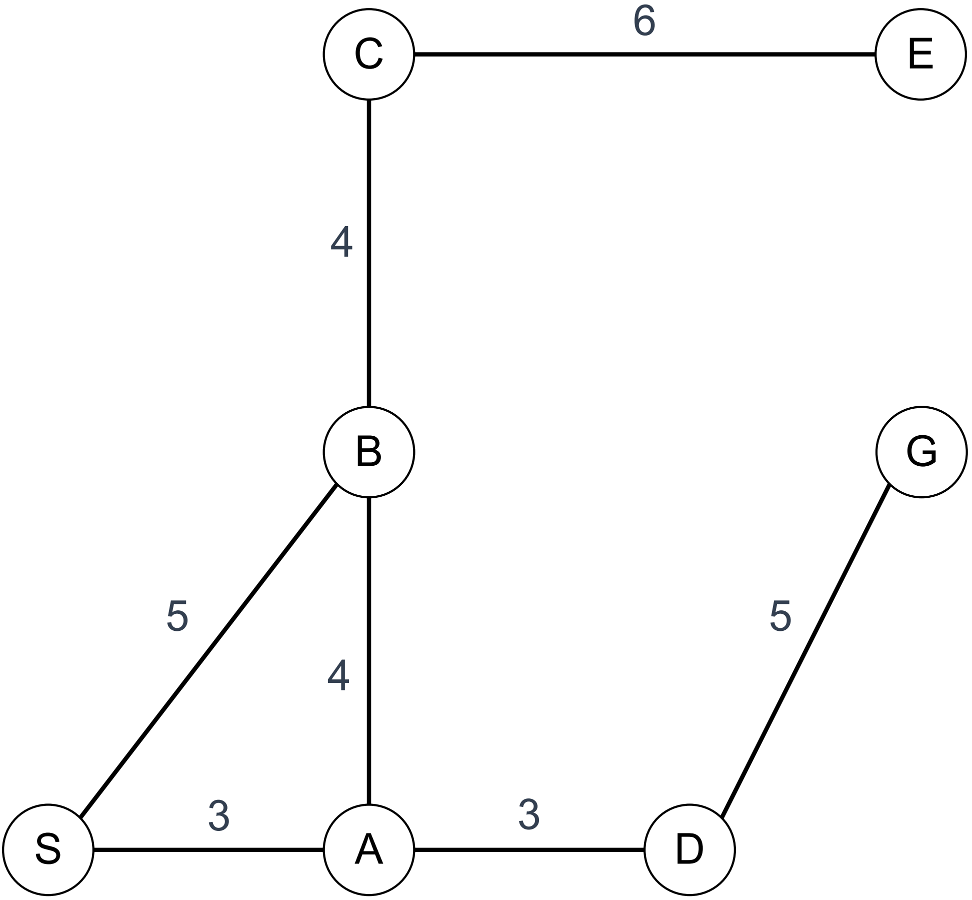




# Hill Climbing Search

- Informed version of depth-first search
- For instance, if straight line distance (SLD) from a node to goal is provided, the idea is to use it, rather than ignore this help



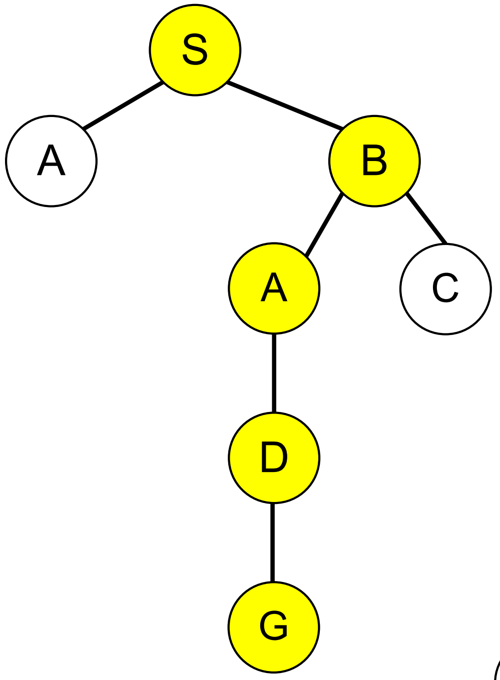


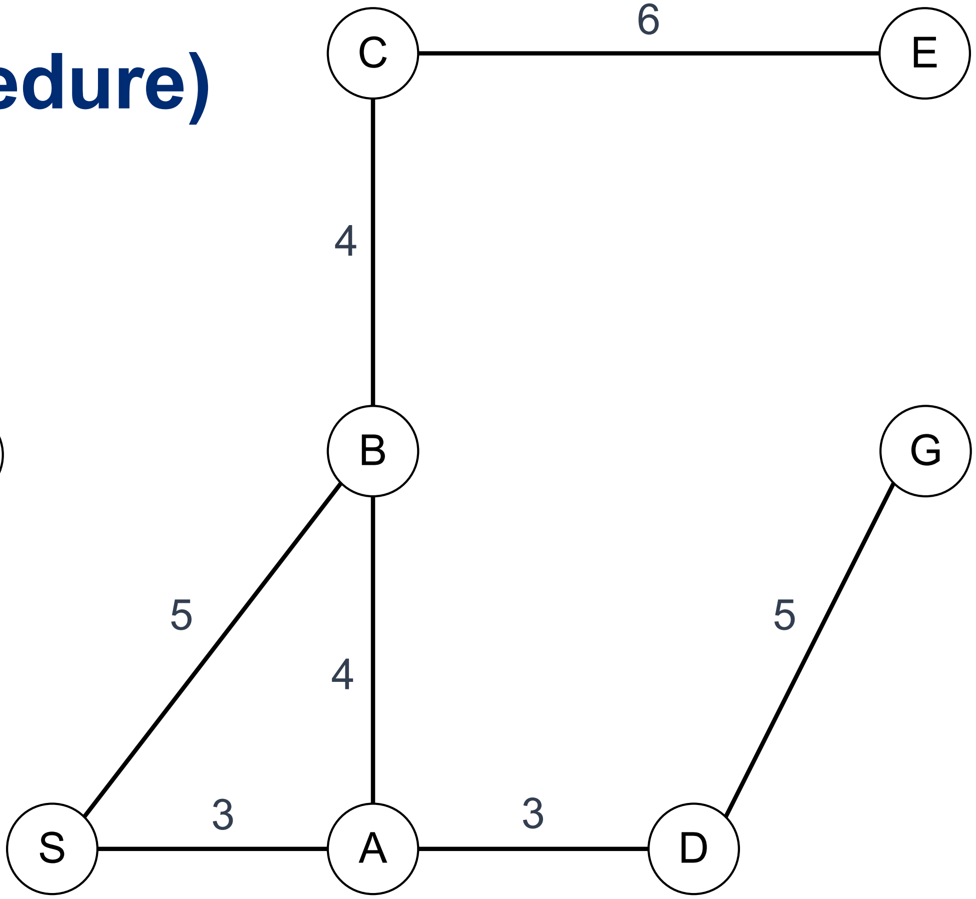
- SLD(A,G) = SLD(C,G)
  - Lexical ordering wins

# Hill Climbing Search (Procedure)

- DONE.

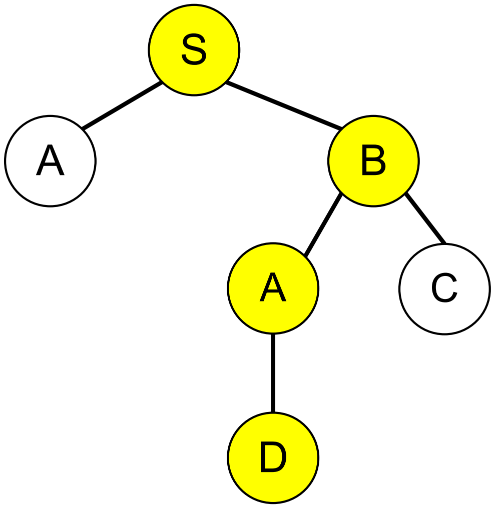
- Procedure same as Depth-First search, but sorted

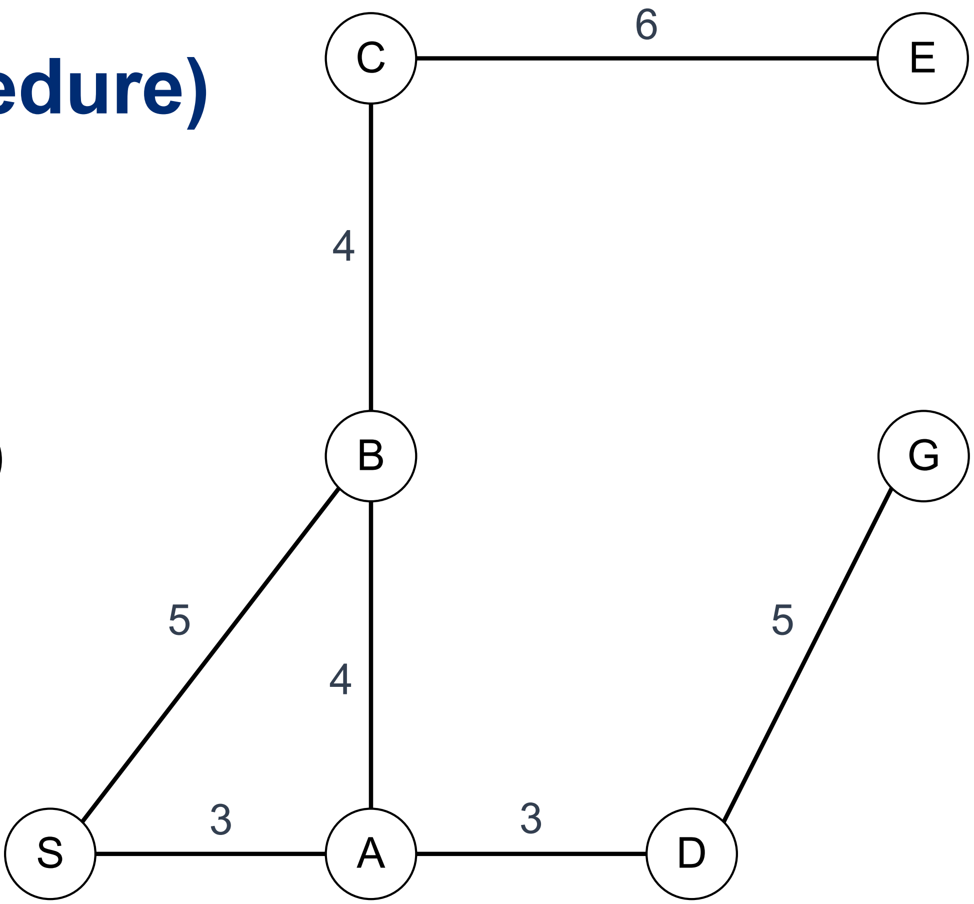




# Hill Climbing Search (Procedure)

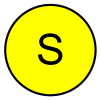
1. Initialize queue
2. Extend first path in queue (check if goal in that path being extended)
3. Enqueue: Place extensions in the ***front*** of the queue (after ***sorting*** by "helpful" information)

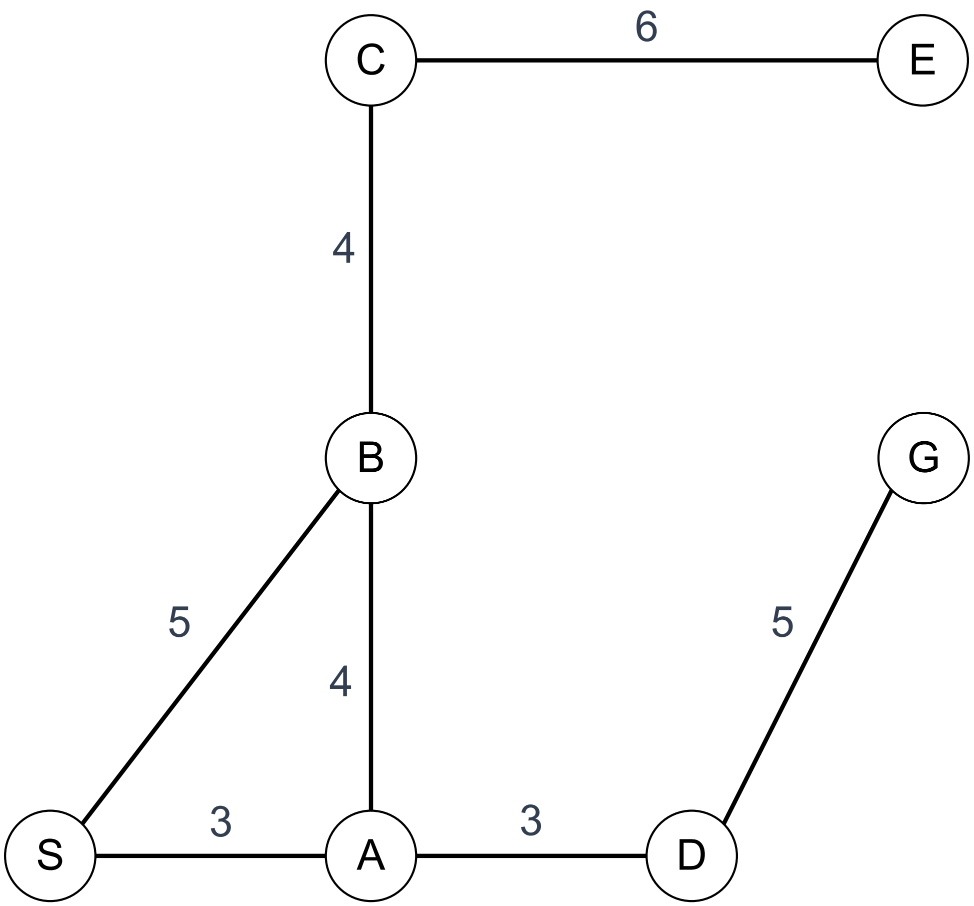




# Beam Search

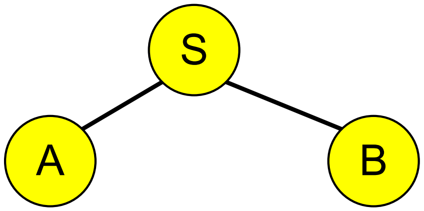
- Informed version of breadth-first search
- At each level, use “helpful” information to consider only the *w* “best” nodes (the beam width)





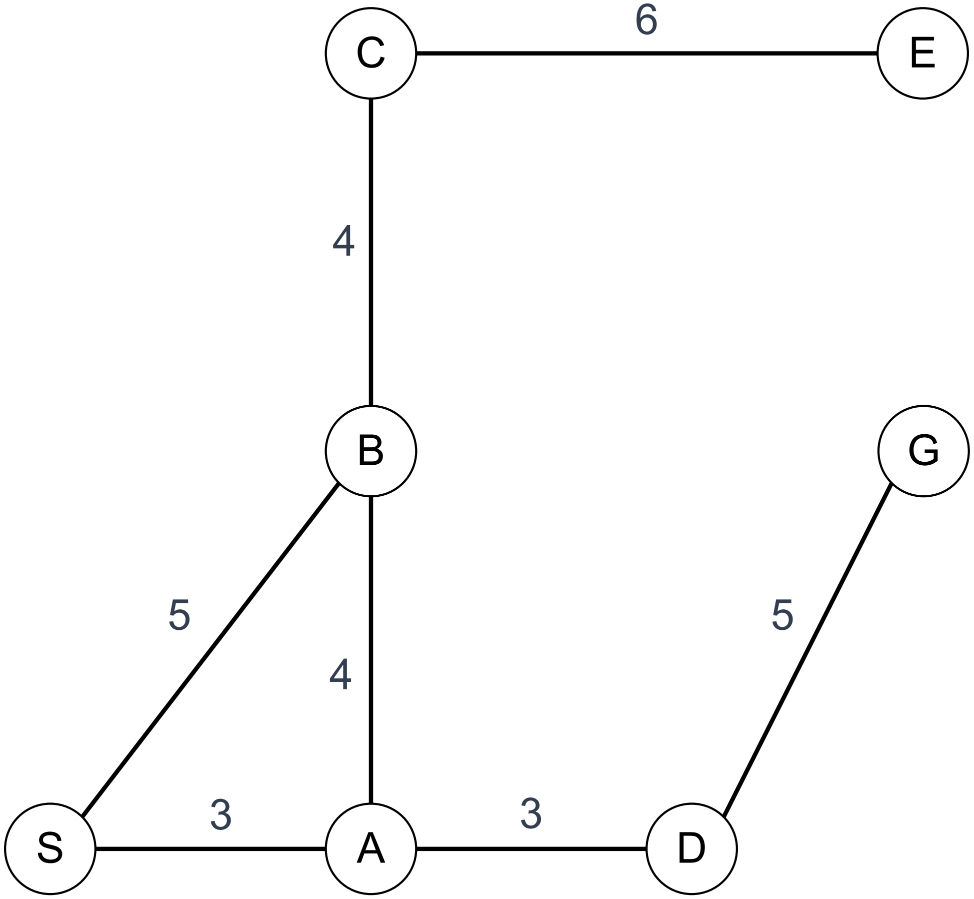
# Beam Search

- Informed version of breadth-first search



- At each level, use "helpful" information to consider only the *w* "best" nodes (the beam width)

- Let *w* = 2
    - We consider all the nodes at this level

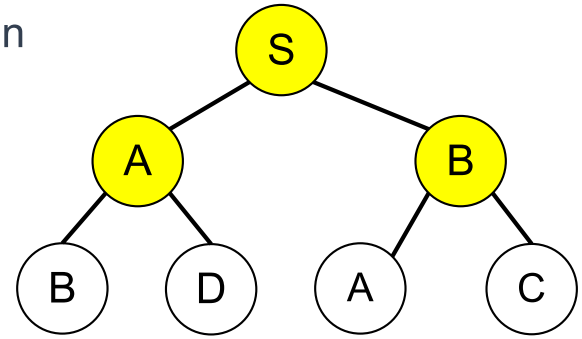


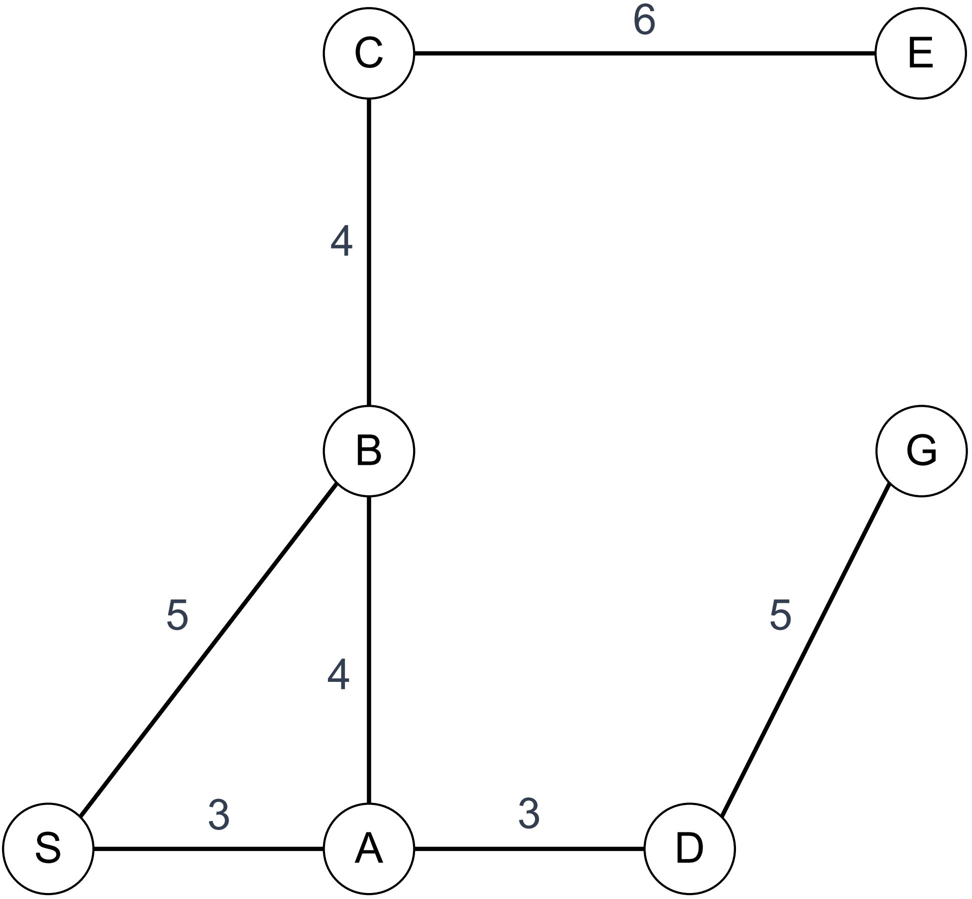
# Beam Search

- Informed version of breadth-first search

- At each level, use “helpful” information to consider only the *w* “best” nodes (the beam width)

- Let *w* = 2
  - We consider the top two nodes at this level



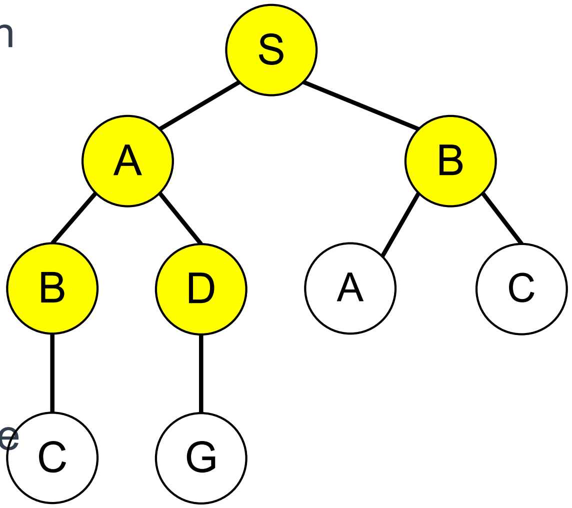


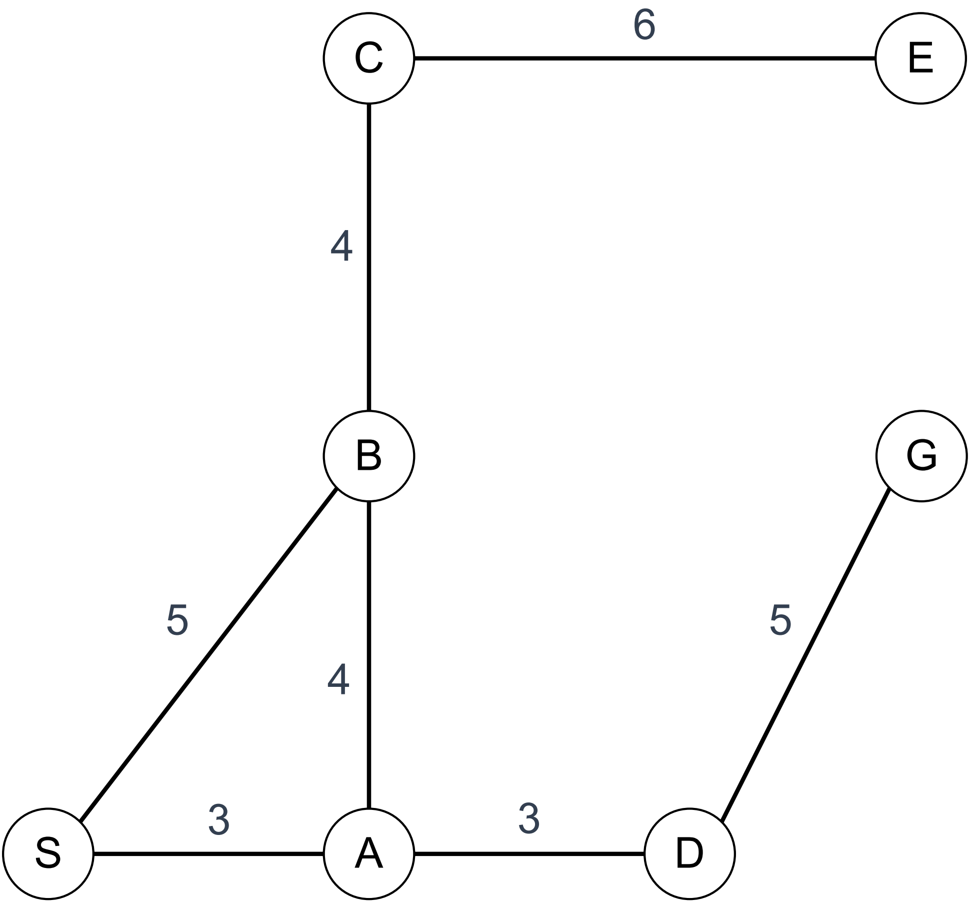
# Beam Search

- Informed version of breadth-first search

- At each level, use “helpful” information to consider only the *w* “best” nodes (the beam width)

- Let *w* = 2
  - We consider the top two nodes at this level



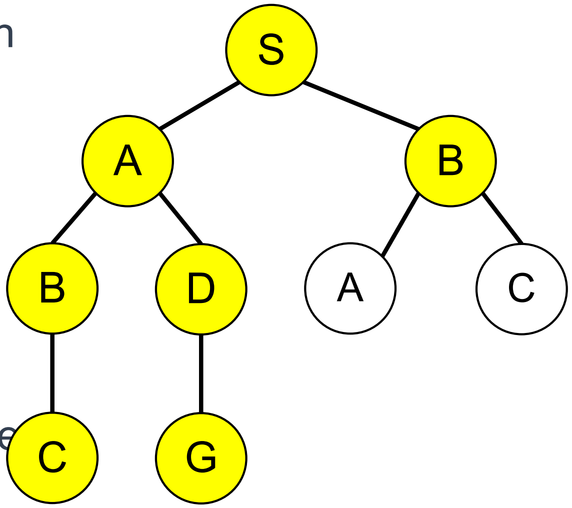


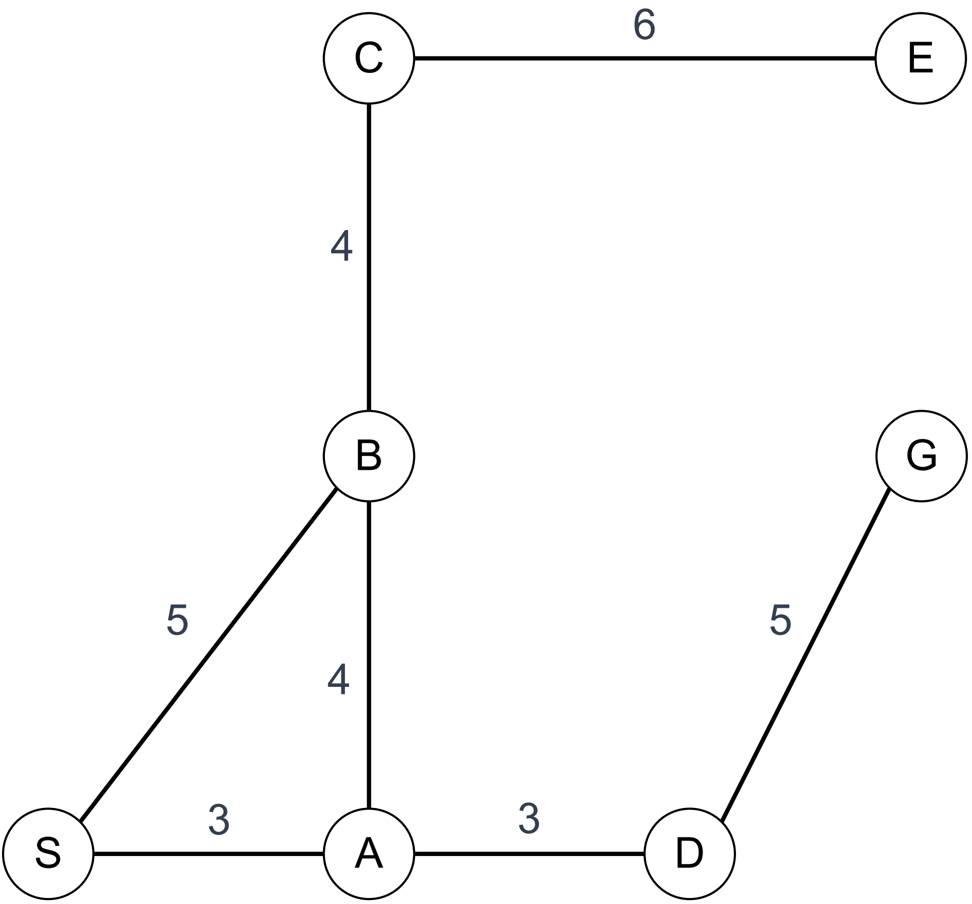
# Beam Search

- Informed version of breadth-first search

- At each level, use “helpful” information to consider only the *w* “best” nodes (the beam width)

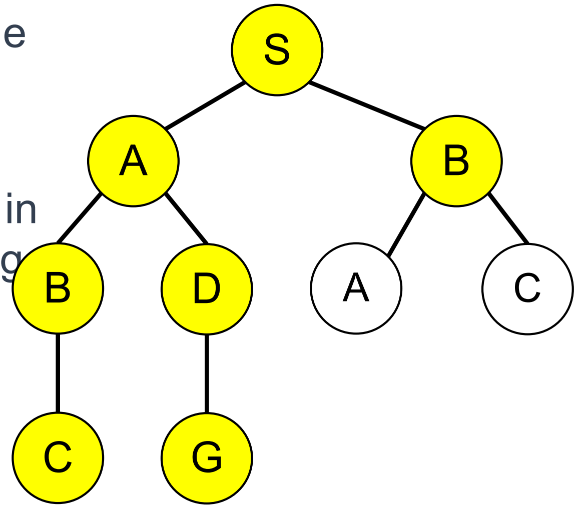
- Let *w* = 2
  - We consider all the nodes at this level

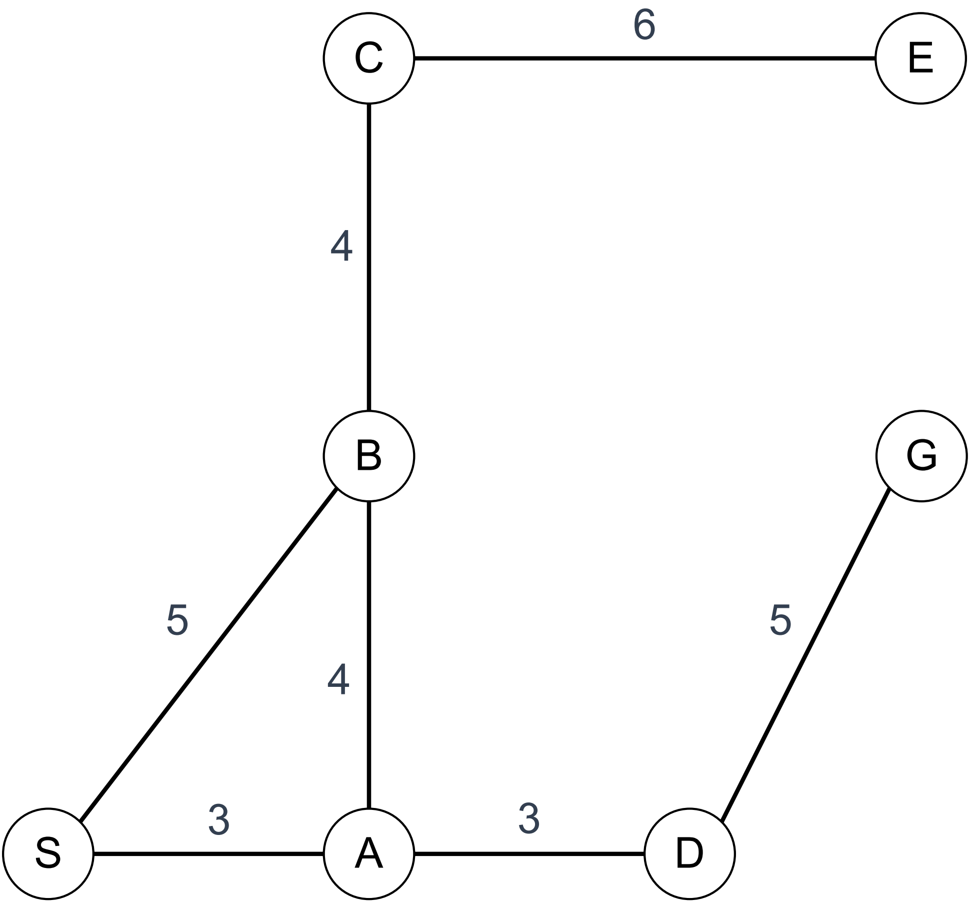




# Beam Search (Procedure)

1. Initialize queue
2. Extend first path in queue (check if goal in that path being extended)
3. Enqueue: Place extensions ***anywhere*** on the queue (***keeping*** the best *w* nodes)





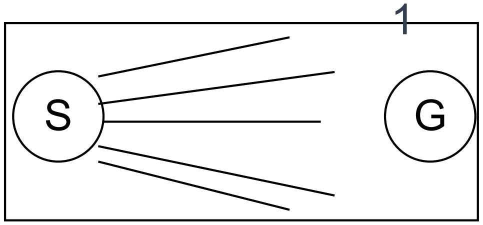
# Flourishes

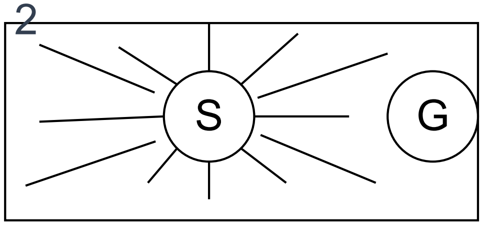
- Use the concept of Extended Lists to improve efficiency
  - Don’t extend nodes that have already been extended
  - Can be added as a feature to any of the four searches described above

- Backtracking doesn’t make sense for Breadth-First type searches

- Hill Climbing and Beam Search make use of information

Information like SLD perhaps not so helpful in this case



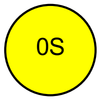


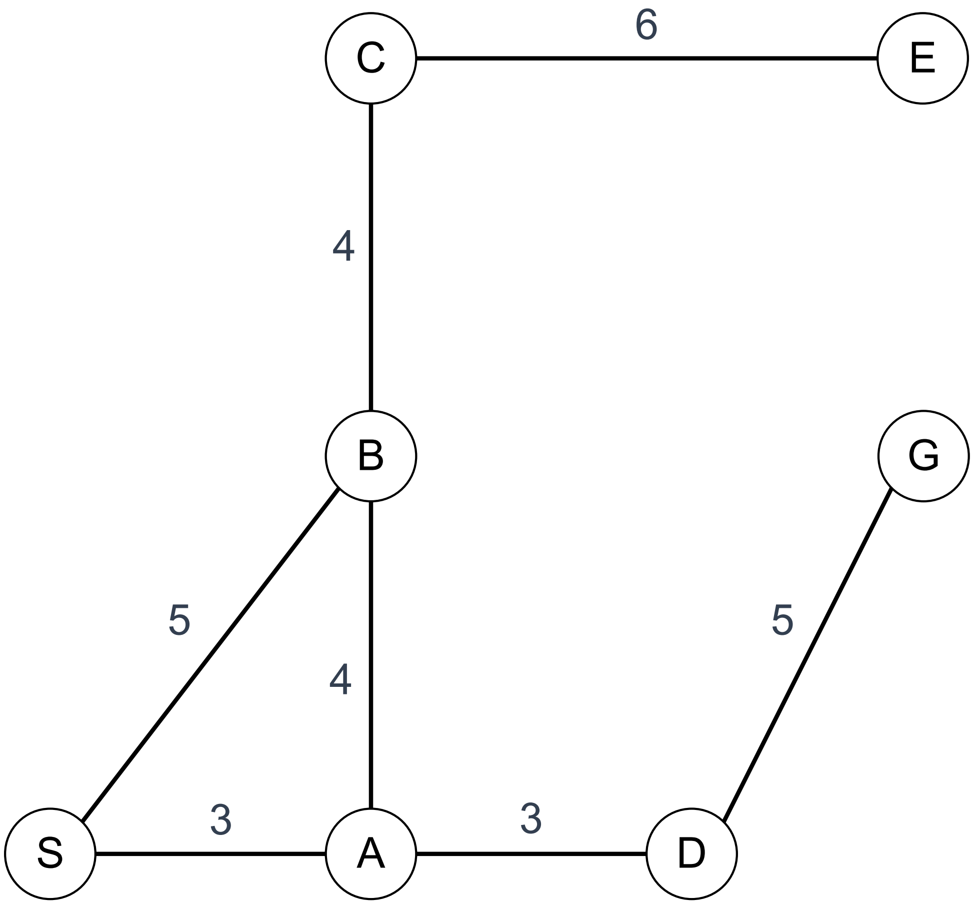
Information like SLD helps us ignore the left

# Branch and Bound

- Find the optimal path instead of just locating goal

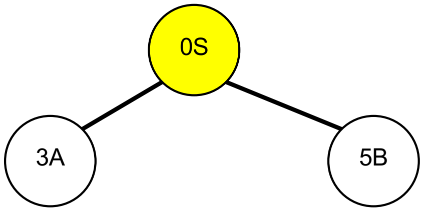
- Extend the shortest *accumulated* path
  - Placed next to name of node



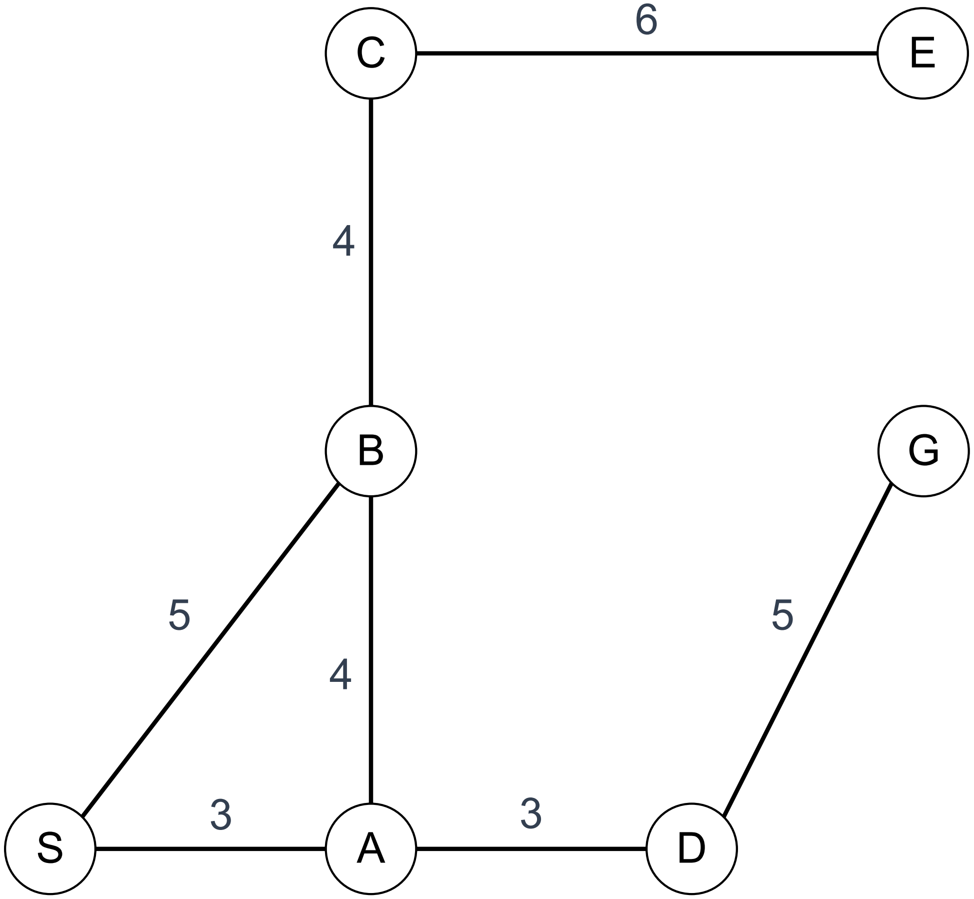


# Branch and Bound

- Find the optimal path instead of just locating goal

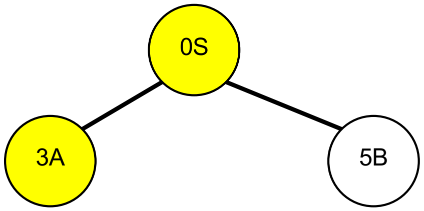


- Extend the shortest ***accumulated*** path

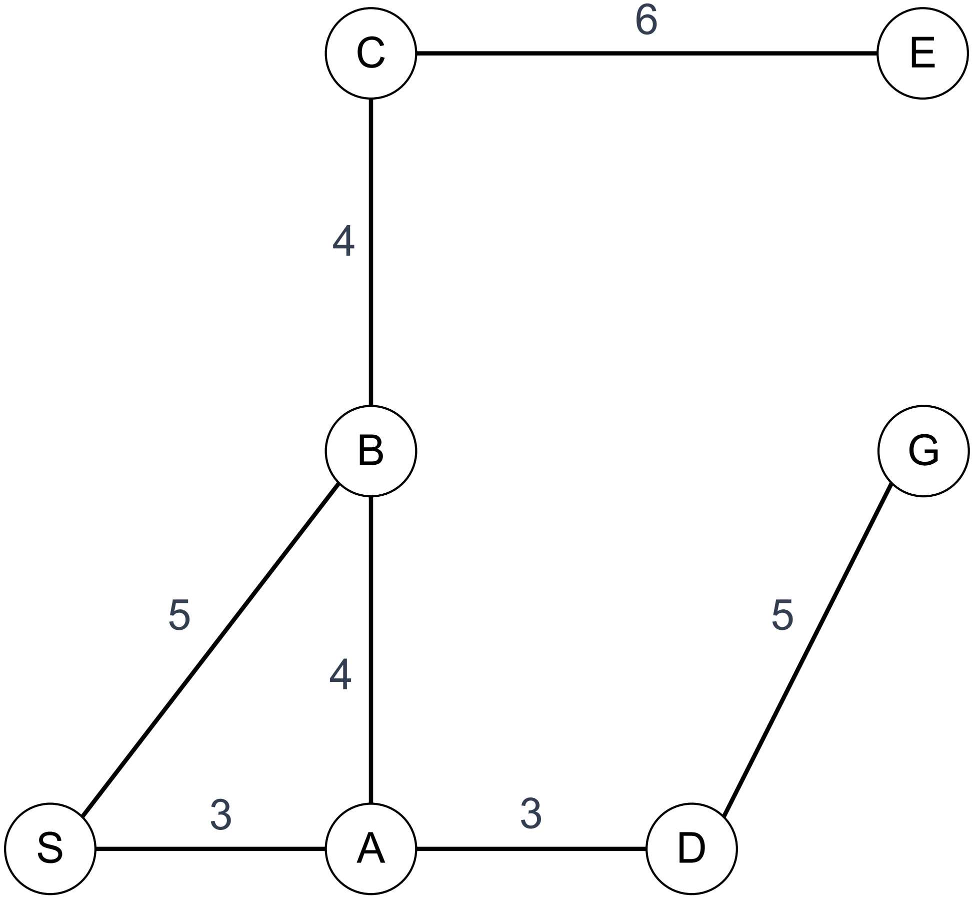


# Branch and Bound

- Find the optimal path instead of just locating goal



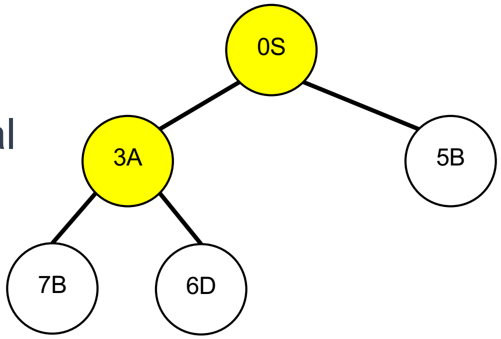
- Extend the shortest *accumulated* path
  - 3A shorter than 5B

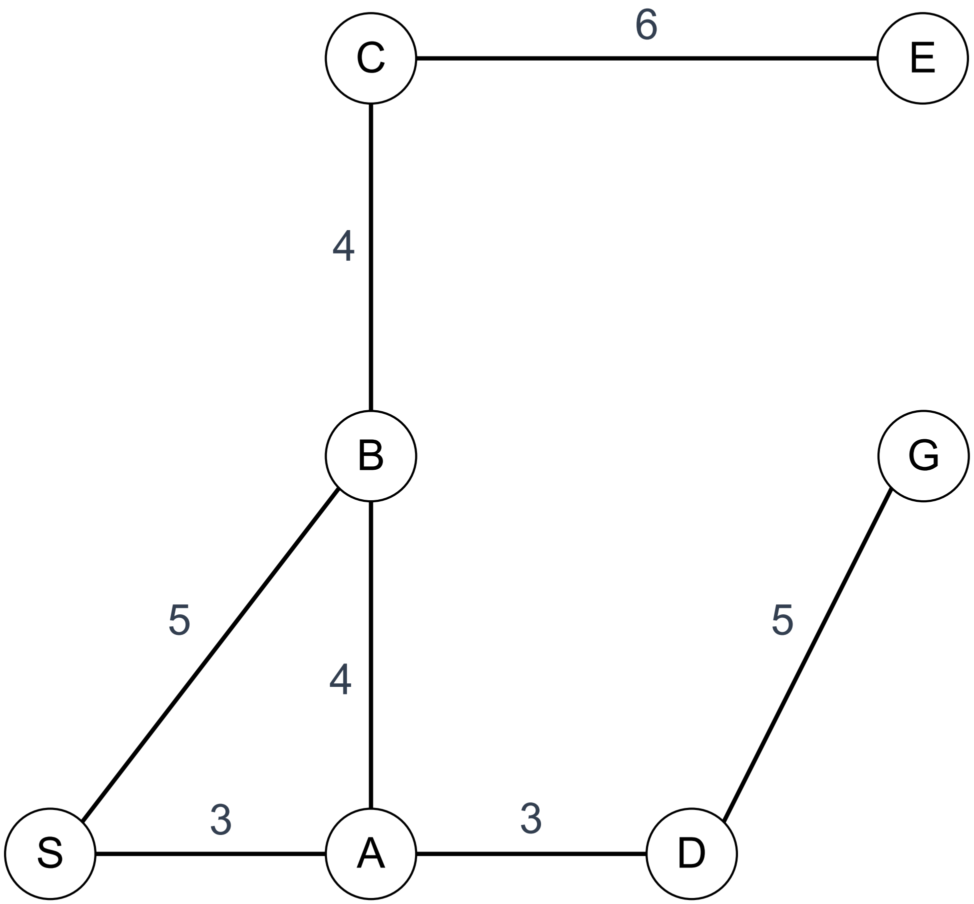


# Branch and Bound

- Find the optimal path instead of just locating goal

- Extend the shortest *accumulated* path
  - 5B shorter than 6D and 7B

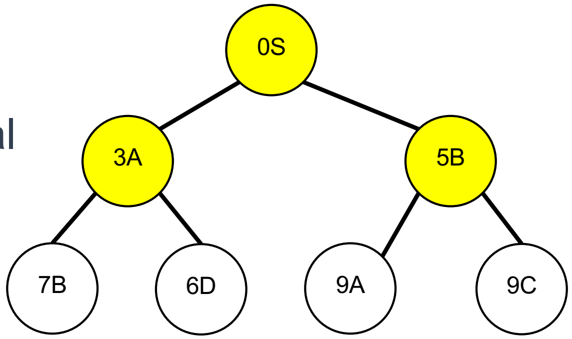


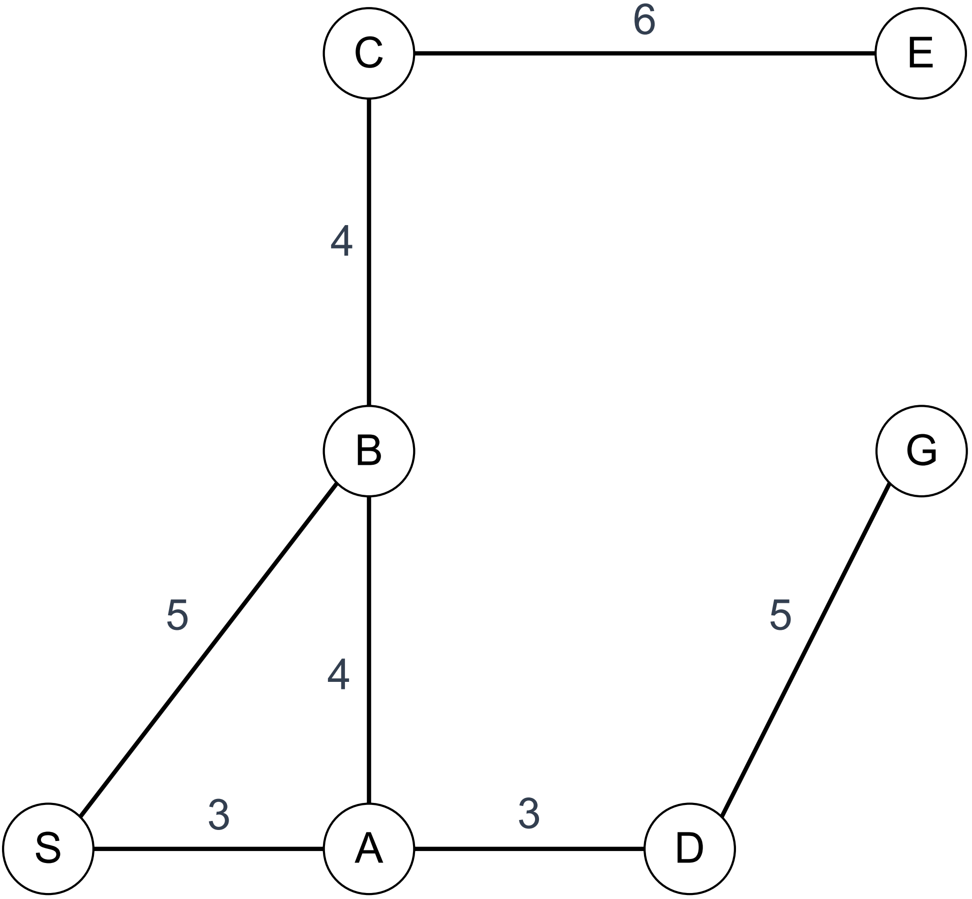


# Branch and Bound

- Find the optimal path instead of just locating goal

- Extend the shortest *accumulated* path
  - 6D shorter than 7B and 9A and 9C

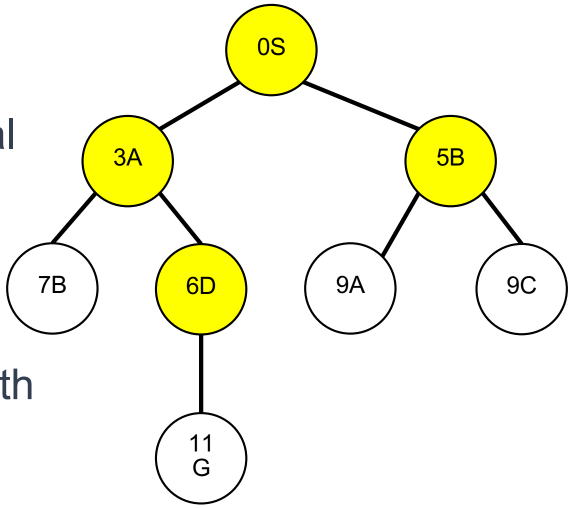


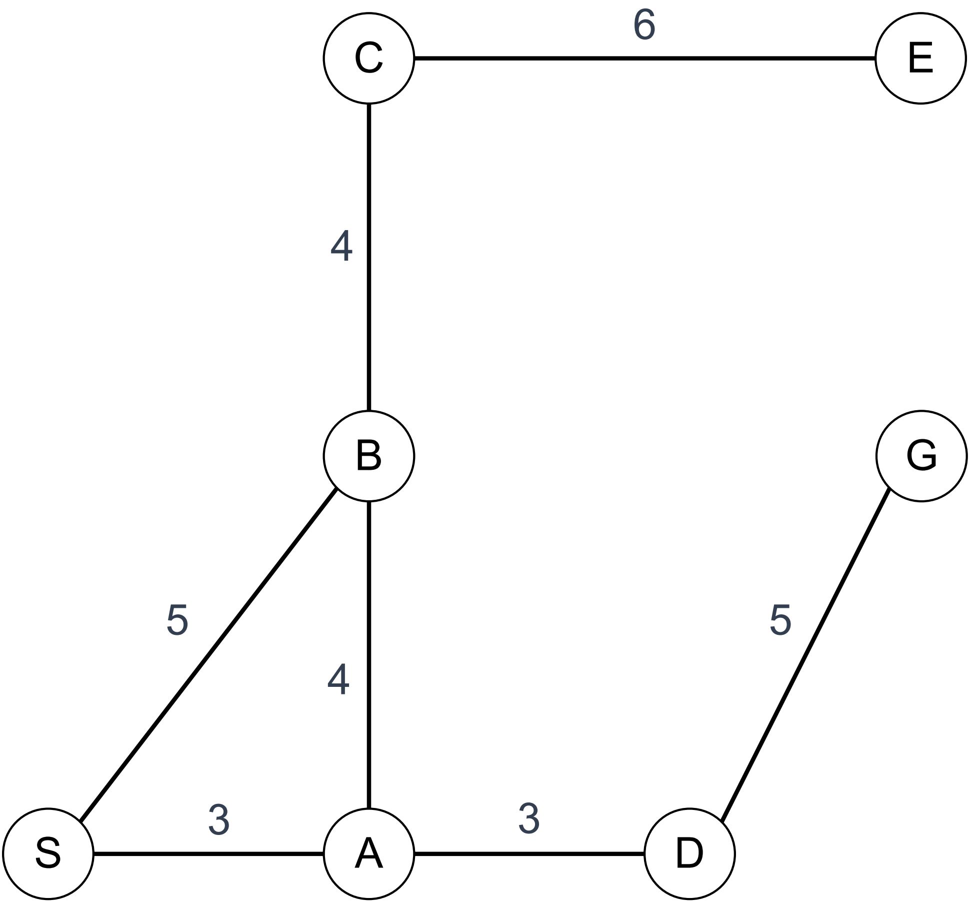


# Branch and Bound

- Find the optimal path instead of just locating goal

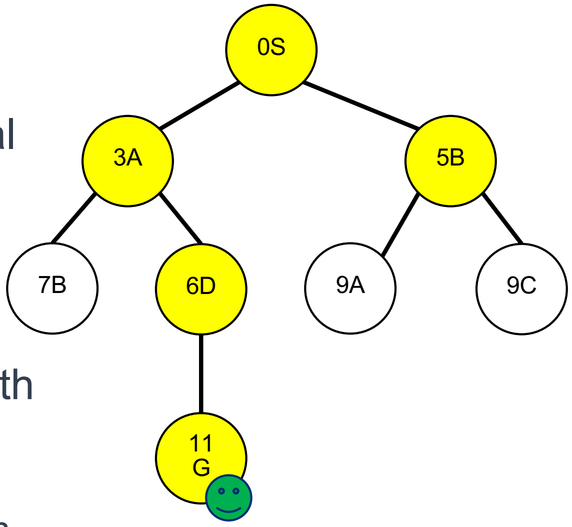
- Extend the shortest *accumulated* path
  - Found goal
  - Shouldn't stop since the number to beat is 11 and we have a 7 and two 9s still unexplored

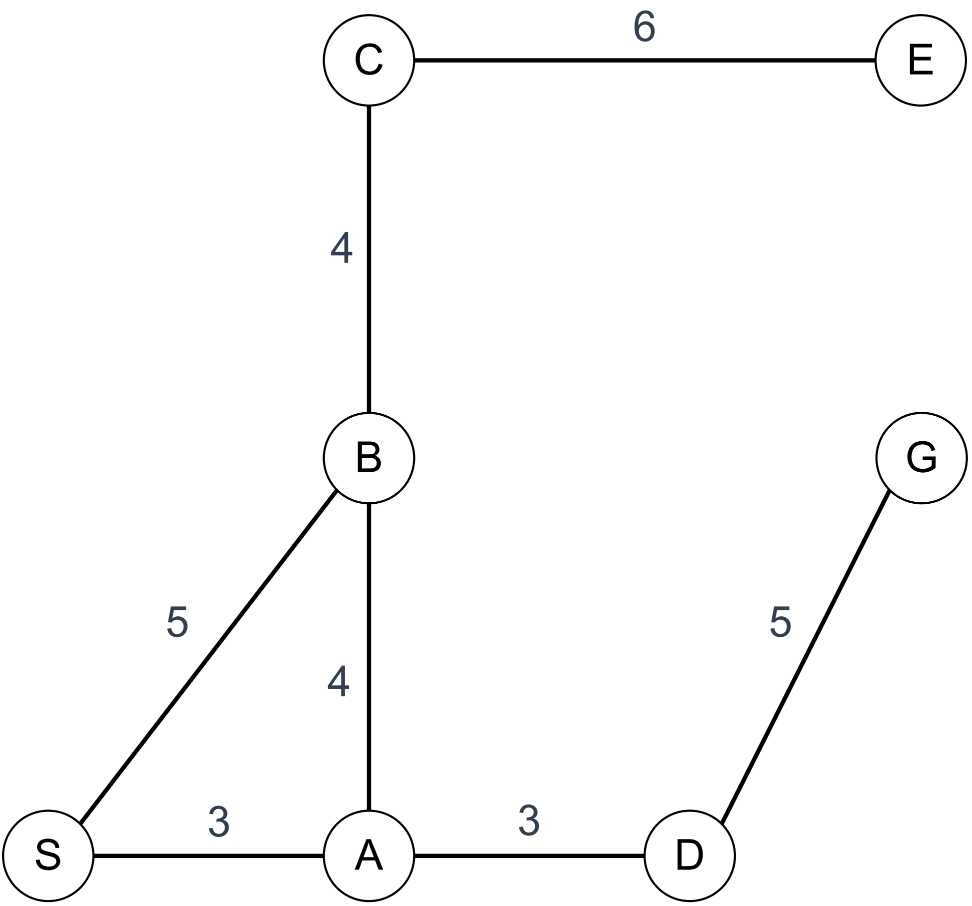




# Branch and Bound

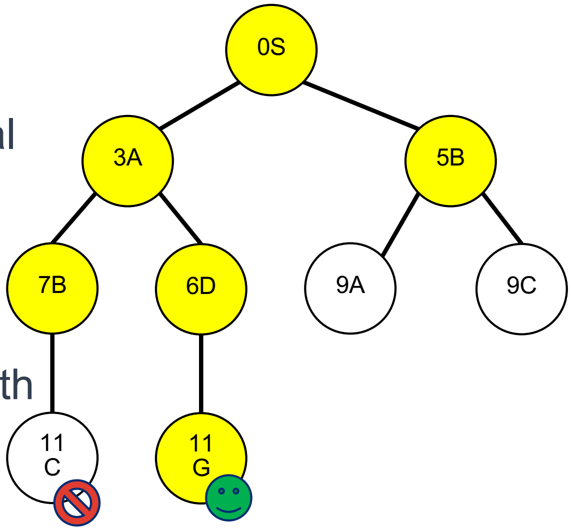
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - The number to beat is 11
  - 7B shorter than 9A and 9C



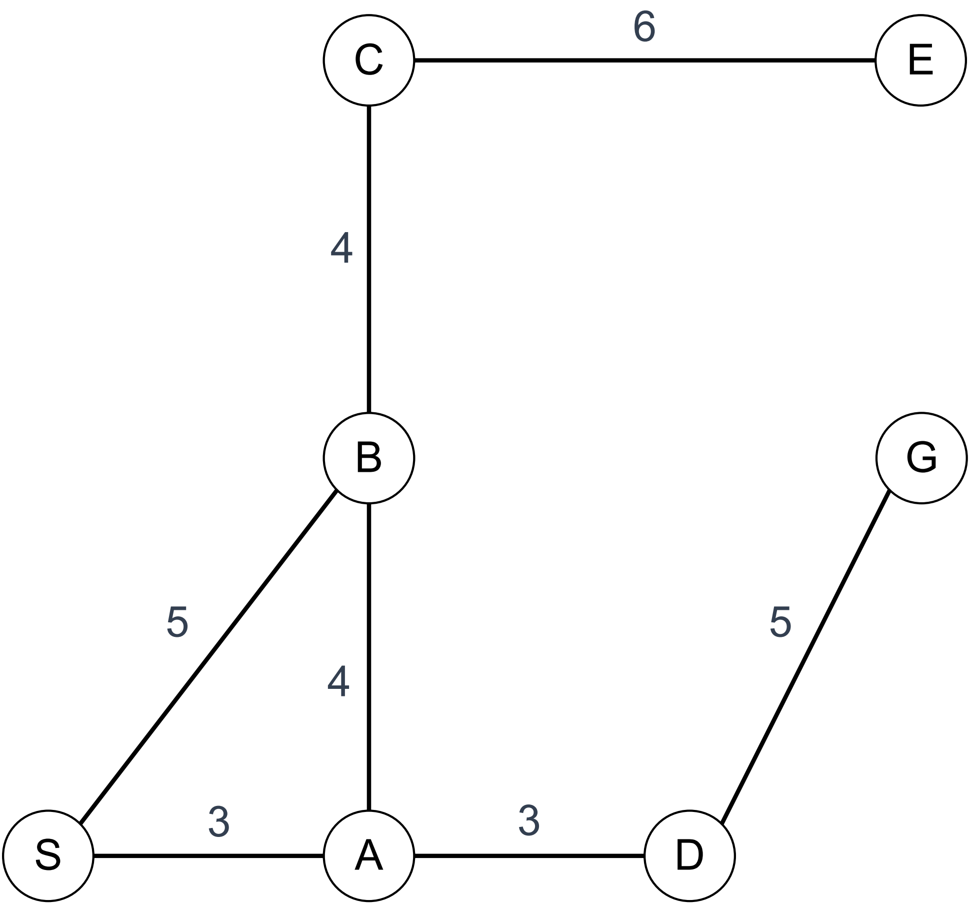


# Branch and Bound

- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - The number to beat is 11
  - 11C hopeless
  - 9A chosen over 9C (lexical)



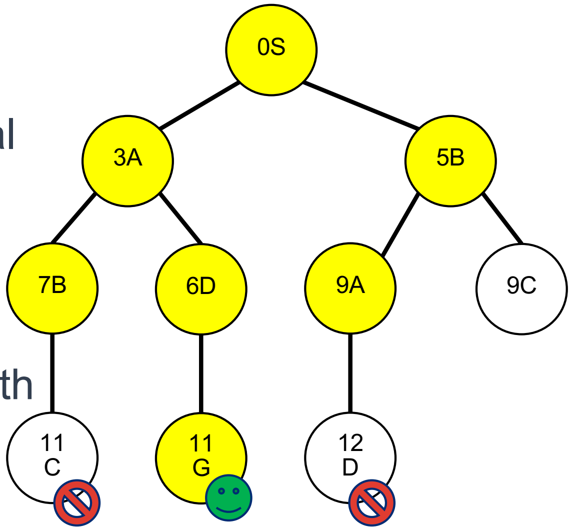
🚫 Stopped because of cost

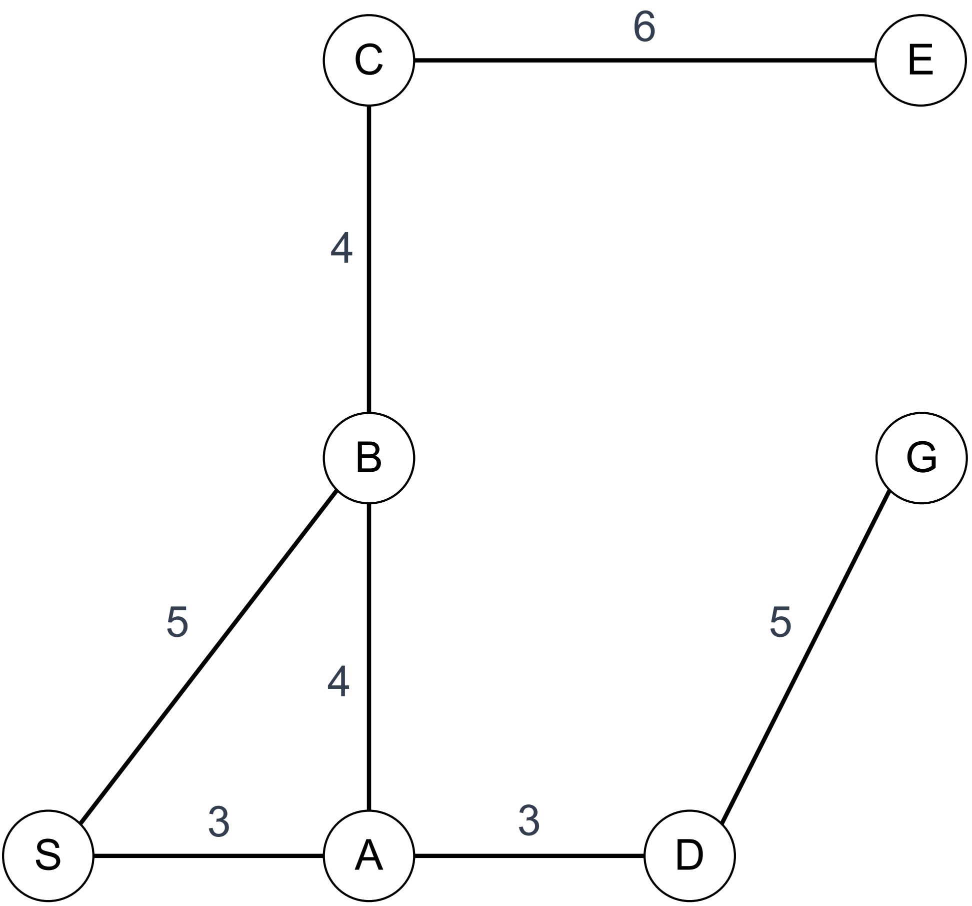


# Branch and Bound

- Find the optimal path instead of just locating goal

- Extend the shortest *accumulated* path
    - The number to beat is 11
    - 12D hopeless

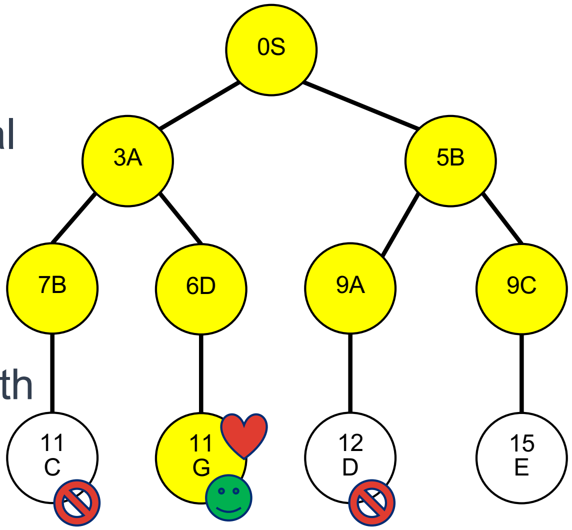


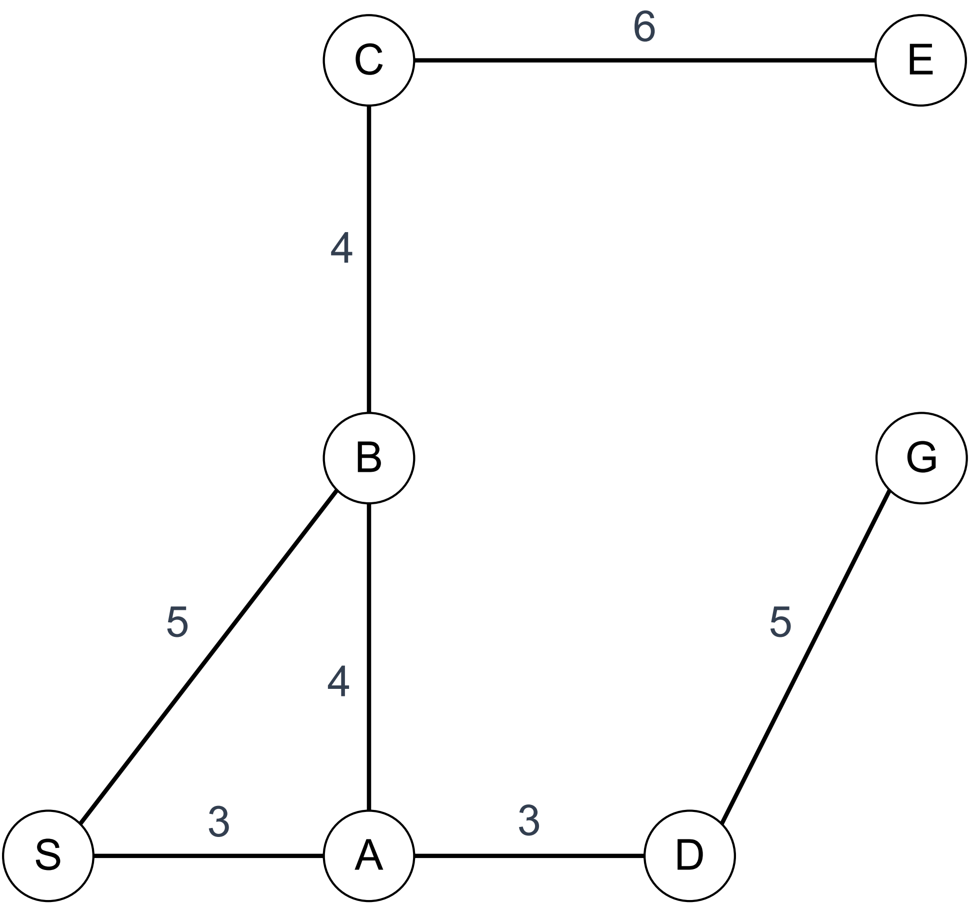


# Branch and Bound

- Find the optimal path instead of just locating goal

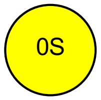
- Extend the shortest *accumulated* path
    - The number to beat is 11
    - 15E is a dead end (and hopeless)

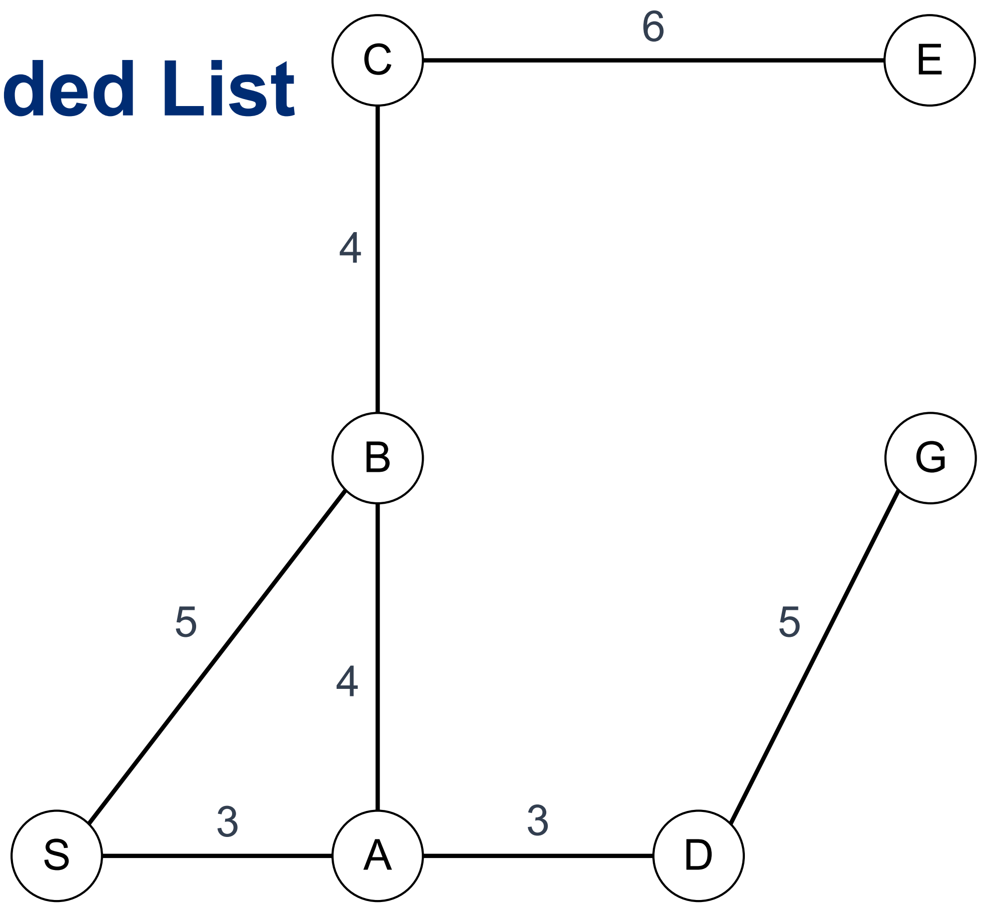




# Branch and Bound + Extended List

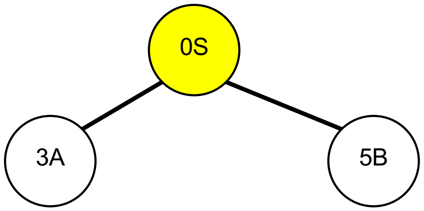
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
    - Placed next to name of node
- Don’t waster resources extending a node that has already been extended



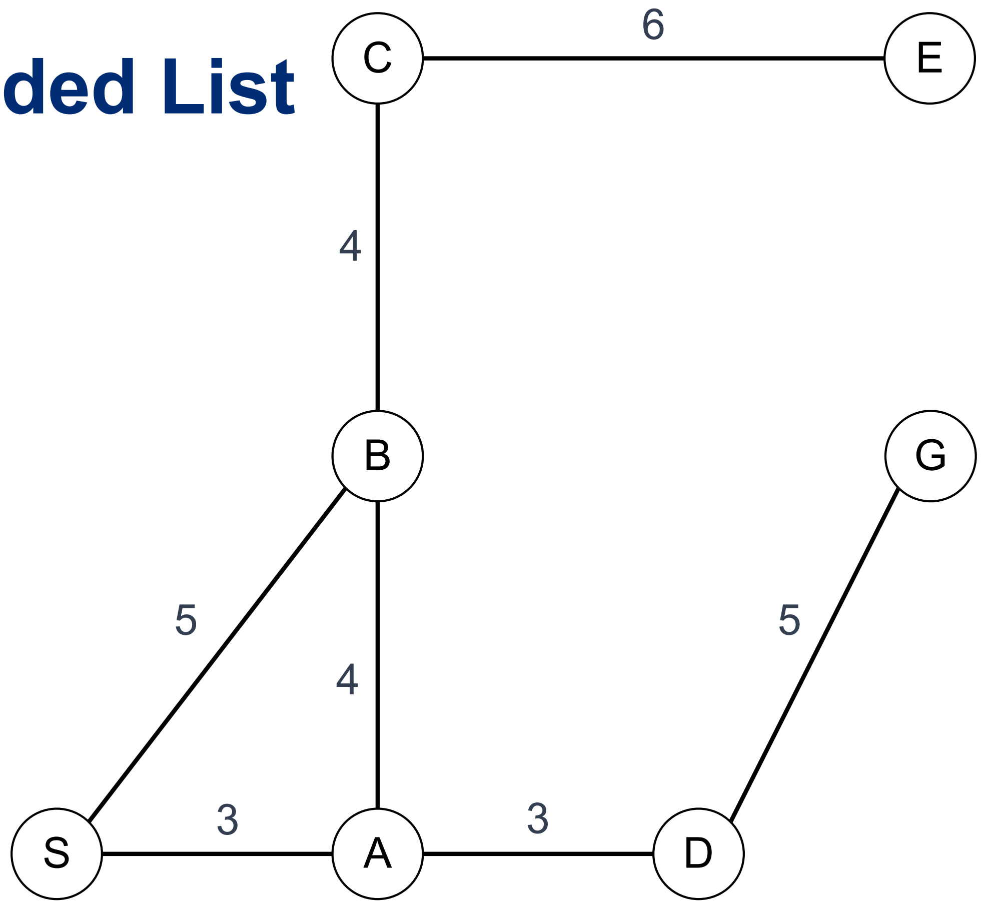


# Branch and Bound + Extended List

- Find the optimal path instead of just locating goal

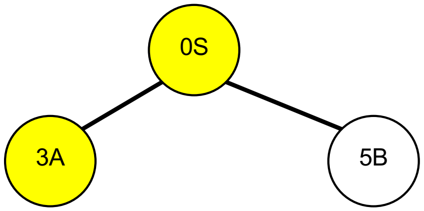


- Extend the shortest *accumulated* path
  - Placed next to name of node
- Don’t waster resources extending a node that has already been extended



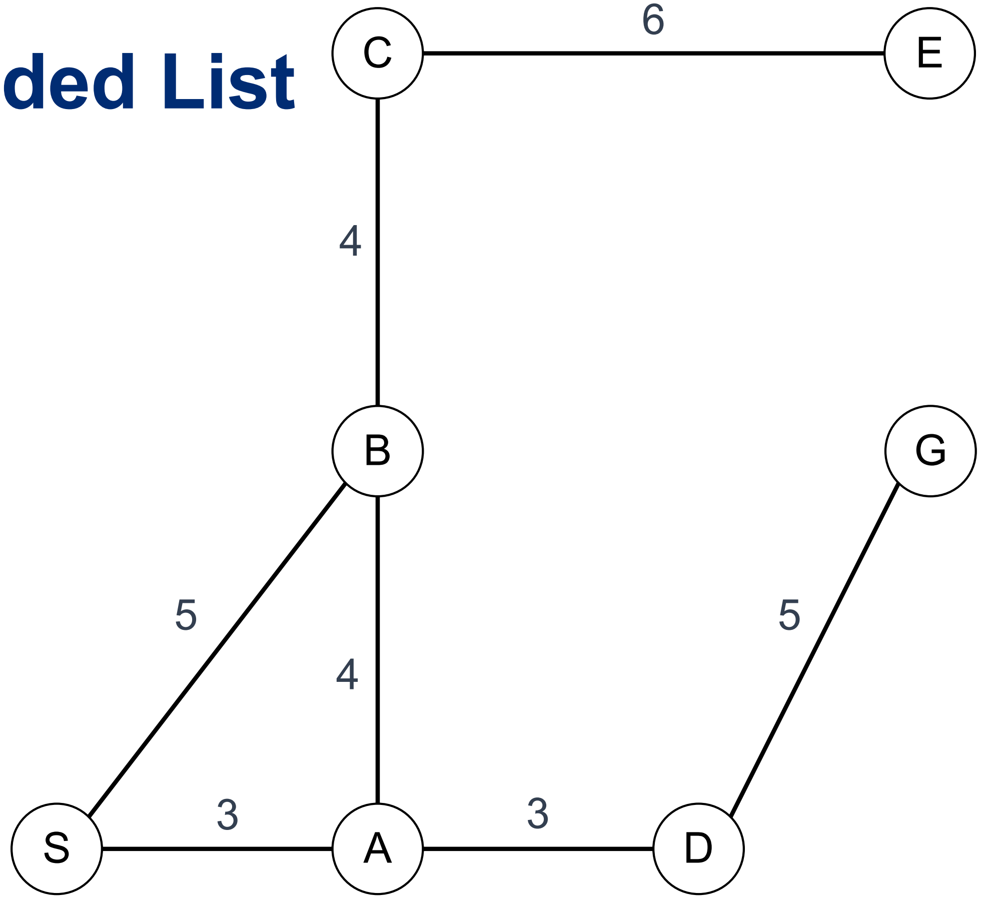
# Branch and Bound + Extended List

- Find the optimal path instead of just locating goal



- Extend the shortest *accumulated* path
    - Placed next to name of node

- Don’t waster resources extending a node that has already been extended

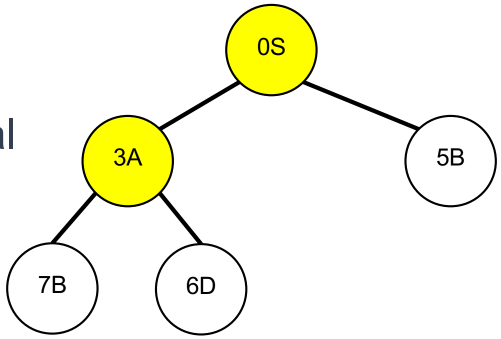


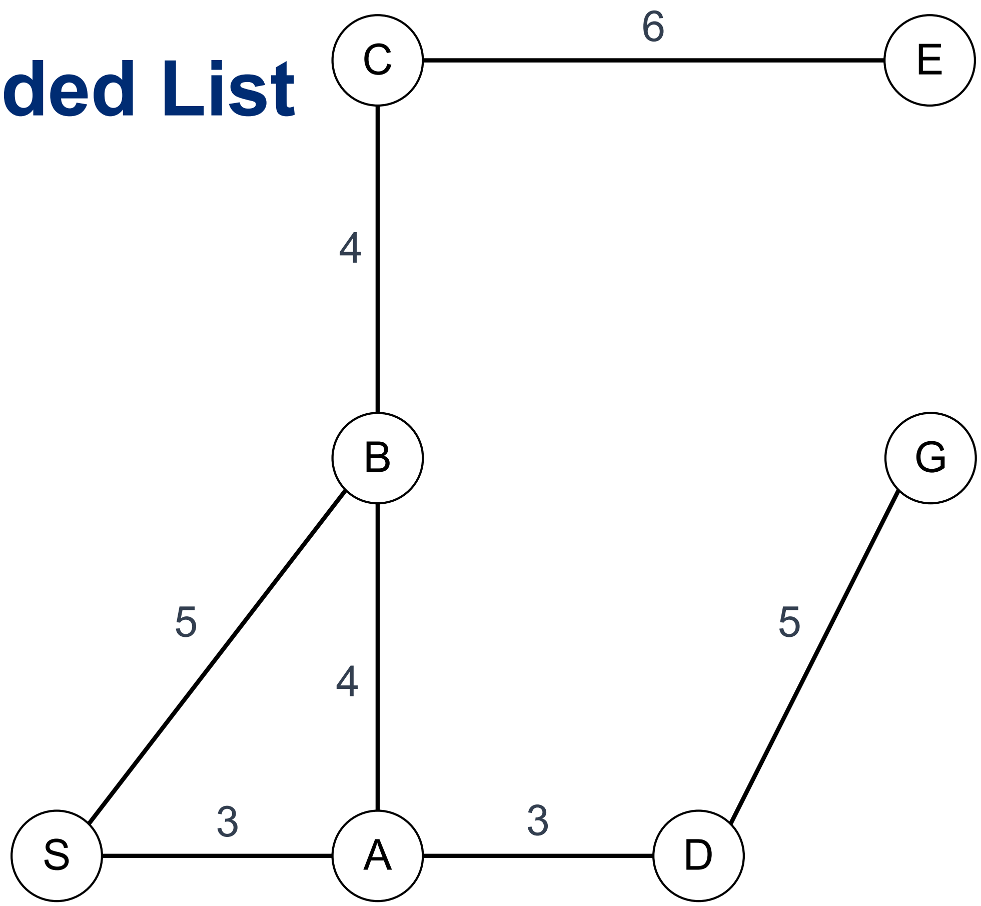
# Branch and Bound + Extended List

- Find the optimal path instead of just locating goal

- Extend the shortest *accumulated* path
  - Placed next to name of node

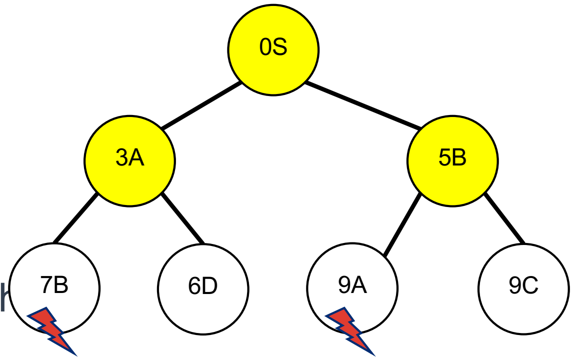
- Don’t waster resources extending a node that has already been extended

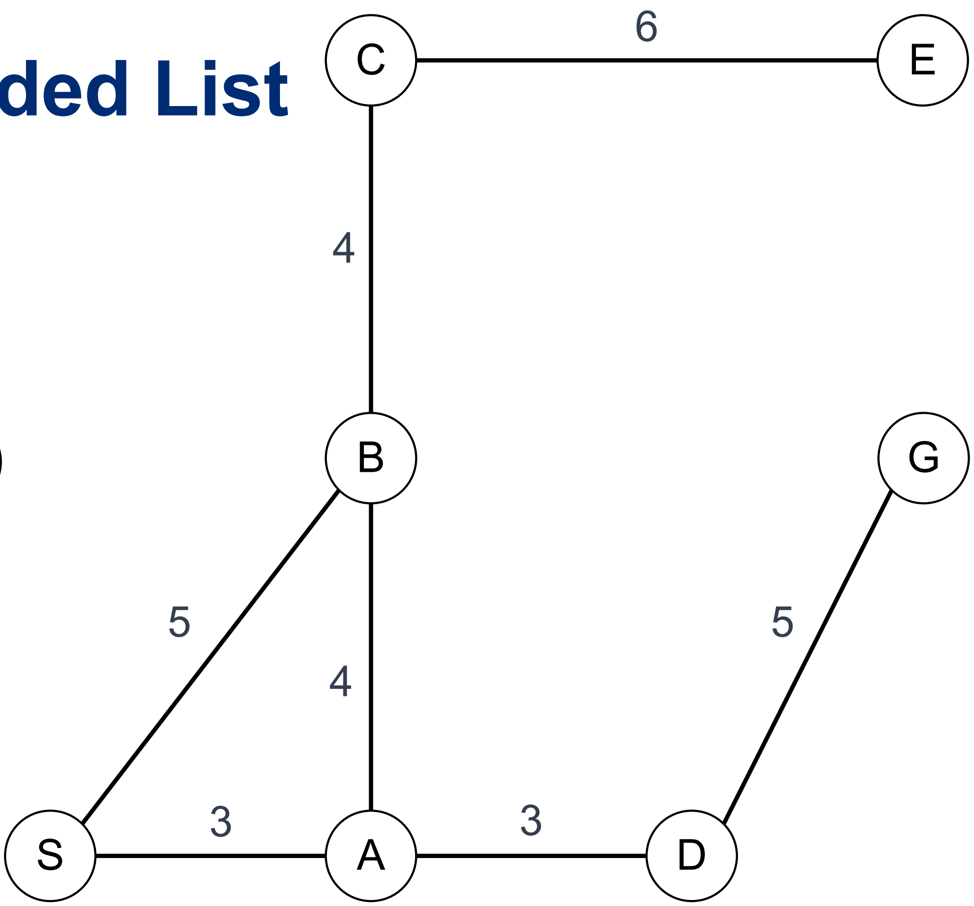




# Branch and Bound + Extended List

- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- Don’t waster resources extending a node that has already been extended
  - 7B never extended because of 5B
  - 9A never extended because of 3A

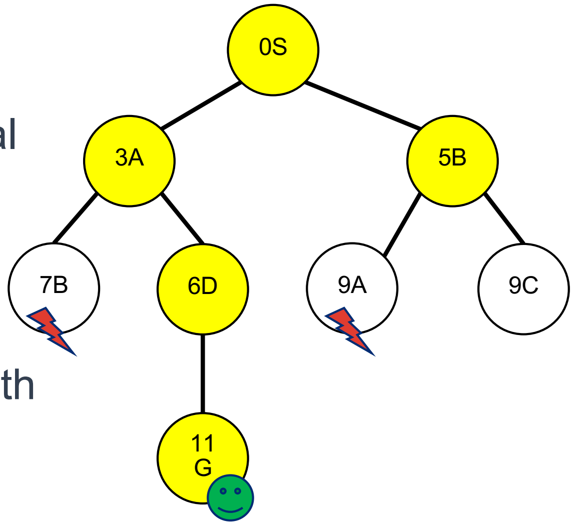


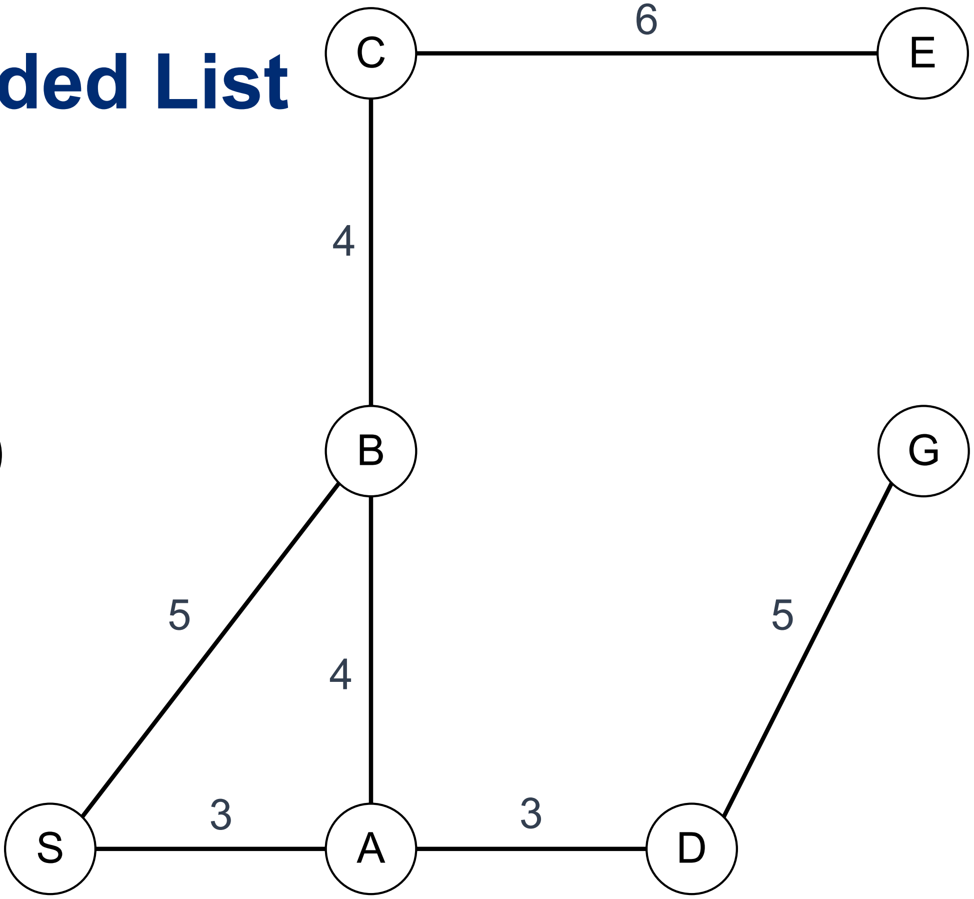


⚡ Stopped because we’ve been there before

# Branch and Bound + Extended List

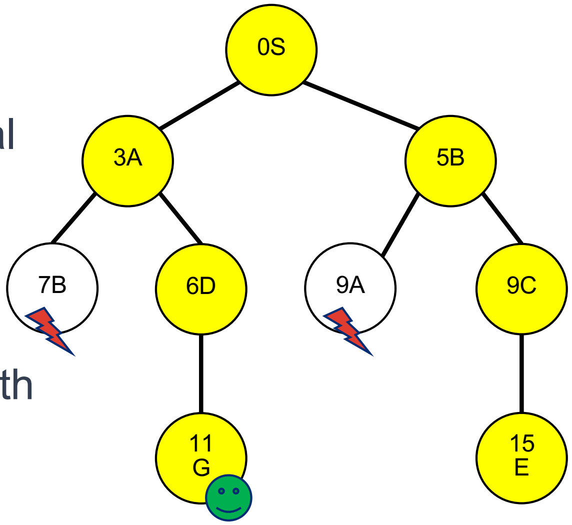
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- Don’t waster resources extending a node that has already been extended

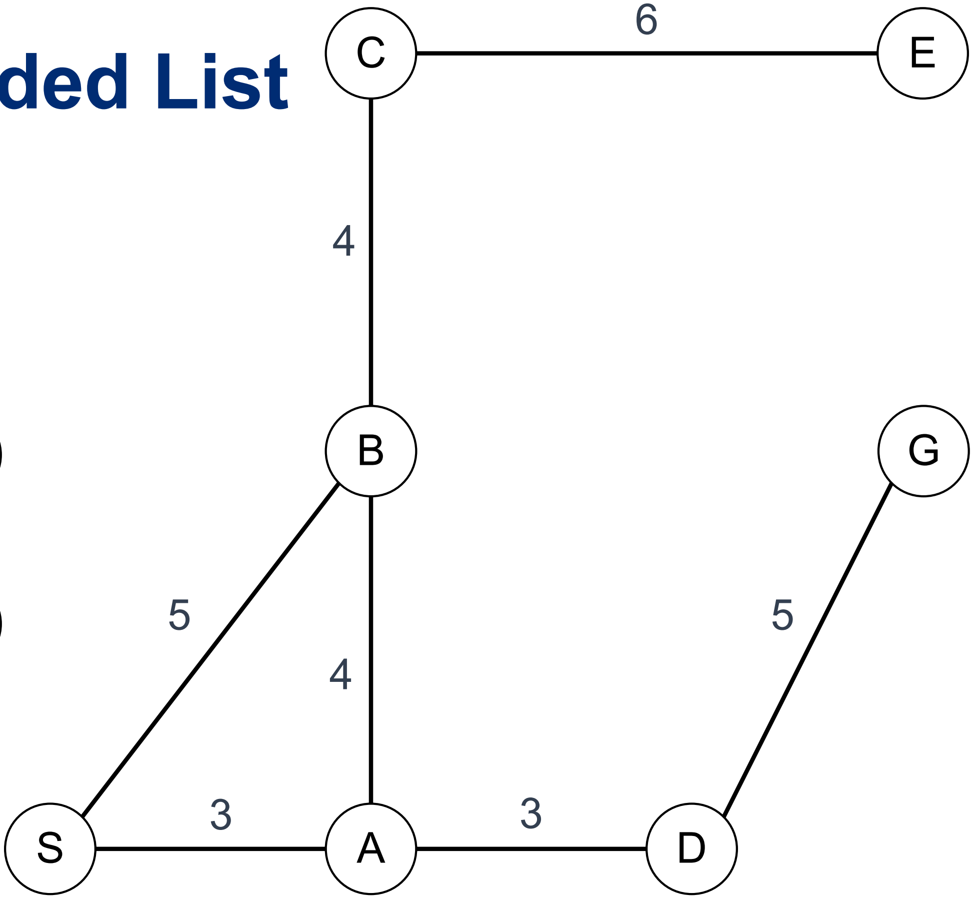




# Branch and Bound + Extended List

- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- Don’t waster resources extending a node that has already been extended





# Heuristic

- Some additional piece of information (a rule, function, or constraint) that informs an otherwise brute-force algorithm to act in a more optimal manner
  - Generally a good idea to make use of this help
- The evaluation function at a node adds the accumulated path cost to get to that node and the heuristic information available at that node

Evaluation function

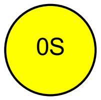
Heuristic, SLD for instance

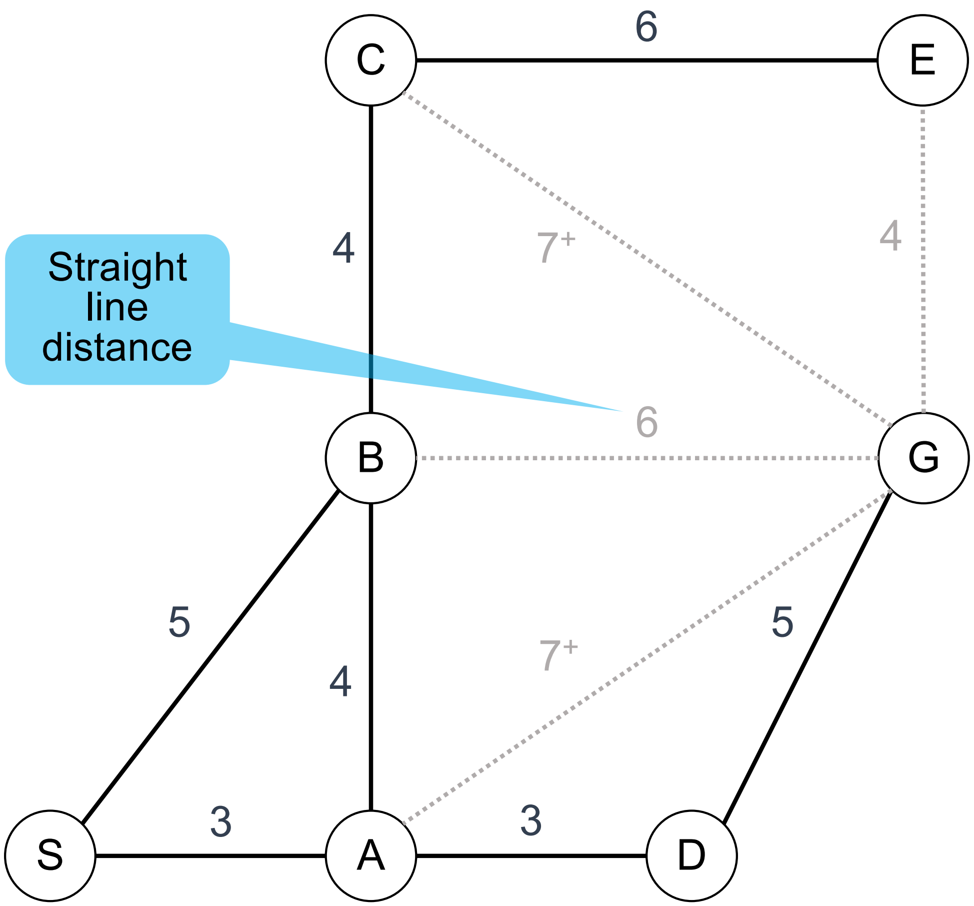
$$f(n) = g(n) + h(n)$$

**Accumulated** path cost

# A*

- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- ***Use*** the evaluation function and assume h(n) is given in the form of SLD to goal

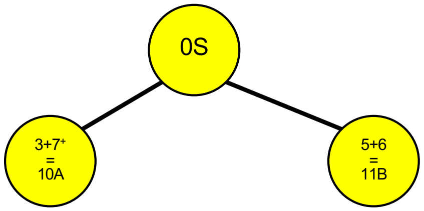


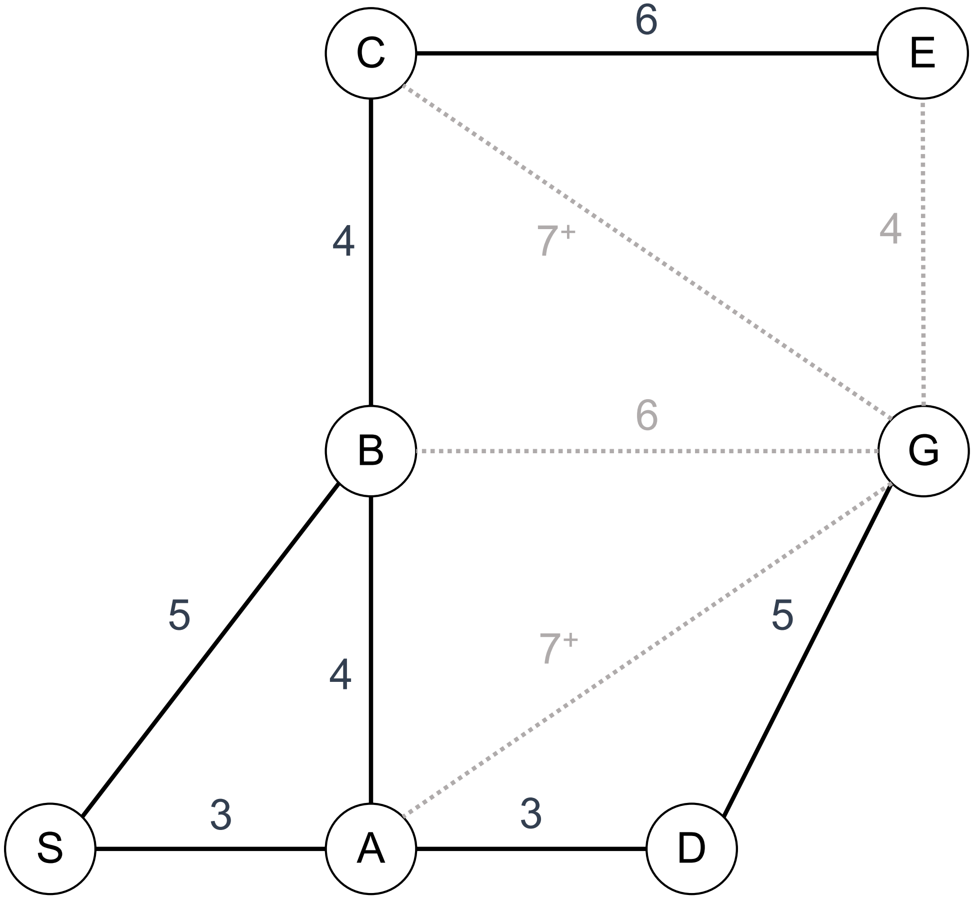


Straight line distance

# A*

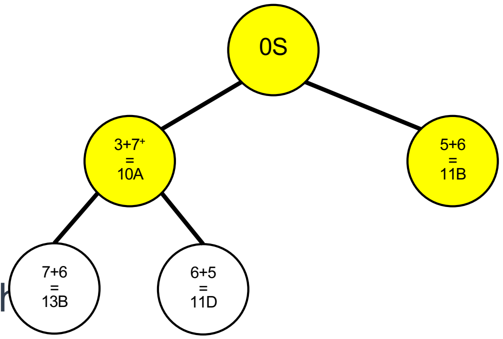
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- ***Use*** the evaluation function and assume h(n) is given in the form of SLD to goal

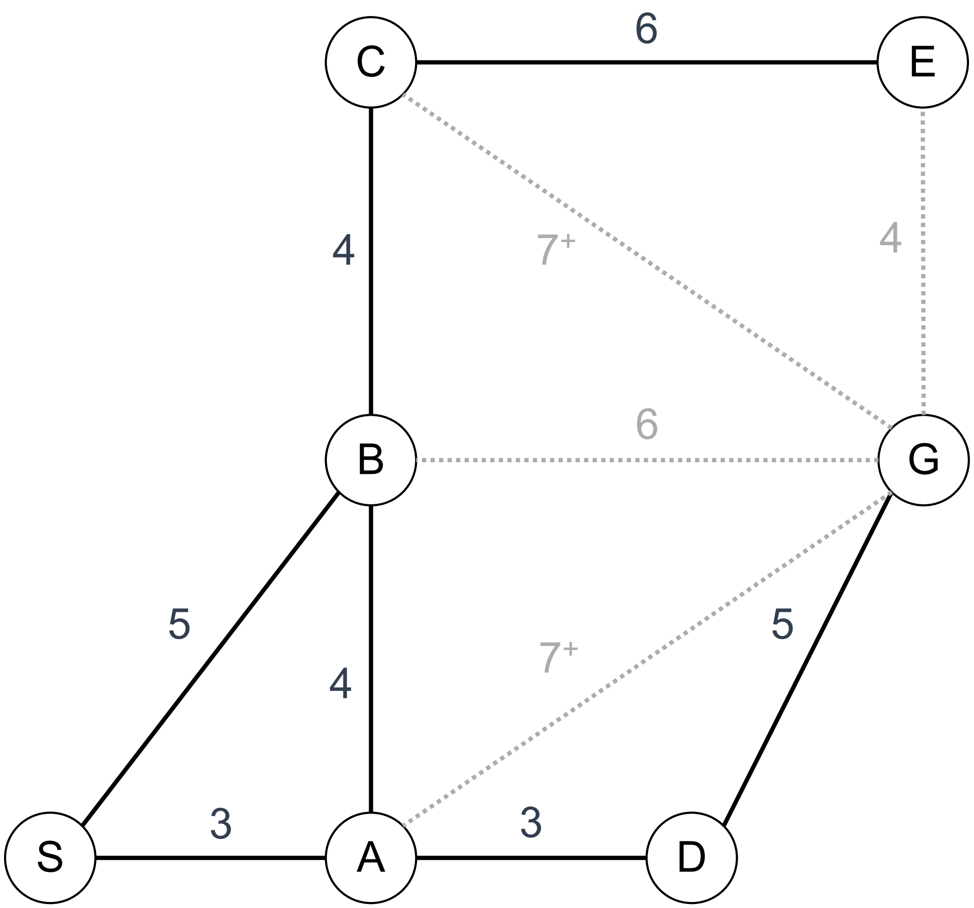




# A*

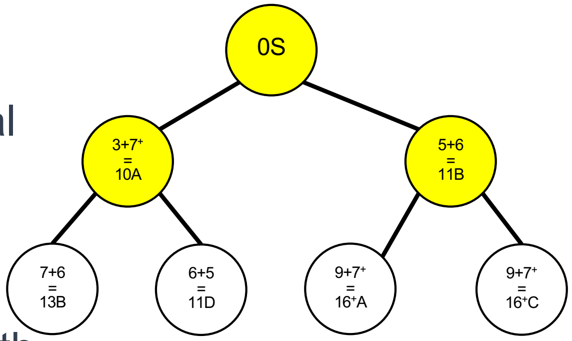
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- ***Use*** the evaluation function and assume h(n) is given in the form of SLD to goal
- Expand 11B (lexical)

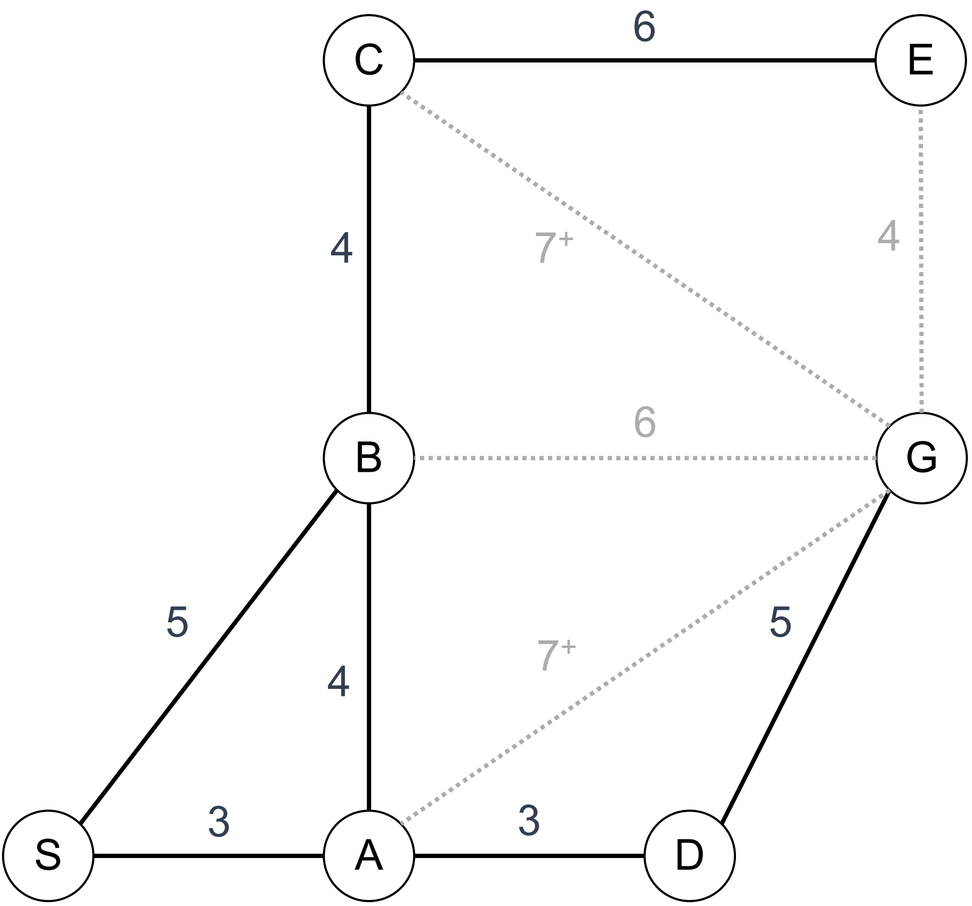




# A*

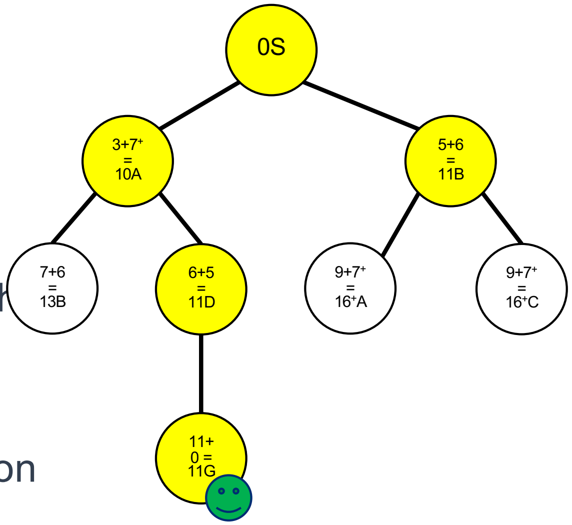
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- ***<u>Use</u>*** the evaluation function and assume h(n) is given in the form of SLD to goal

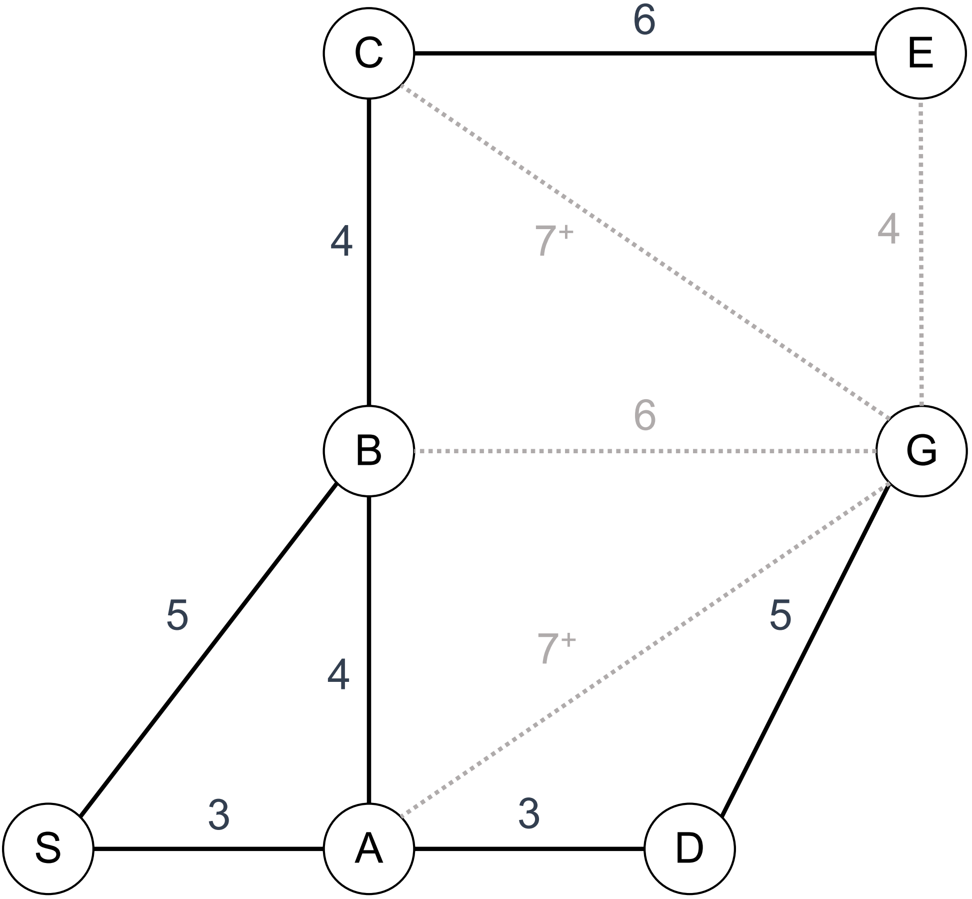




# A*

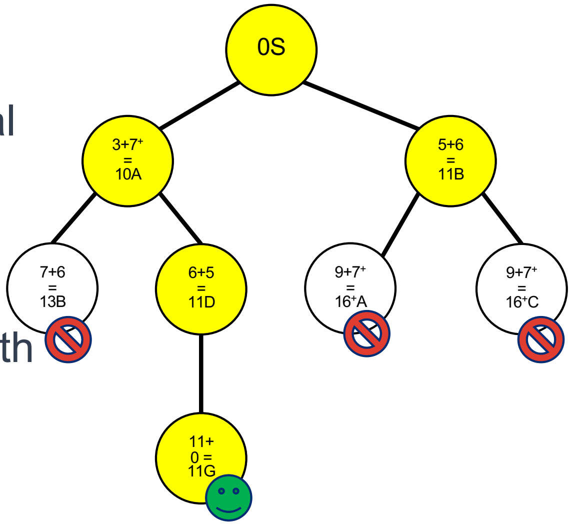
- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
    - Placed next to name of node
- ***Use*** the evaluation function and assume h(n) is given in the form of SLD to goal
- The number to beat is 11

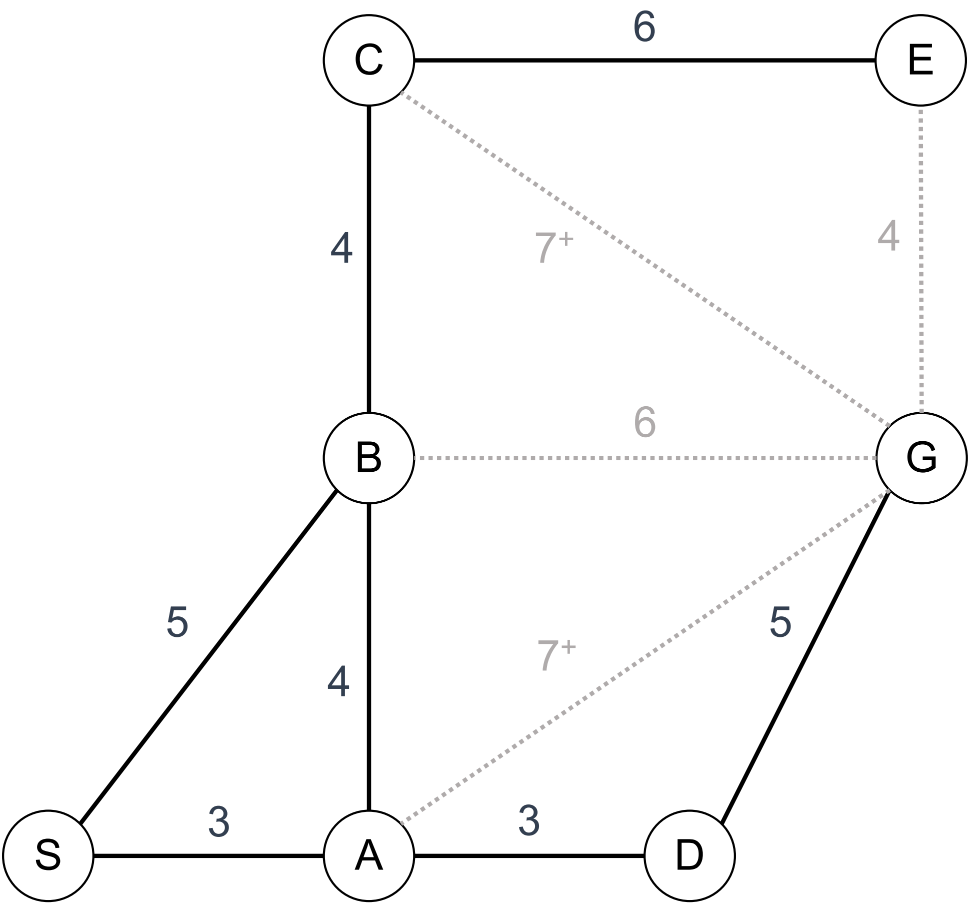




# A*

- Find the optimal path instead of just locating goal
- Extend the shortest *accumulated* path
  - Placed next to name of node
- ***Use*** the evaluation function and assume h(n) is given in the form of SLD to goal





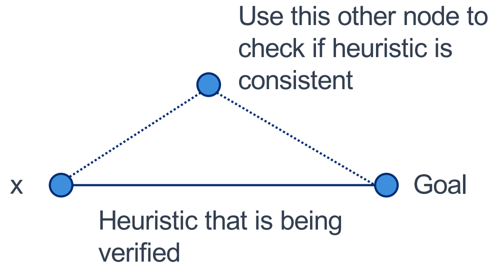
# Heuristics

- An **admissible** heuristic is an estimate that does not overestimate the true measure
  - The estimate from node *$n_i$* to goal can't be more than the actual "distance"
  - SLD doesn't overestimate; hence it is an ***admissible*** heuristic

$$\mathcal{H}(n_i, G) \leq \mathcal{D}(n_i, G)$$

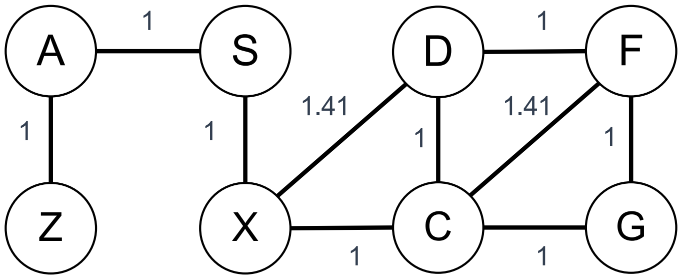
- ***Consistency*** is a stronger condition than admissibility
  - Use the goal as a point of reference to triangulate
  - *Triangular Inequality Theory* states that that the size of each side of a triangle is less than the sum of the other two sides

$$|\mathcal{H}(n_x, G) - \mathcal{H}(n_y, G)| \leq \mathcal{D}(x, y)$$


Use this other node to check if heuristic is consistent
x
Goal
Heuristic that is being verified

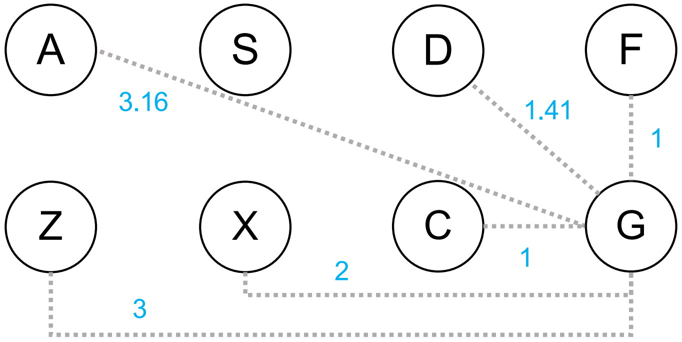
# Problem #2

- Start at (S) and find a *path* to goal (G)

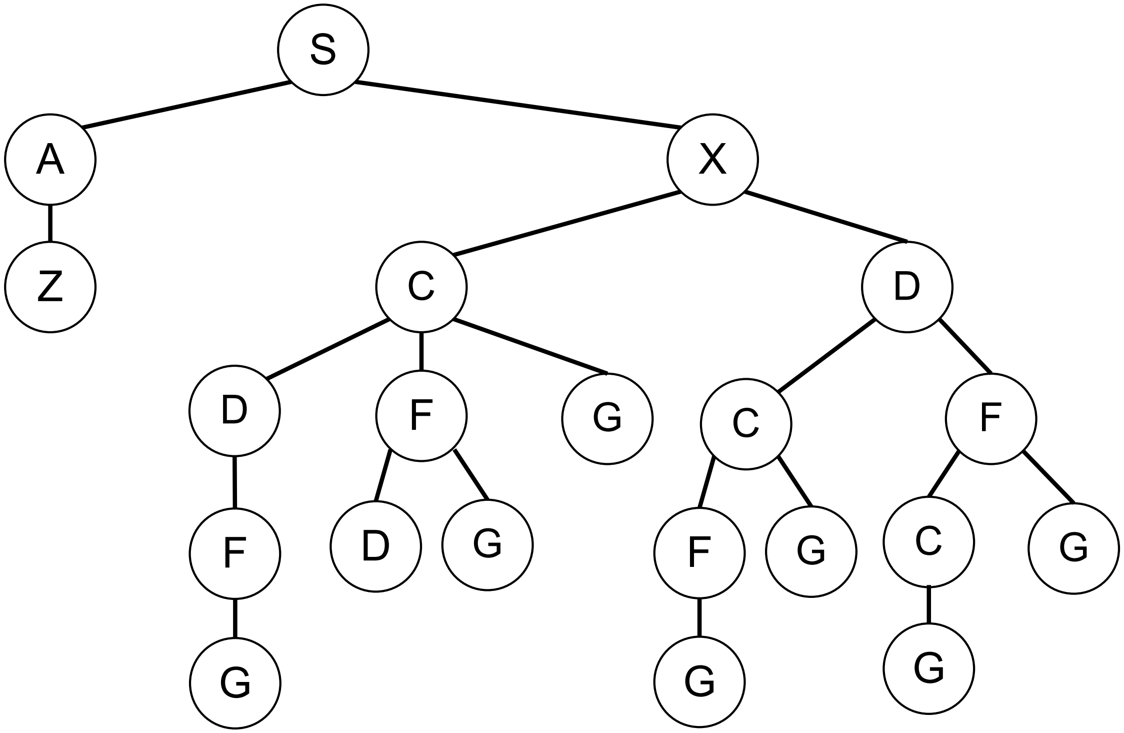


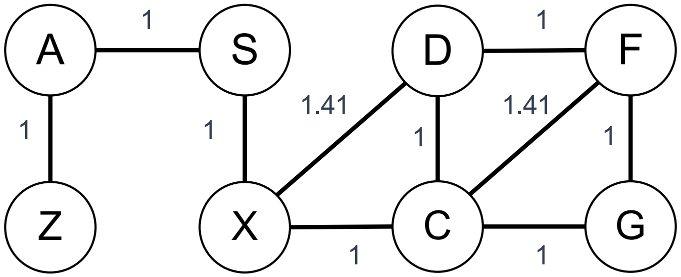
Path costs

SLD Heuristic



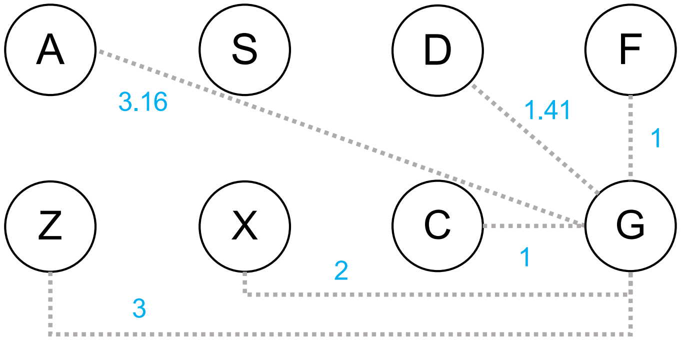
- British Museum



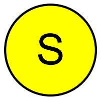


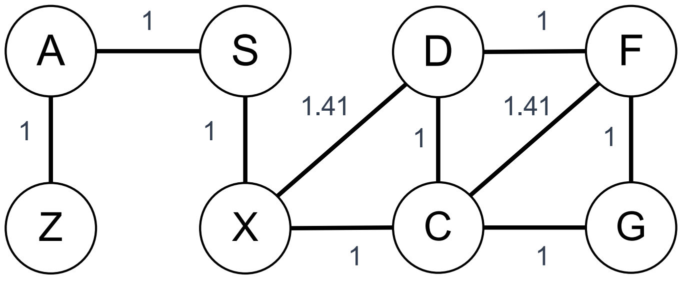
Path costs

SLD Heuristic



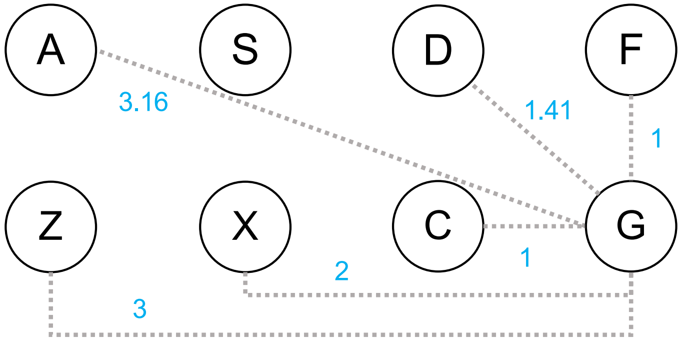
# Depth-First Search



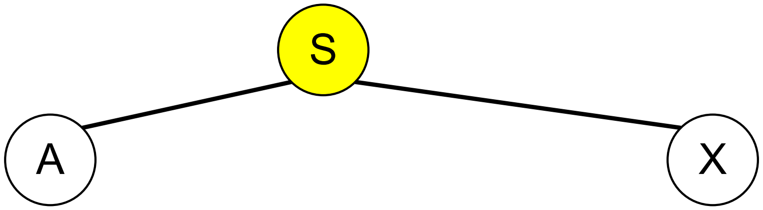


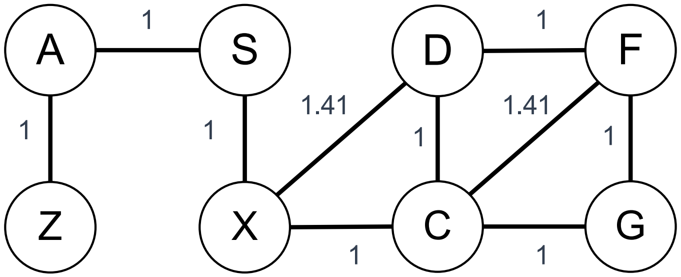
Path costs

SLD Heuristic



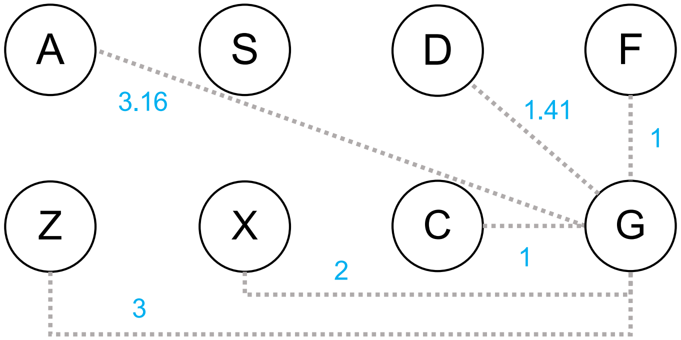
# Depth-First Search



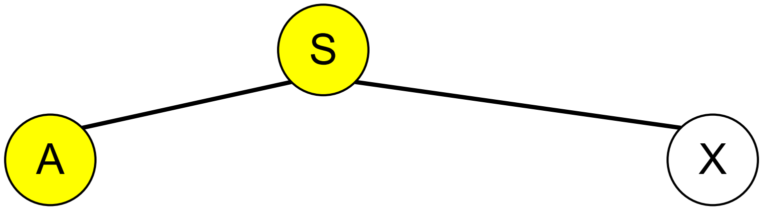


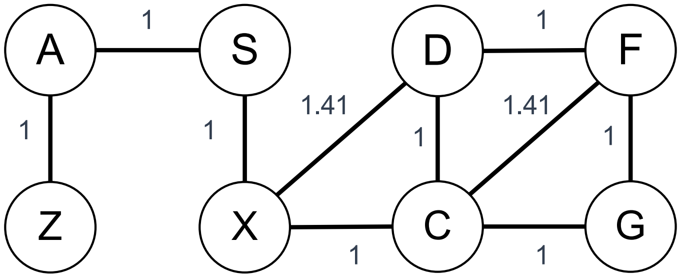
Path costs

SLD Heuristic



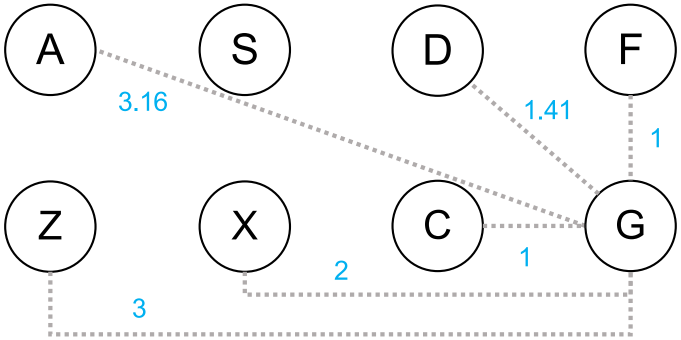
# Depth-First Search



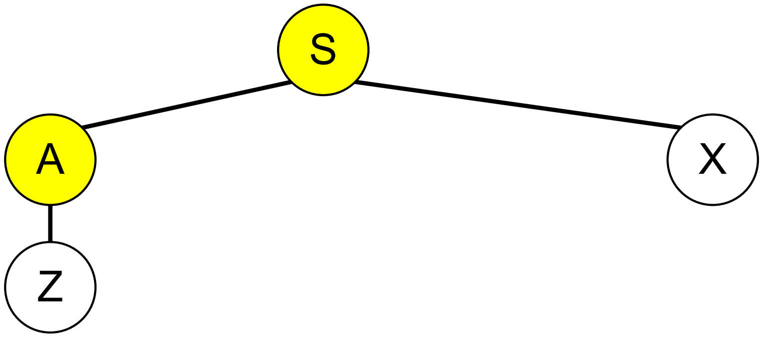


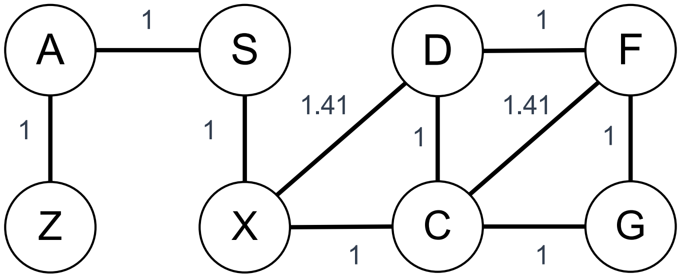
Path costs

SLD Heuristic



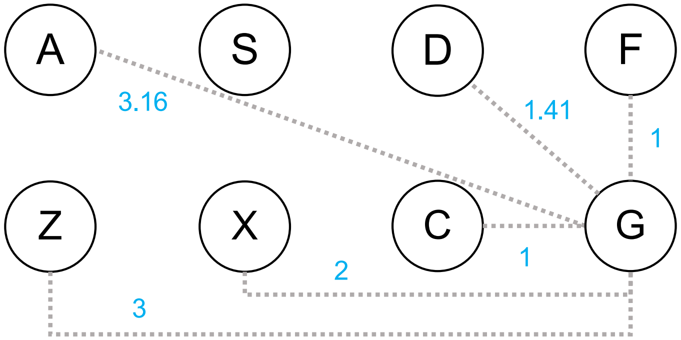
# Depth-First Search



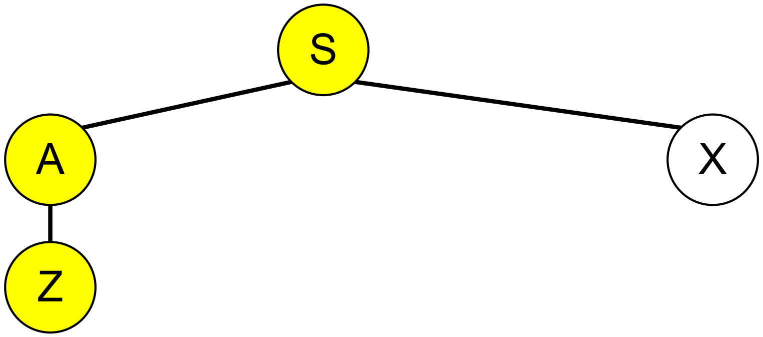


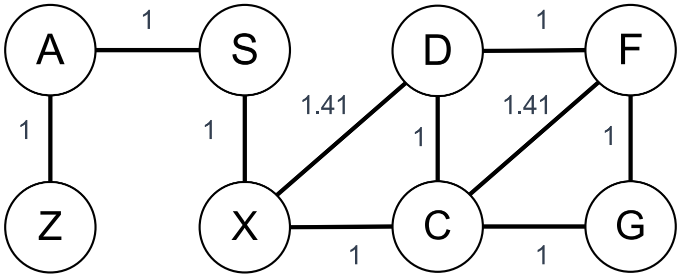
Path costs

SLD Heuristic



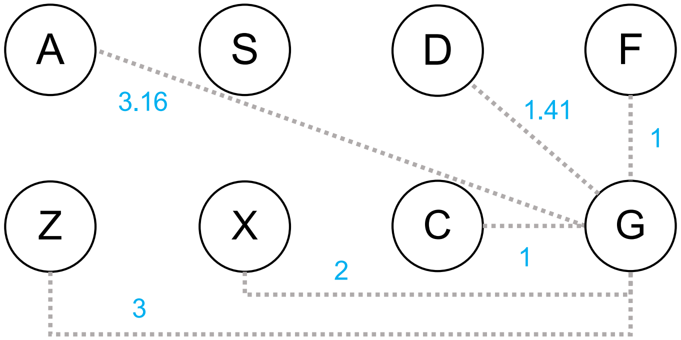
# Depth-First Search



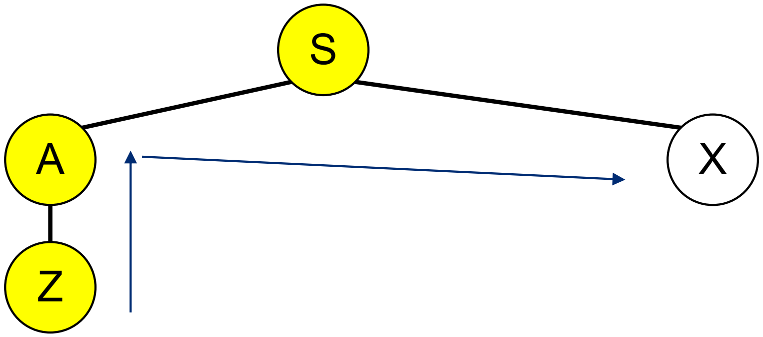


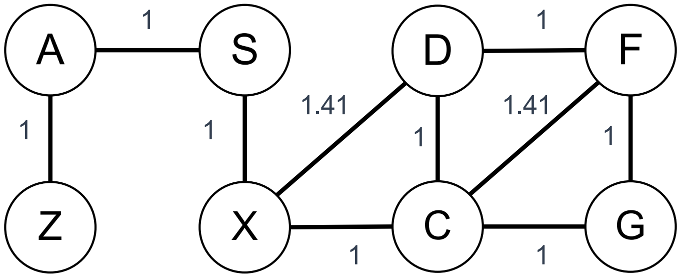
Path costs

SLD Heuristic



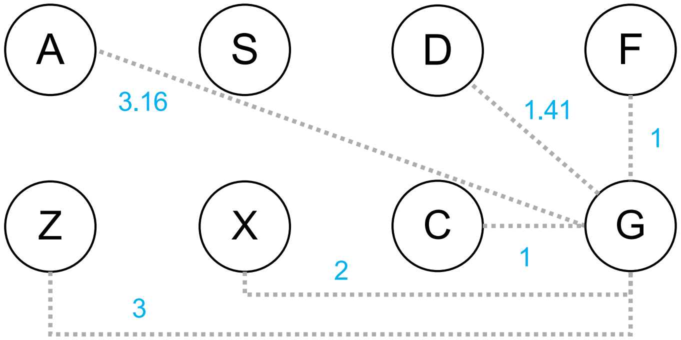
# Depth-First Search



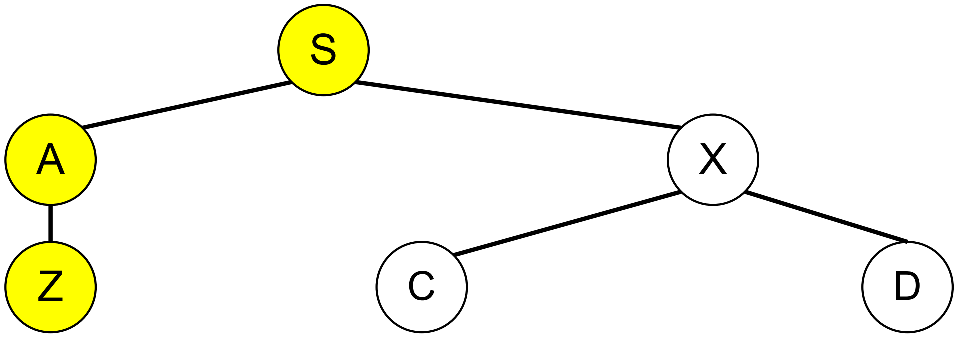


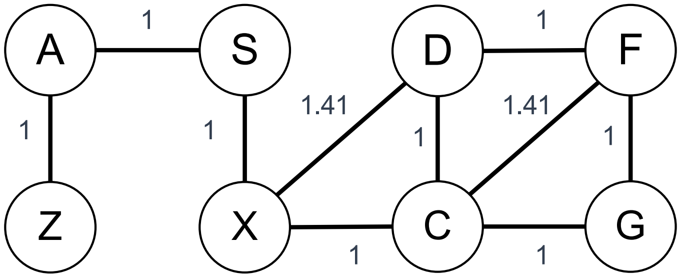
Path costs

SLD Heuristic



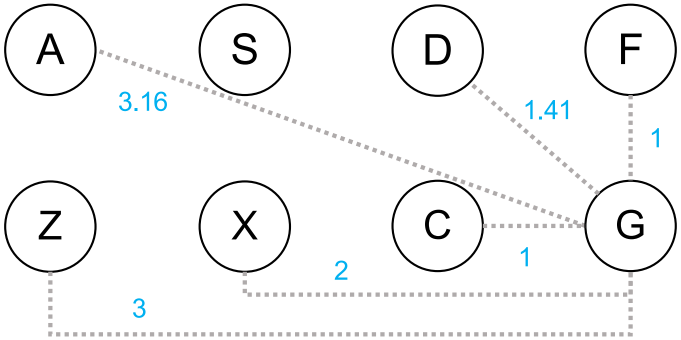
# Depth-First Search



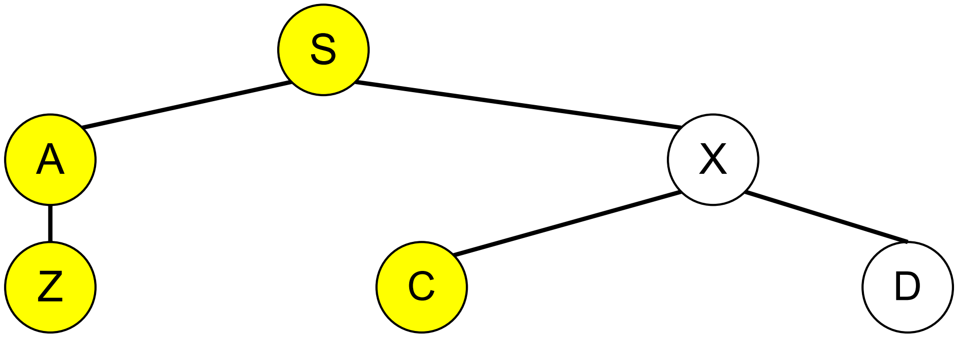


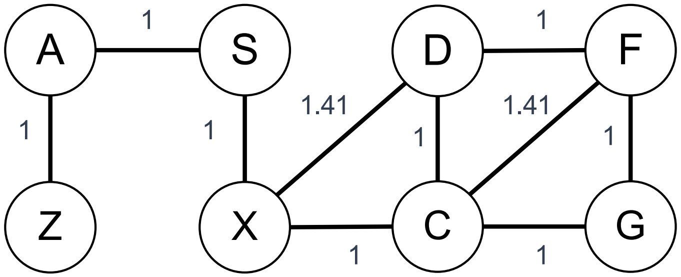
Path costs

SLD Heuristic



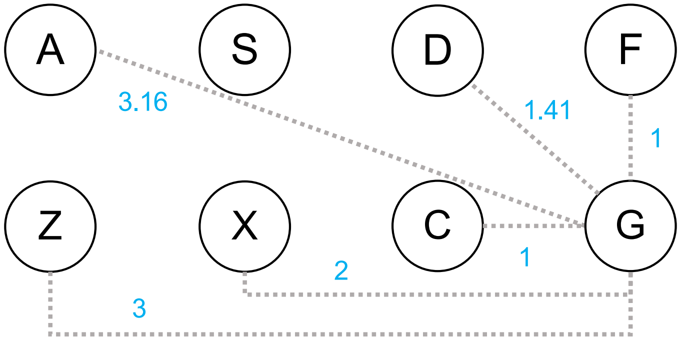
# Depth-First Search



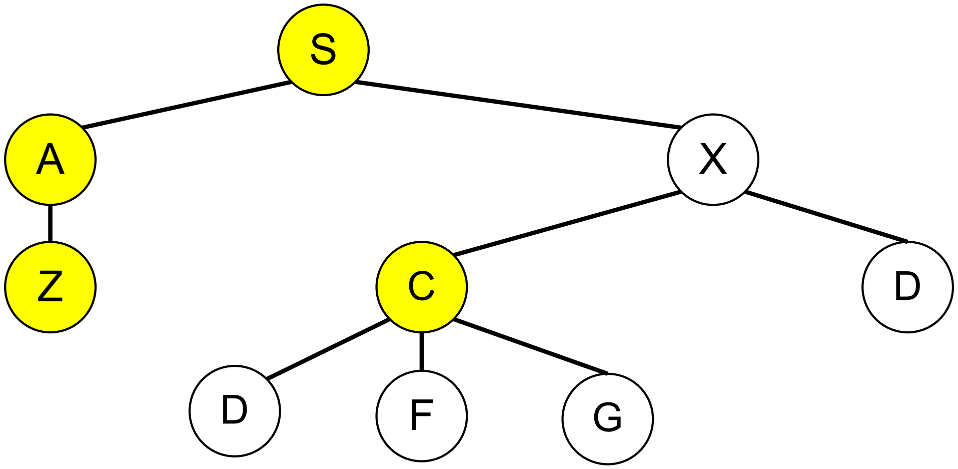


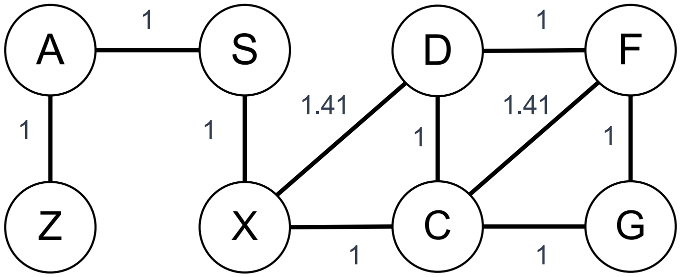
Path costs

SLD Heuristic



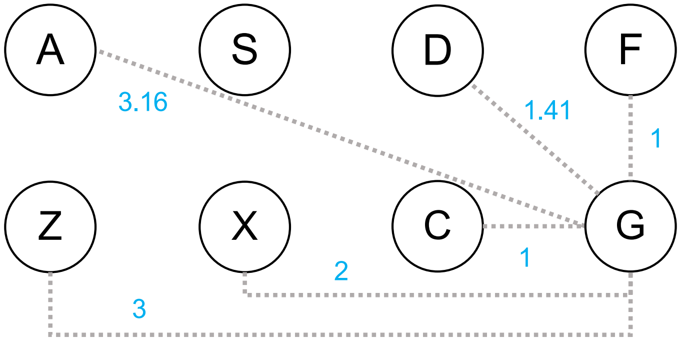
# Depth-First Search



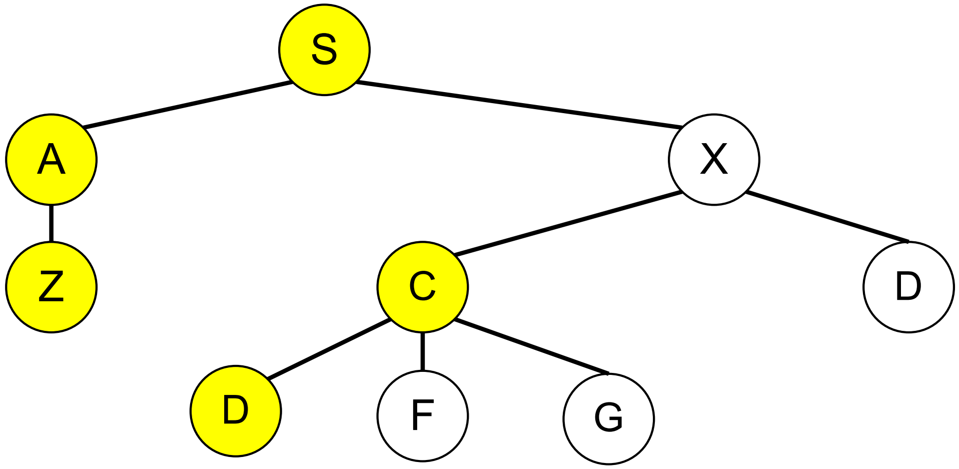


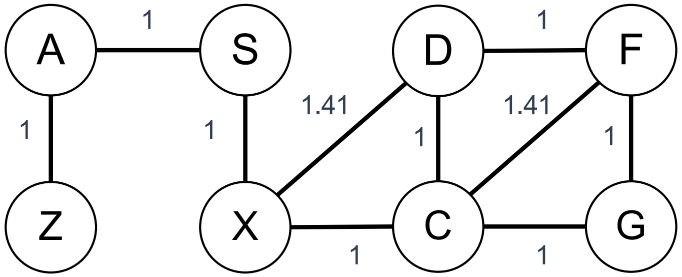
Path costs

SLD Heuristic



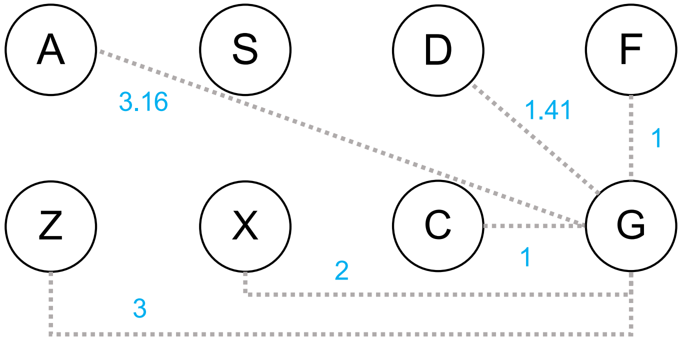
# Depth-First Search



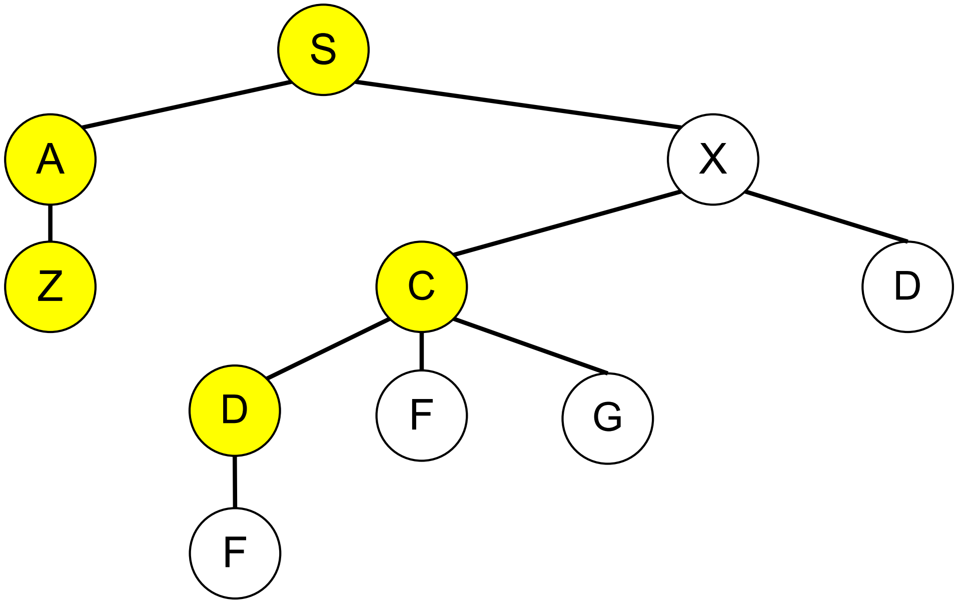


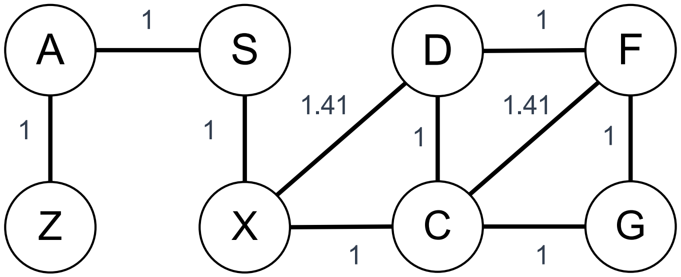
Path costs

SLD Heuristic



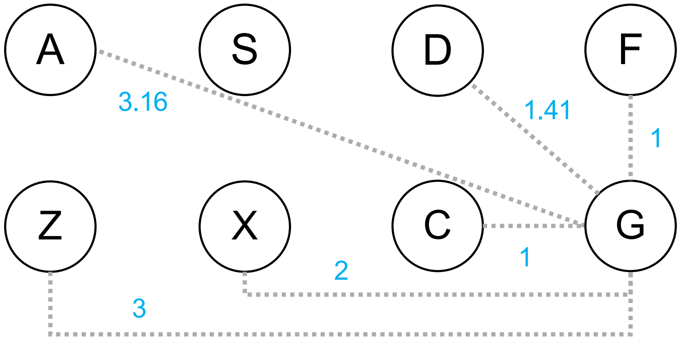
# Depth-First Search



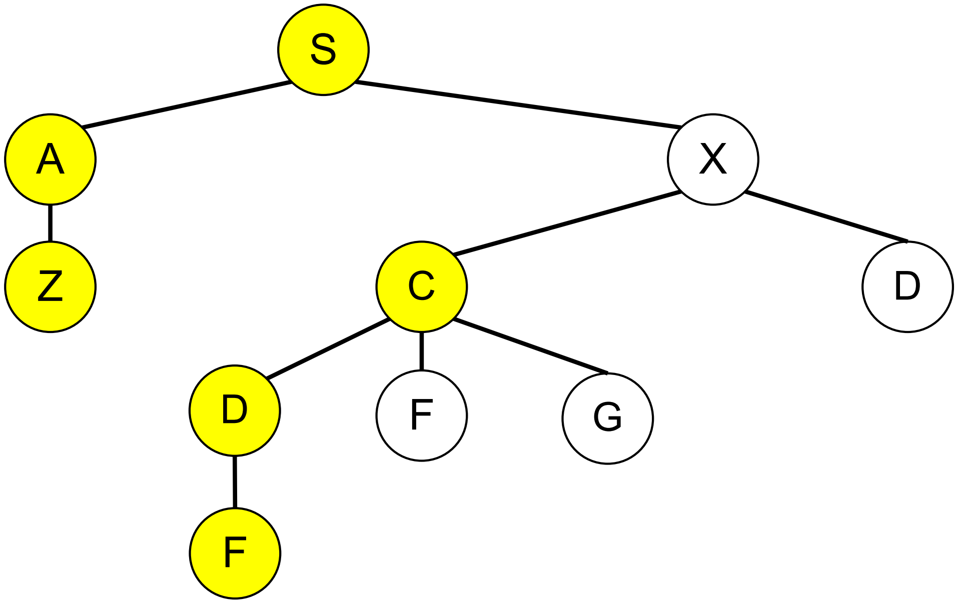


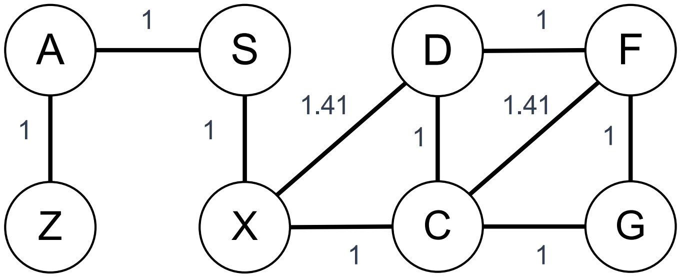
Path costs

SLD Heuristic



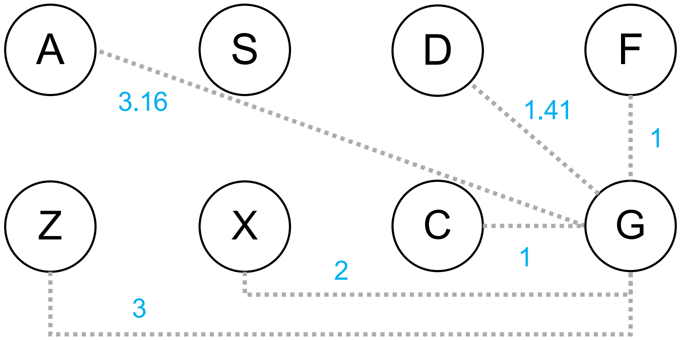
# Depth-First Search



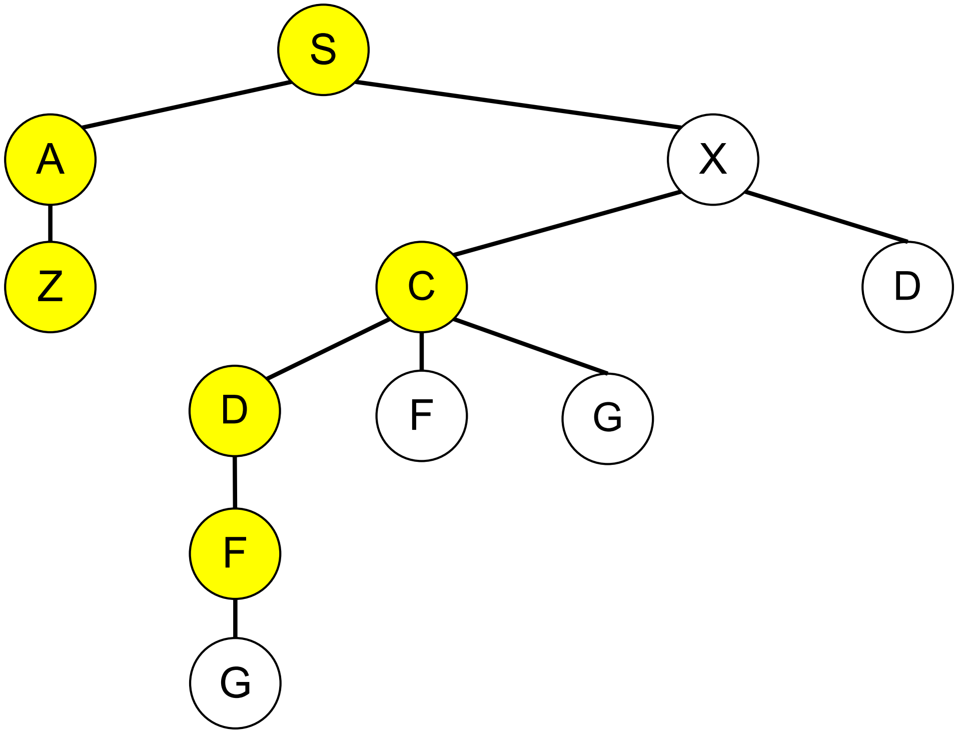


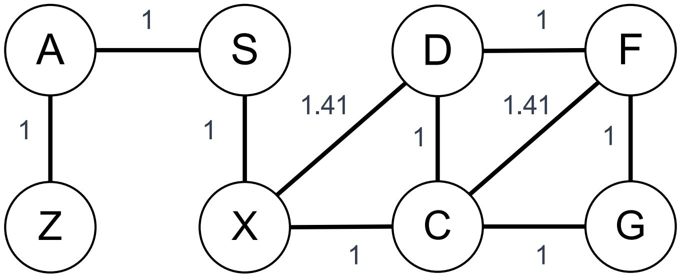
Path costs

SLD Heuristic



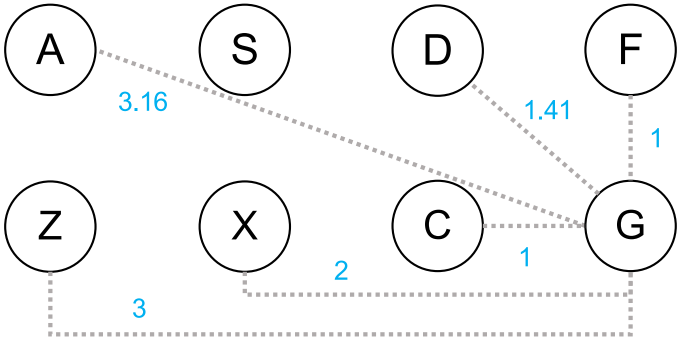
# Depth-First Search



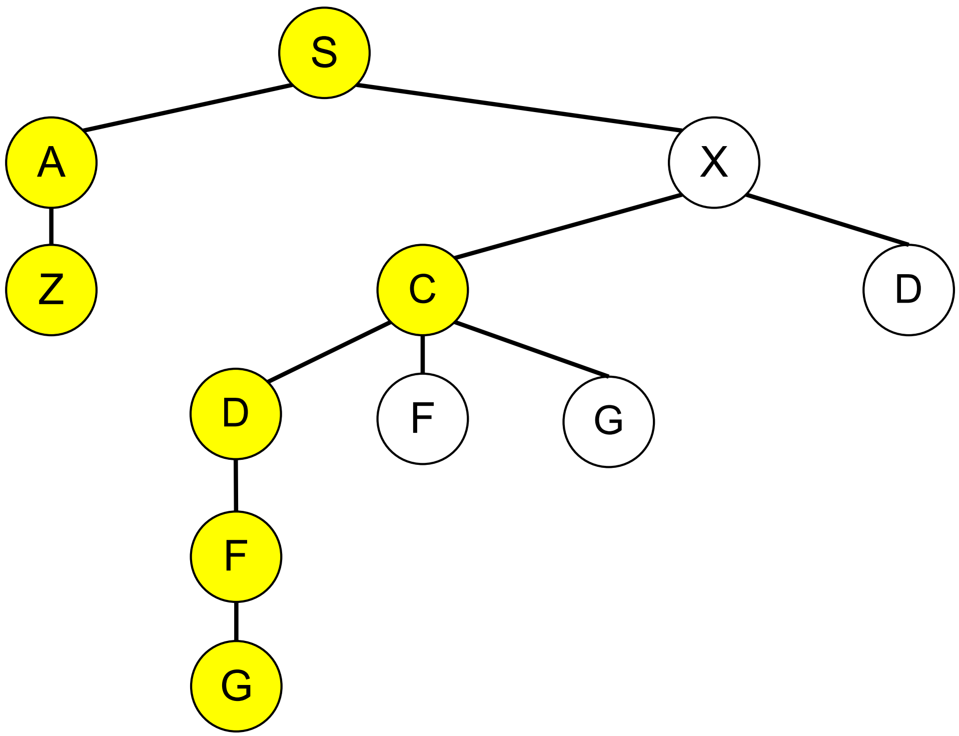


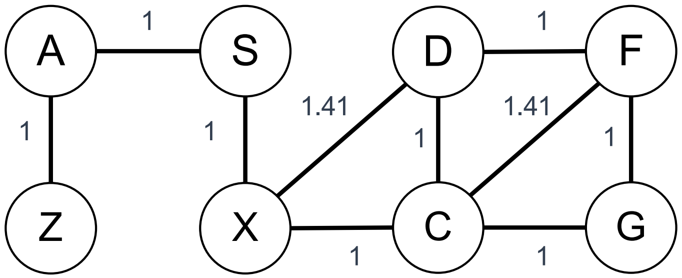
Path costs

SLD Heuristic



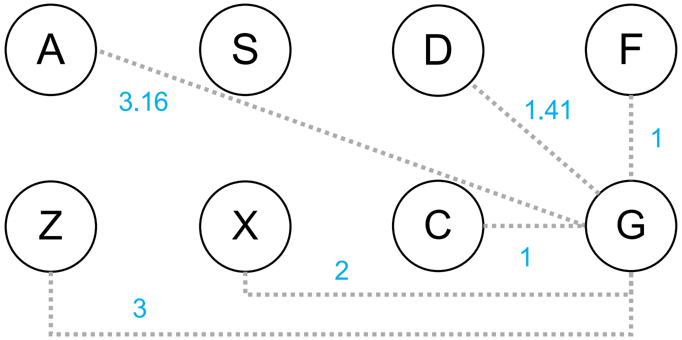
# Depth-First Search





Path costs

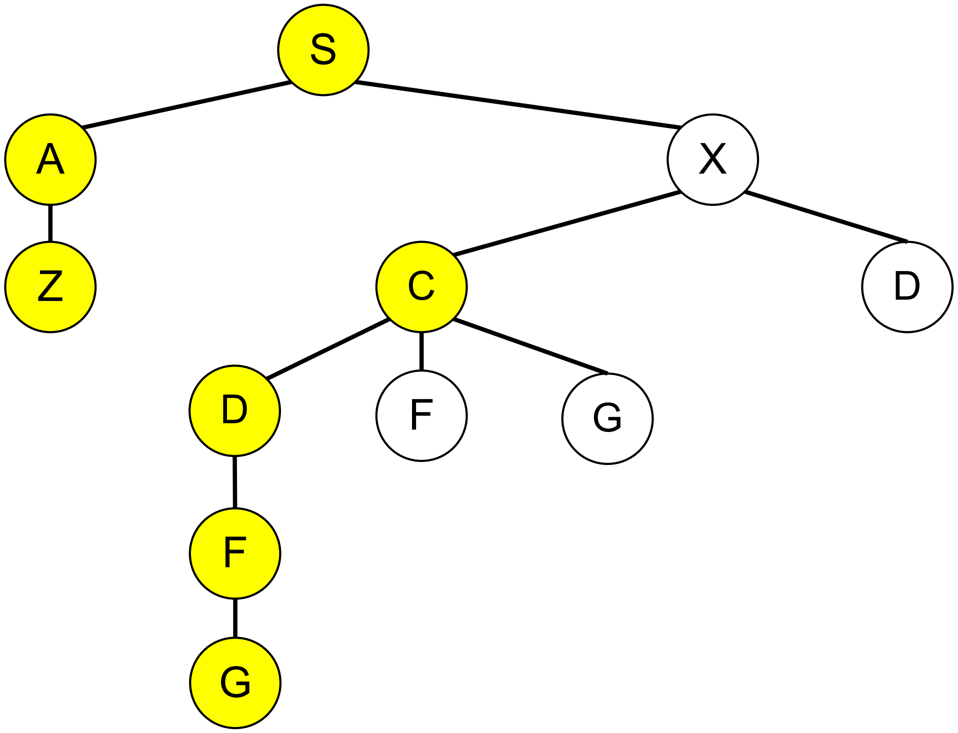
SLD Heuristic



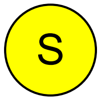
# Depth-First Search (Procedure)

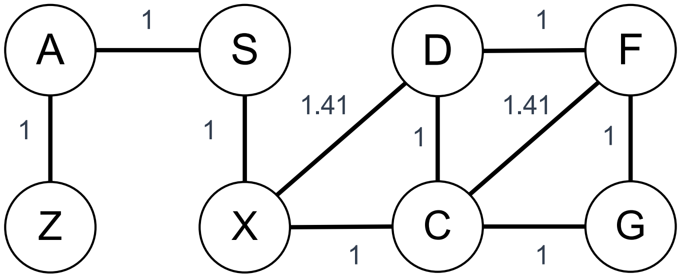
1. (S)
2. (SA)(SX)
3. (SAZ)(SX)
4. (SX)
5. (SXC)(SXD)
6. (SXCD)(SXCF)(SXCG)(SXD)
7. (SXCDF)(SXCF)(SXCG)(SXD)
8. (SXCDFG)(SXCF)(SXCG)(SXD)
9. DONE

What's being checked



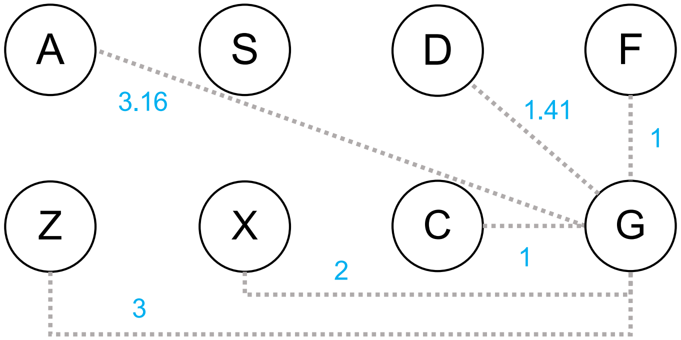
# Breadth-First Search



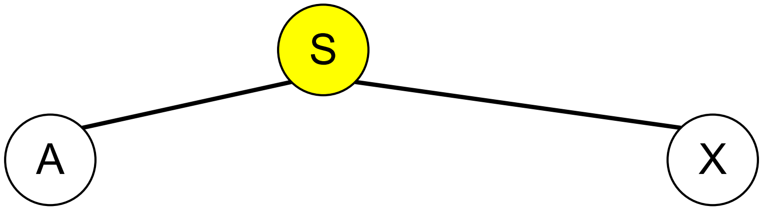


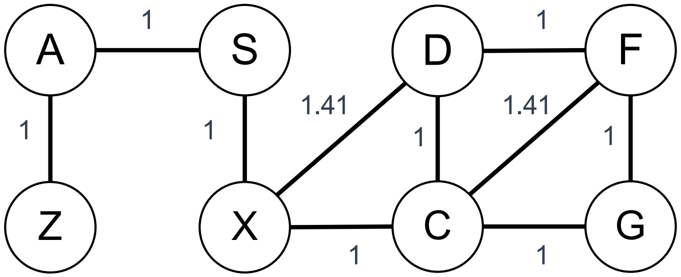
Path costs

SLD Heuristic



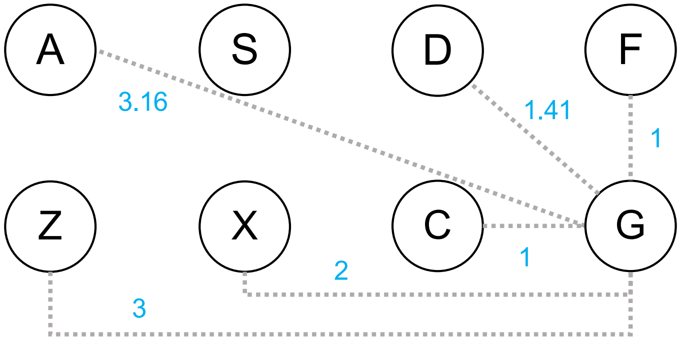
# Breadth-First Search



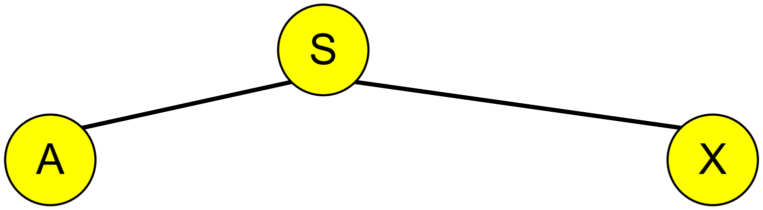


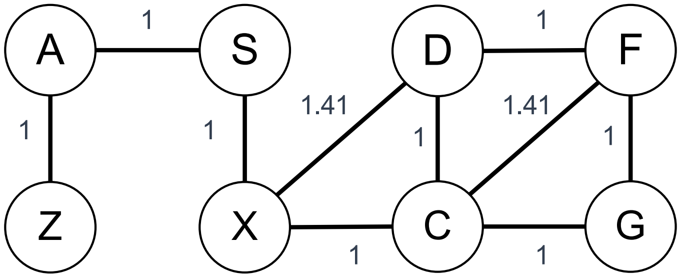
Path costs

SLD Heuristic



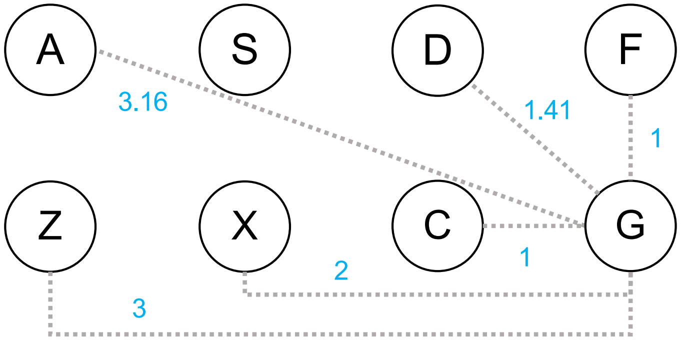
# Breadth-First Search



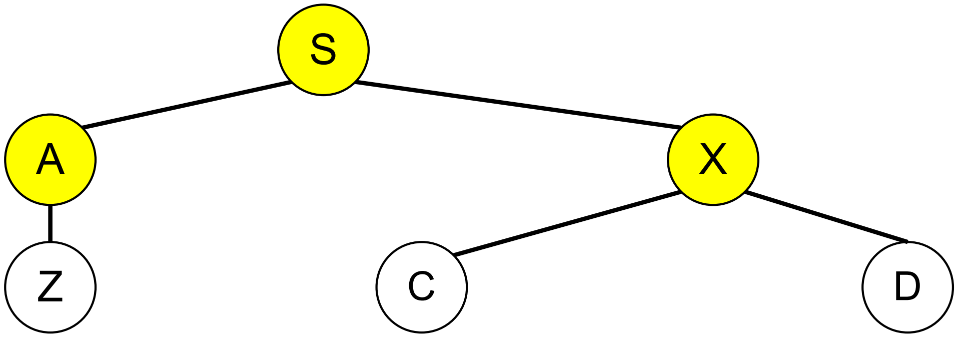


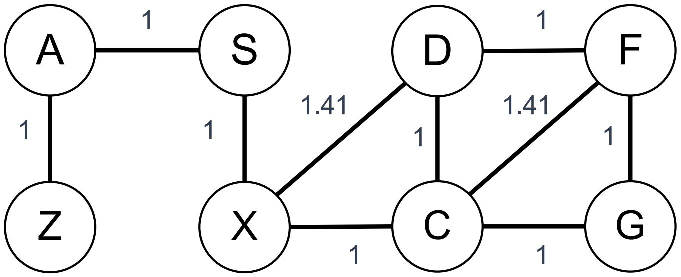
Path costs

SLD Heuristic



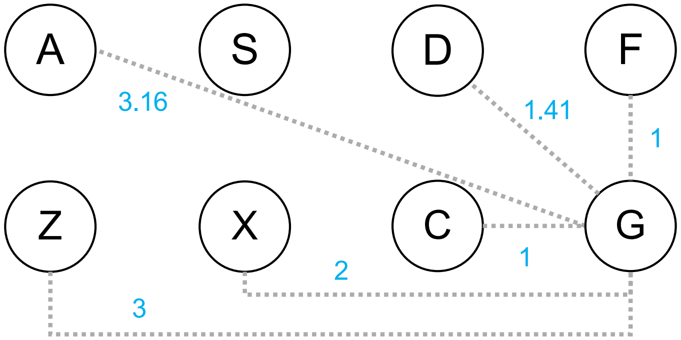
# Breadth-First Search



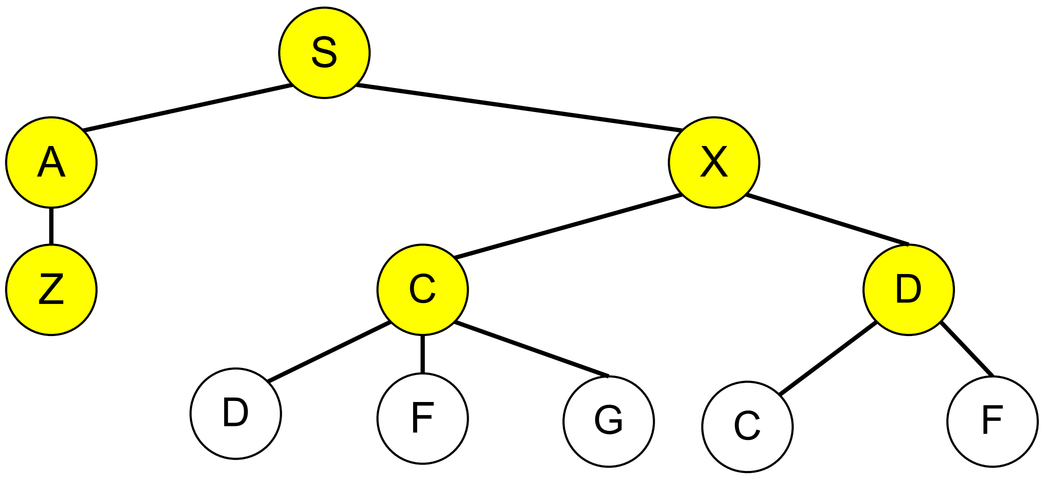


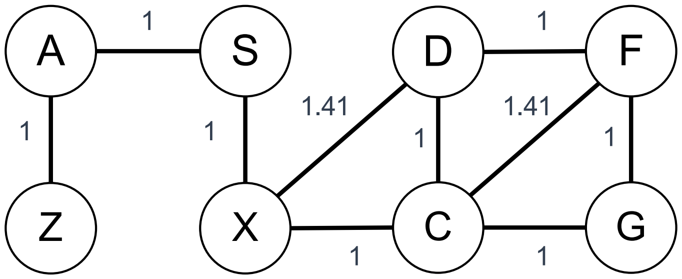
Path costs

SLD Heuristic



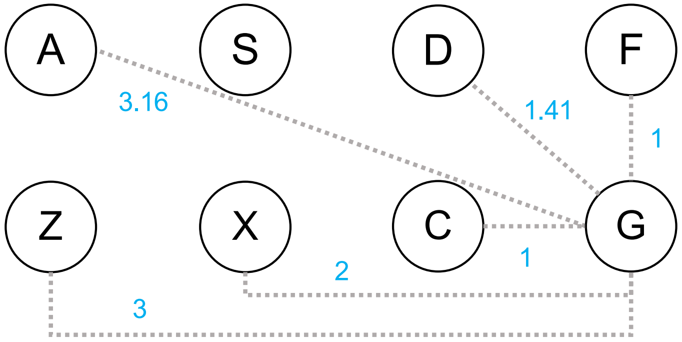
# Breadth-First Search



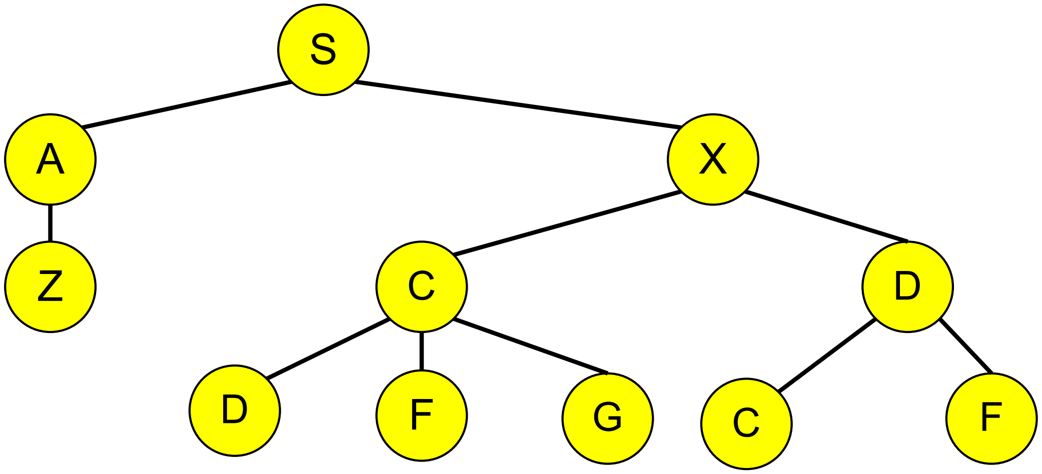


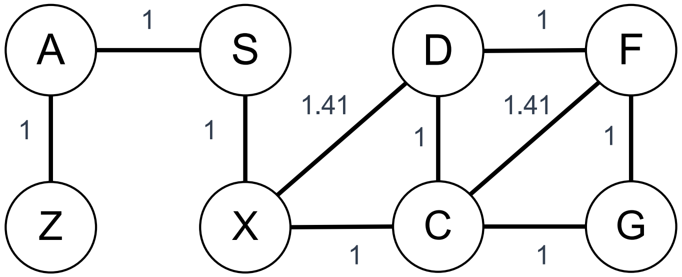
Path costs

SLD Heuristic



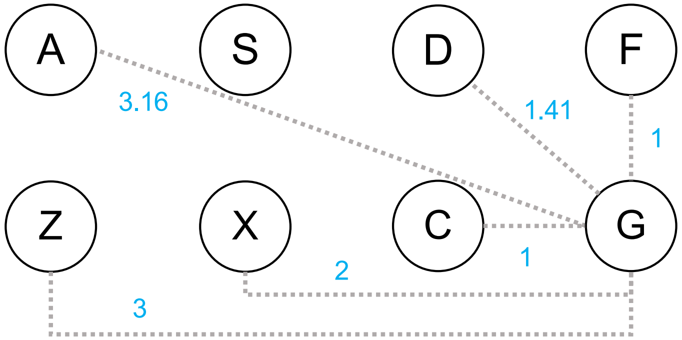
# Breadth-First Search





Path costs

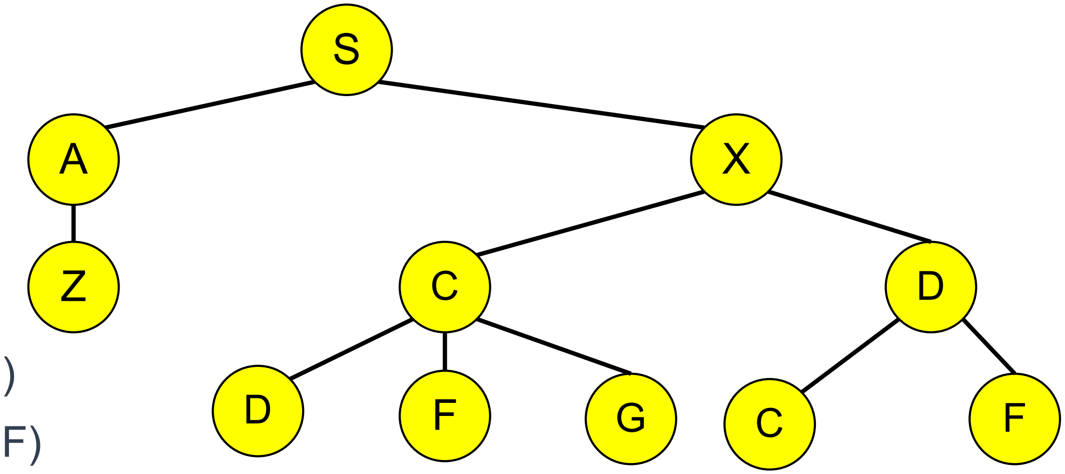
SLD Heuristic



# Breadth-First Search (Procedure)

1. (S)
2. (SA)(SX)
3. (SX)(SAZ)
4. (SAZ)(SXC)(SXD)
5. (SXC)(SXD)
6. (SXD)(SXCD)(SXCF)(SXCG)
7. (SXCD)(SXCF)(SXCG)(SXDC)(SXDF)
8. (SXCF)(SXCG)(SXDC)(SXDF)(SXCDF)
9. (SXCG)(SXDC)(SXDF)(SXCDF)(SXCFD)
10. DONE

What's being checked



# DFS Vs BFS

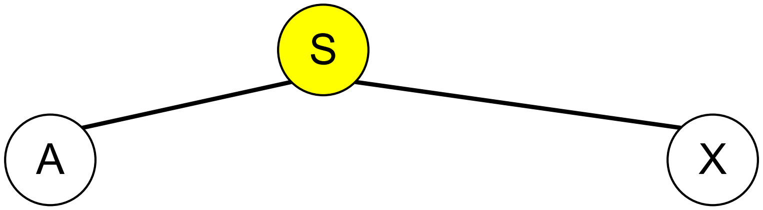
1. (S)
2. (SA)(SX)
3. (SAZ)(SX)
4. (SX)
5. (SXC)(SXD)
6. (SXCD)(SXCF)(SXCG)(SXD)
7. (SXCDF)(SXCF)(SXCG)(SXD)
8. (SXCDFG)(SXCF)(SXCG)(SXD)
9. DONE

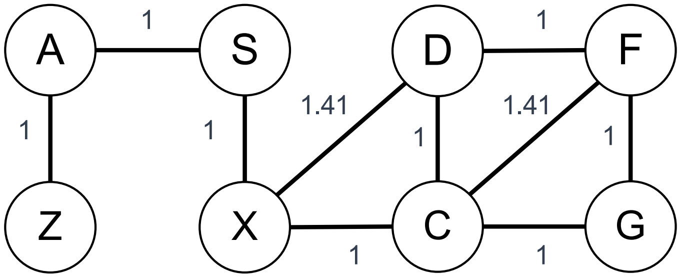
What's being checked

1. (S)
2. (SA)(SX)
3. (SX)(SAZ)
4. (SAZ)(SXC)(SXD)
5. (SXC)(SXD)
6. (SXD)(SXCD)(SXCF)(SXCG)
7. (SXCD)(SXCF)(SXCG)(SXDC)(SXDF)
8. (SXCF)(SXCG)(SXDC)(SXDF)(SXCDF)
9. (SXCG)(SXDC)(SXDF)(SXCDF)(SXCFD)
10. DONE

What's being checked

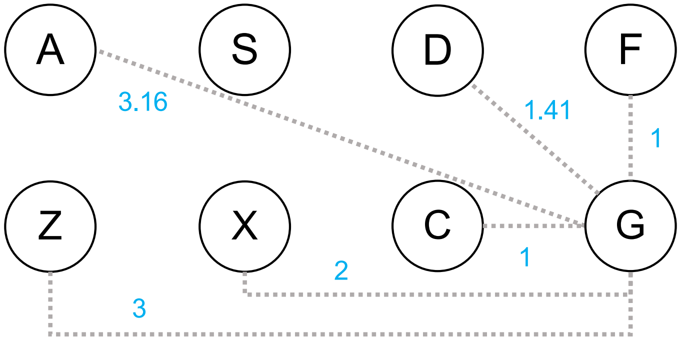
# Hill Climbing Search



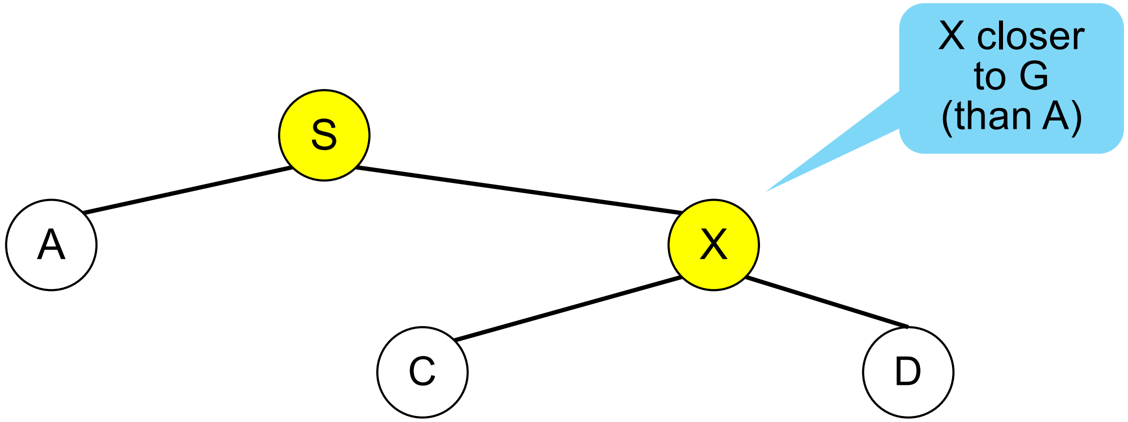


Path costs

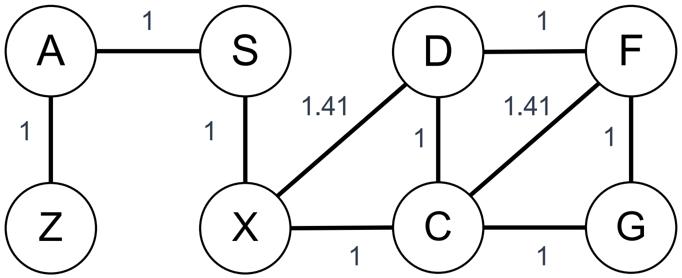
SLD Heuristic



# Hill Climbing Search

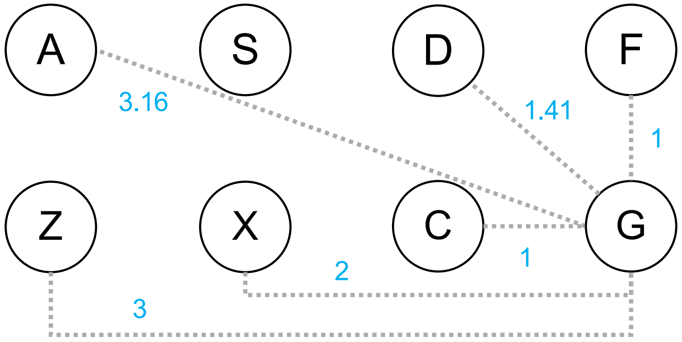


X closer to G (than A)

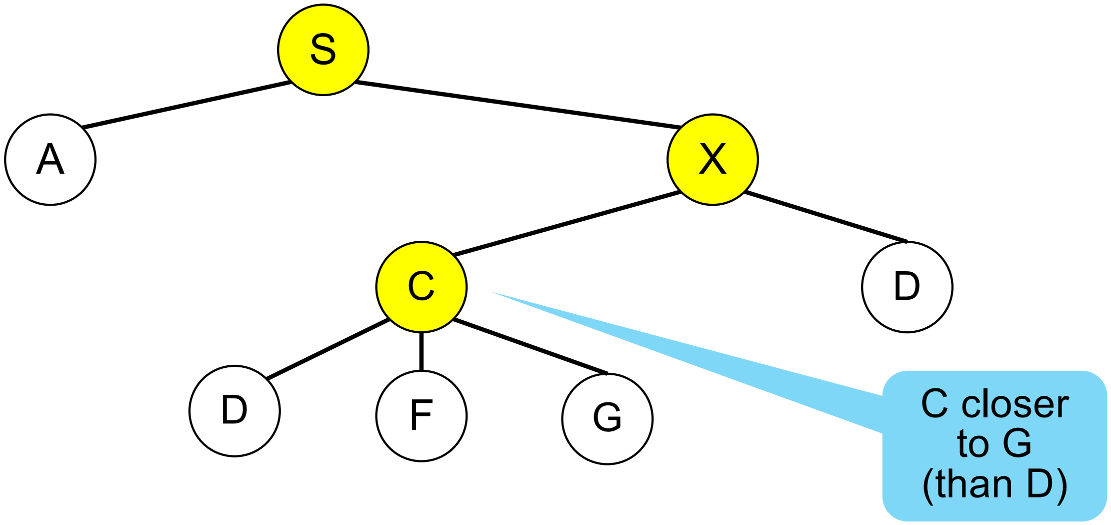


Path costs

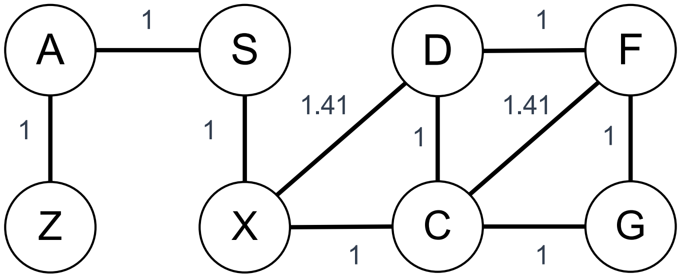
SLD Heuristic



# Hill Climbing Search

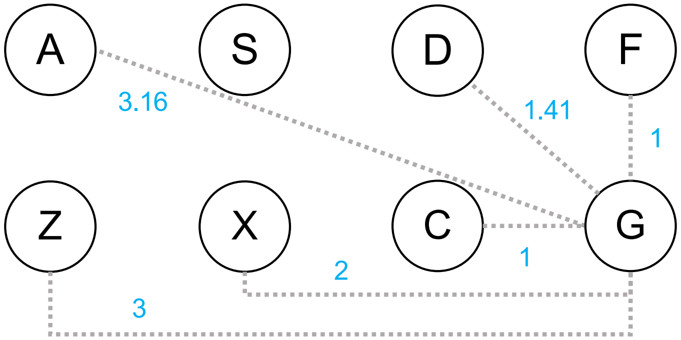


C closer to G (than D)

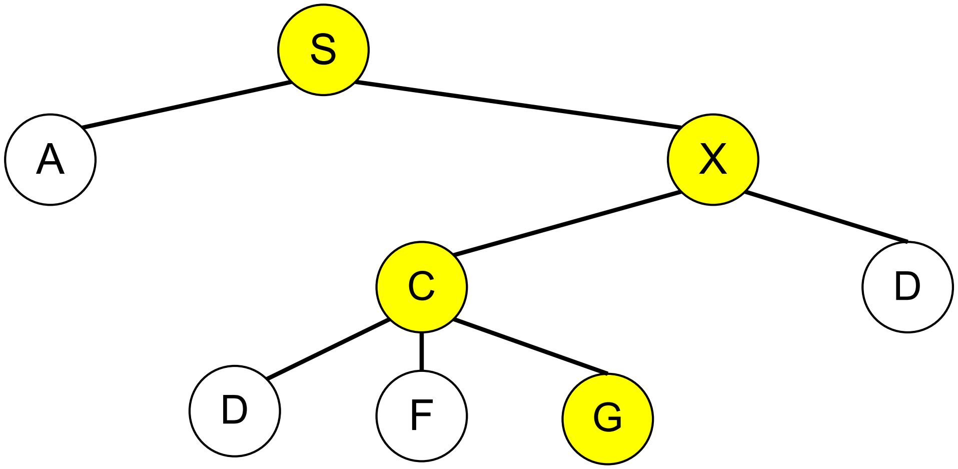


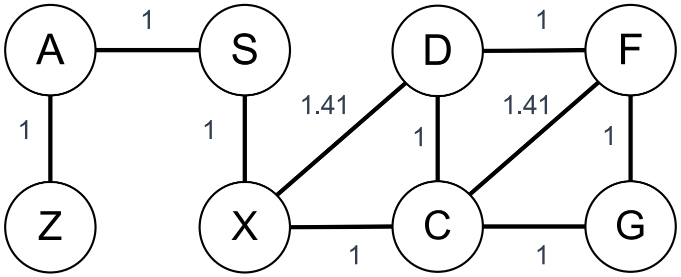
Path costs

SLD Heuristic



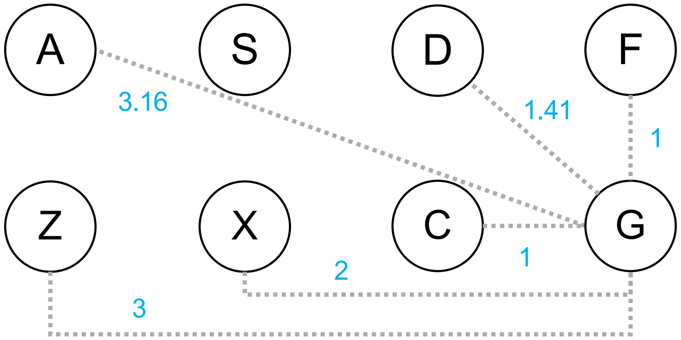
# Hill Climbing Search



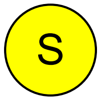


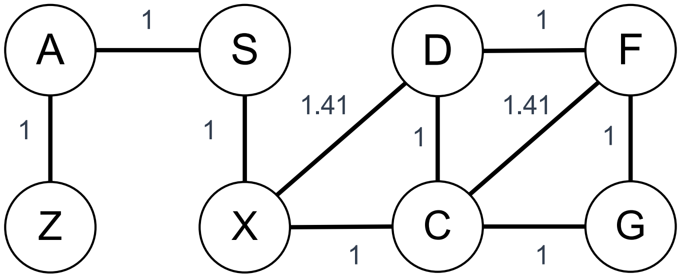
Path costs

SLD Heuristic



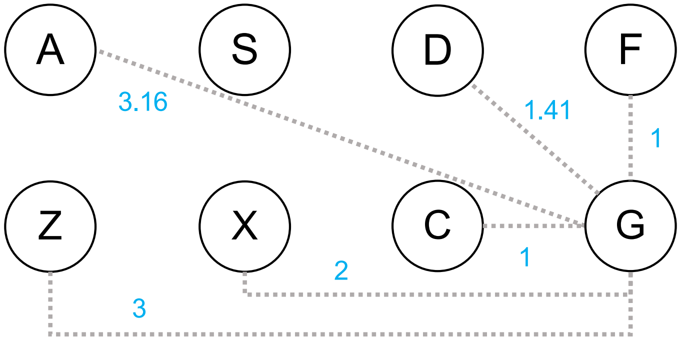
# Beam Search





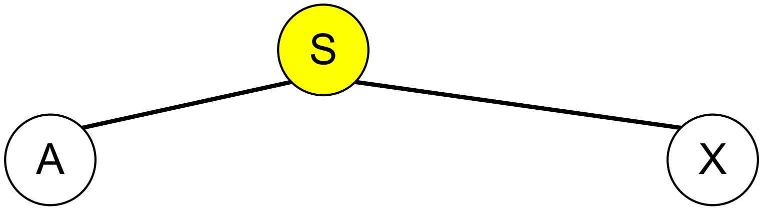
Path costs

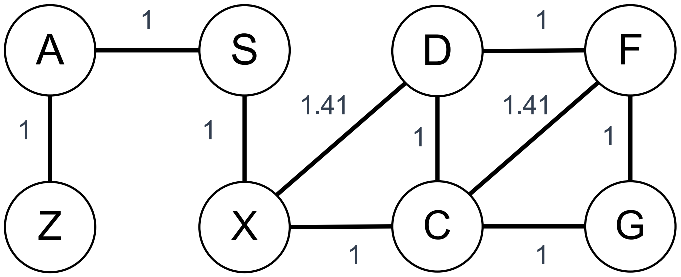
SLD Heuristic



- Let *w* = 1

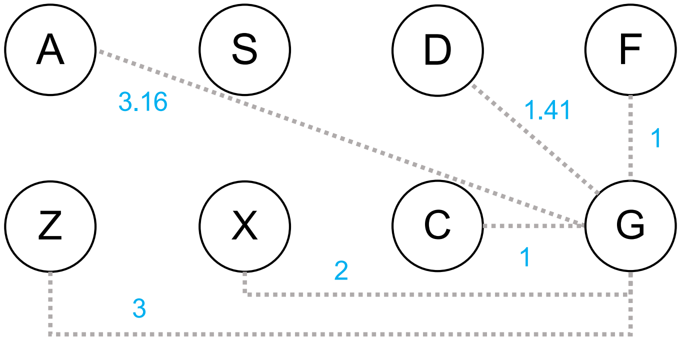
# Beam Search





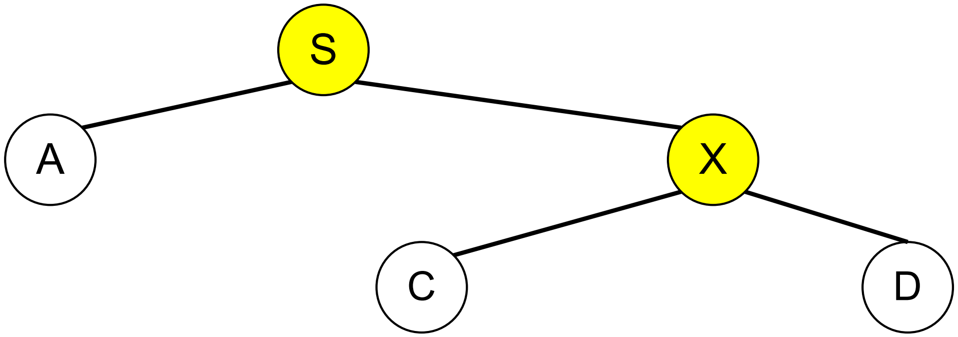
Path costs

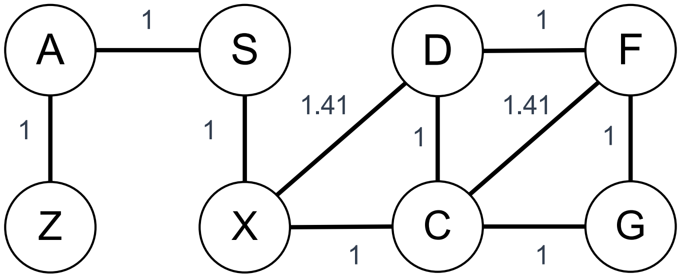
SLD Heuristic



- Let *w* = 1

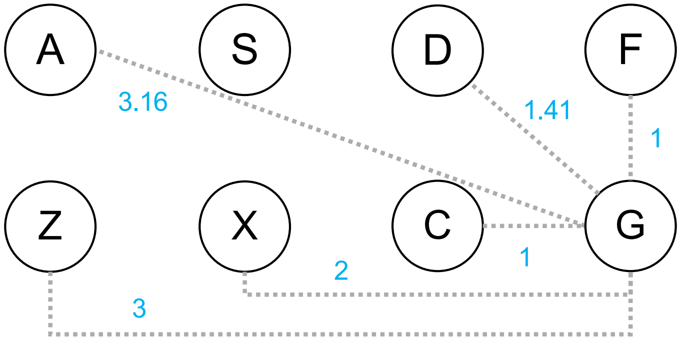
# Beam Search





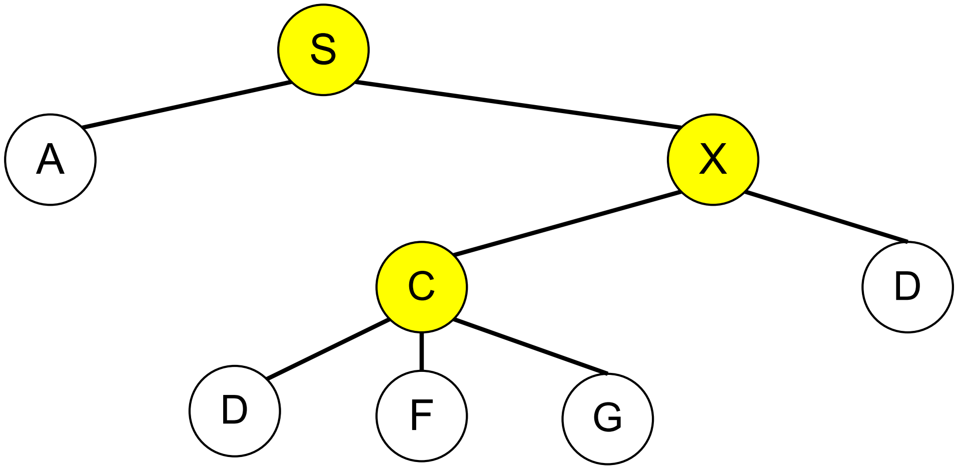
Path costs

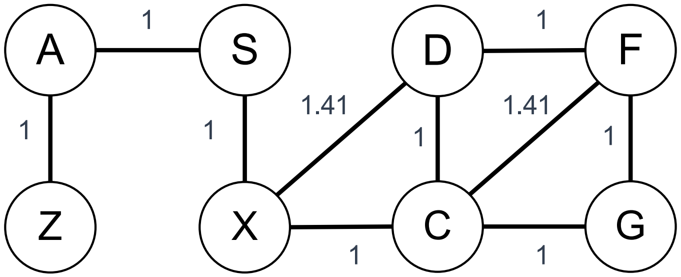
SLD Heuristic



- Let *w* = 1

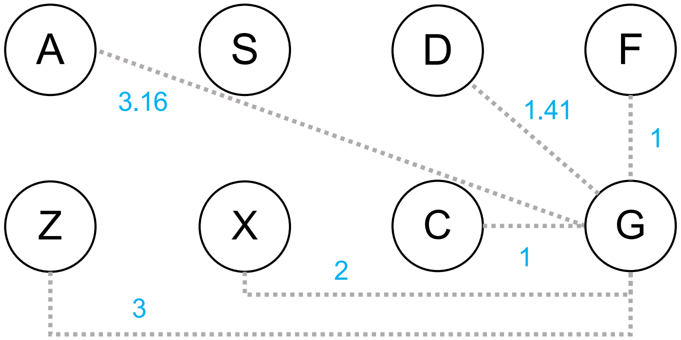
# Beam Search





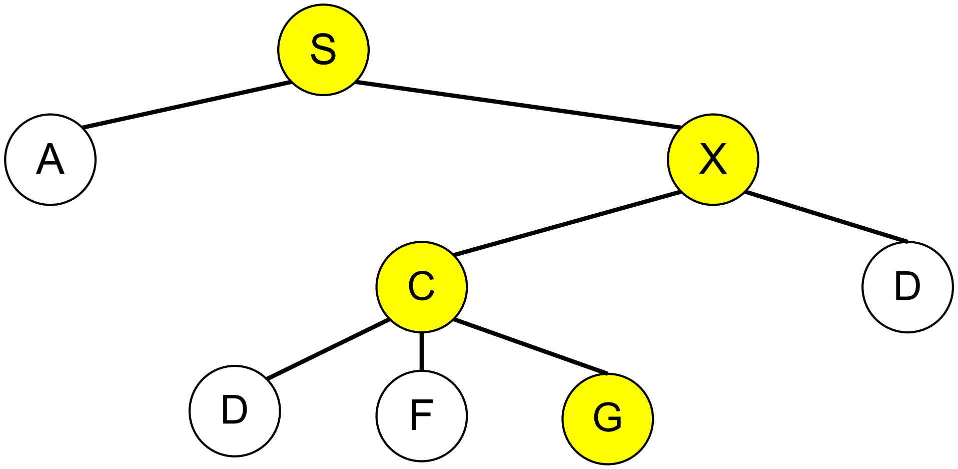
Path costs

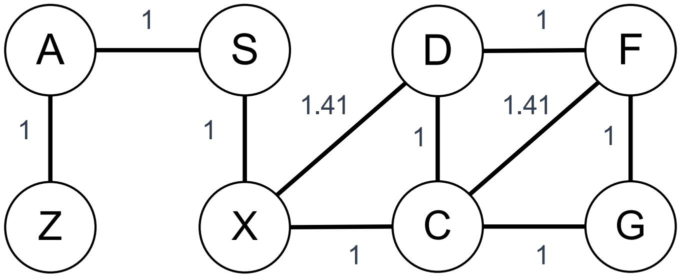
SLD Heuristic



- Let *w* = 1

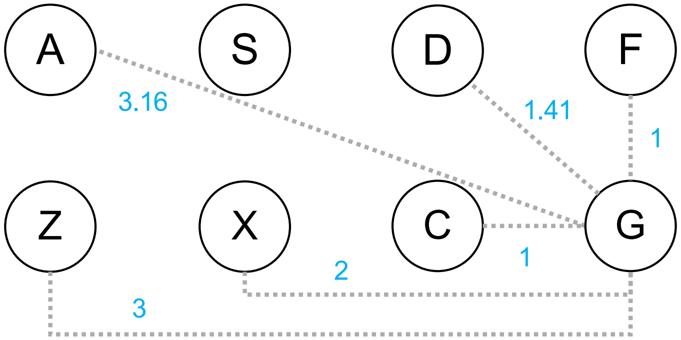
# Beam Search





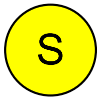
Path costs

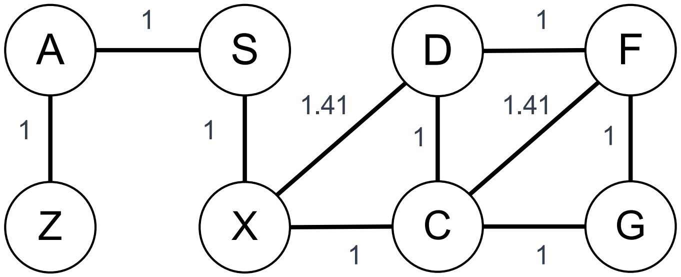
SLD Heuristic



- Let *w* = 1

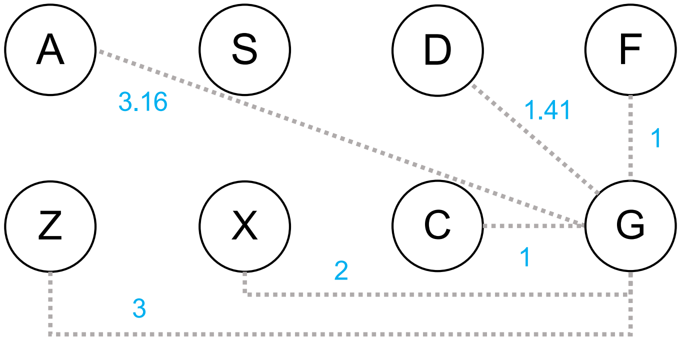
# Beam Search





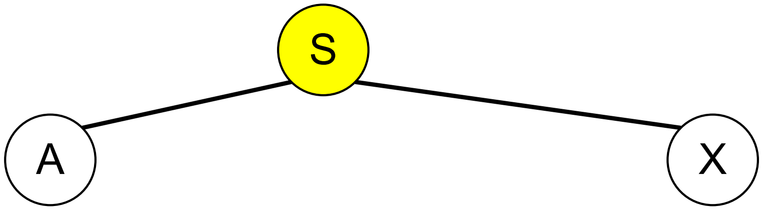
Path costs

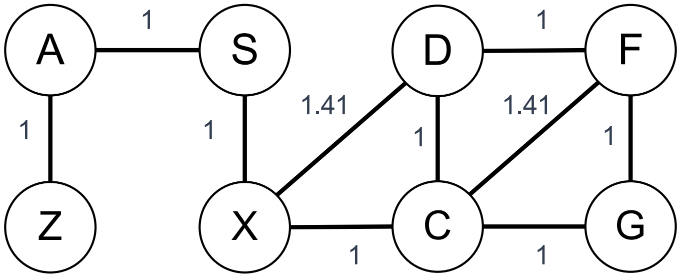
SLD Heuristic



- Let *w* = 2

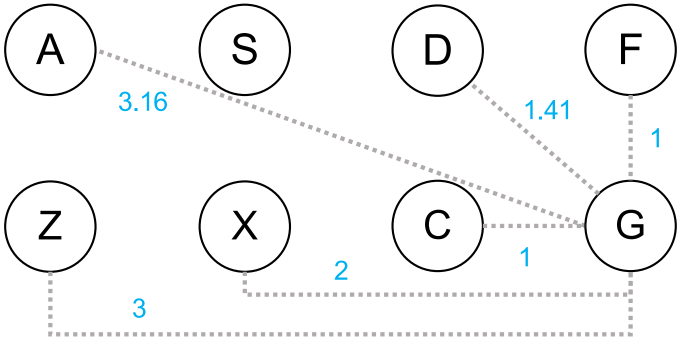
# Beam Search





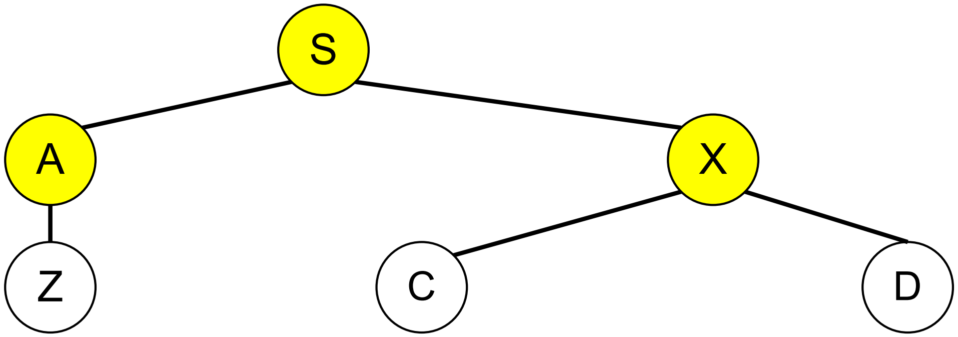
Path costs

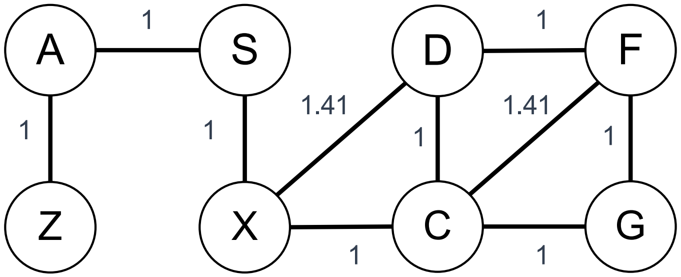
SLD Heuristic



- Let *w* = 2

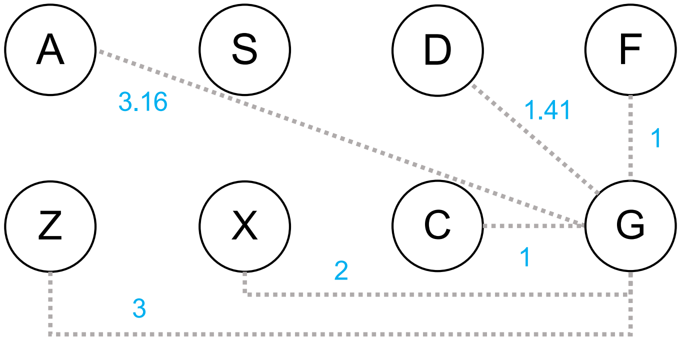
# Beam Search





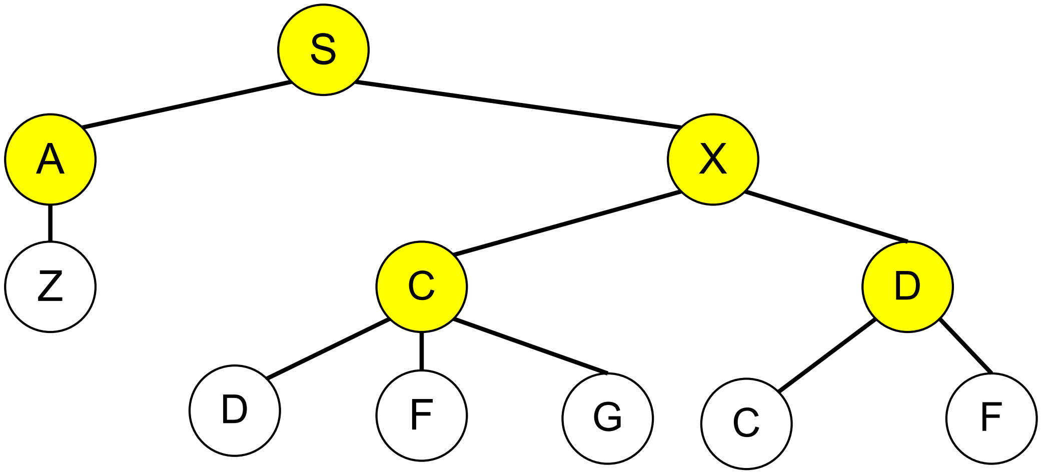
Path costs

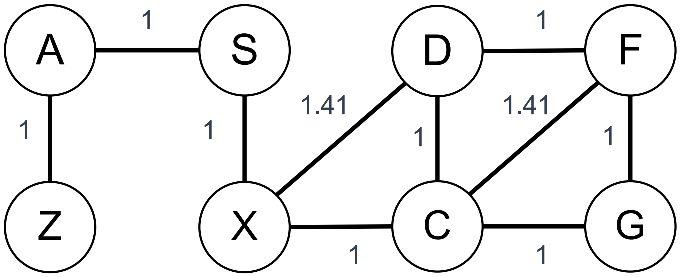
SLD Heuristic



- Let $w = 2$

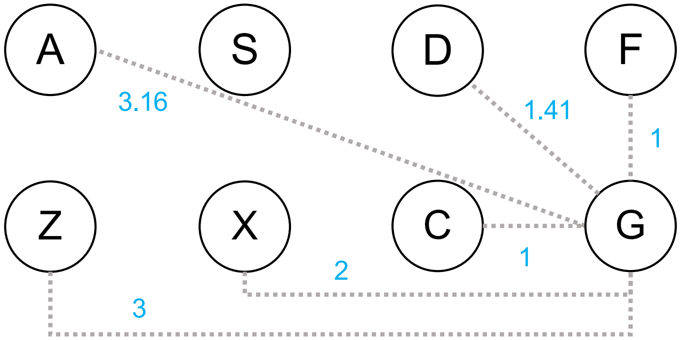
# Beam Search





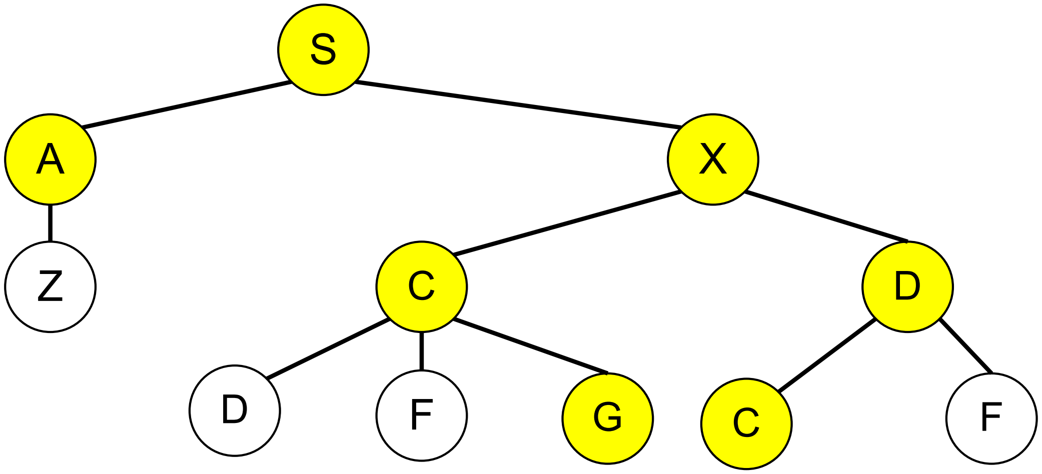
Path costs

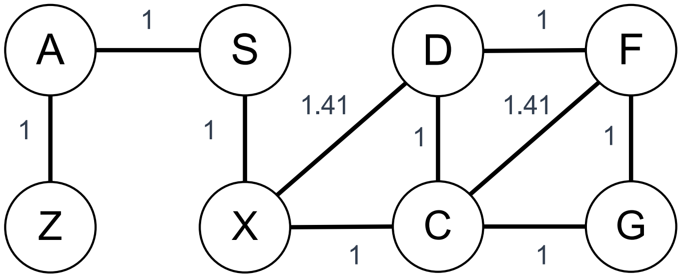
SLD Heuristic



- Let *w* = 3

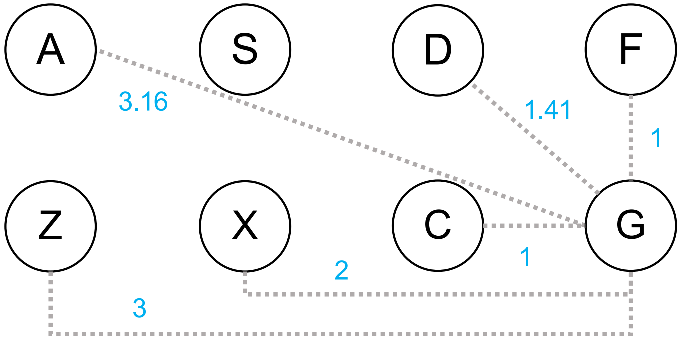
# Beam Search





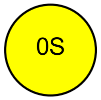
Path costs

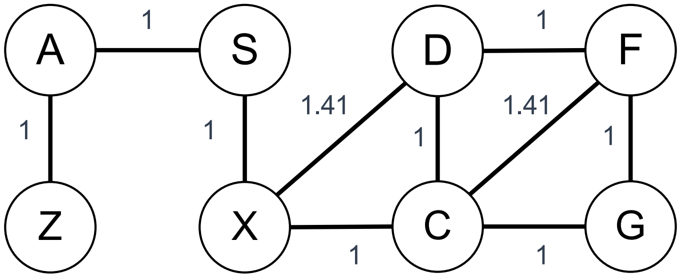
SLD Heuristic



- Let *w* = 2

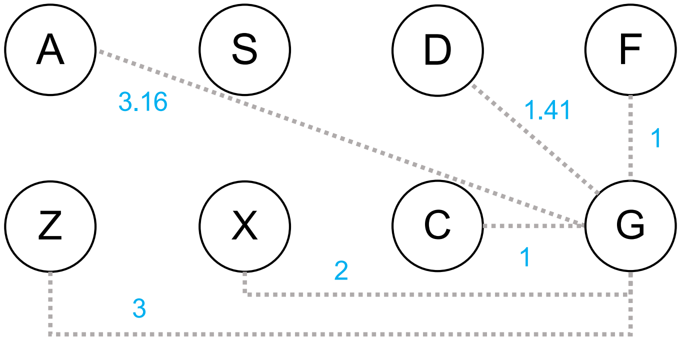
# Branch and Bound



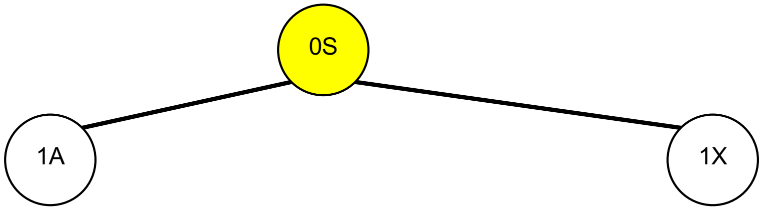


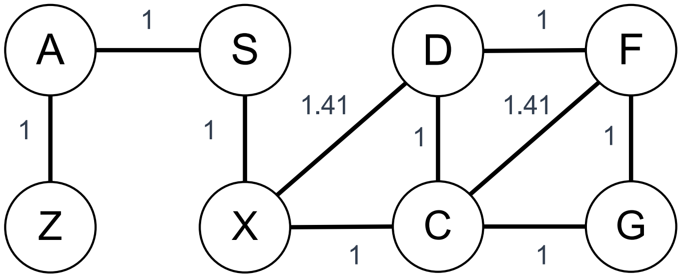
Path costs

SLD Heuristic



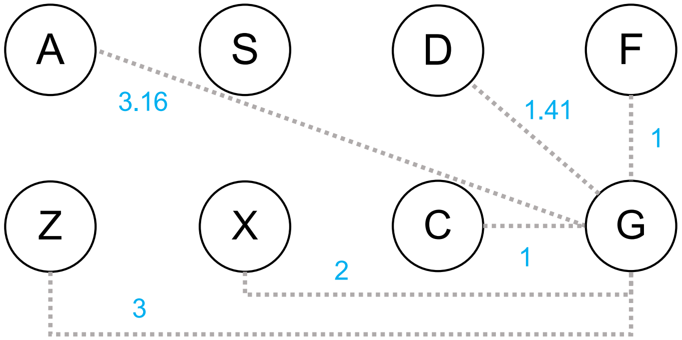
# Branch and Bound



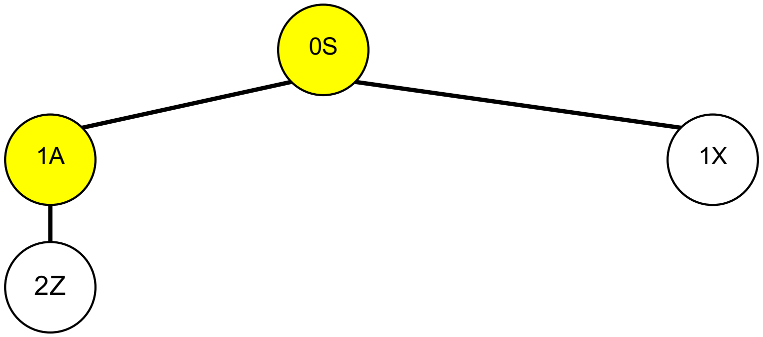


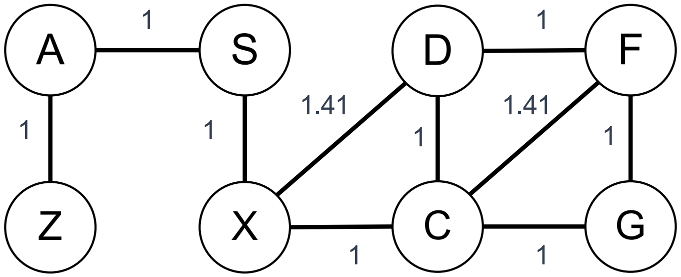
Path costs

SLD Heuristic



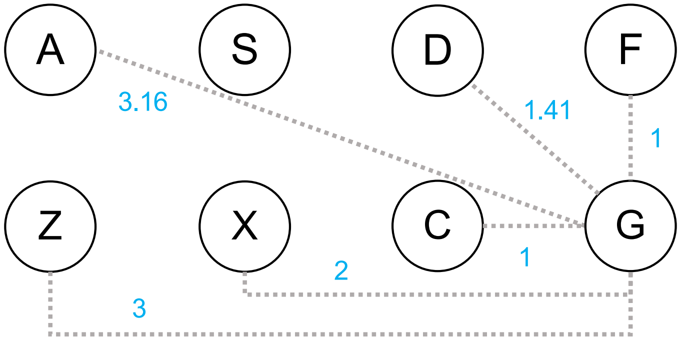
# Branch and Bound



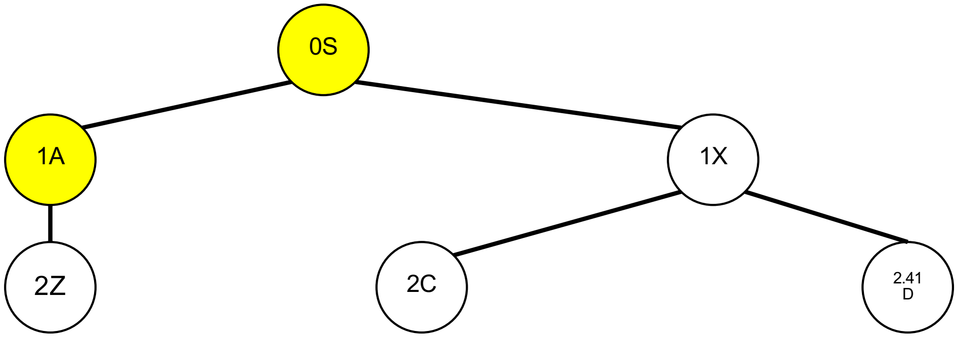


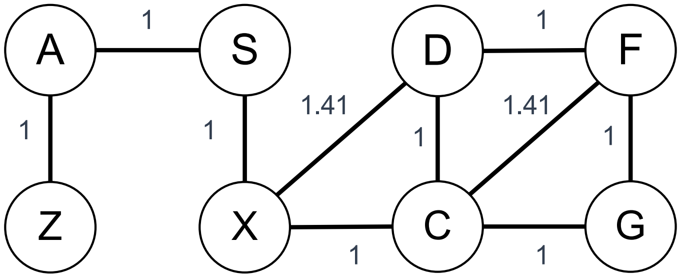
Path costs

SLD Heuristic



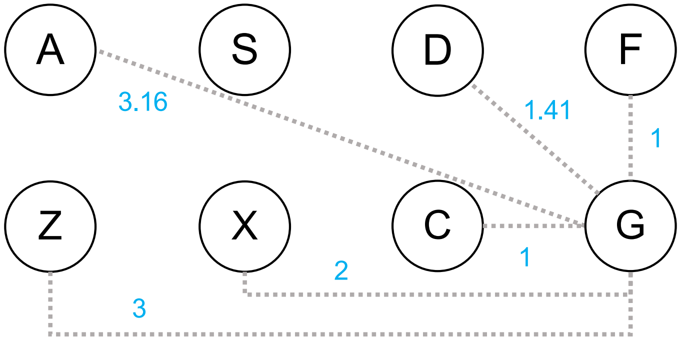
# Branch and Bound



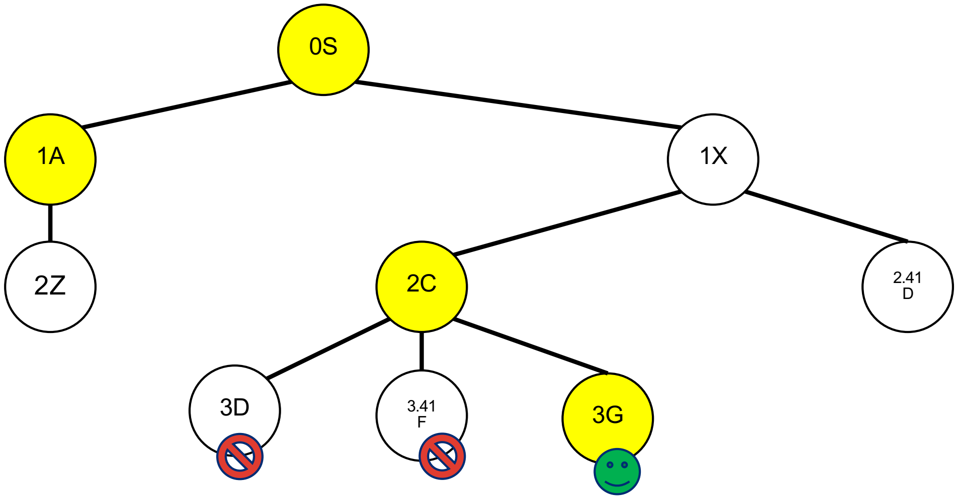


Path costs

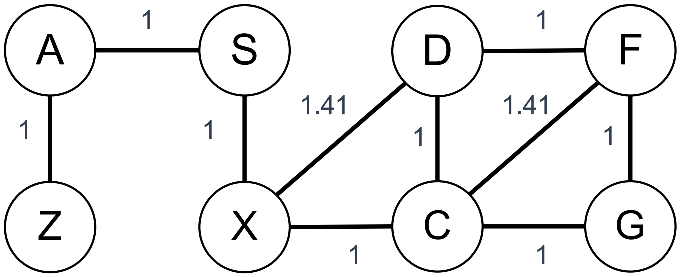
SLD Heuristic



# Branch and Bound

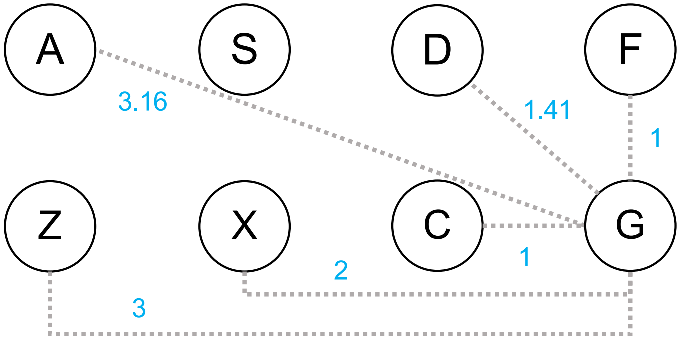


🚫 Stopped because of cost

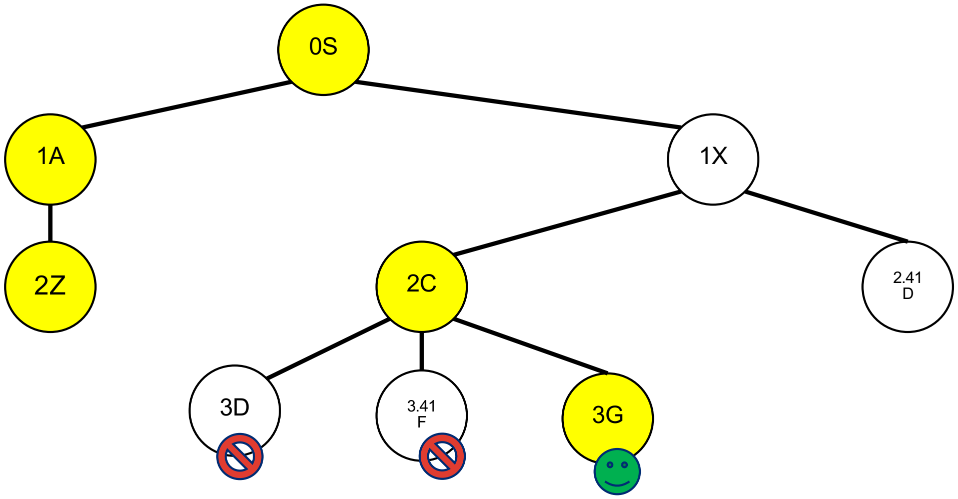


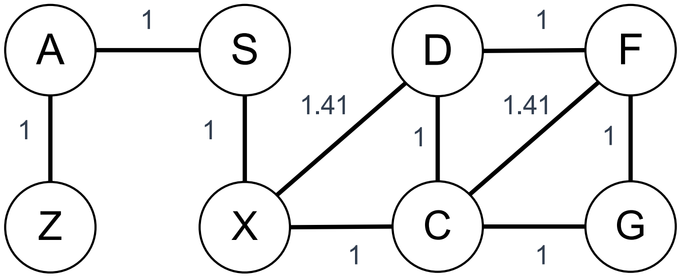
Path costs

SLD Heuristic



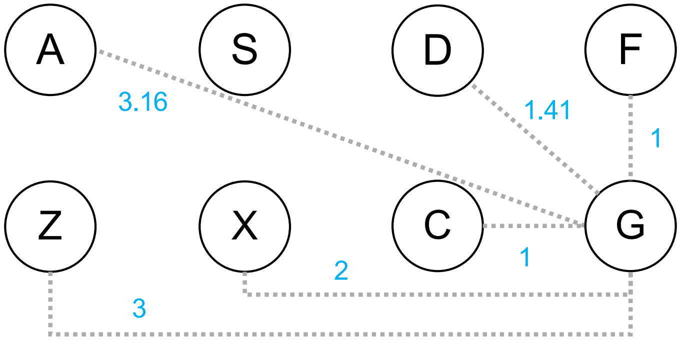
# Branch and Bound





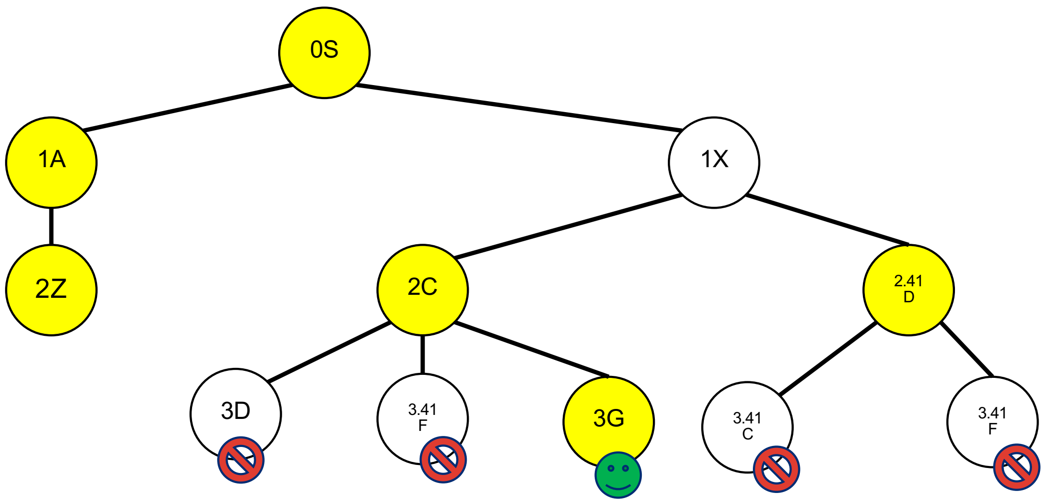
Path costs

SLD Heuristic

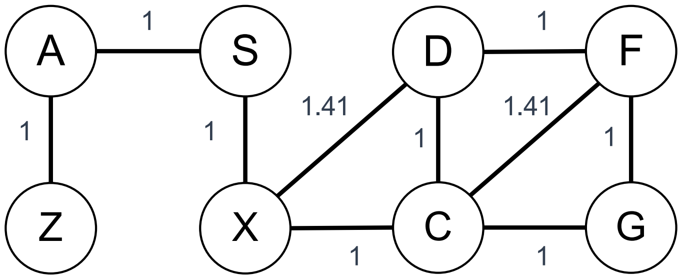


🚫 Stopped because of cost

# Branch and Bound

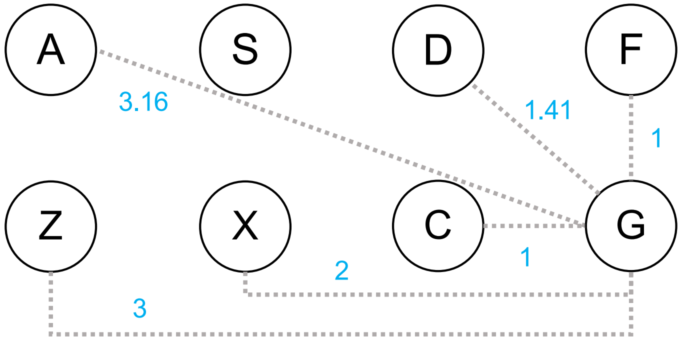


🚫 Stopped because of cost

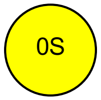


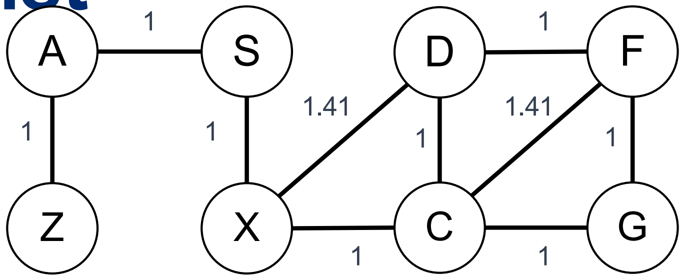
Path costs

SLD Heuristic



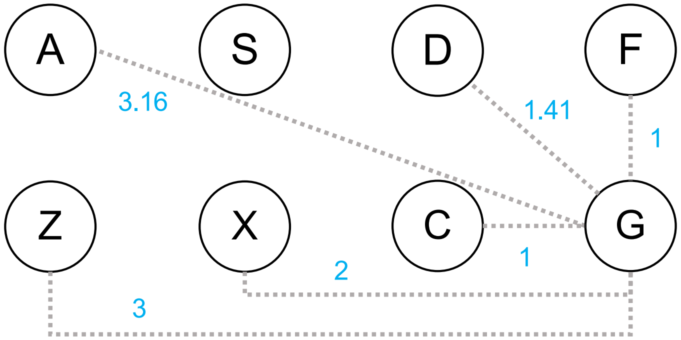
# Branch and Bound + Extended List



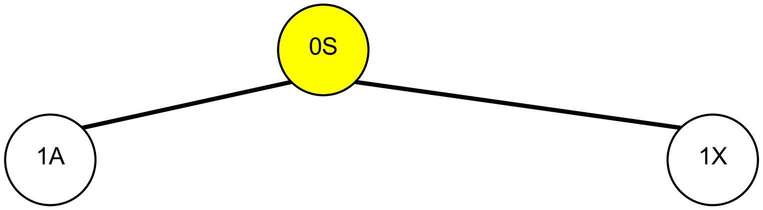


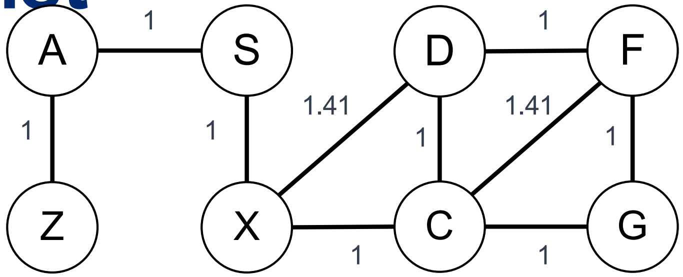
Path costs

SLD Heuristic



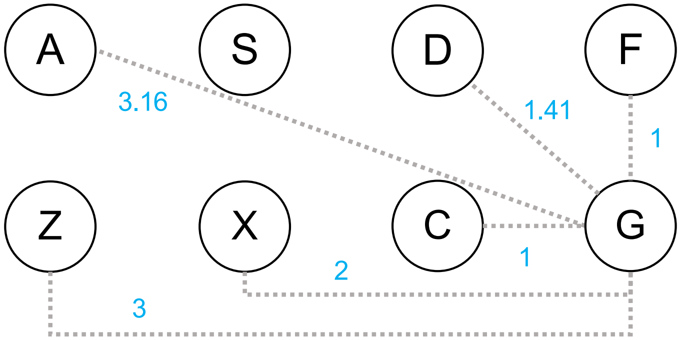
# Branch and Bound + Extended List



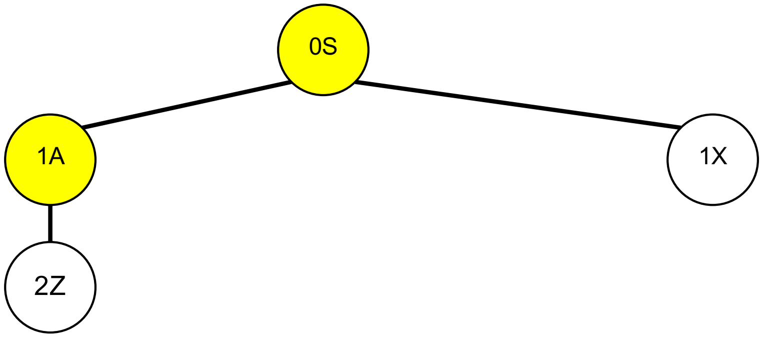


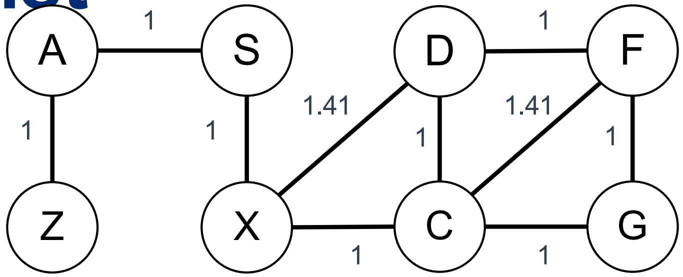
Path costs

SLD Heuristic



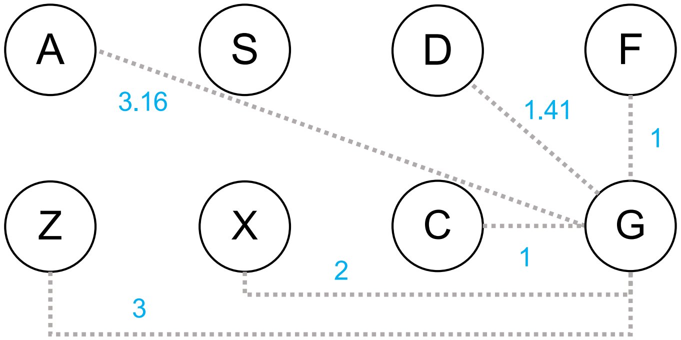
# Branch and Bound + Extended List



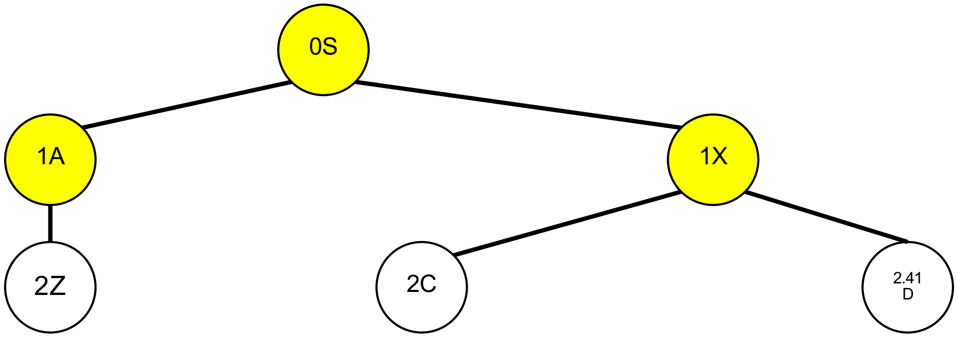


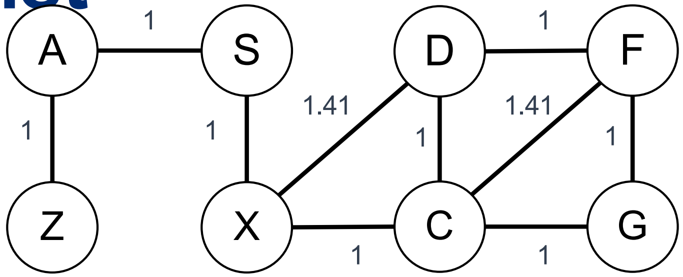
Path costs

SLD Heuristic



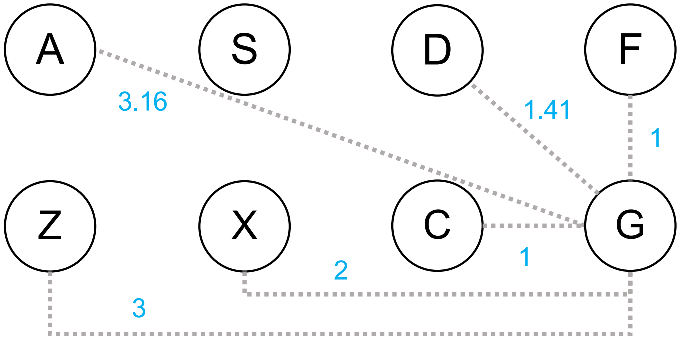
# Branch and Bound + Extended List



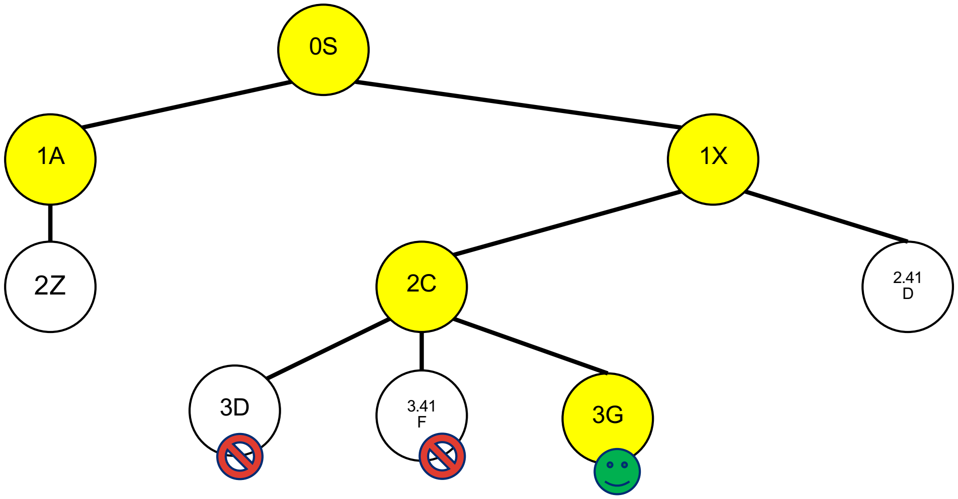


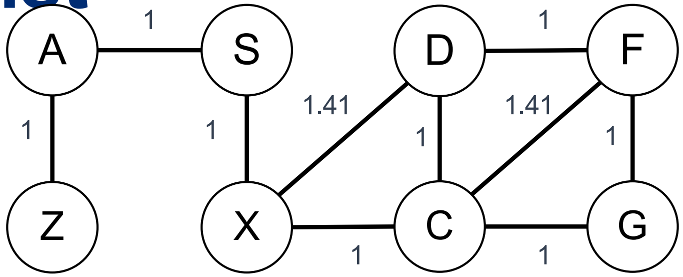
Path costs

SLD Heuristic



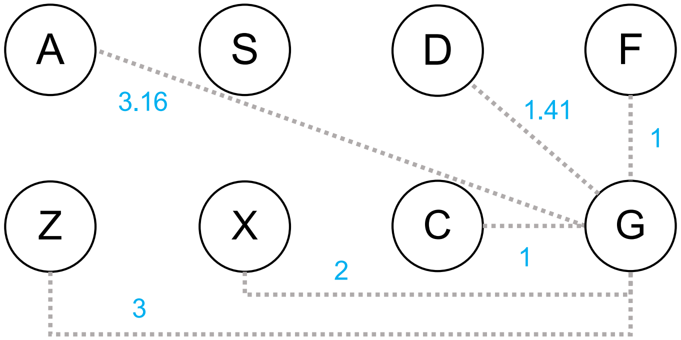
# Branch and Bound + Extended List



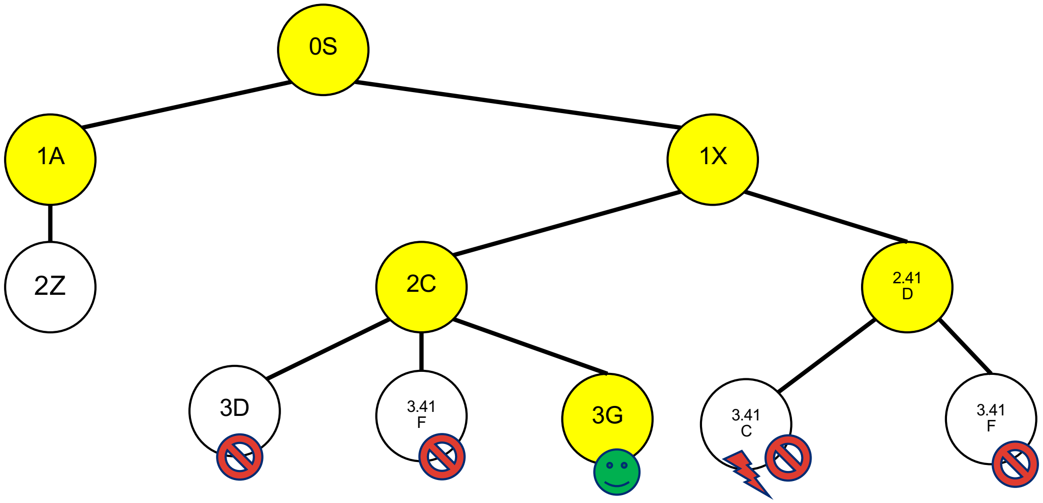


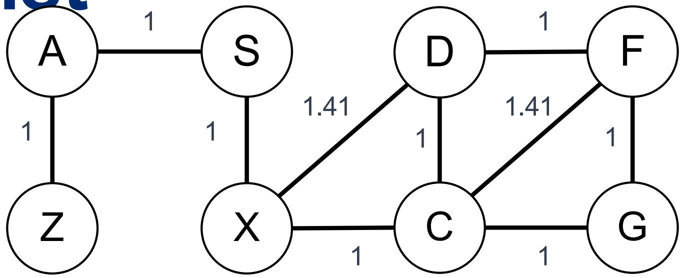
Path costs

SLD Heuristic



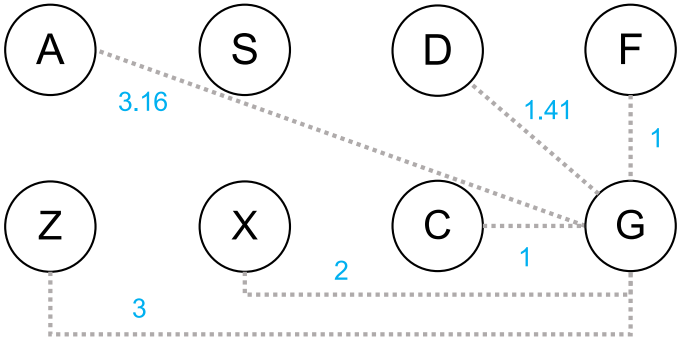
# Branch and Bound + Extended List





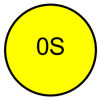
Path costs

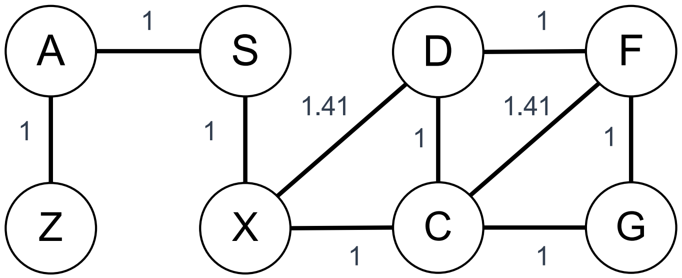
SLD Heuristic



⚡ Stopped because we’ve extended before

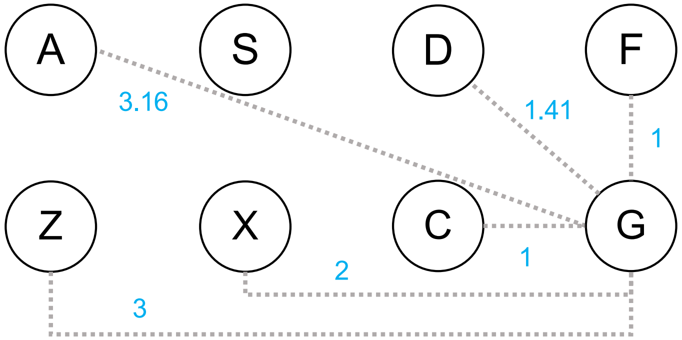
# A*



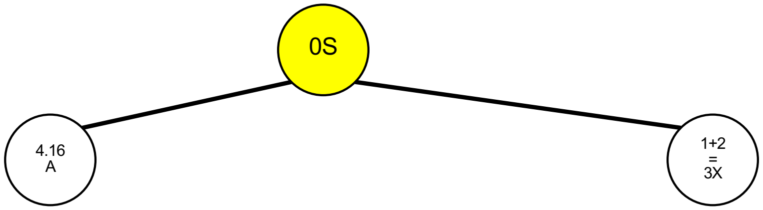


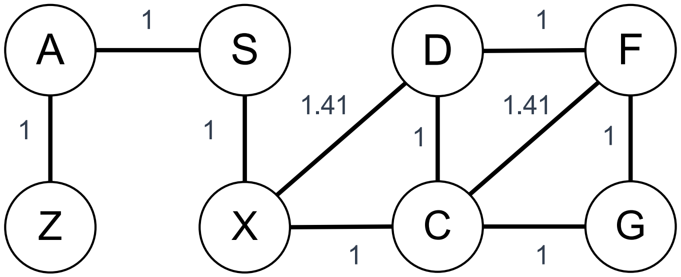
Path costs

SLD Heuristic



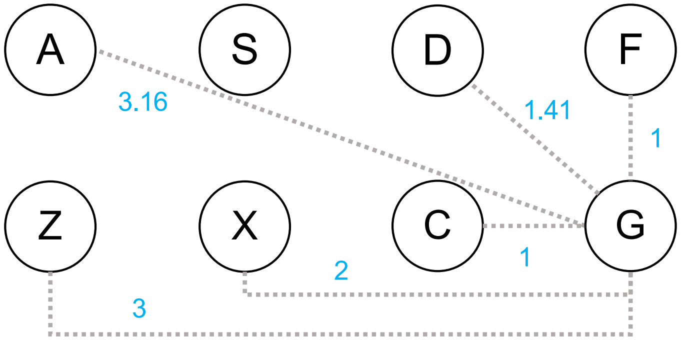
# A*



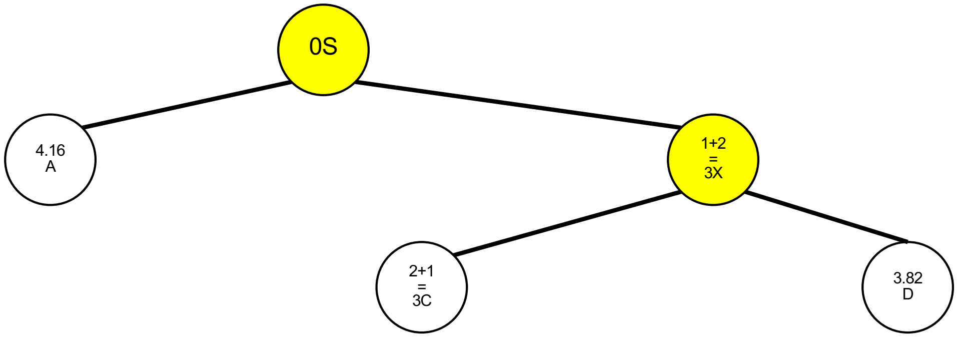


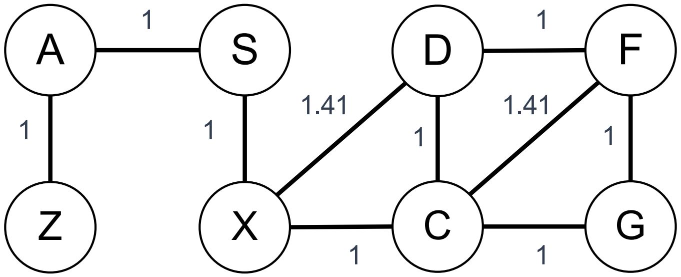
Path costs

SLD Heuristic



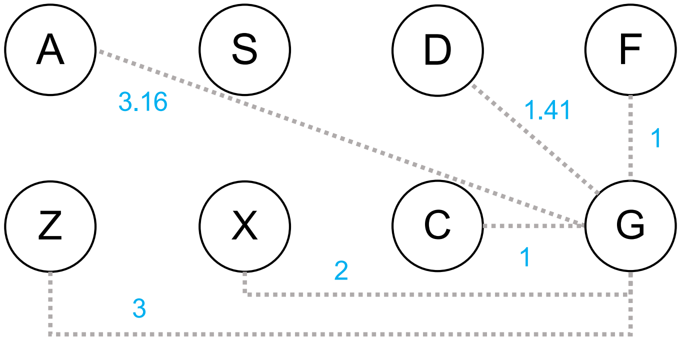
# A*



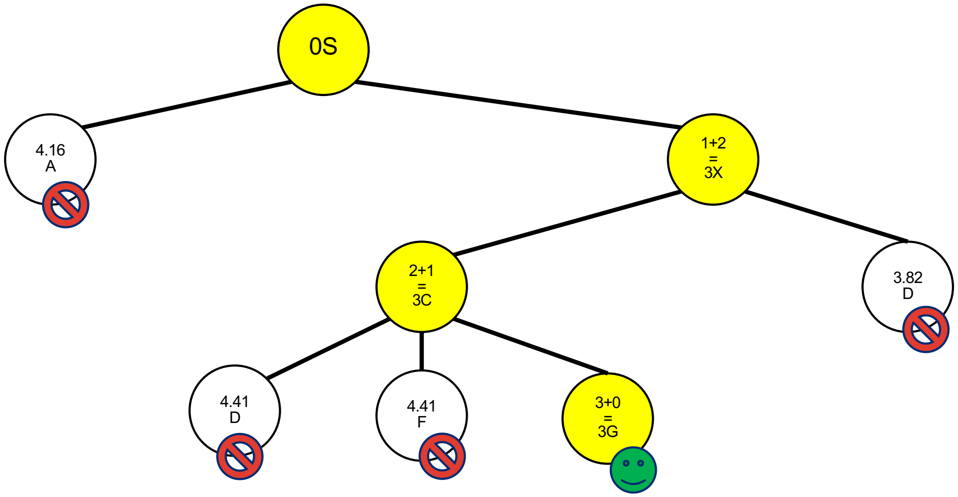


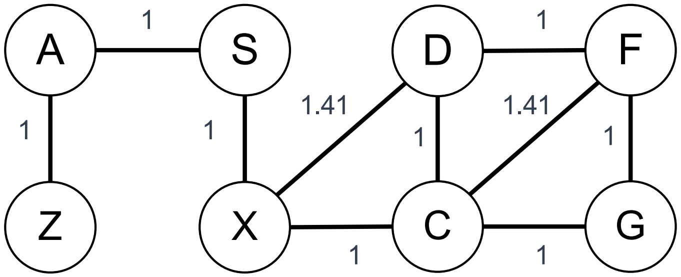
Path costs

SLD Heuristic



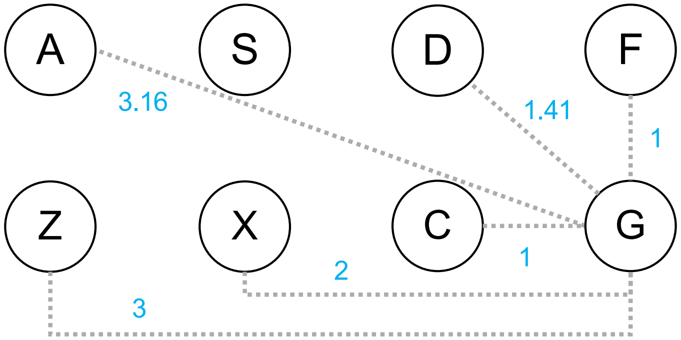
# A*



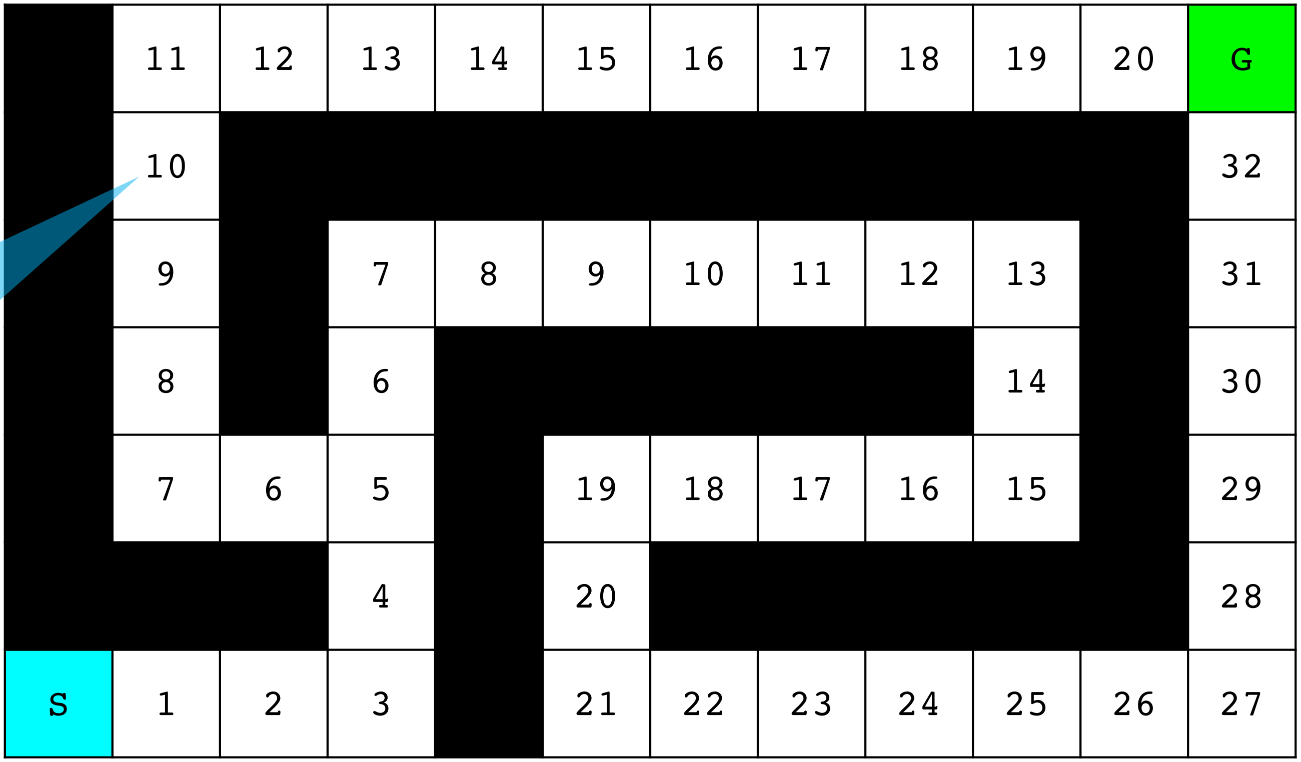


Path costs

SLD Heuristic



# Maze using A*



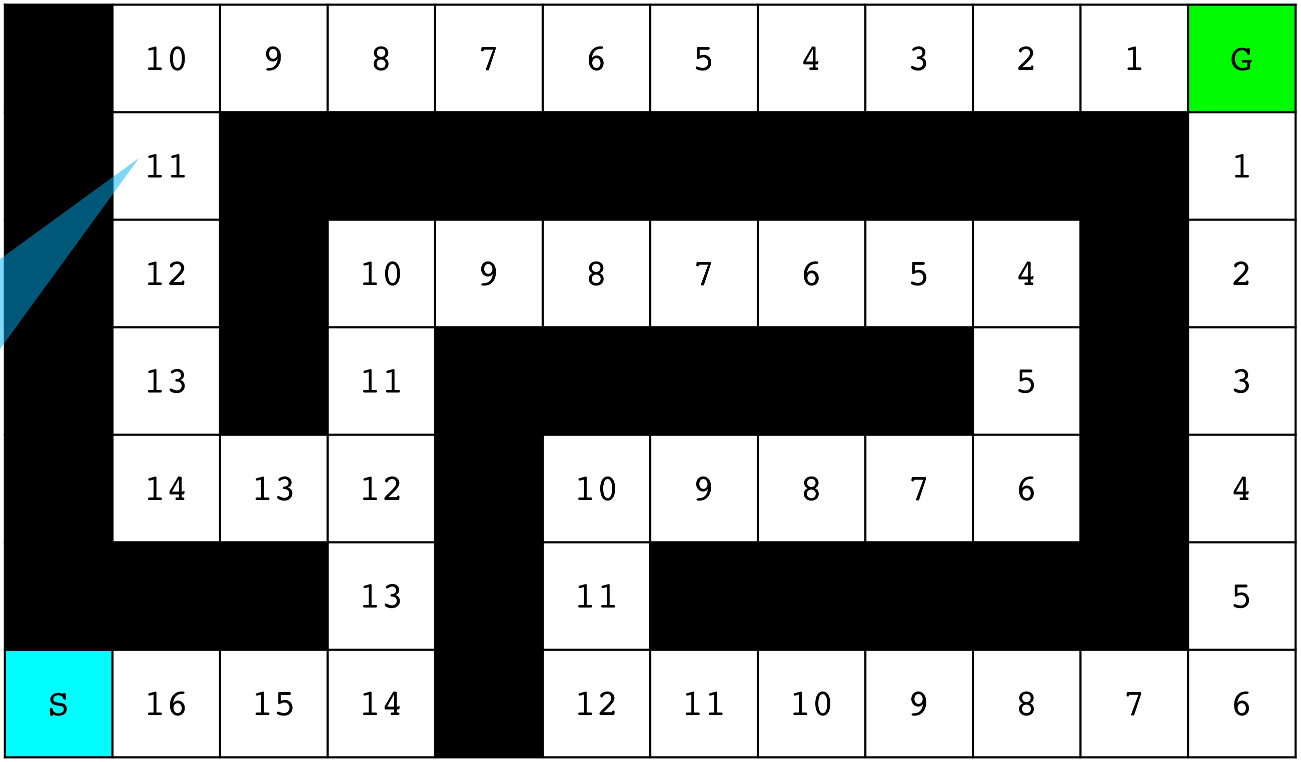
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ■ | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | G |
| ■ | 10 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | 32 |
| ■ | 9 | ■ | 7 | 8 | 9 | 10 | 11 | 12 | 13 | ■ | 31 |
| ■ | 8 | ■ | 6 | ■ | ■ | ■ | ■ | ■ | 14 | ■ | 30 |
| ■ | 7 | 6 | 5 | ■ | 19 | 18 | 17 | 16 | 15 | ■ | 29 |
| ■ | ■ | ■ | 4 | ■ | 20 | ■ | ■ | ■ | ■ | ■ | 28 |
| S | 1 | 2 | 3 | ■ | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

accumulated path cost

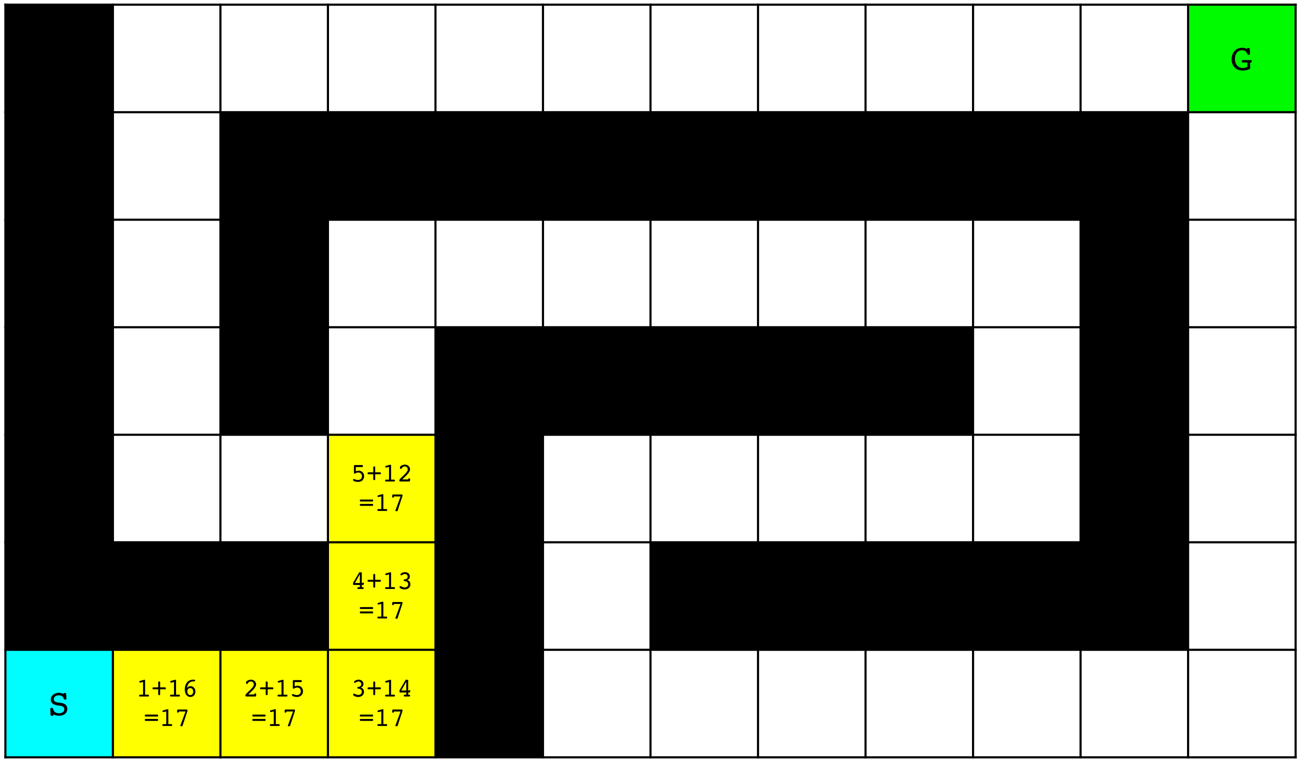
each step cost a point

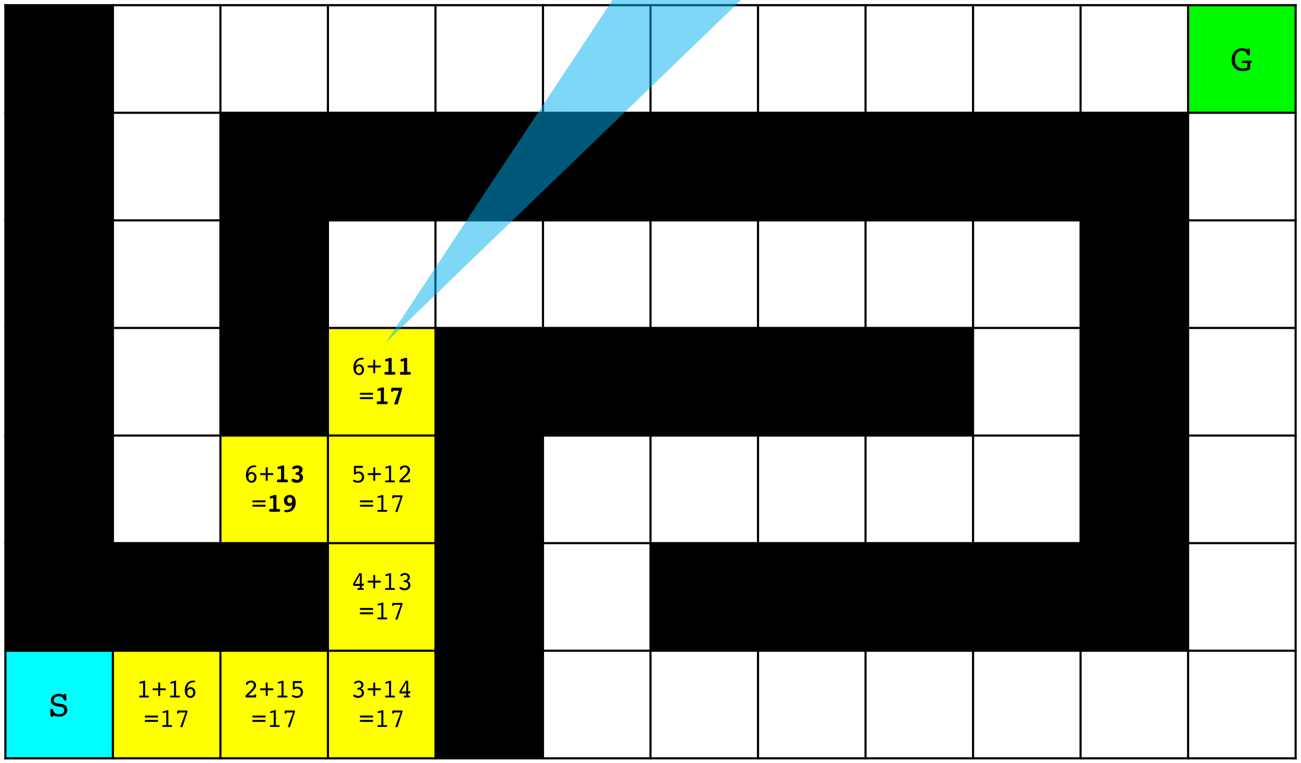
Heuristic: Manhattan Distance (not SLD)

Manhattan Distance between (x1, y1) and (x2, y2) = abs(x1 – x2) + abs(y1 – y2)



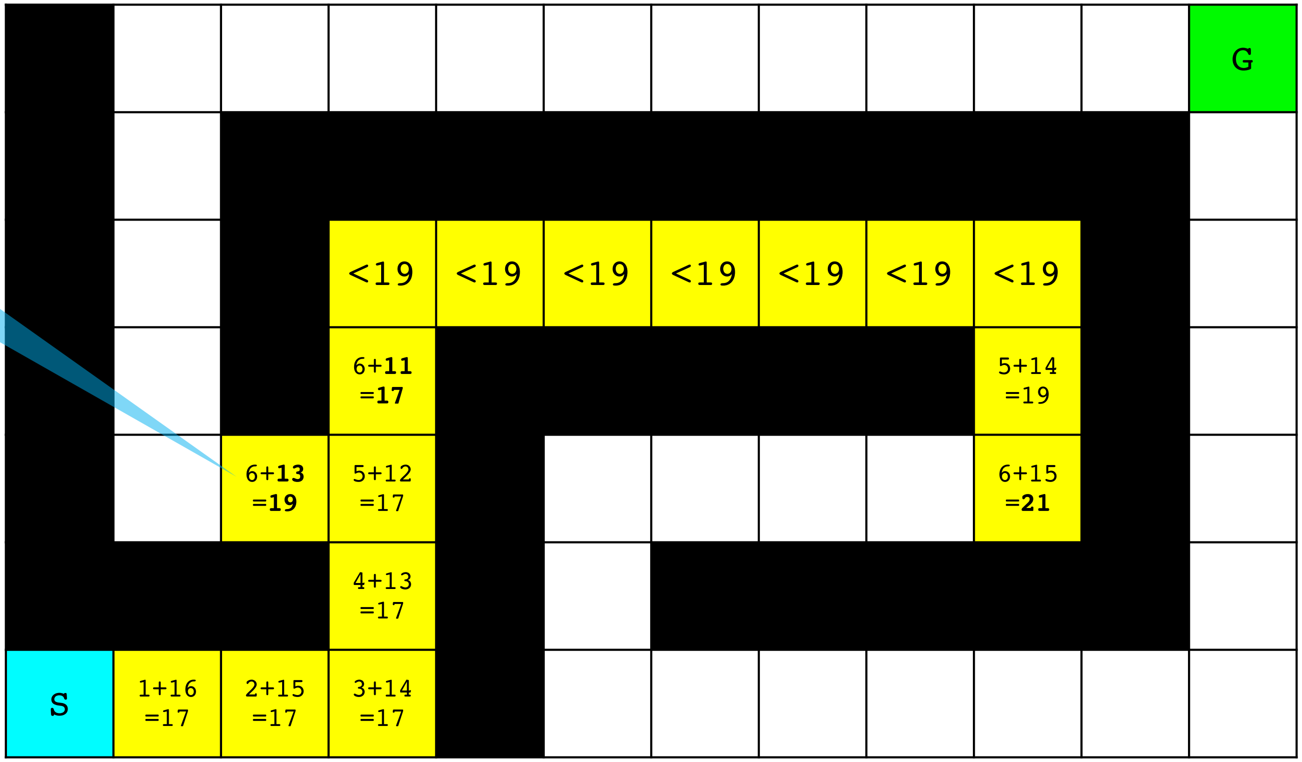
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ■ | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | G |
| ■ | 11 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | 1 |
| ■ | 12 | ■ | 10 | 9 | 8 | 7 | 6 | 5 | 4 | ■ | 2 |
| ■ | 13 | ■ | 11 | ■ | ■ | ■ | ■ | ■ | 5 | ■ | 3 |
| ■ | 14 | 13 | 12 | ■ | 10 | 9 | 8 | 7 | 6 | ■ | 4 |
| ■ | ■ | ■ | 13 | ■ | 11 | ■ | ■ | ■ | ■ | ■ | 5 |
| S | 16 | 15 | 14 | ■ | 12 | 11 | 10 | 9 | 8 | 7 | 6 |


G
5+12
=17
4+13
=17
S
1+16
=17
2+15
=17
3+14
=17

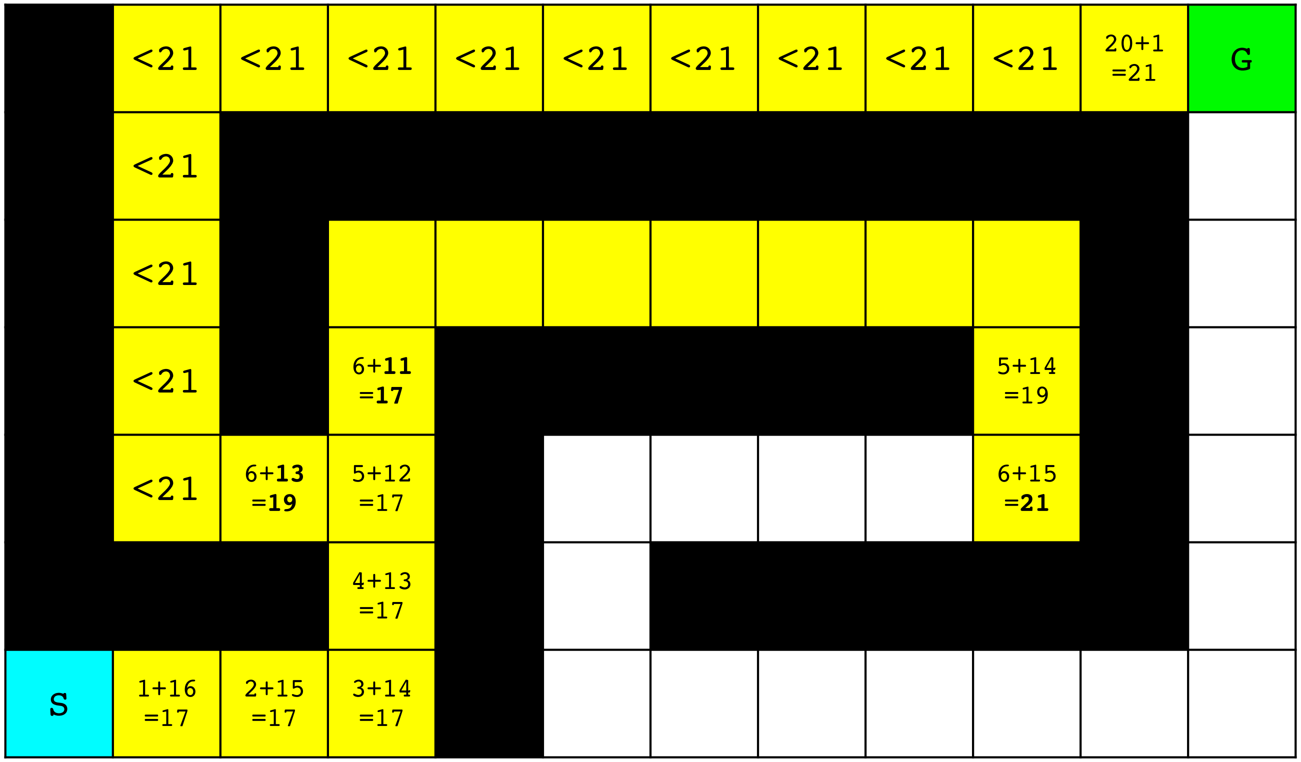


Notice how the longer path is chosen here due to the heuristics and the AI will continue down the longer path until it realizes the path cost is getting too high

This path now looks cheaper



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ■ | | | | | | | | | | | G |
| ■ | | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| ■ | | ■ | <19 | <19 | <19 | <19 | <19 | <19 | <19 | ■ | |
| ■ | | ■ | 6+**11**<br>=**17** | ■ | ■ | ■ | ■ | ■ | 5+14<br>=19 | ■ | |
| ■ | | 6+**13**<br>=**19** | 5+12<br>=17 | ■ | | | | | 6+15<br>=**21** | ■ | |
| ■ | ■ | ■ | 4+13<br>=17 | ■ | | ■ | ■ | ■ | ■ | ■ | |
| S | 1+16<br>=17 | 2+15<br>=17 | 3+14<br>=17 | ■ | | | | | | | |



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ■ | <21 | <21 | <21 | <21 | <21 | <21 | <21 | <21 | <21 | 20+1<br>=21 | G |
| ■ | <21 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| ■ | <21 | ■ | | | | | | | | ■ | |
| ■ | <21 | ■ | 6+**11**<br>=**17** | ■ | ■ | ■ | ■ | ■ | 5+14<br>=19 | ■ | |
| ■ | <21 | 6+**13**<br>=**19** | 5+12<br>=17 | ■ | | | | | 6+15<br>=**21** | ■ | |
| ■ | ■ | ■ | 4+13<br>=17 | ■ | | ■ | ■ | ■ | ■ | ■ | |
| S | 1+16<br>=17 | 2+15<br>=17 | 3+14<br>=17 | ■ | | | | | | | |

JOHNS HOPKINS

APPLIED PHYSICS LABORATORY