AI Problem Solving by Searching
&
Robot Path Planning

Lecture by Margrit Betke and Mahir Patel

Reading: Russell and Norvig, Chapter 3
Winston

What you need to do to specify an agent-based AI Problem:

- Initial state that the agent starts in
- Actions available to agent
- Transition model & state space: Path through state space = sequence of states = sequence of actions
- Goal test
- Path cost (e.g. sum of step costs)

“AI Toy Problems” useful for learning concepts

- 8 Puzzle: Start state:

7	2	4
5		6
8	3	1

Goal state:

	1	2
3	4	5
6	7	8

- 8 Queens Problem: 8x8 chess board

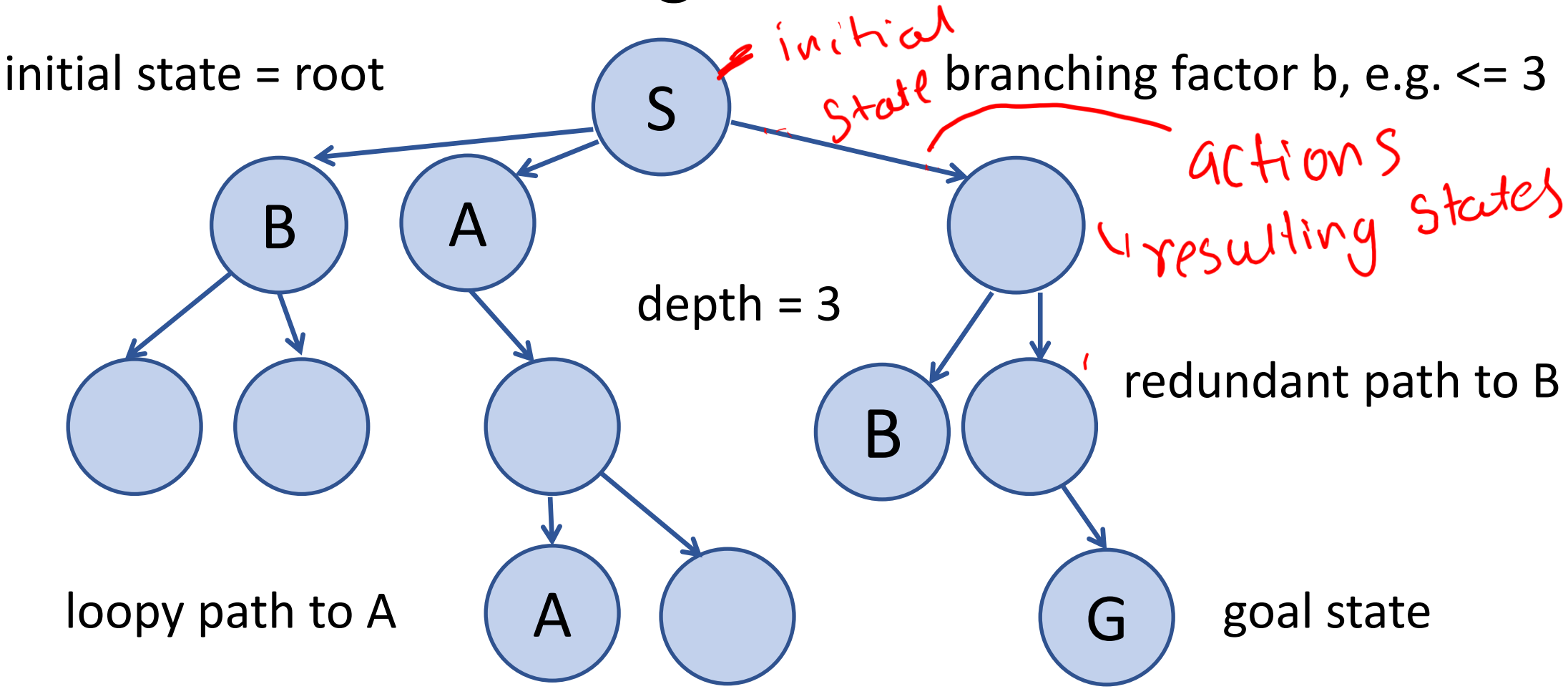
Place queens so that none attacks any other queen.

“Attack state” = 2 queens are on the same row, column or diagonal.

Problem Solving by Searching: “Real-World Problems”

- Route finding problems
- Touring problems, e.g., Traveling Salesperson (efficient path for visiting every city once)
- VLSI layout
- Robot navigation
- Drone navigation
- Protein design
- Cancer detection

Solution: Sequence of Actions = Path through a Search Tree



Evaluation of Search Algorithm Performance

- Completeness: Is it guaranteed to find a solution?
- Optimality:
  - Shortest path?
  - Lowest cost?
- Time Complexity
- Space Complexity

Path-based Search Algorithms

Task: Find shortest path through a graph

Applications: Games, robot path planning

Lots of algorithms!

Path-based Search Strategies

- Exhaustive search: Explore all paths

Exhaustive Search

- Winston calls this strategy the "British Museum Procedure"
- Find all paths and select the shortest
- Search tree: root level 1 node

  2^{nd} level b nodes

  3^{rd} level b*b nodes

  4^{th} level b*b*b nodes

  ....

  d^{th} level b^d nodes

If b=10, d=10:

10^{10} = 10 billion paths

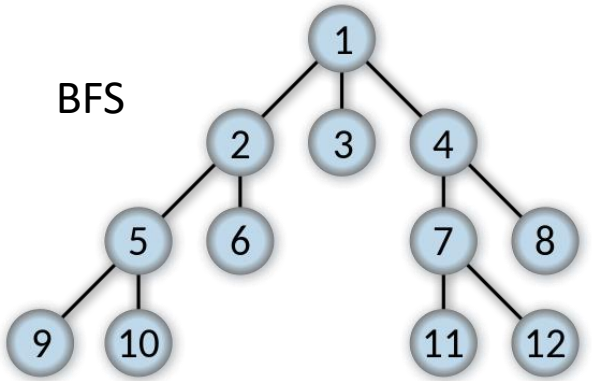
Too many to test!

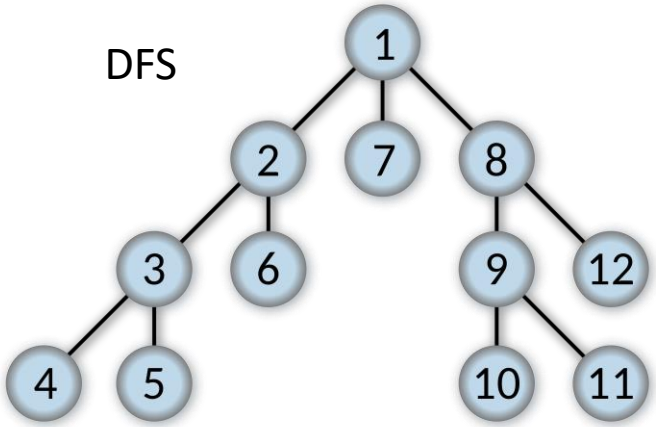
Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node

Cost per edge: 1

BFS & DFS typically covered in your previous classes





Illustrations: Drichel, Wikipedia

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Iterative Deepening: Gradually increasing depth d = 0, 1, 2, …
- Bidirectional search: Search from start S and goal G nodes, hoping to meet

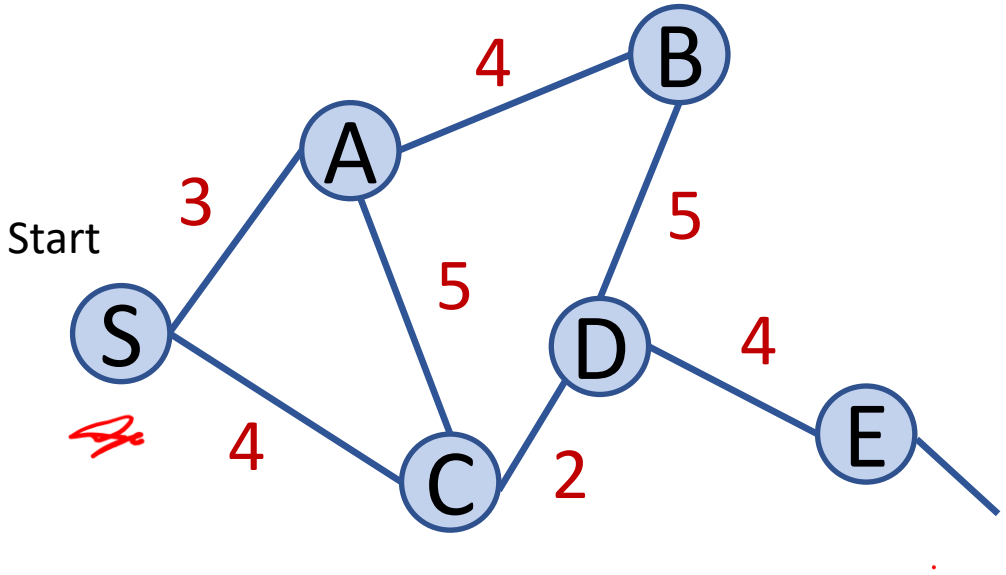
Path-based Search Strategies

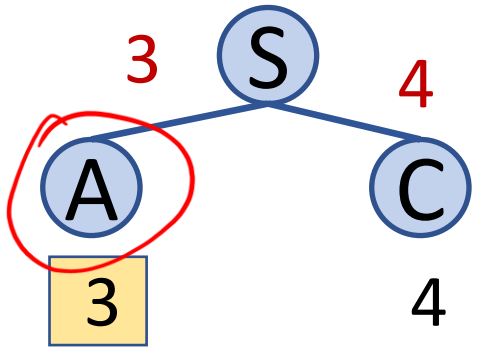
- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost

  Used in AI when graphs are extremely large or infinite

Example for Uniform Cost Search

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost

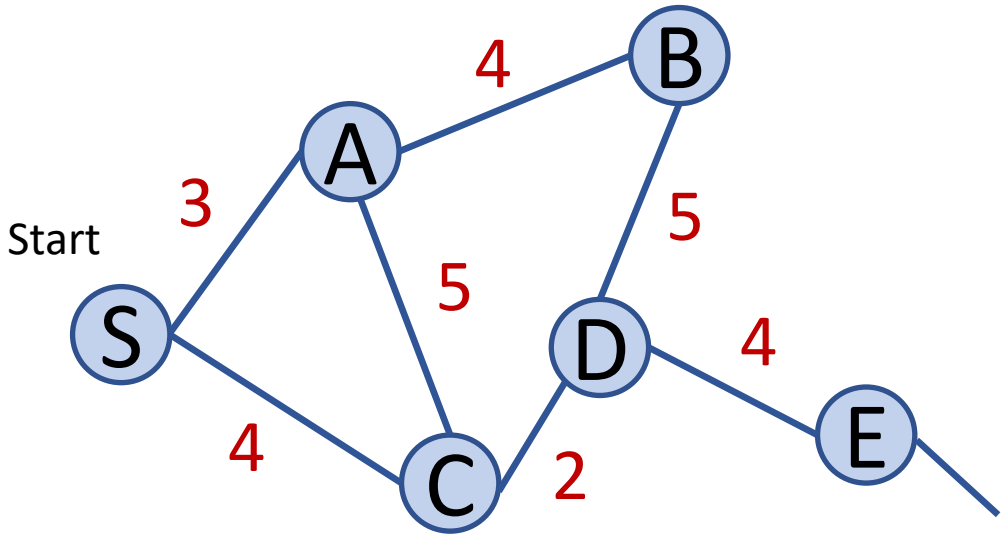


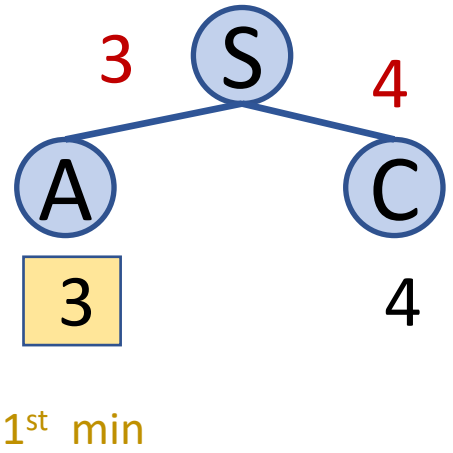


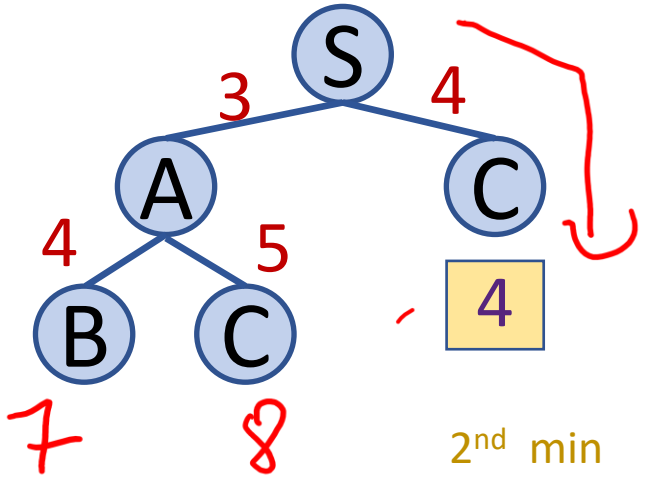
1st min

Example for Uniform Cost Search

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost

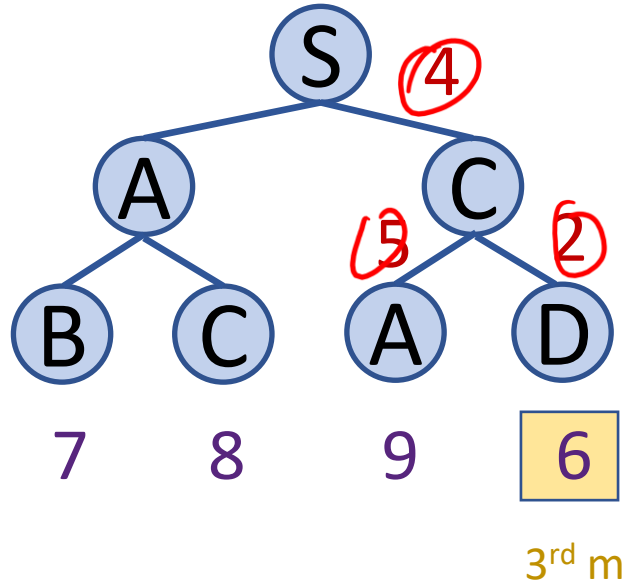


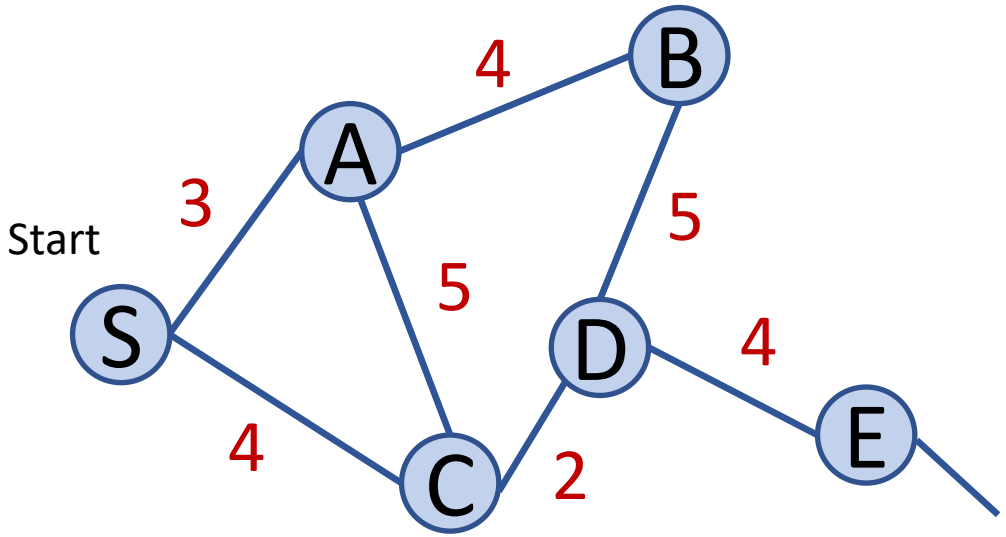


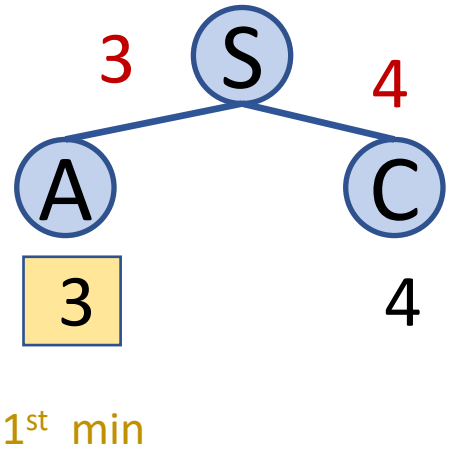


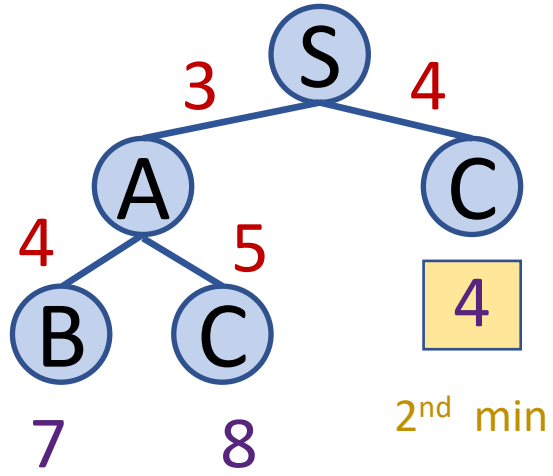
Example for Uniform Cost Search

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost







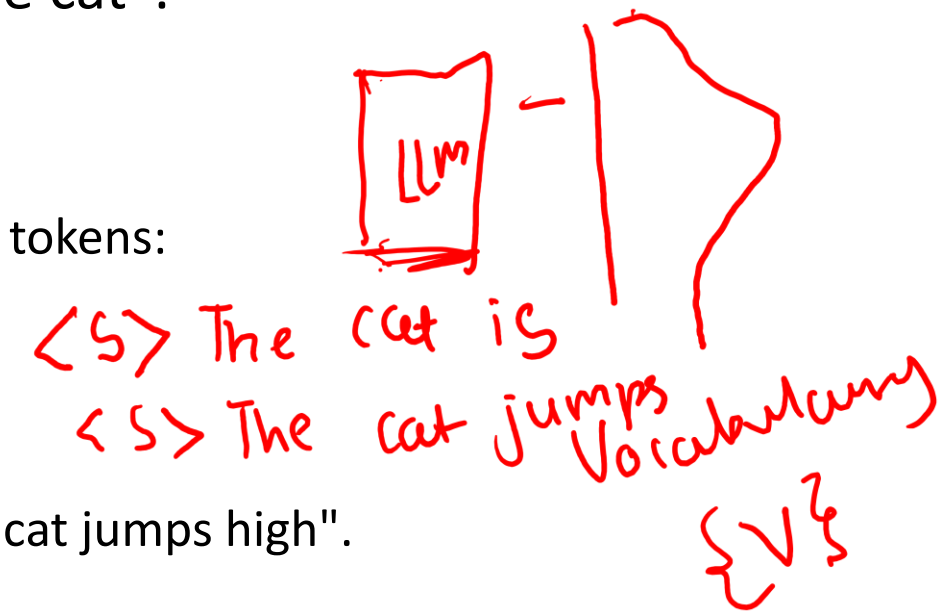


Beam Search

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number w of best nodes at each level, the beam width w
  - Same as BFS with w = infinite
  - The greater w is the fewer states are pruned
  - Useful in AI if BFS search tree is too large to fit in memory
  - Not guaranteed to find optimal solution

Deep Learning Application!

- Beam Search isused in modern generative LLMs to generate better responses.
- At every decoding step, k possible words with highest likelihood are selected.
- Imagine generating a sentence starting with "The cat":
  1. **Initial beam (k=2)**:
     1. Beam 1: "The cat is" (prob: 0.5)
     2. Beam 2: "The cat jumps" (prob: 0.45)
  2. **Next step**: Expand each beam by appending possible tokens:
     1. "The cat is hungry" (prob: 0.4)
     2. "The cat is happy" (prob: 0.3)
     3. "The cat jumps high" (prob: 0.35)
     4. "The cat jumps down" (prob: 0.25)
  3. **Prune to top k=2**: Keep "The cat is hungry" and "The cat jumps high".
  4. **Repeat until termination**.



Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit (Optimal? Complete?)

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Progressive Deepening (also called Iterative Deepening):

  Gradually increasing depth d = 0, 1, 2, …

  We have seen this algorithm used for adversarial game playing.

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Progressive Deepening (also called Iterative Deepening): (Optimal? Complete?)

  Gradually increasing depth d = 0, 1, 2, …

  We have seen this algorithm used for adversarial game playing.

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost (same as BFS with cost =1)
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Iterative Deepening: Gradually increasing depth d = 0, 1, 2, …
- Bidirectional search: Search from start S and goal G nodes, hoping to meet

Path-based Search Strategies

- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost (same as BFS with cost =1)
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Iterative Deepening: Gradually increasing depth d = 0, 1, 2, …
- Bidirectional search: Search from start S and goal G nodes, hoping to meet (Optimal? Complete?)

Path-based Search Strategies

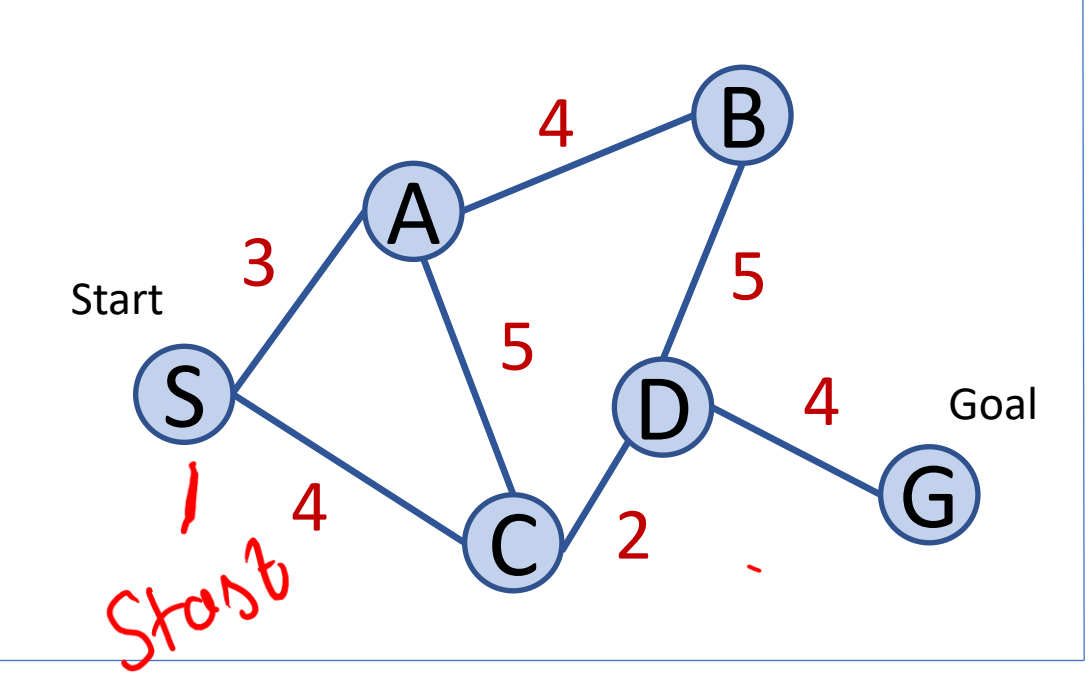
- Exhaustive search: Explore all paths
- Breadth-first search (BFS): Expand shallowest node
- Depth-first search (DFS): Expand deepest node
- Uniform cost search: Expand node with smallest cost (same as BFS with cost =1)
- Beam search: BFS but only keep limited number of best nodes
- Depth-limited search: Predetermined depth limit
- Iterative Deepening: Gradually increasing depth d = 0, 1, 2, …
- Bidirectional search: Search from start S and goal G nodes, hoping to meet
- Greedy search = branch & bound search
- Greedy search with pruning
- A* = Greedy search with pruning and underestimates of remaining distance

Greedy Search = Branch & Bound Search

Phase 1: Extend shortest partial path until goal is reached.

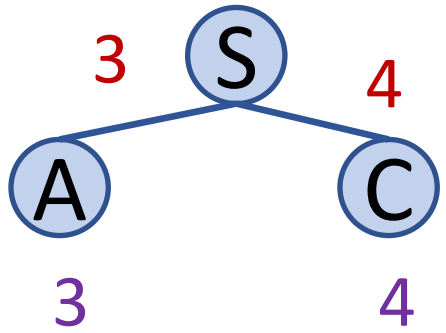
Reject loops.

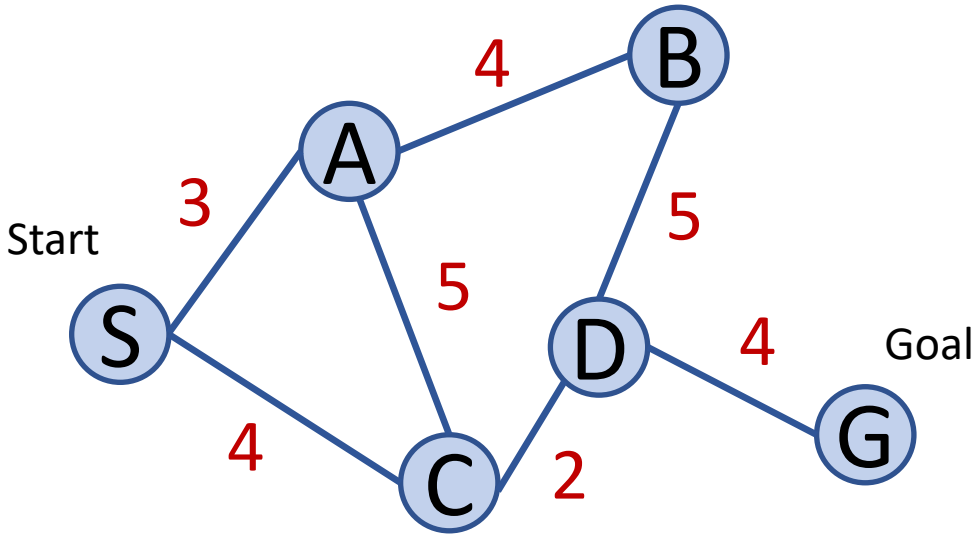
Phase 2: Extend all partial paths
until their length >= complete path to goal



Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

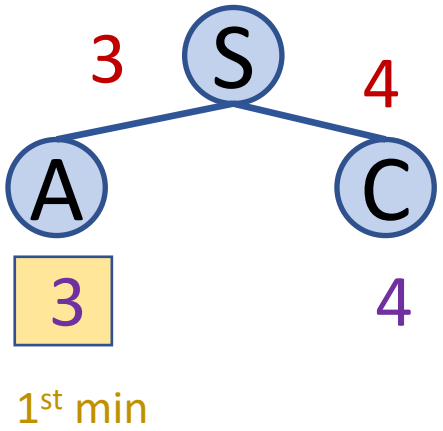
Phase 1:

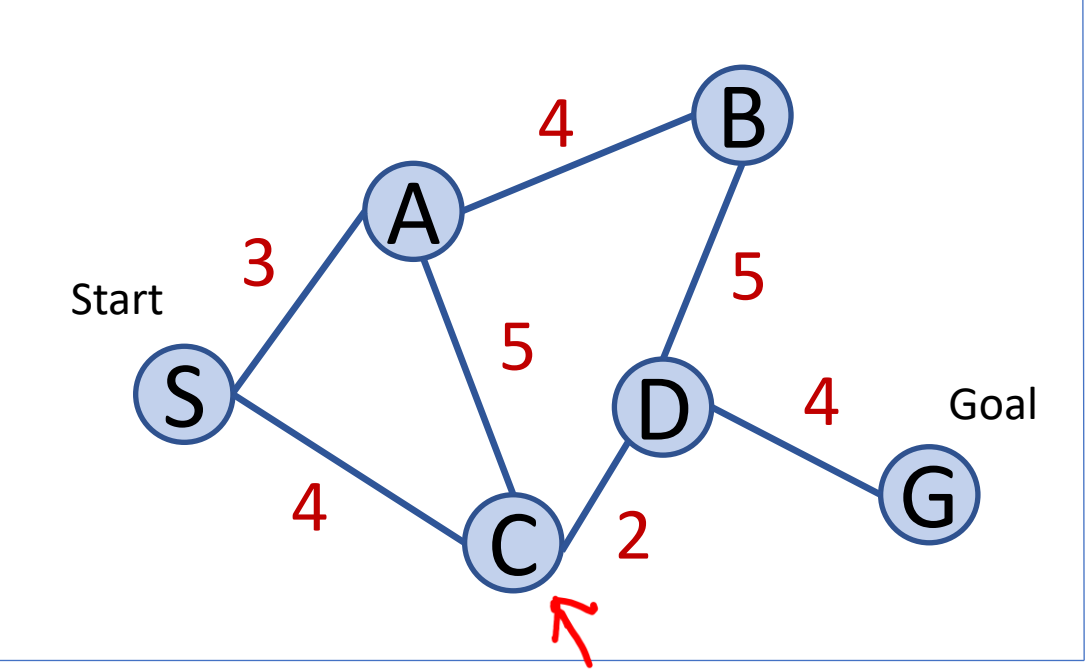




Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

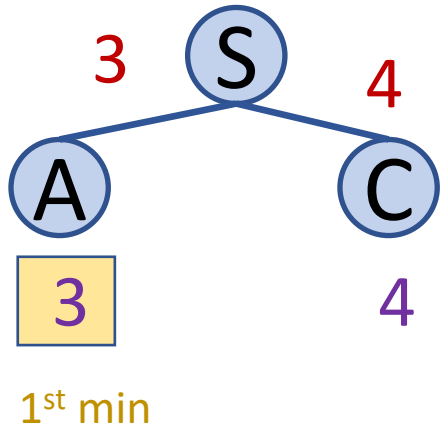
Phase 1:

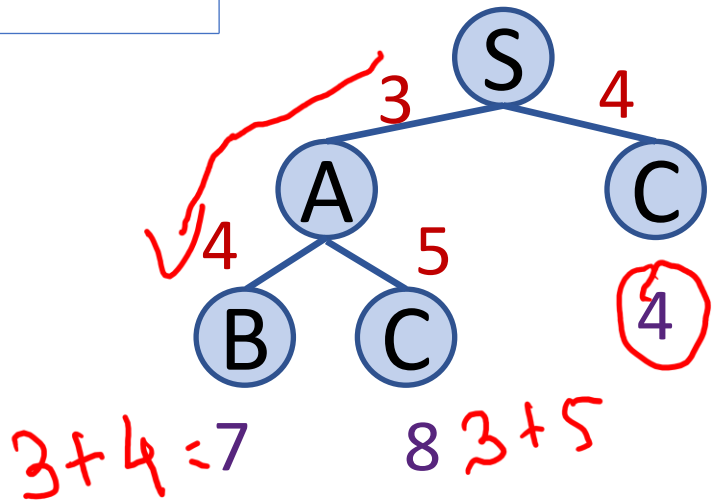


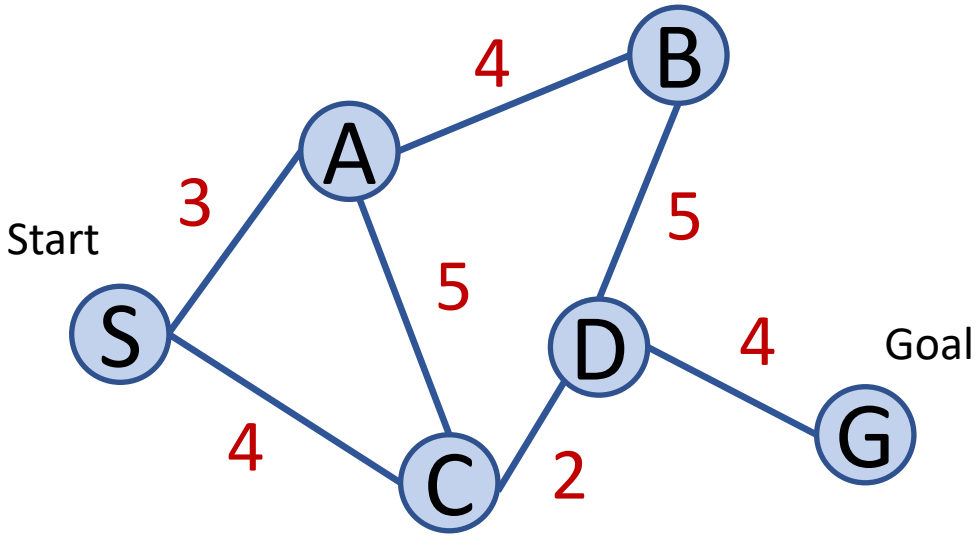


Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

Phase 1:

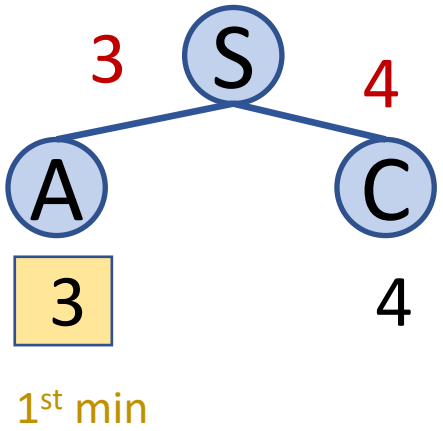


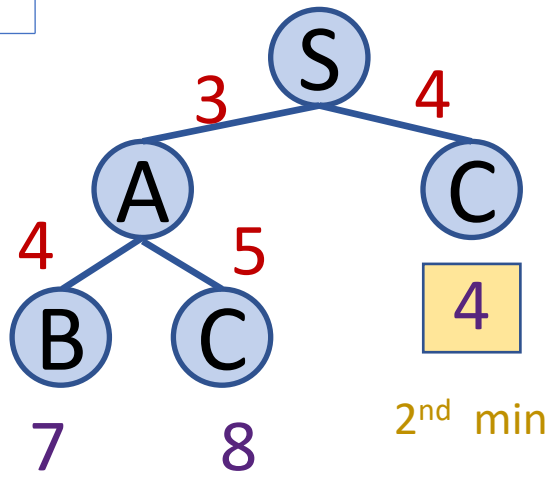


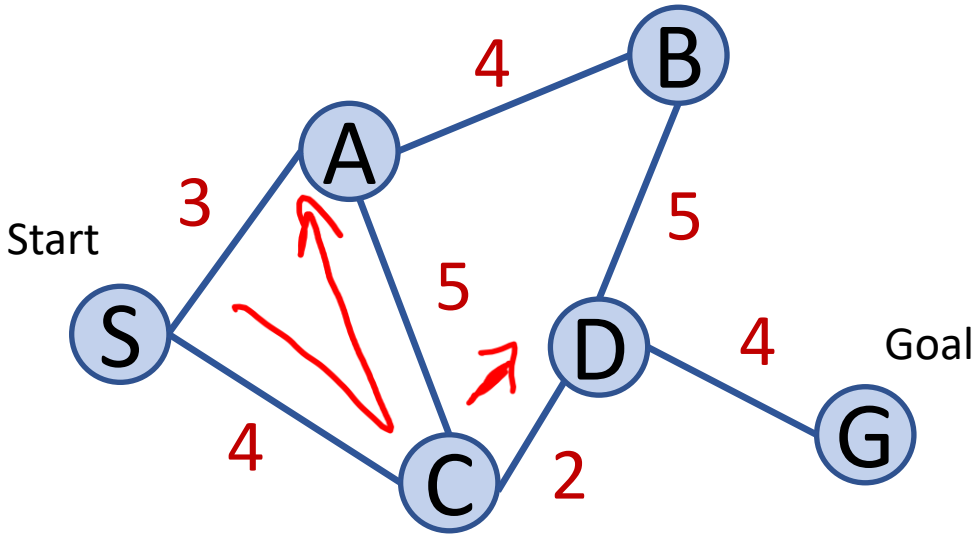


Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

Phase 1:

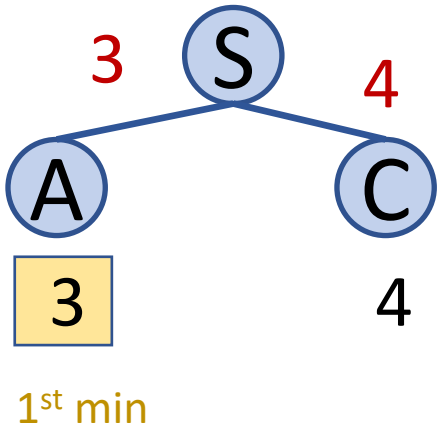


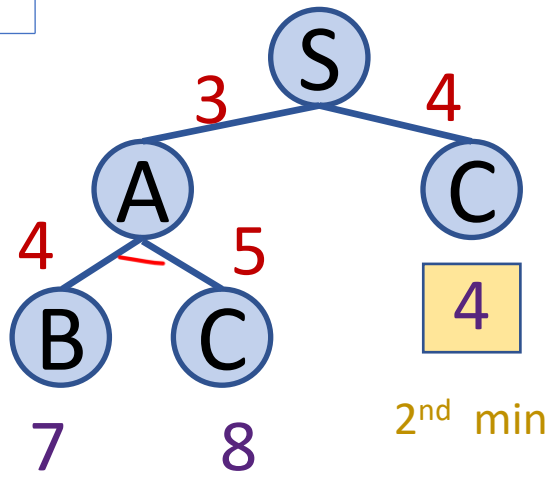


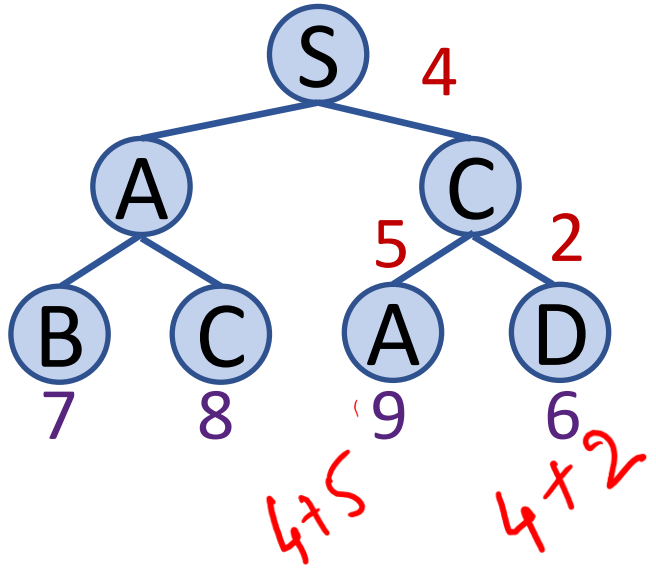


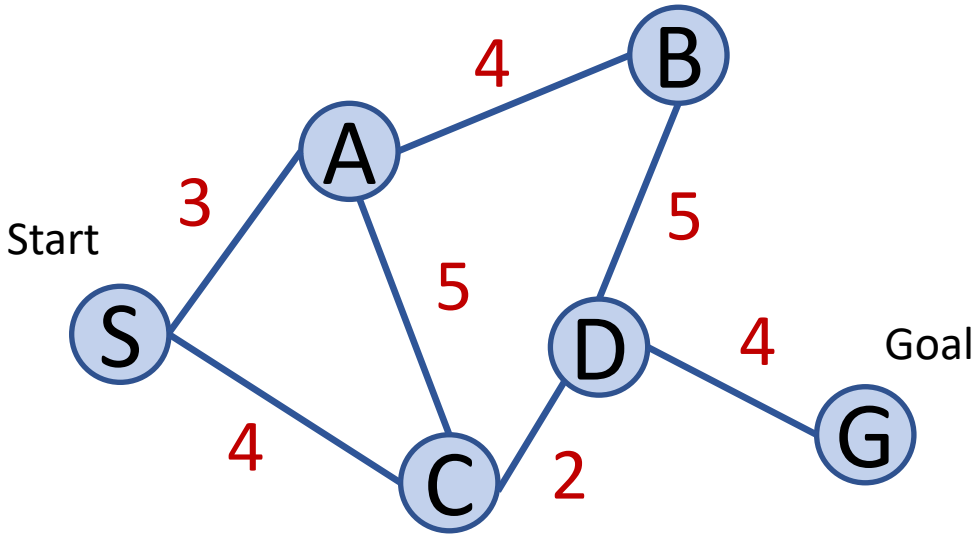
Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

Phase 1:



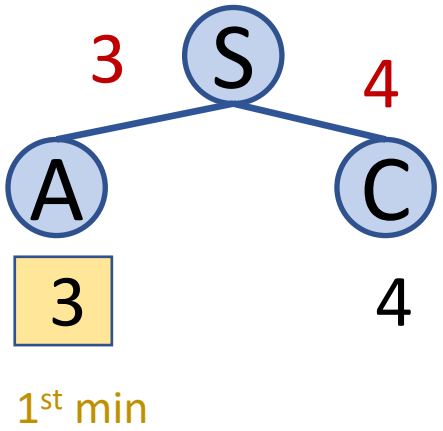


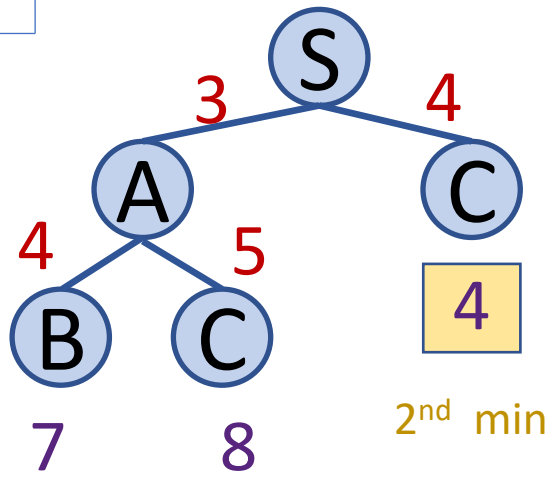


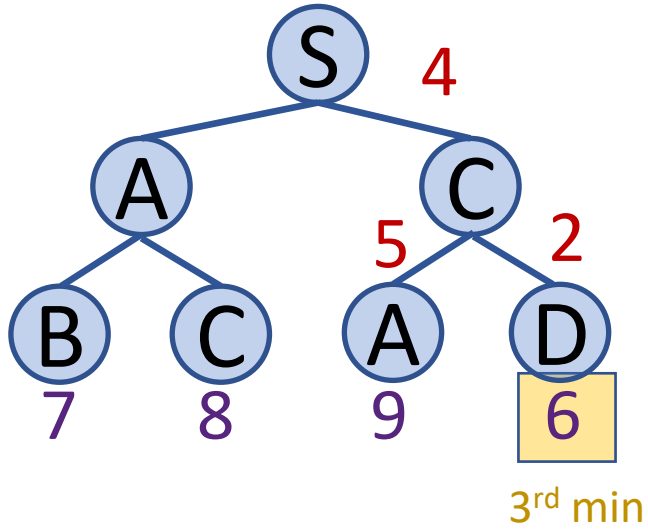


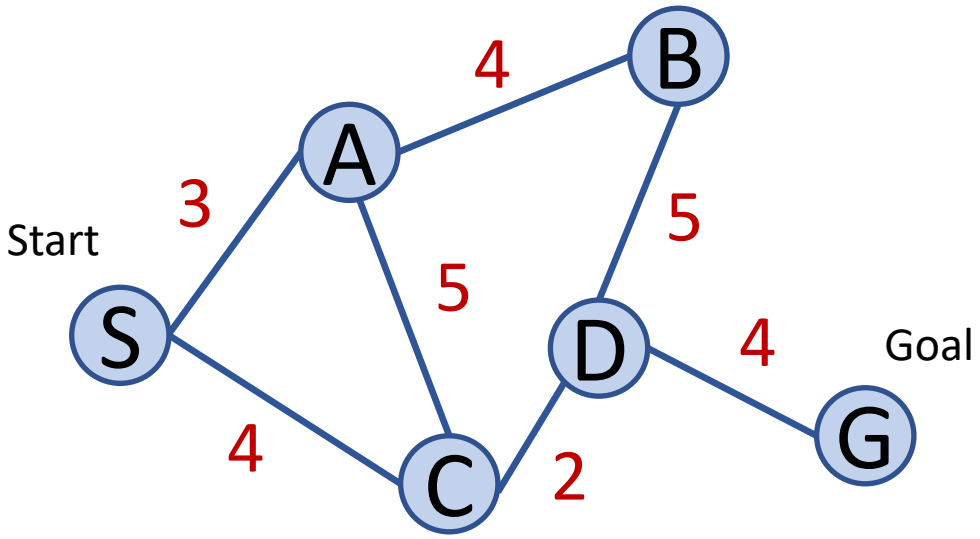
Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

Phase 1:





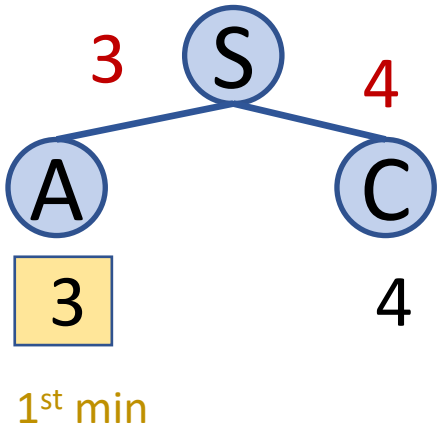


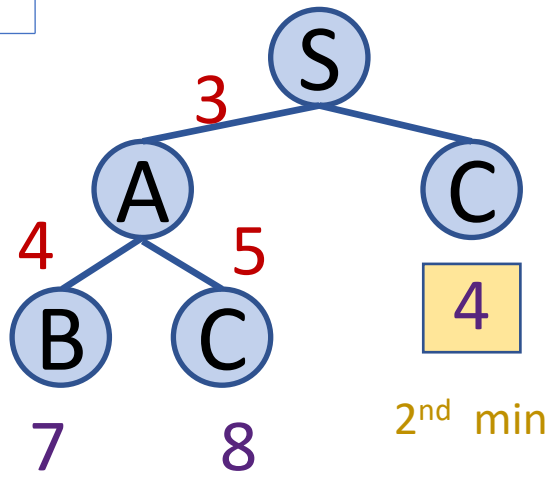


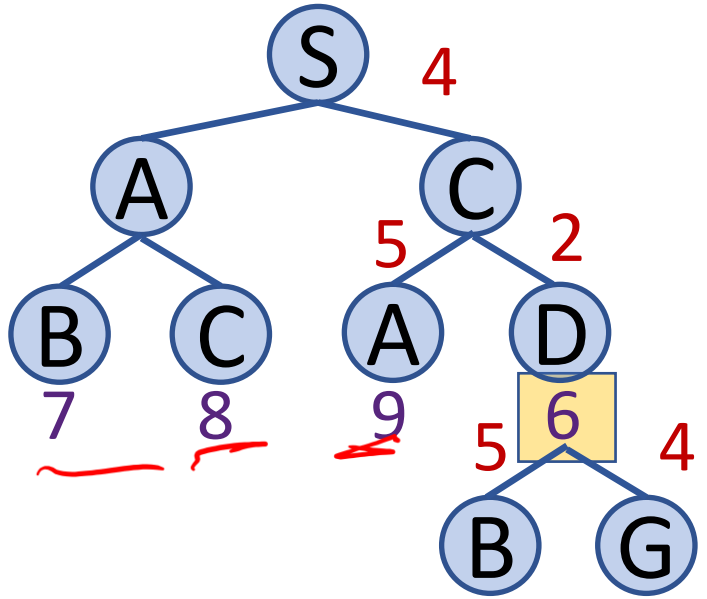
Phase 1: Extend shortest partial path until goal is reached.
Reject loops.

4+2+4 = 10

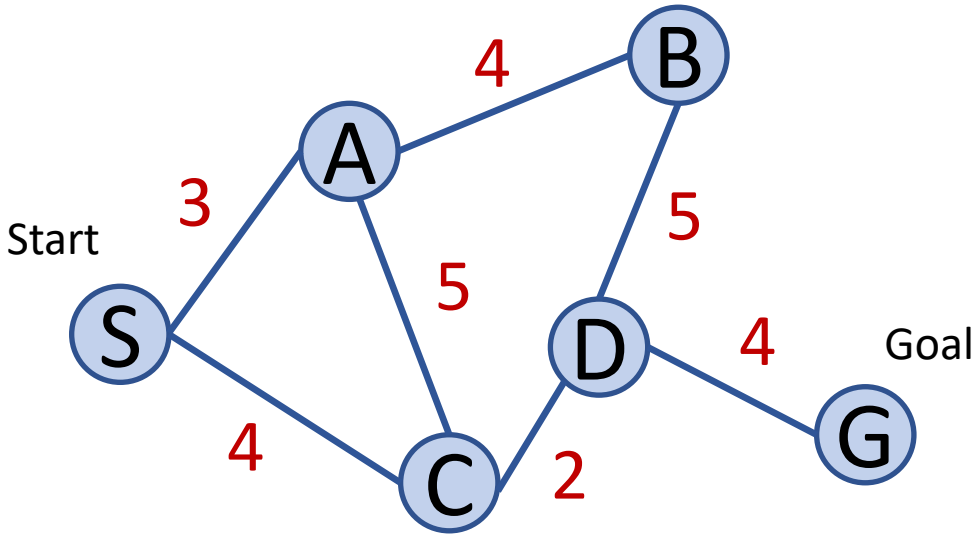
Phase 1:



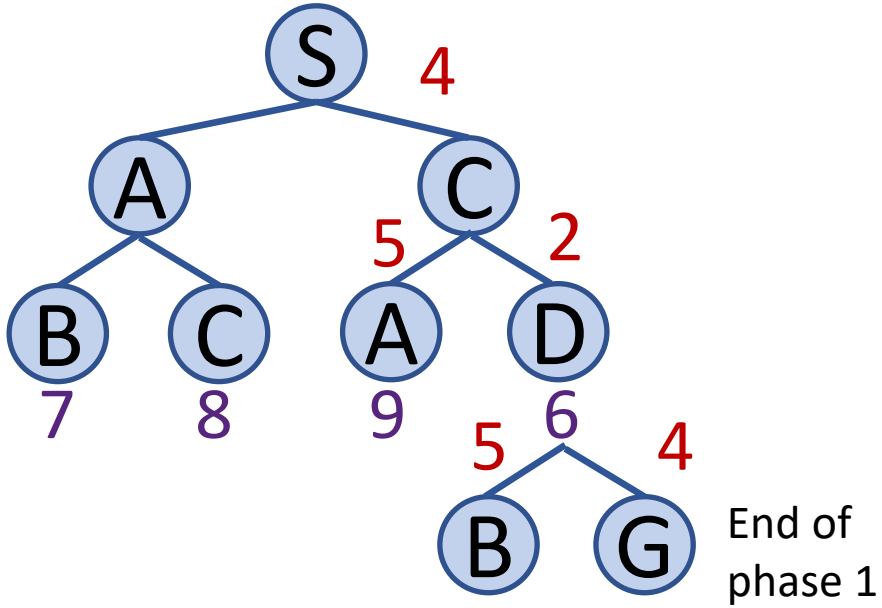




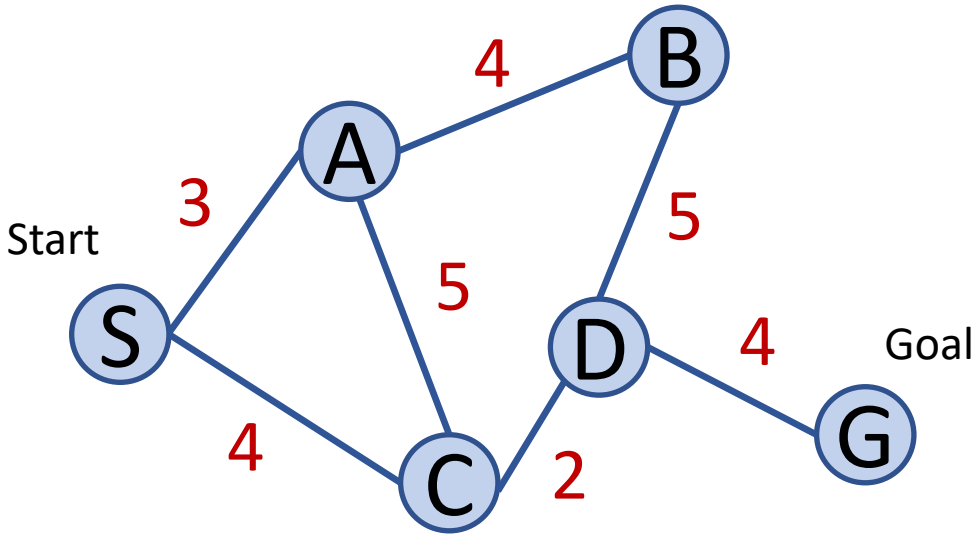
End of phase 1: G reached



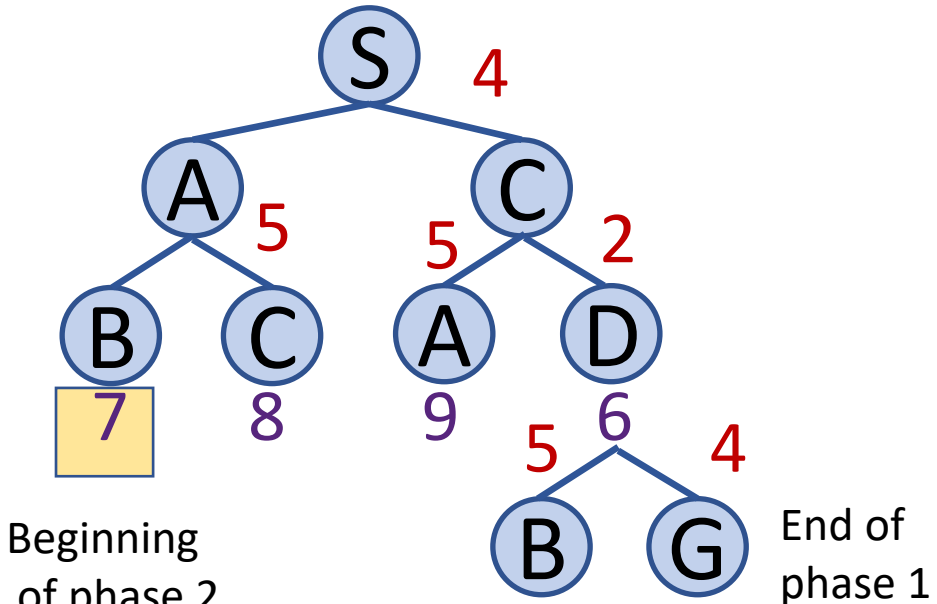
Phase 2:



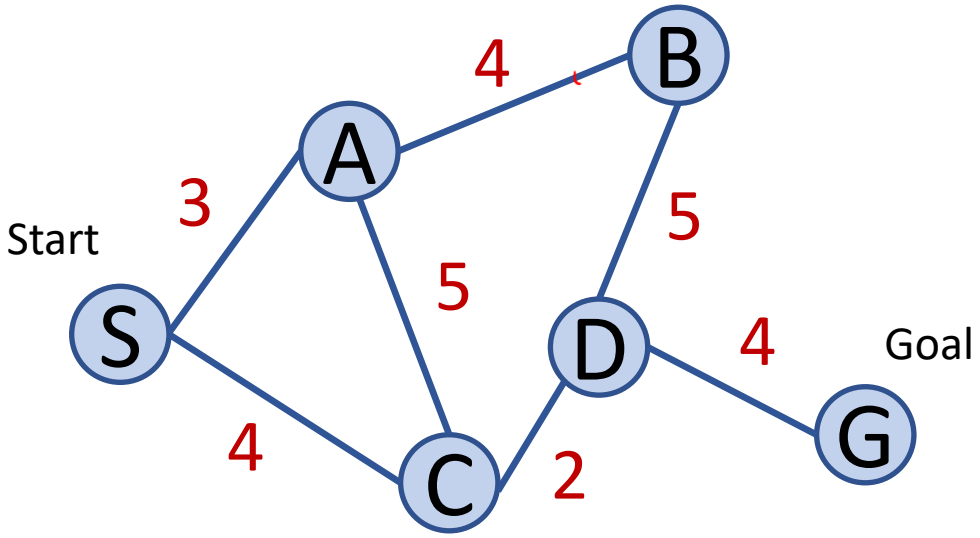
Phase 2: Extend all partial paths
until their length >= complete path to goal



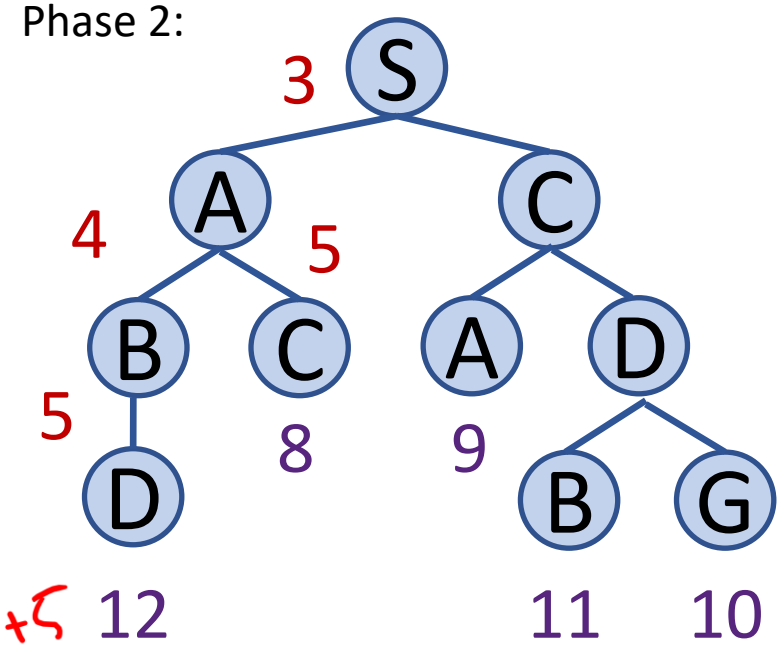
Phase 2:



Phase 2: Extend all partial paths
until their length >= complete path to goal

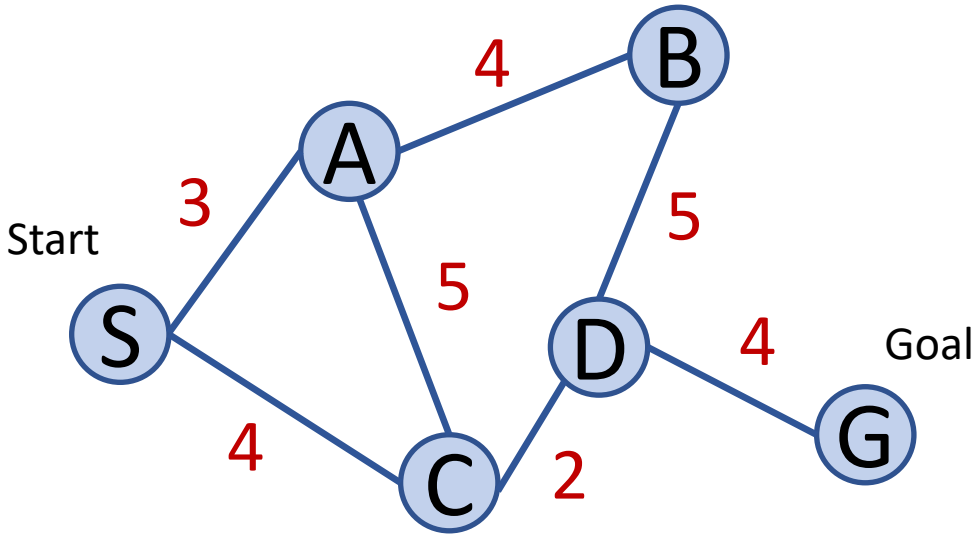


Phase 2:

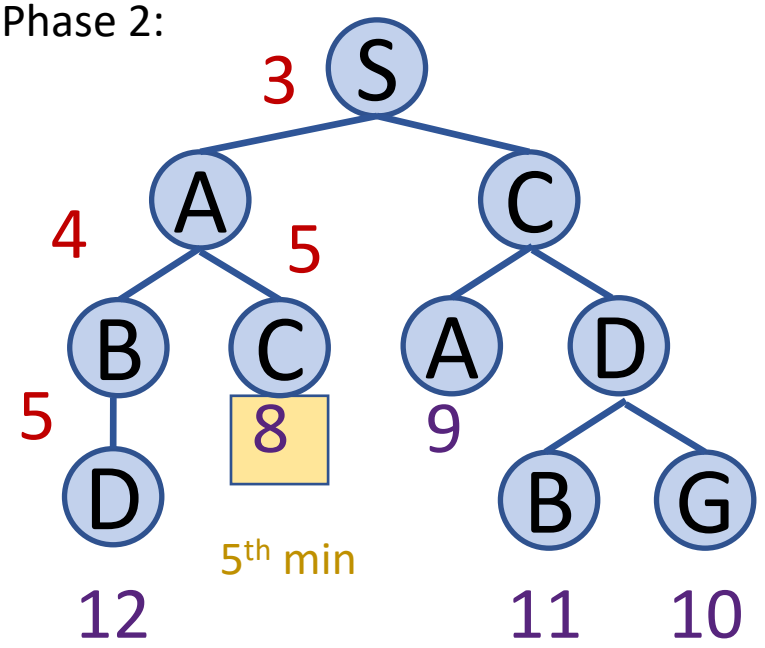


3+4+5

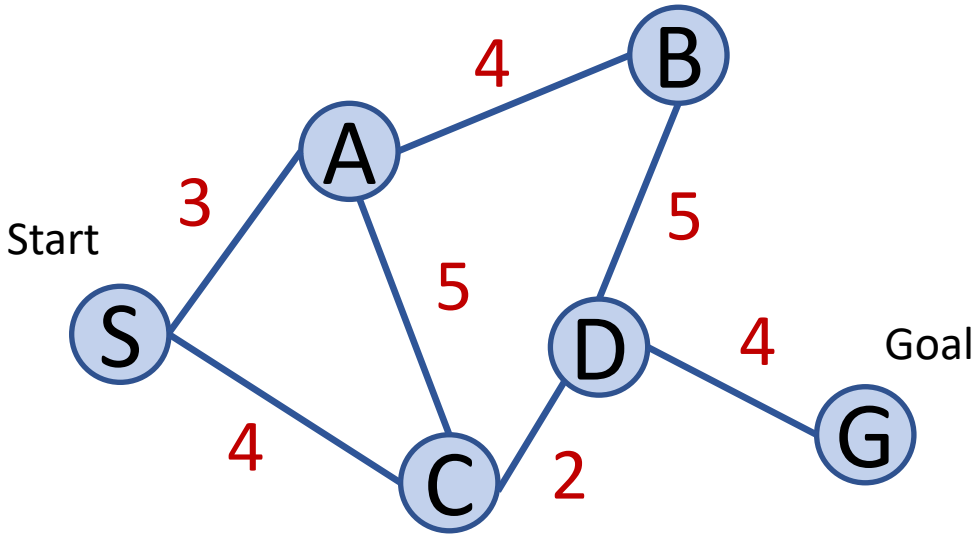
Phase 2: Extend all partial paths
until their length >= complete path to goal



Phase 2:

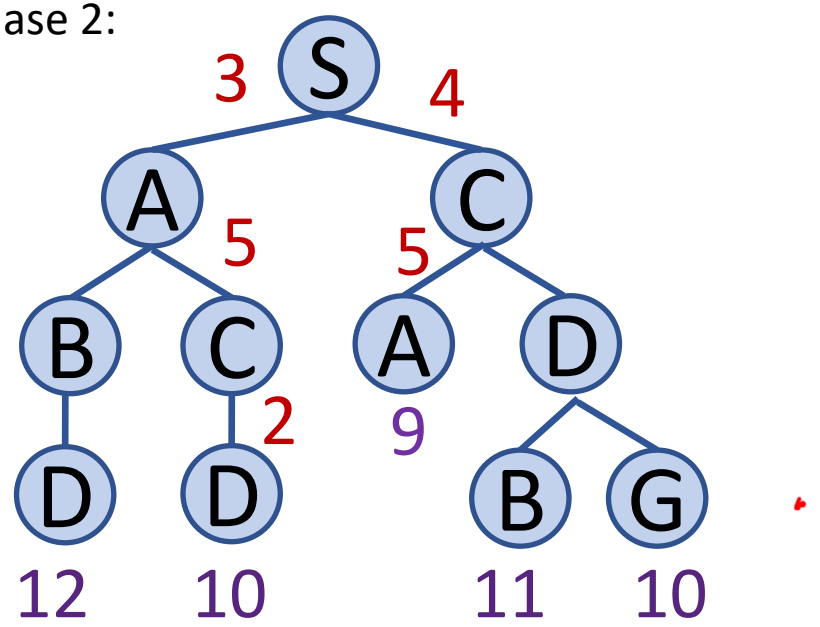


Phase 2: Extend all partial paths
until their length >= complete path to goal

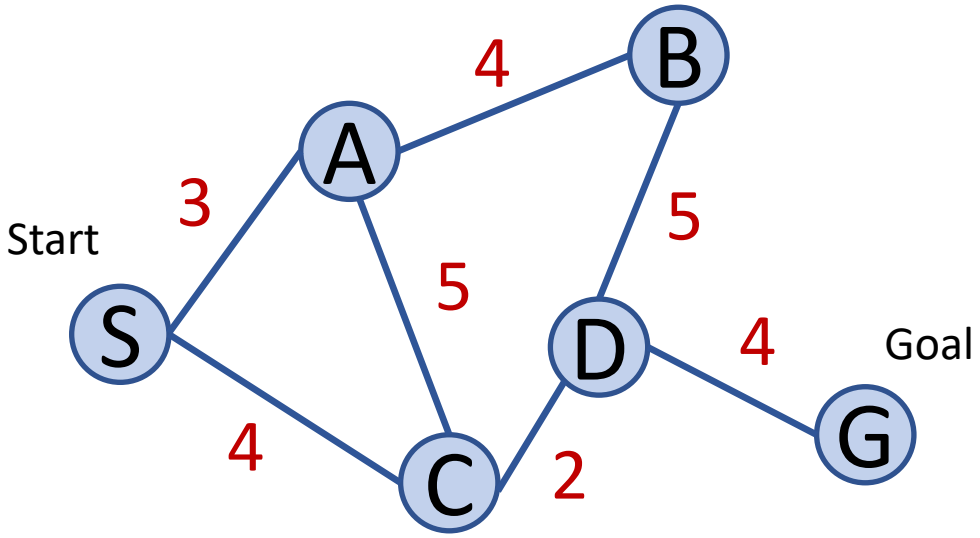


optimal : 10

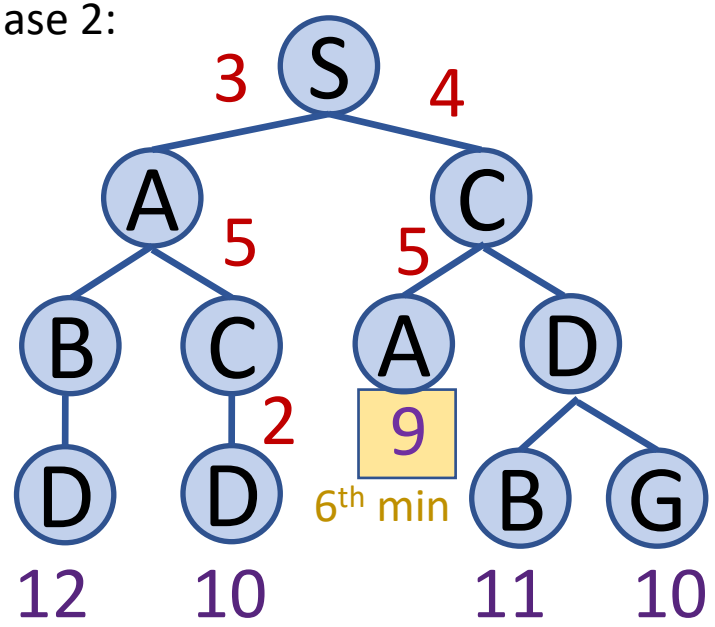
Phase 2:



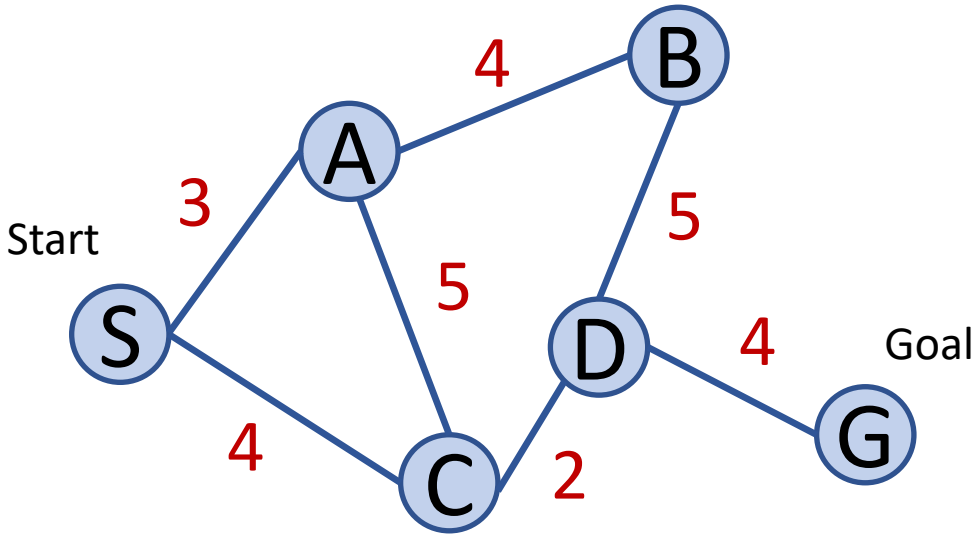
Phase 2: Extend all partial paths
until their length >= complete path to goal



Phase 2:

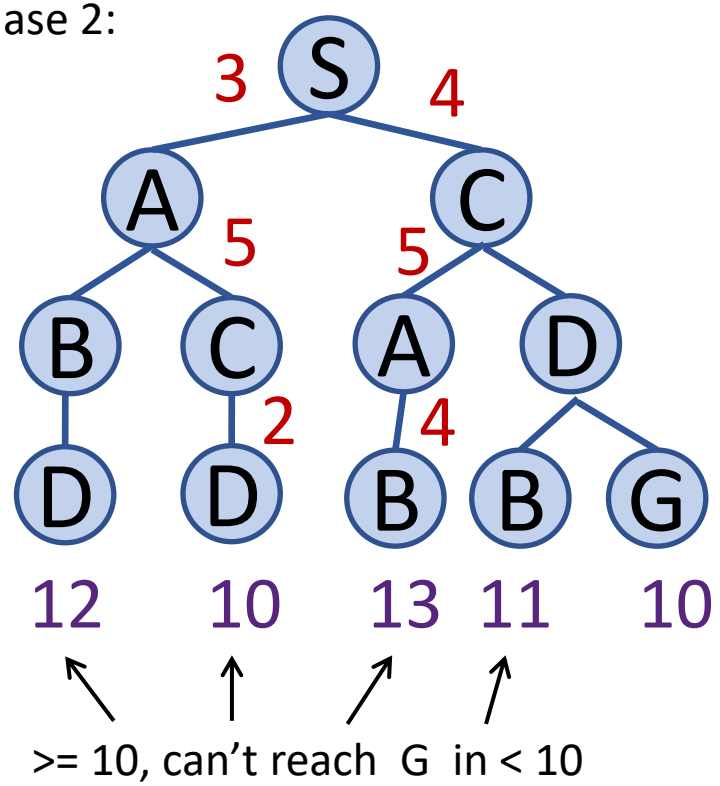


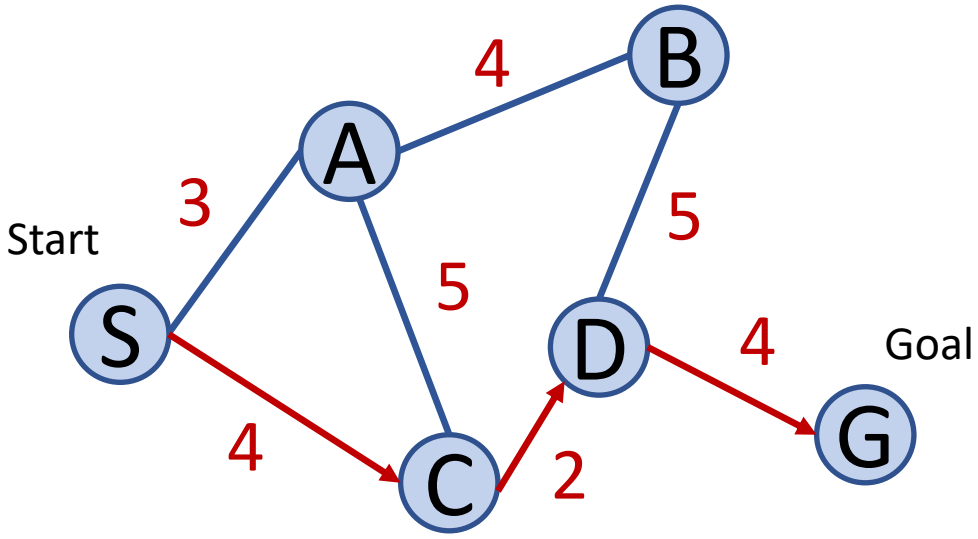
Phase 2: Extend all partial paths
until their length >= complete path to goal



Phase 2: Extend all partial paths
until their length >= complete path to goal

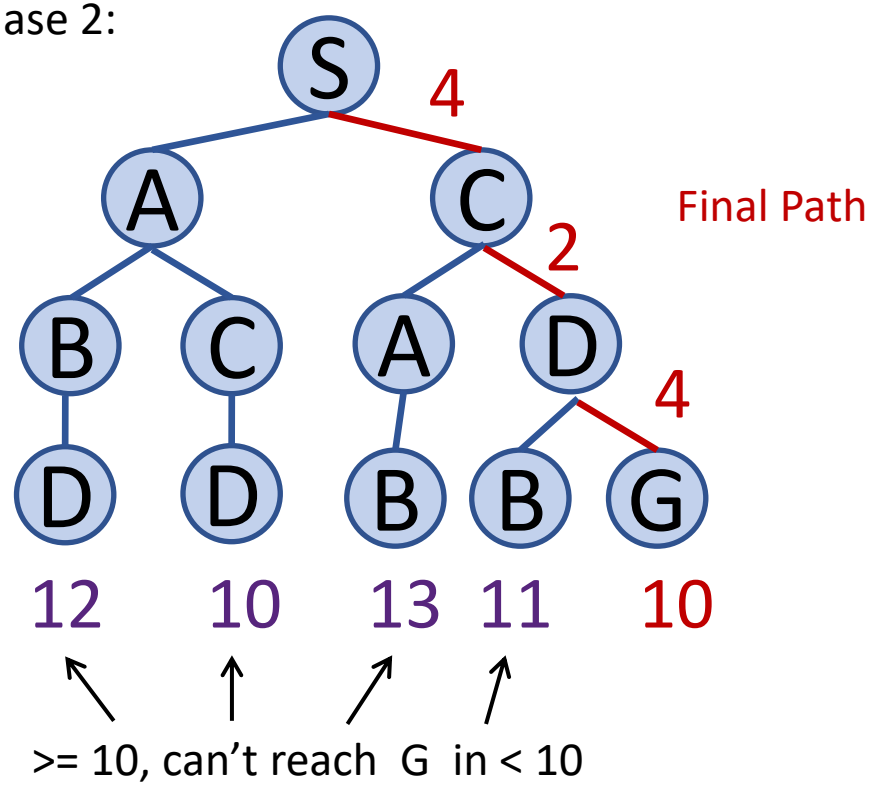
Phase 2:





Phase 2: Extend all partial paths
until their length >= complete path to goal

Phase 2:



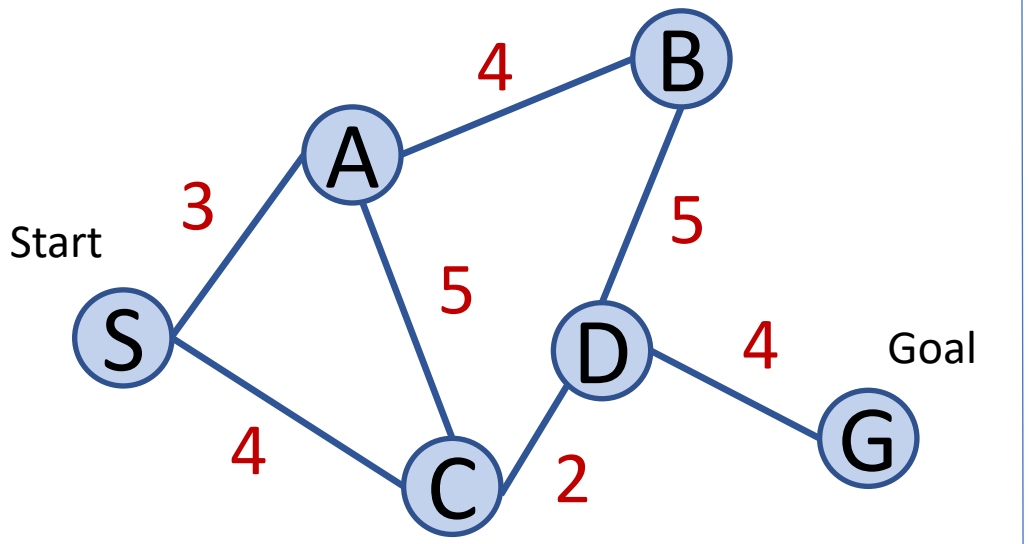
Final Path

Greedy Search with Pruning

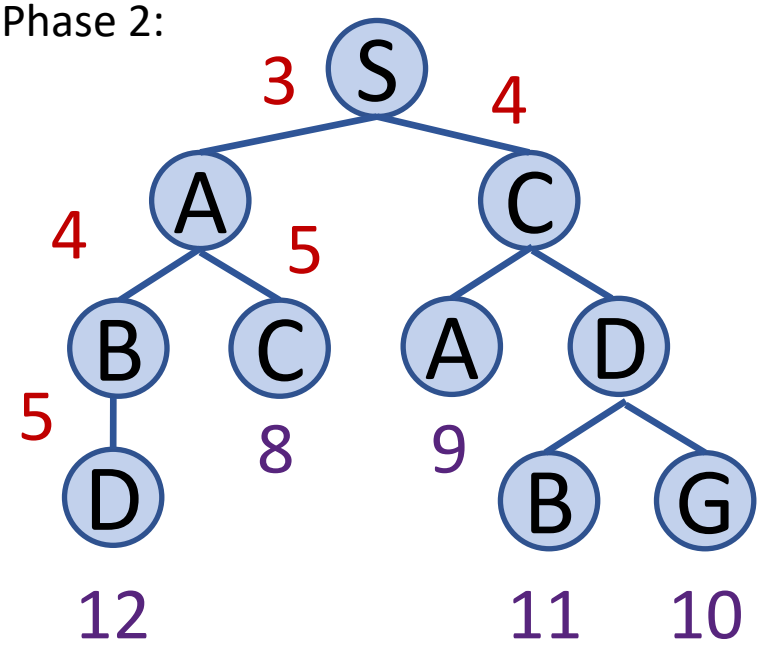
Dynamic Programming Principle:

If two or more paths reach a common node, delete all paths except the minimum cost path.

Greedy Search with Pruning

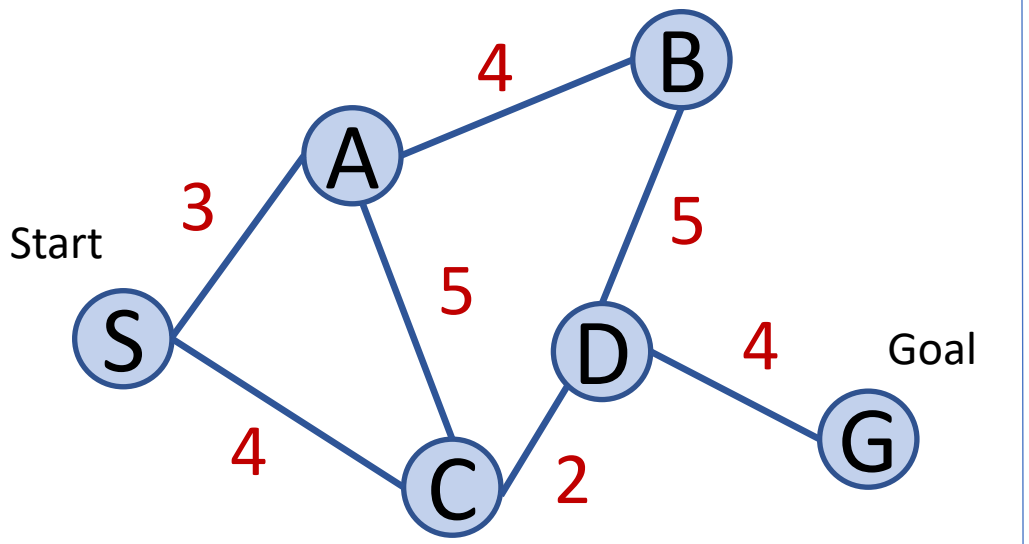


Phase 2:

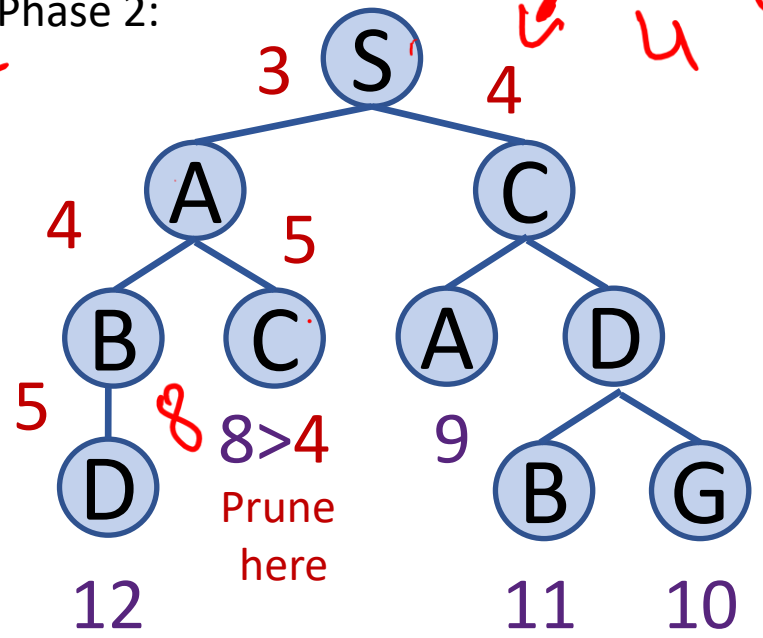


Phase 2: Extend all partial paths
until their length >= complete path to goal

Greedy Search with Pruning

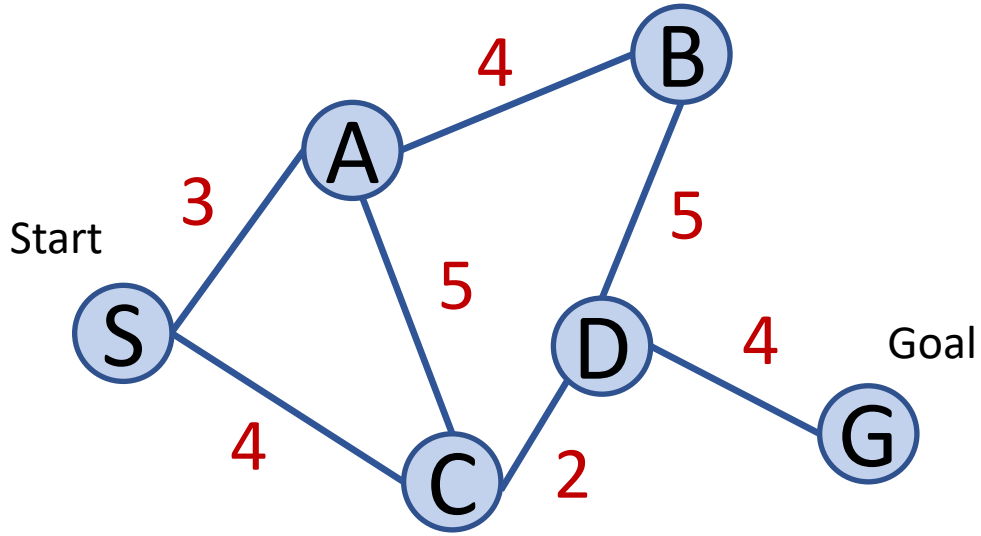


Phase 2:

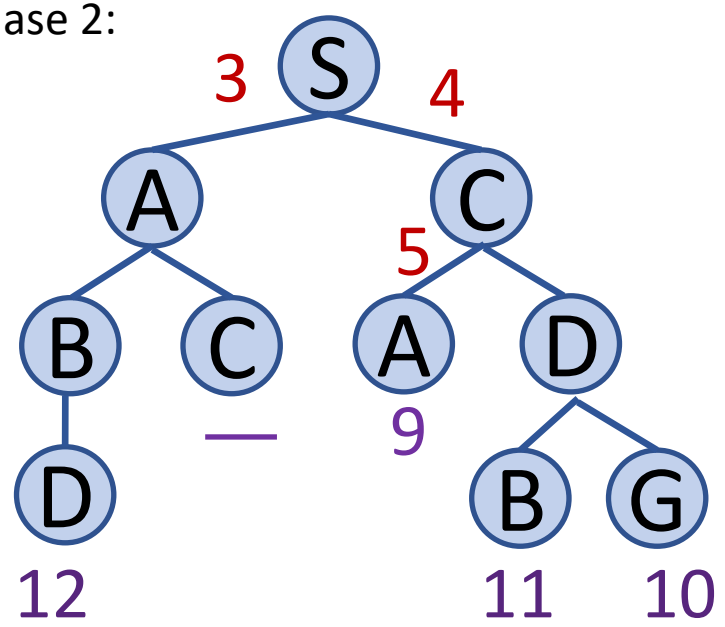


Phase 2: Extend all partial paths
until their length >= complete path to goal

Greedy Search with Pruning

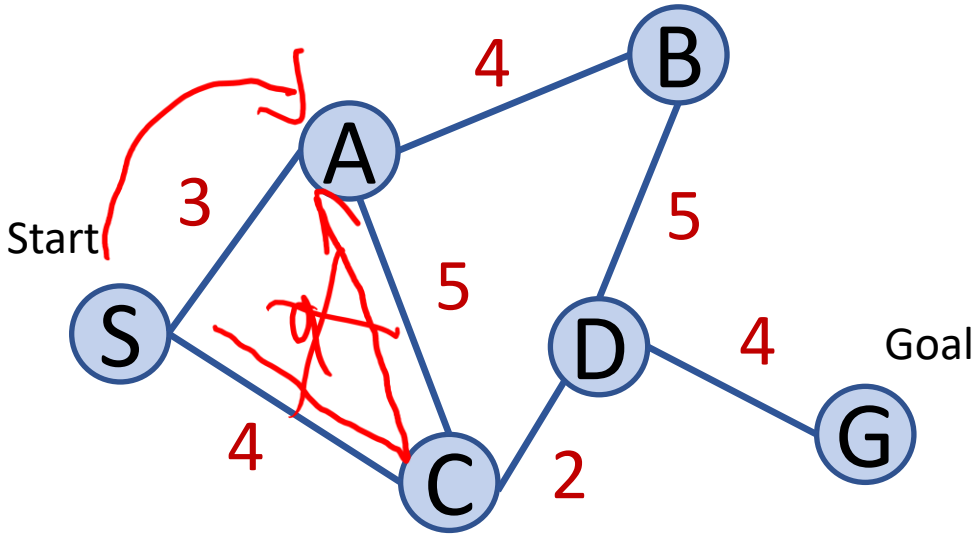


Phase 2:



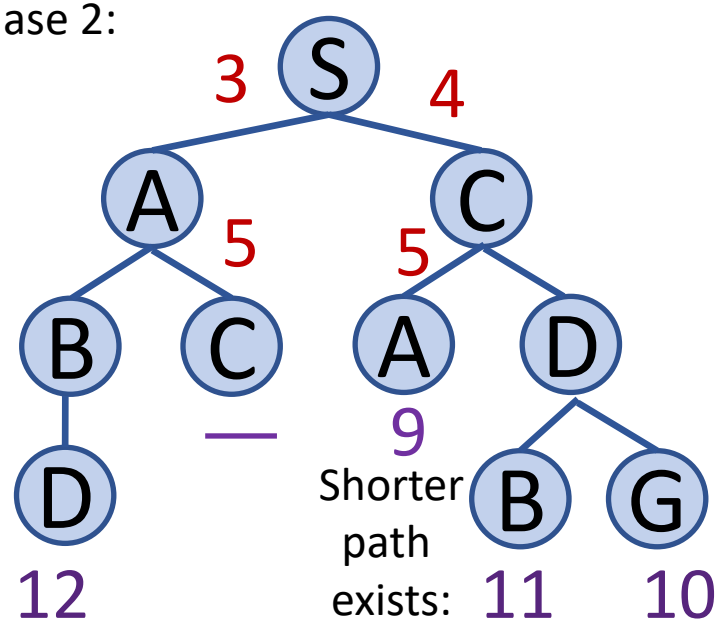
Phase 2: Extend all partial paths
until their length >= complete path to goal

Greedy Search with Pruning



Phase 2: Extend all partial paths
until their length >= complete path to goal

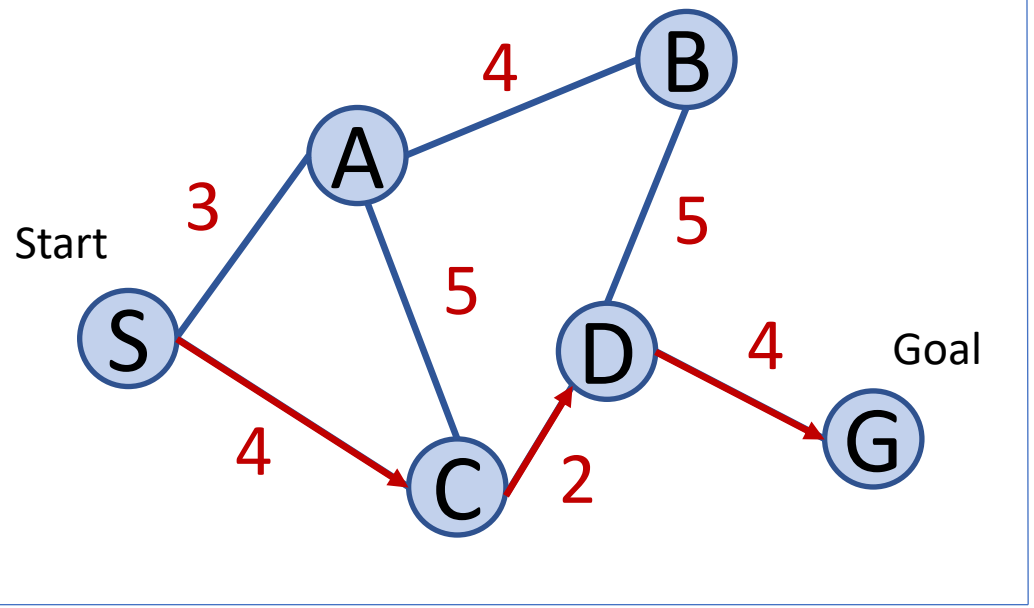
Phase 2:



Shorter path exists:

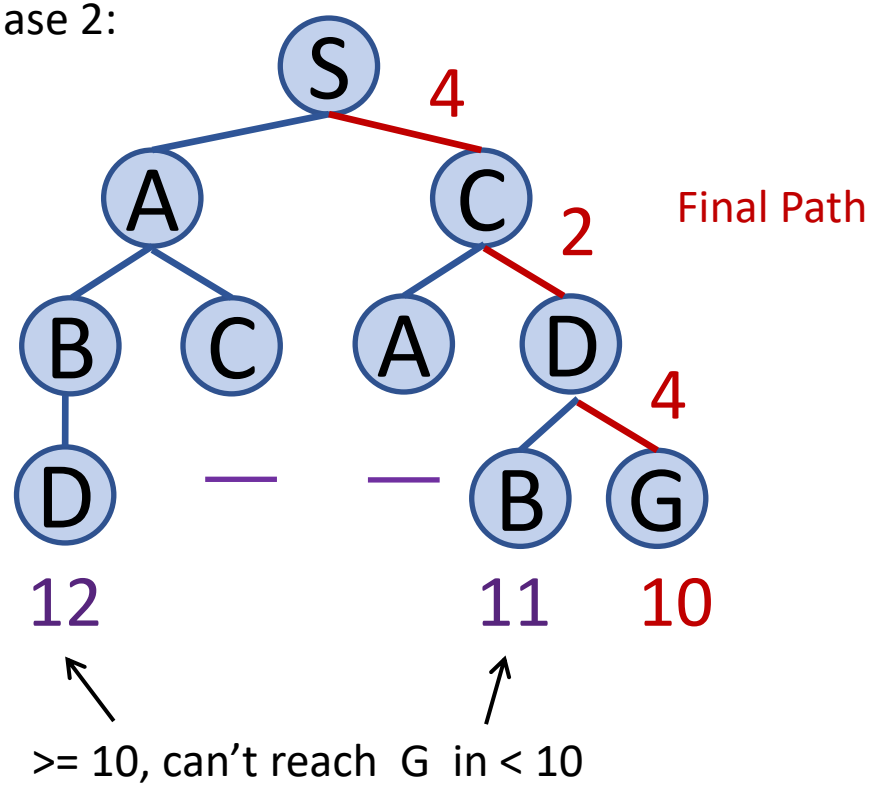
Prune!

Greedy Search with Pruning



Phase 2: Extend all partial paths
until their length >= complete path to goal

Phase 2:



Final Path

Implementation of Greedy Search with Pruning

Data Structure: Queue
Elements of queue: Partial paths

Initialize: Place start node in queue
Until a path in queue reaches goal node or queue is empty:
- Remove 1st queue element & extend path to its children
- Reject loops
- Add new paths to queue
- Prune
- Sort

Connection to Dijkstra's Algorithm

Same as Greedy Search with Pruning except

- Dijkstra's algorithm computes the all-pairs shortest paths
- while "Greedy Search with Pruning" computes the shortest path between a single start state and a single goal state.

A* Algorithm = Greedy Search with Pruning and Underestimates of Remaining Distance

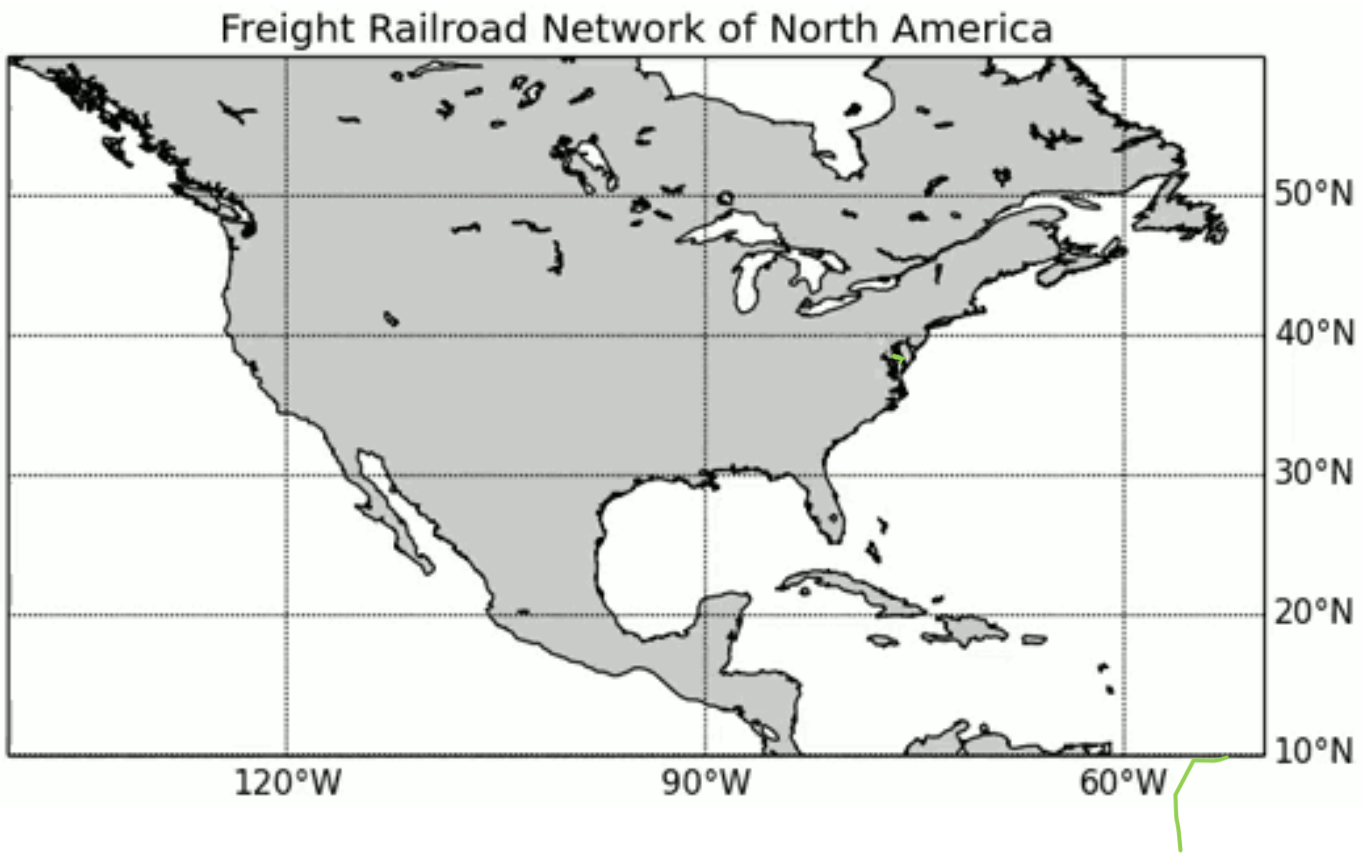
- Remaining distance = e.g., straight-line distance on a highway map
- In each step:

  Estimate of total path length =

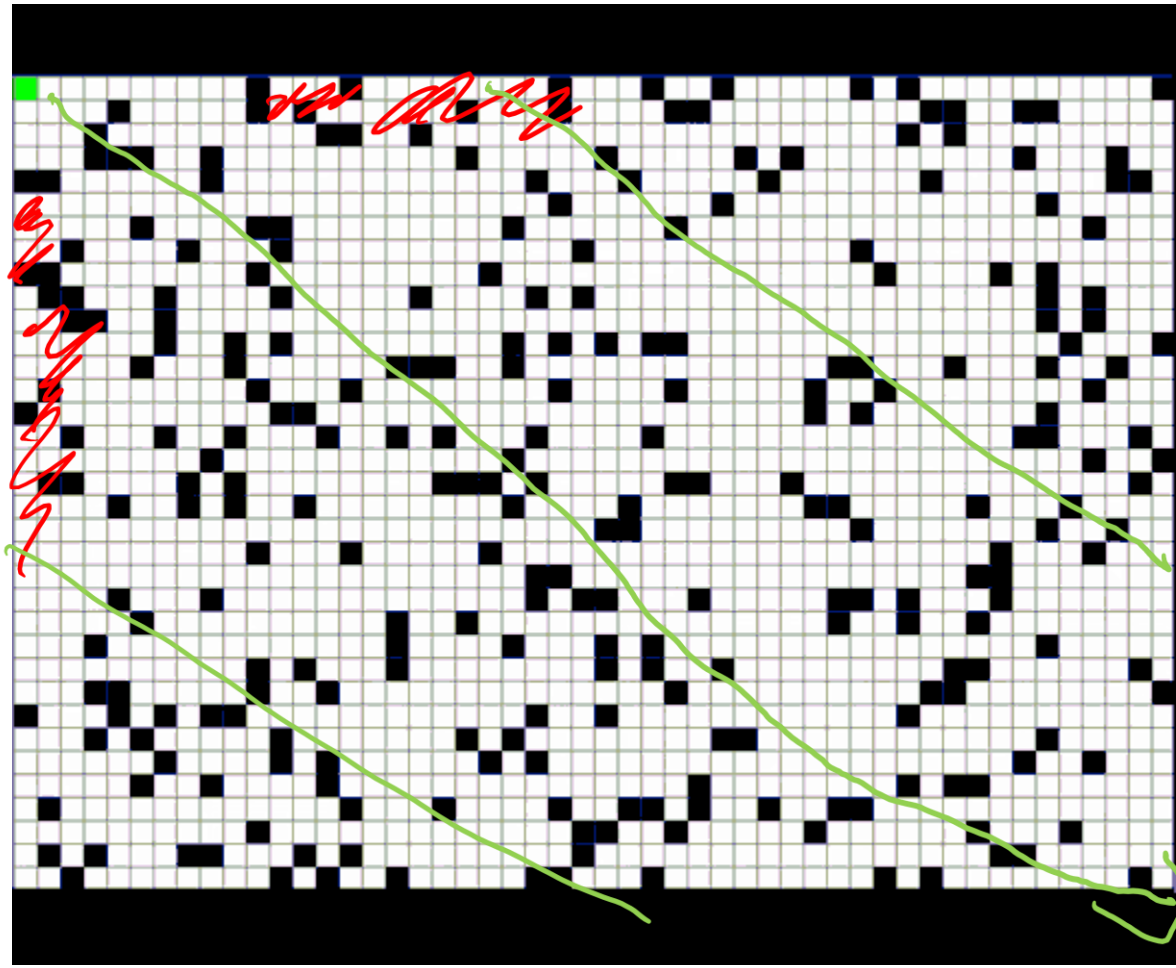
  length of partial path + underestimate of remaining

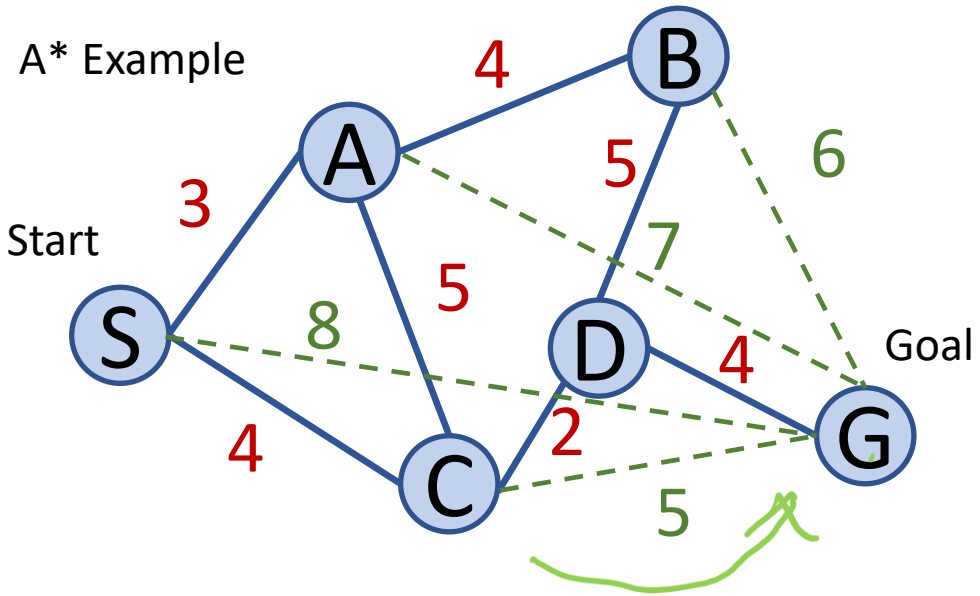
  "This path is at least this bad."

An animation of the A* algorithm as it explores the North American freight train network to find the optimum path between Washington, D.C. and Los Angeles.



A* pathfinding algorithm navigating around a randomly-generated maze

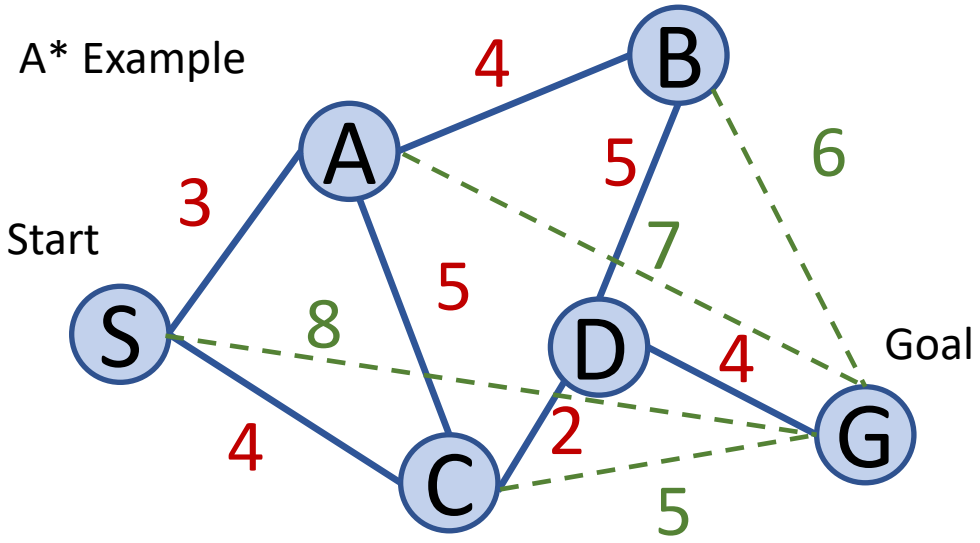




A* Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.

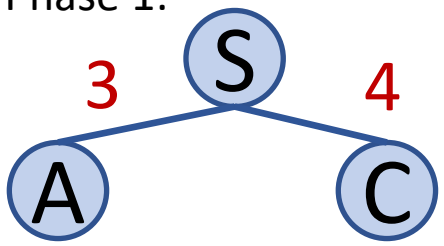
Phase 2: Extend all estimated partial paths
until their length >= complete path to goal

A* Example

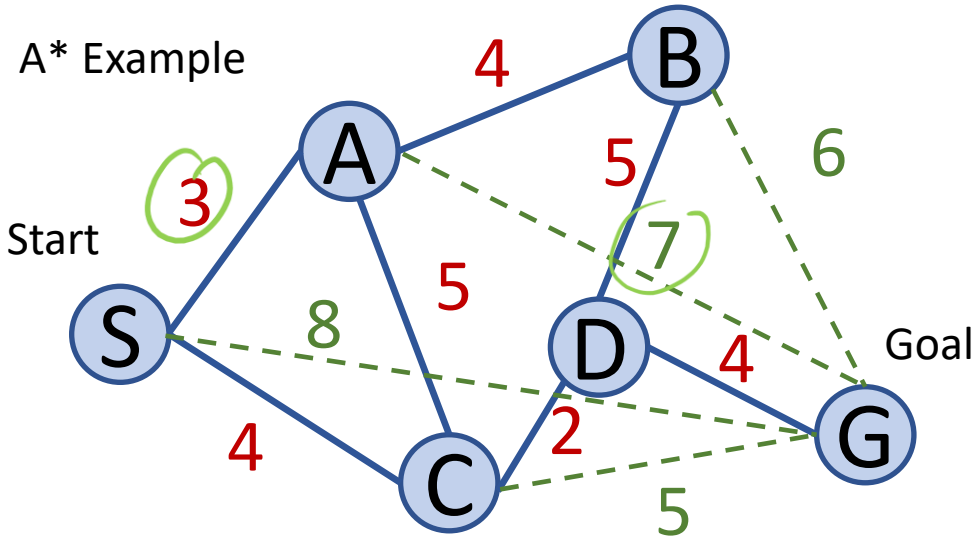


Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.

Phase 1:



A* Example



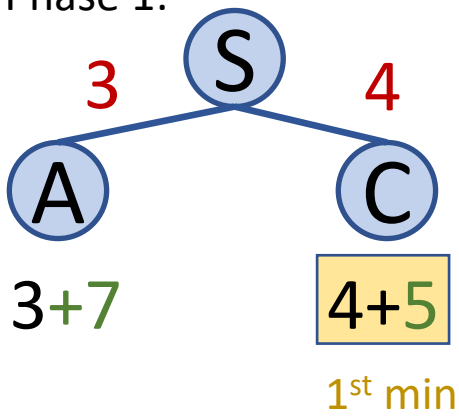
Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.

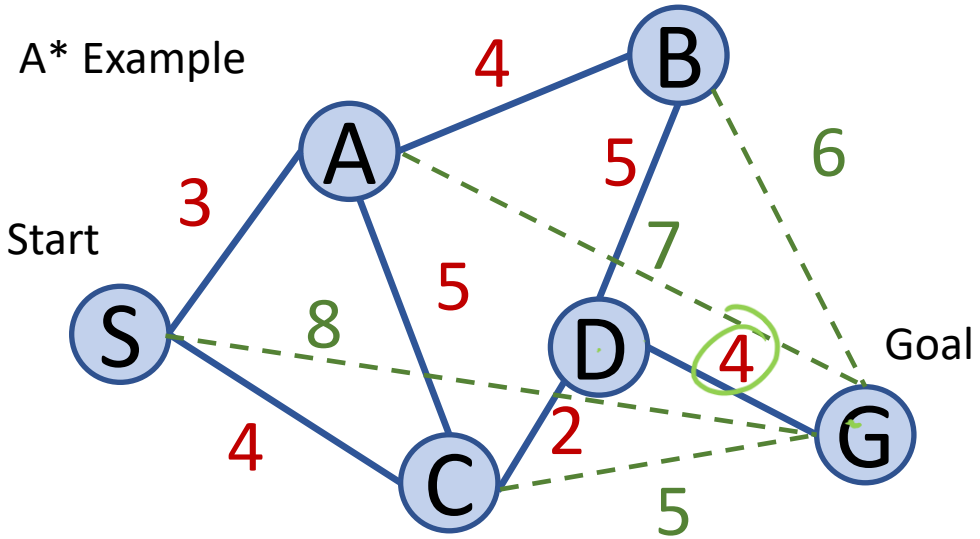
$$f(x) = g(x) + h(x)$$

↑ cost for reaching x

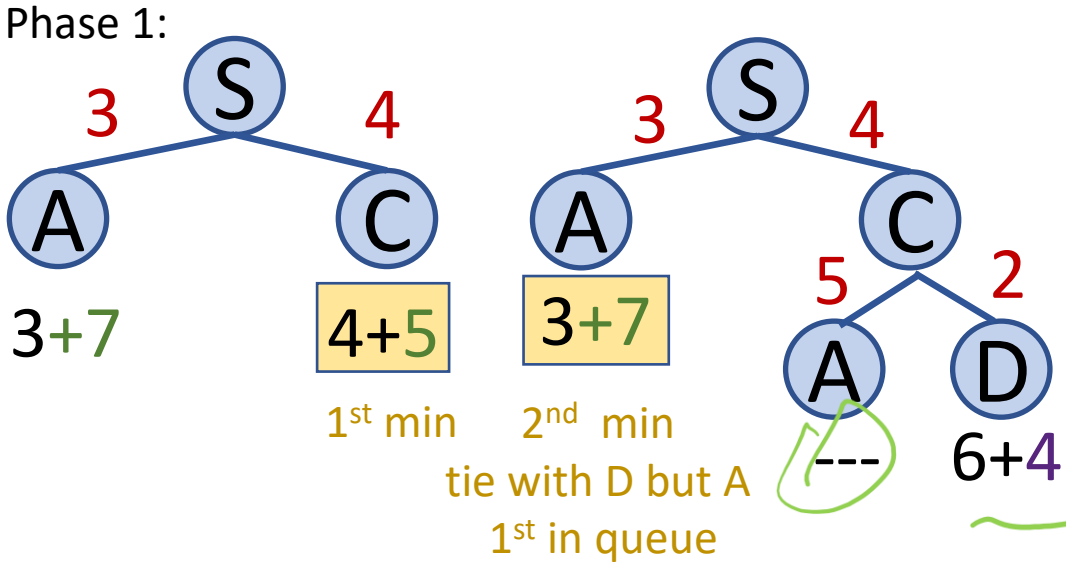
| estimate of cost of reaching x from

Phase 1:





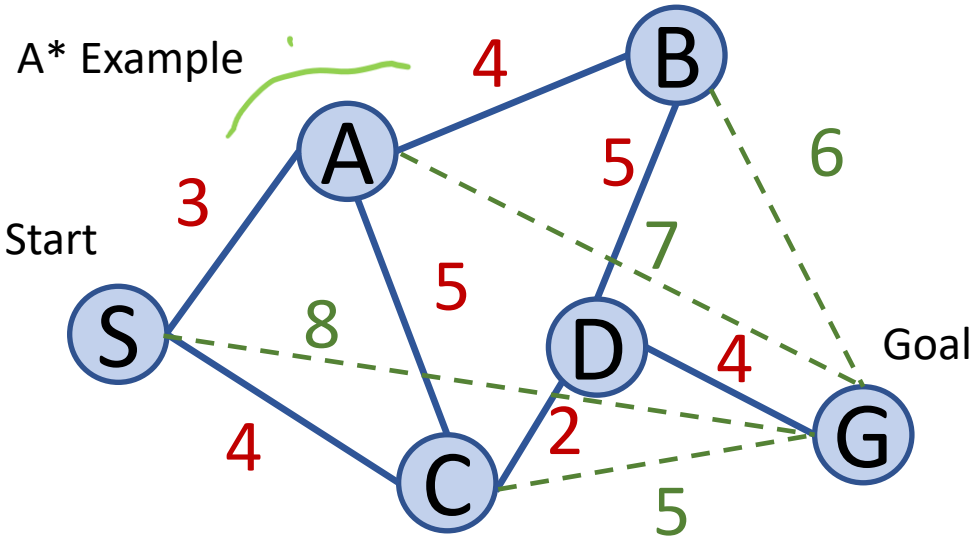
Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.



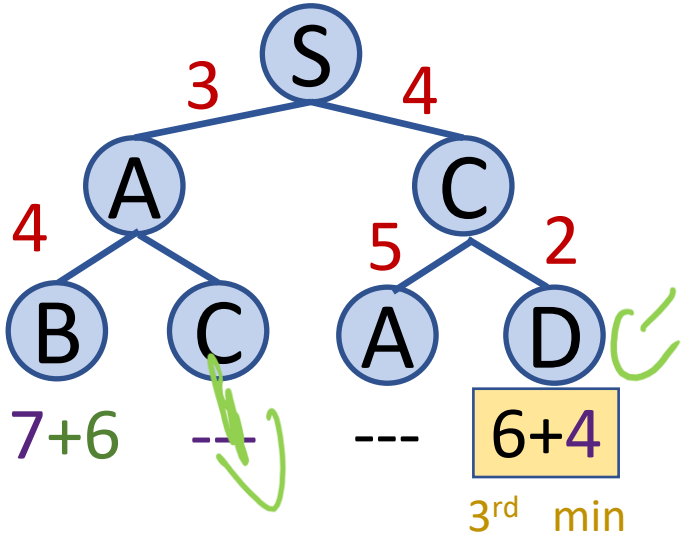
$f(x) = g + h$

$c(D,S) \approx h(D)$

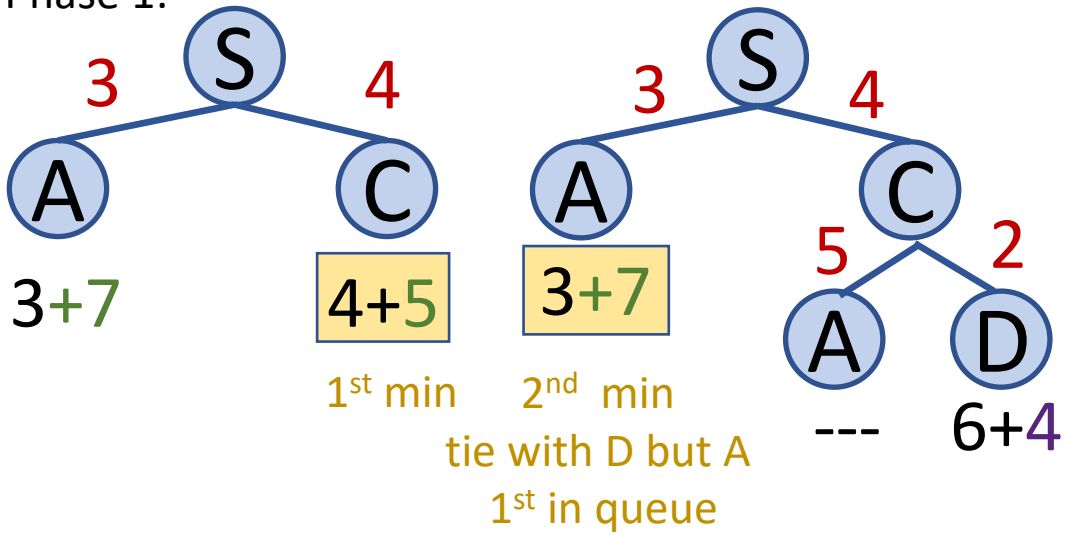
$c(S,C) + c(C,D)$



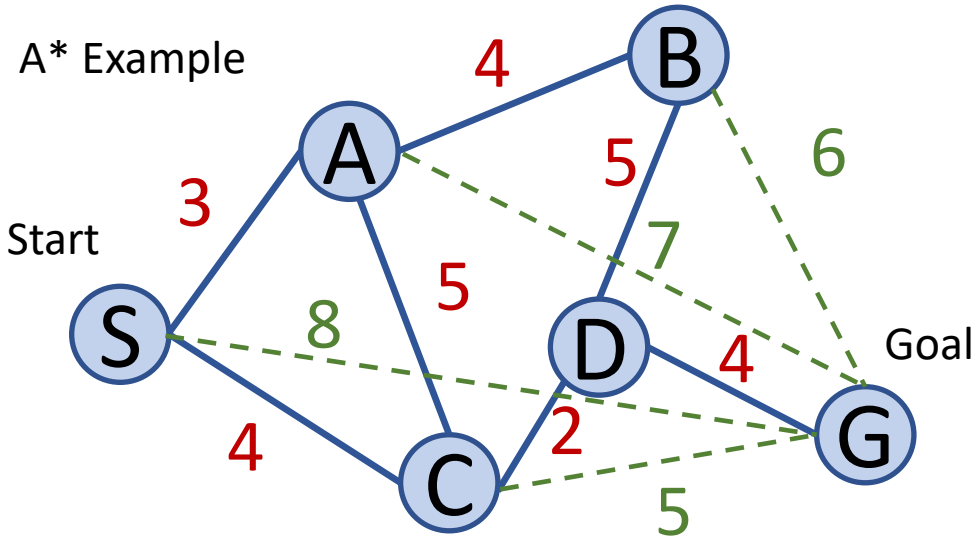
Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.



Phase 1:

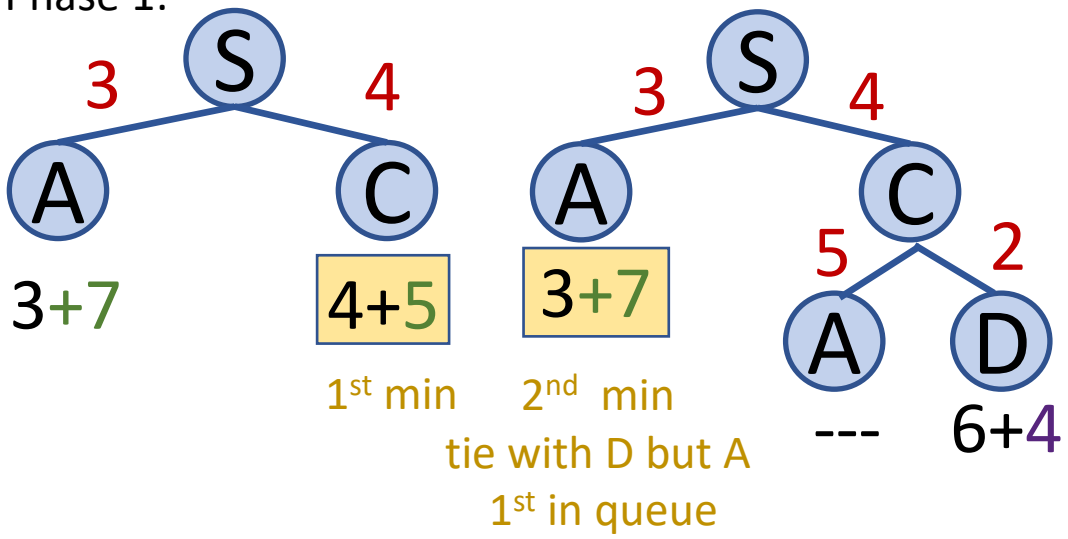


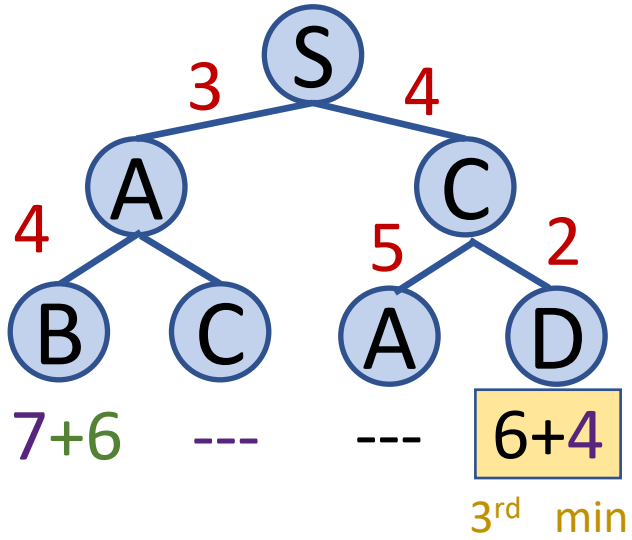
A* Example

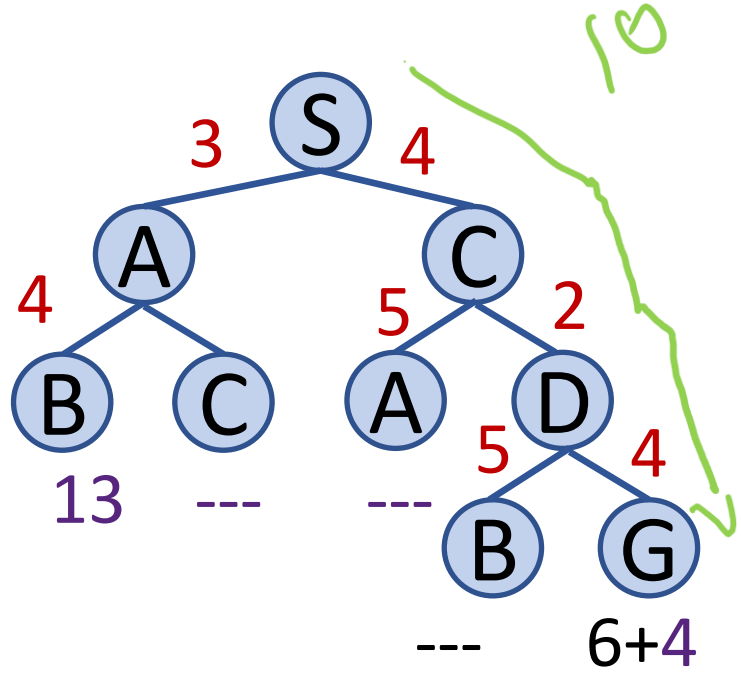


Phase 1: Extend shortest estimated partial path (= length of partial path + underestimate of remaining) until goal is reached. Reject loops.

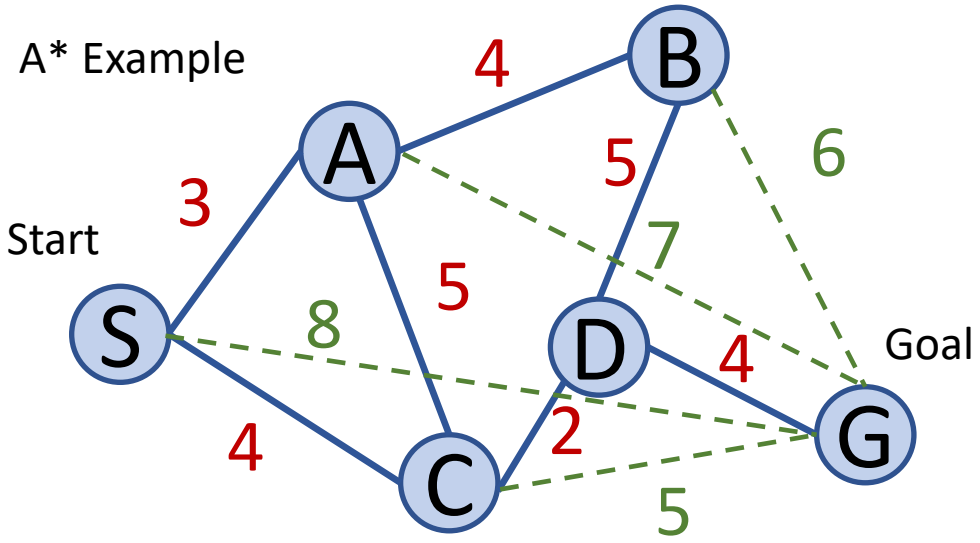
Phase 1:





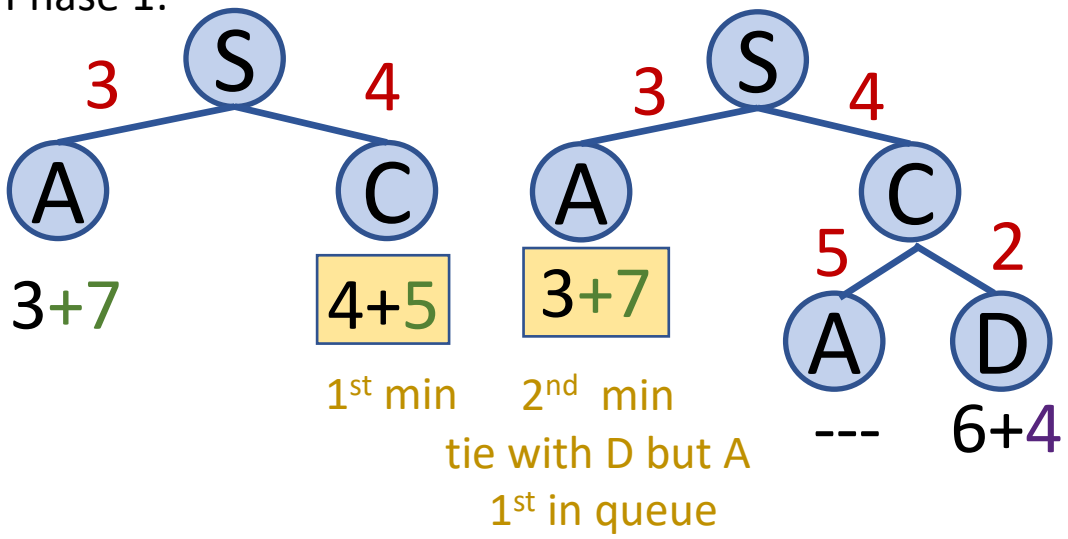


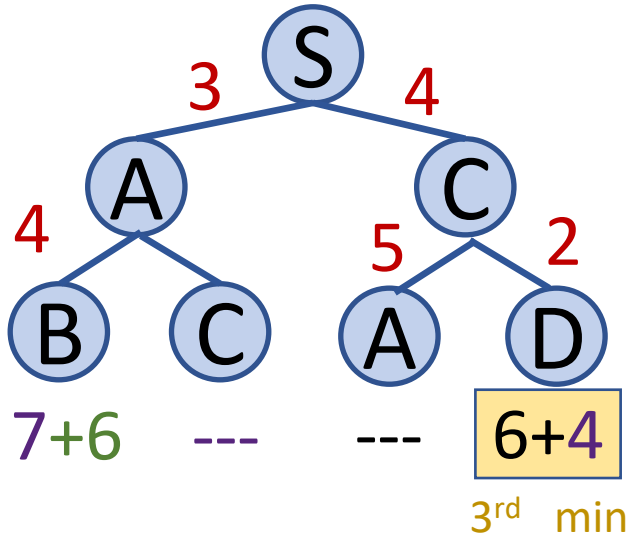
A* Example

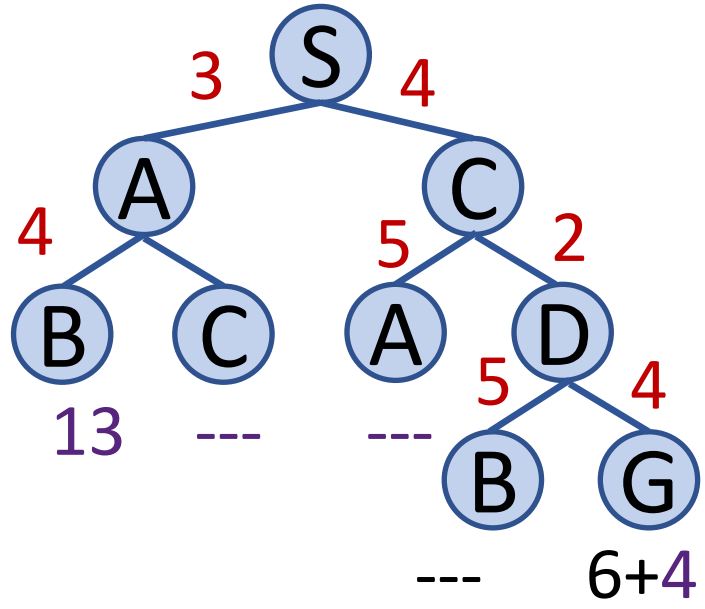


Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.

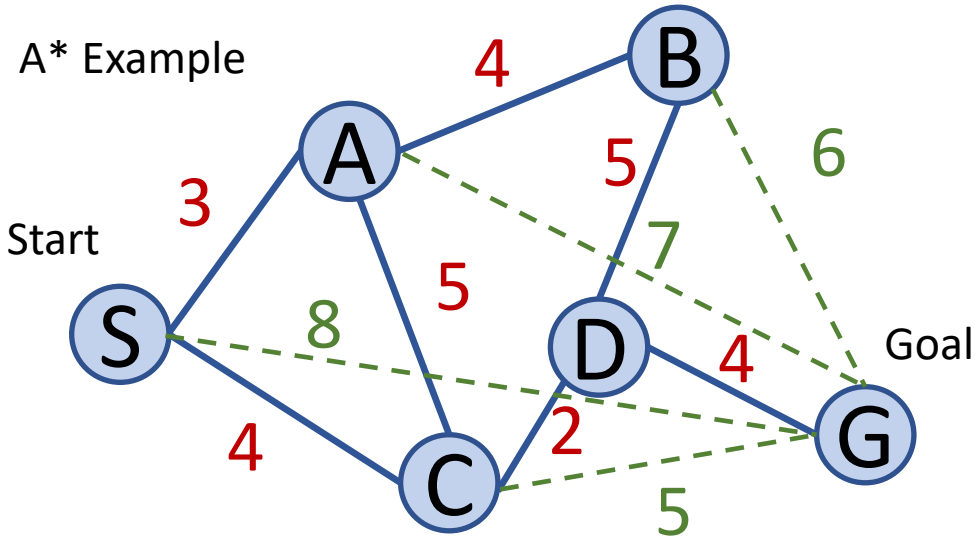
Phase 1:







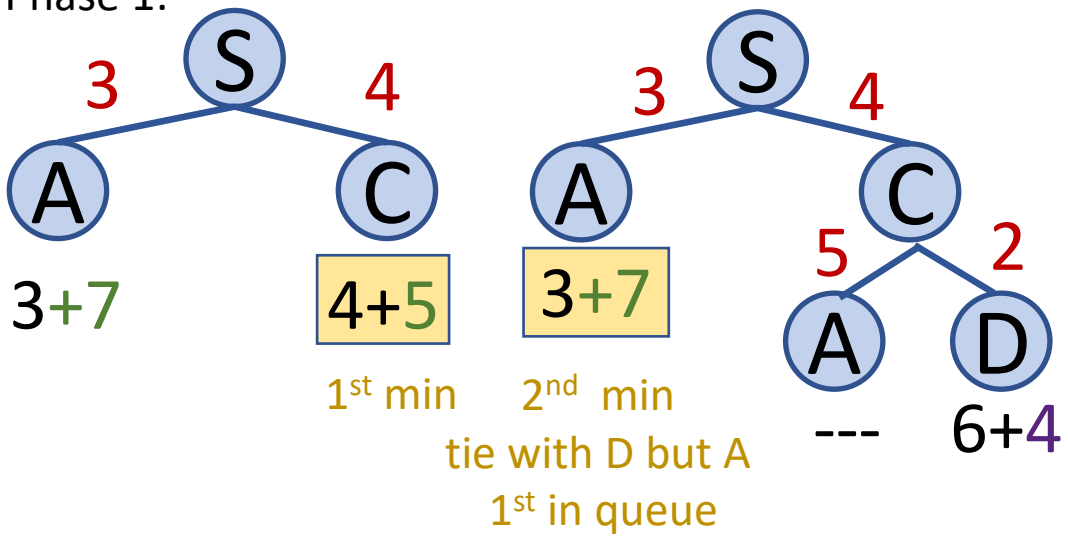
End of phase 1: G reached

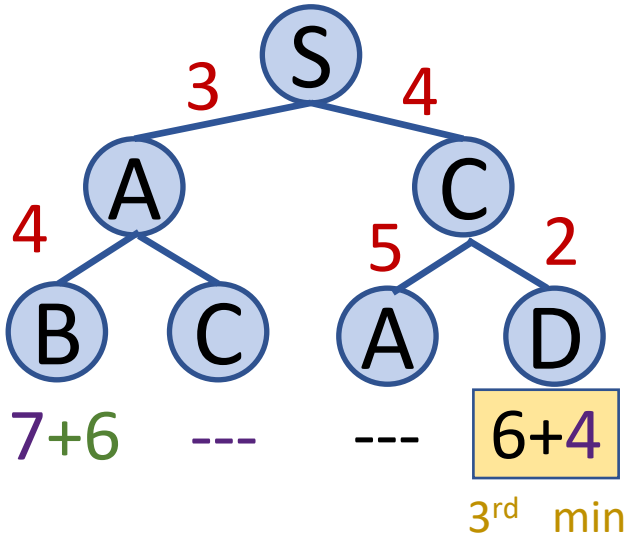


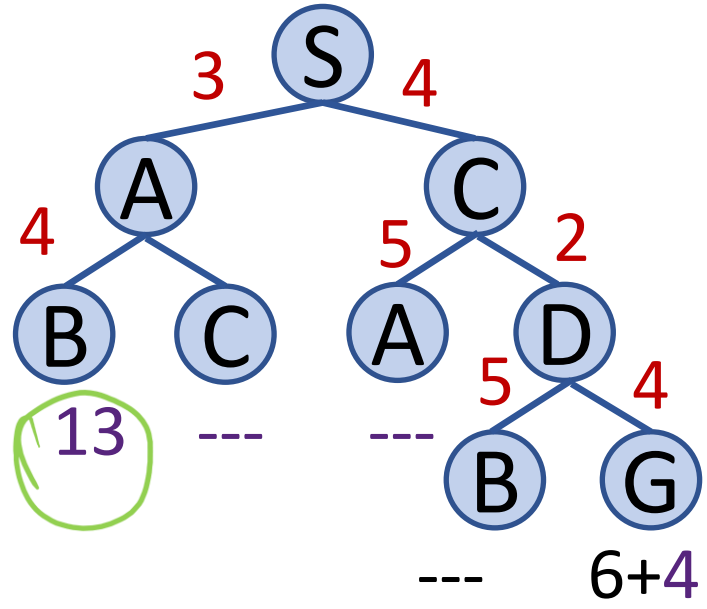
Phase 1: Extend shortest estimated partial path
(= length of partial path + underestimate of remaining)
until goal is reached.
Reject loops.

Phase 2: Extend all estimated partial paths
until their length >= complete path to goal

Phase 1:





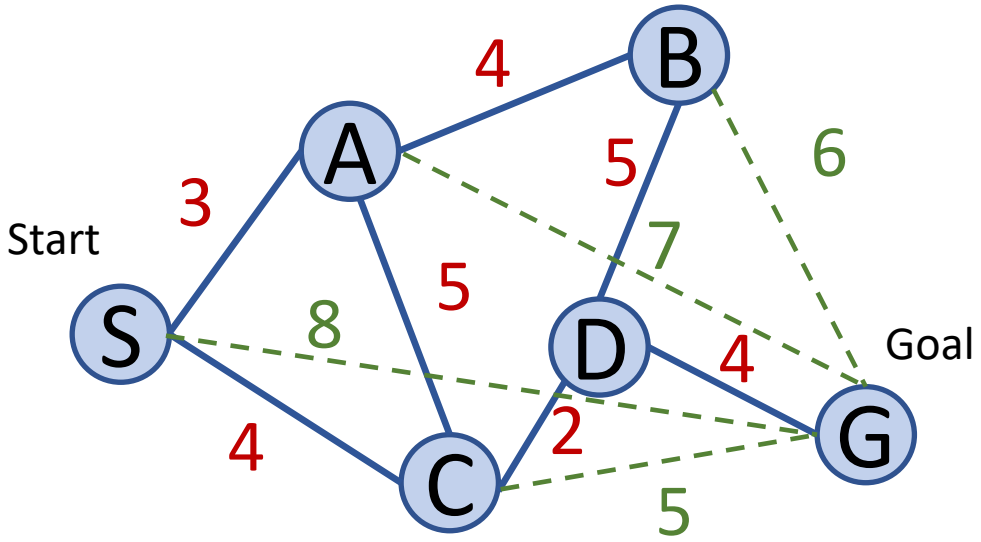


End of phase 1: G reached

End of phase 2: 13>10

Admissible Heuristic

- If the heuristic function never overestimates the actual cost to get to the goal, then A* is guaranteed to return a least-cost path from start to goal.



Admissible Heuristic

- If the heuristic function never overestimates the actual cost to get to the goal, then A* is guaranteed to return a least-cost path from start to goal.

Time Complexity

- If goal state exists and is reachable from start state:

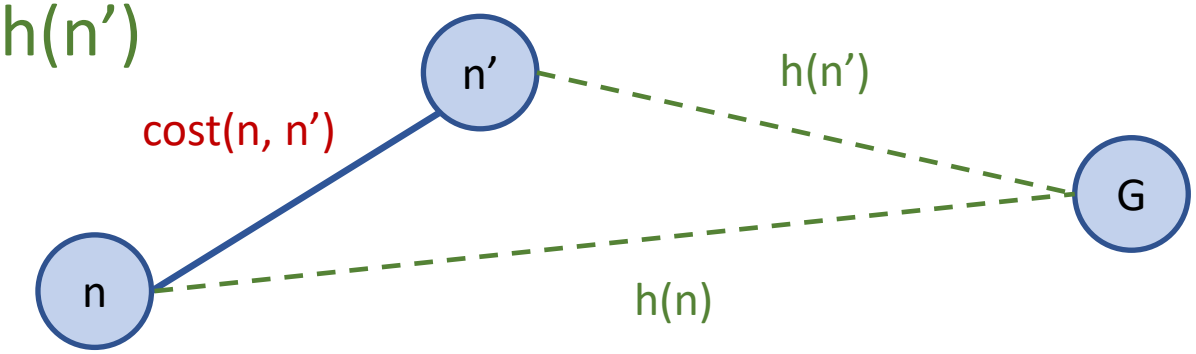
  Worst case $O(b^d)$ where d = depth(start,goal), b branching factor

- Otherwise, A* will not terminate
- A good heuristic function allows A* to prune away many of the b^d nodes.

Monotone Underestimates of Remaining Distance

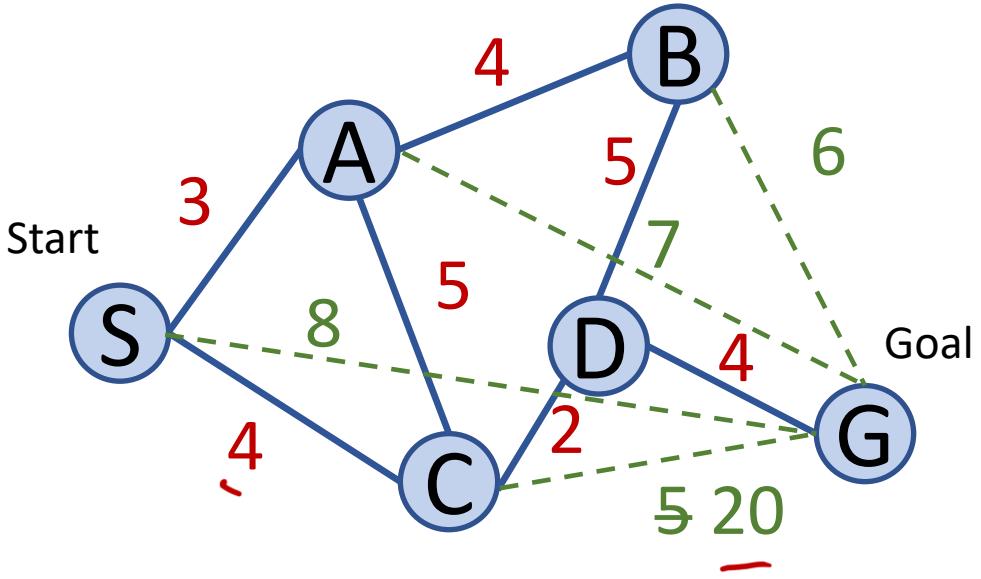
Heuristic function h(n) is "monotone" if and only if it satisfies the triangle inequality:

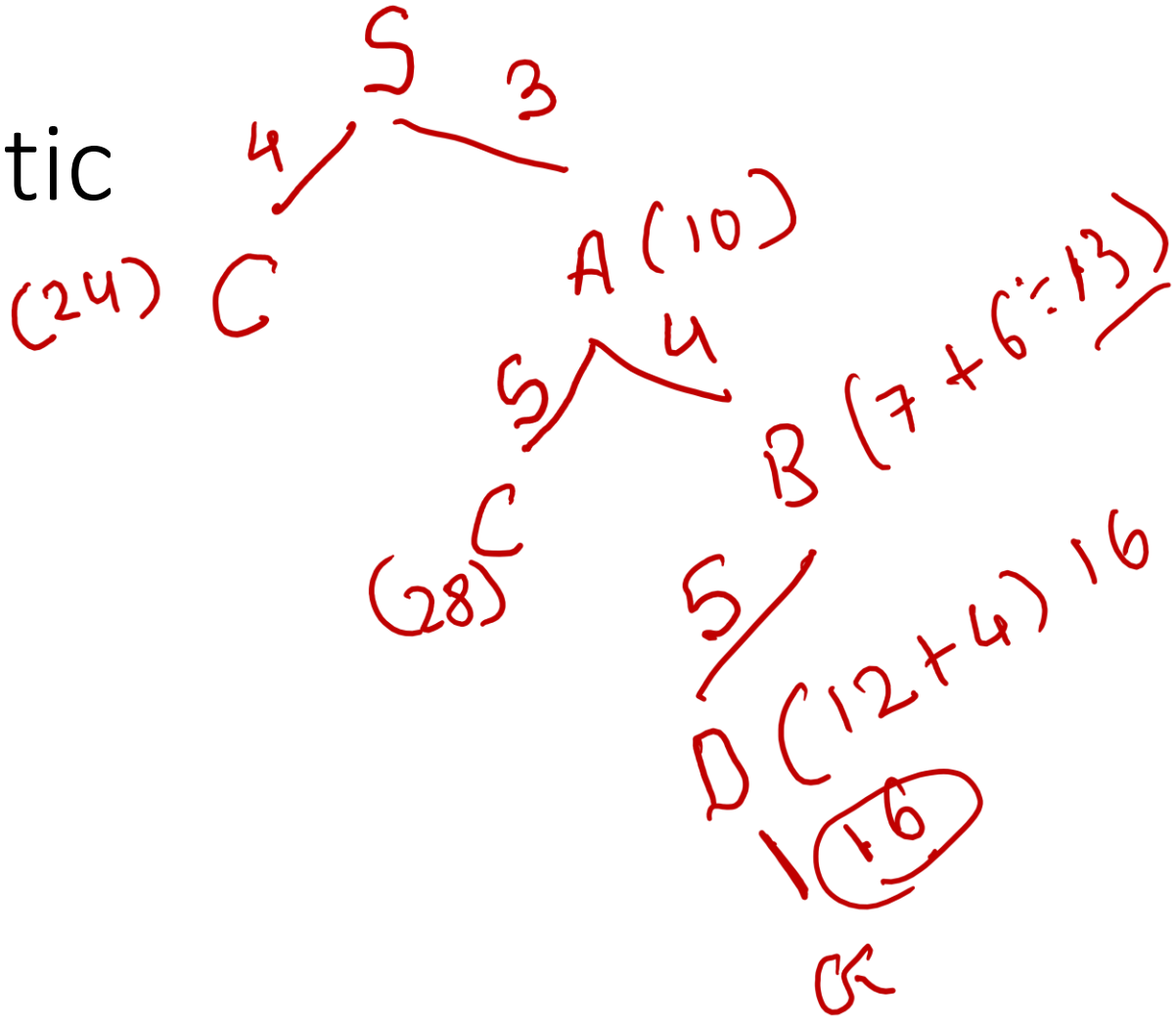
- h(n) = 0 if n=goal state
- 0 <= h(n) <= cost(n, n') + h(n')



With a monotone heuristic, A* is guaranteed to find an optimal path without processing any node more than once.

Example of Bad Heuristic





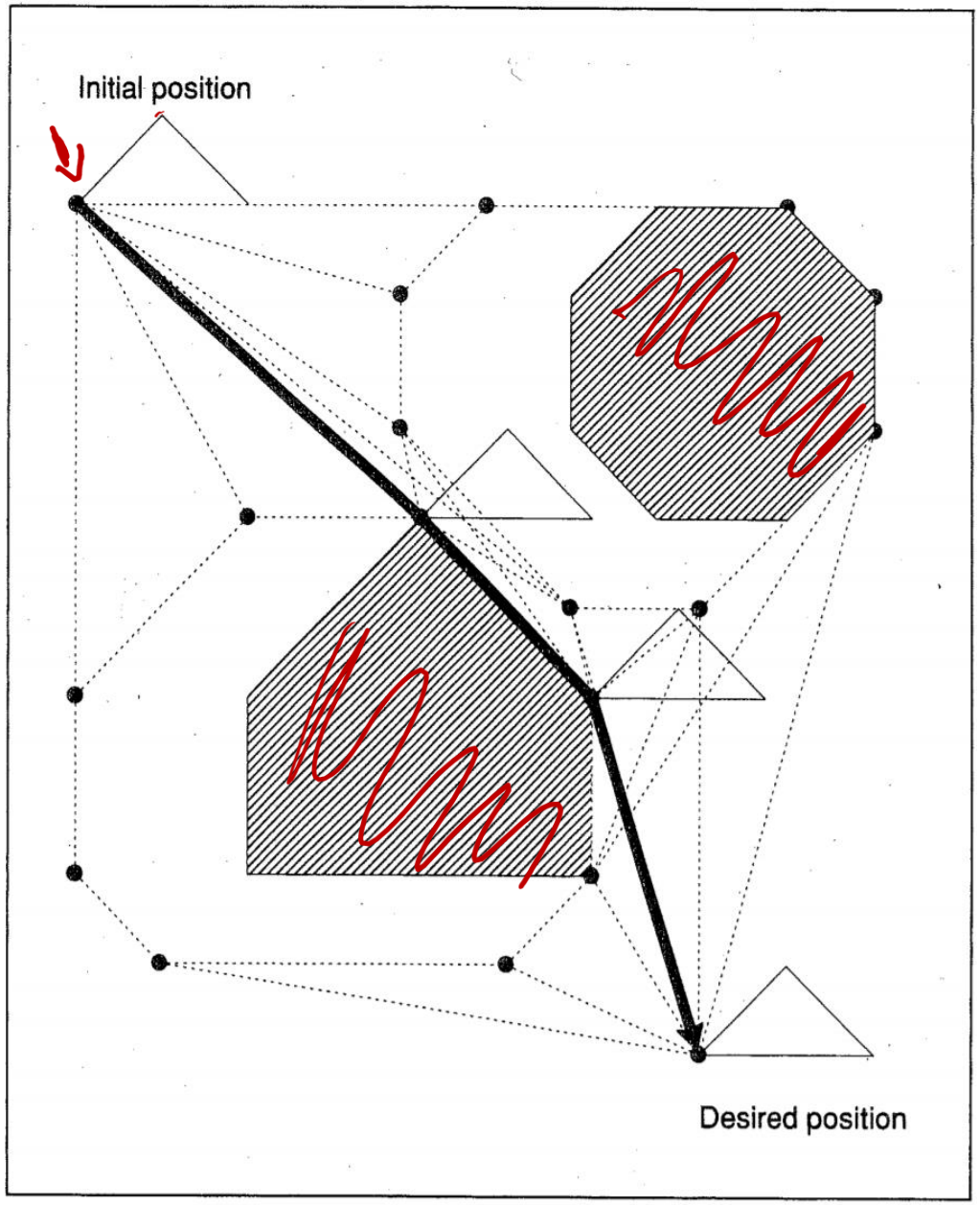
Robot Path Planning with A*

- Convert 2D Map into

  "Configuration Space" C
  - Robot represented as point
  - Obstacles represented as obstacles + fence = O

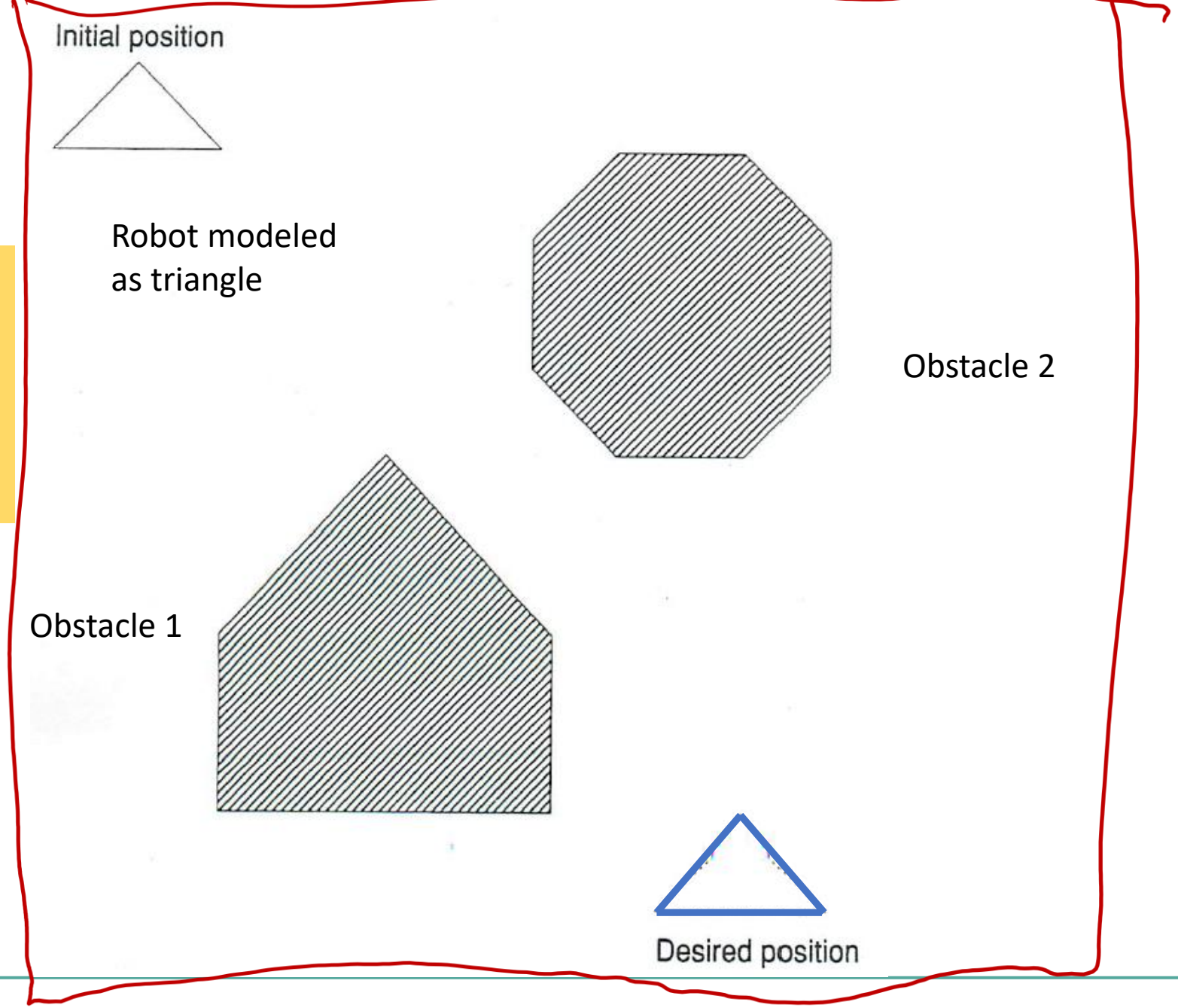
- Run A* on Visibility Graph in

  "Free Space" F = C - O



AI System Task:

Find shortest path for robot to move from initial to goal position without robot hitting the obstacles



Create a fence around each obstacle:

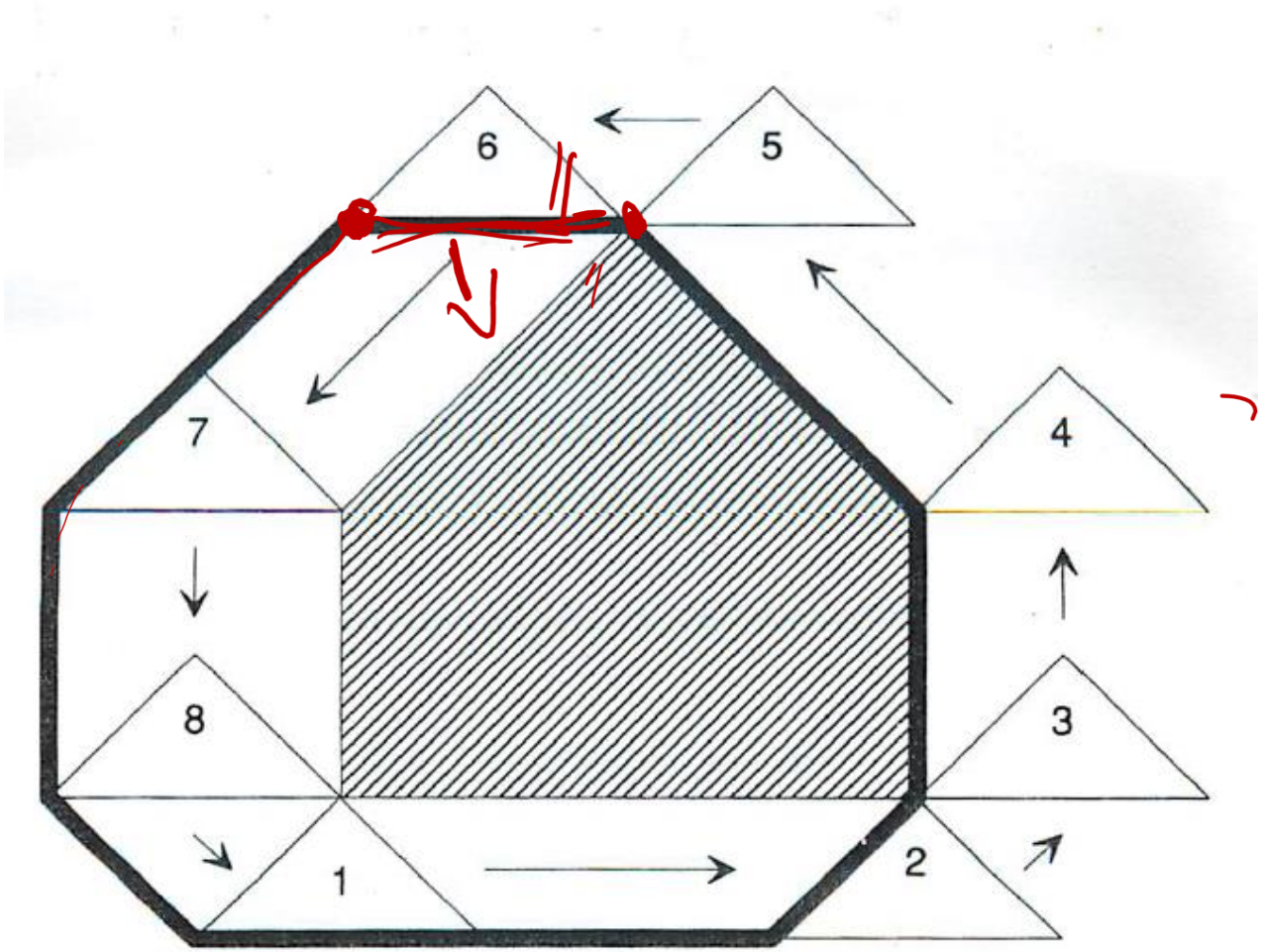
1. Select robot reference point
2. Slide robot shape around obstacle
3. Mark locations of reference point as fence

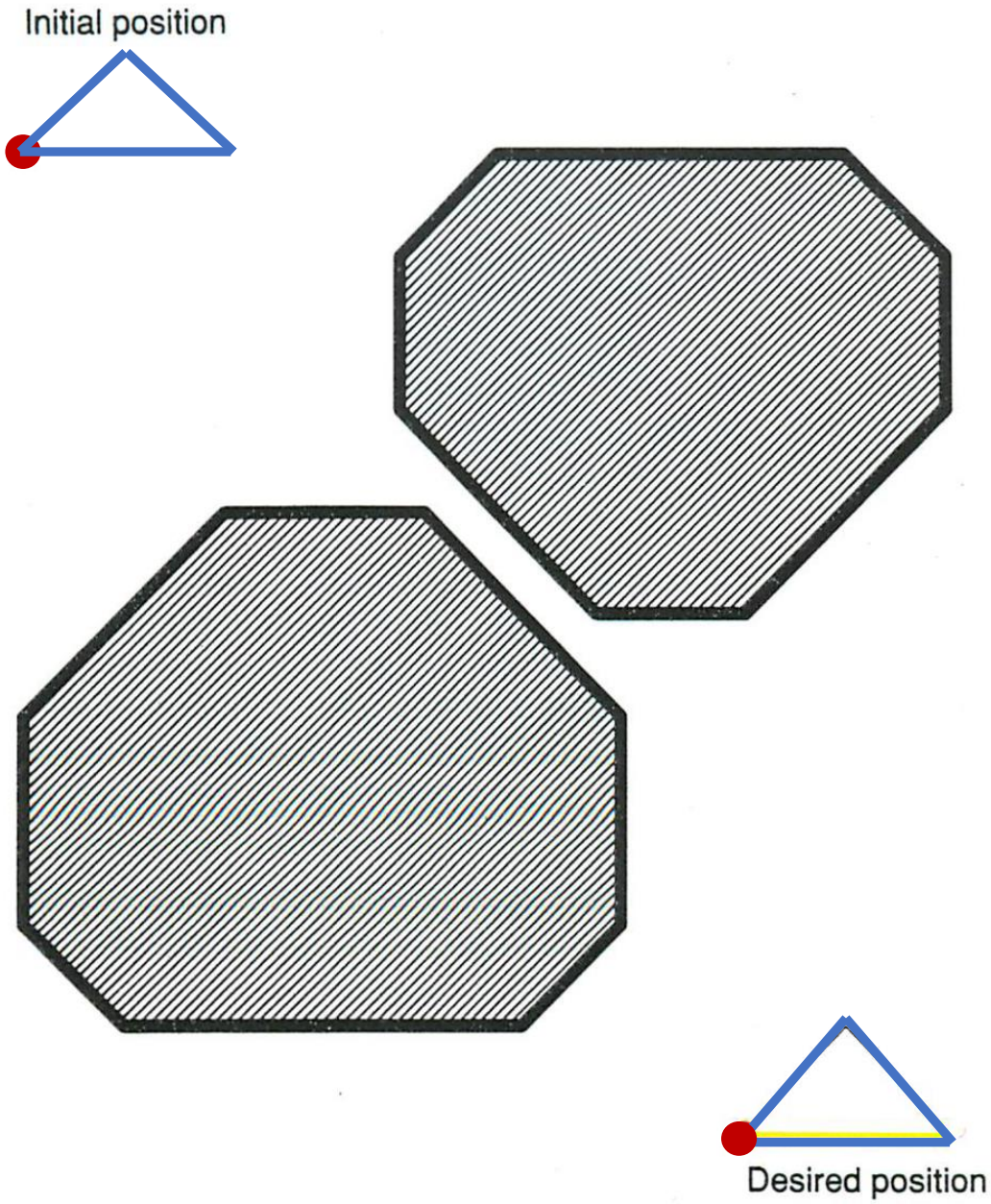
HERE:

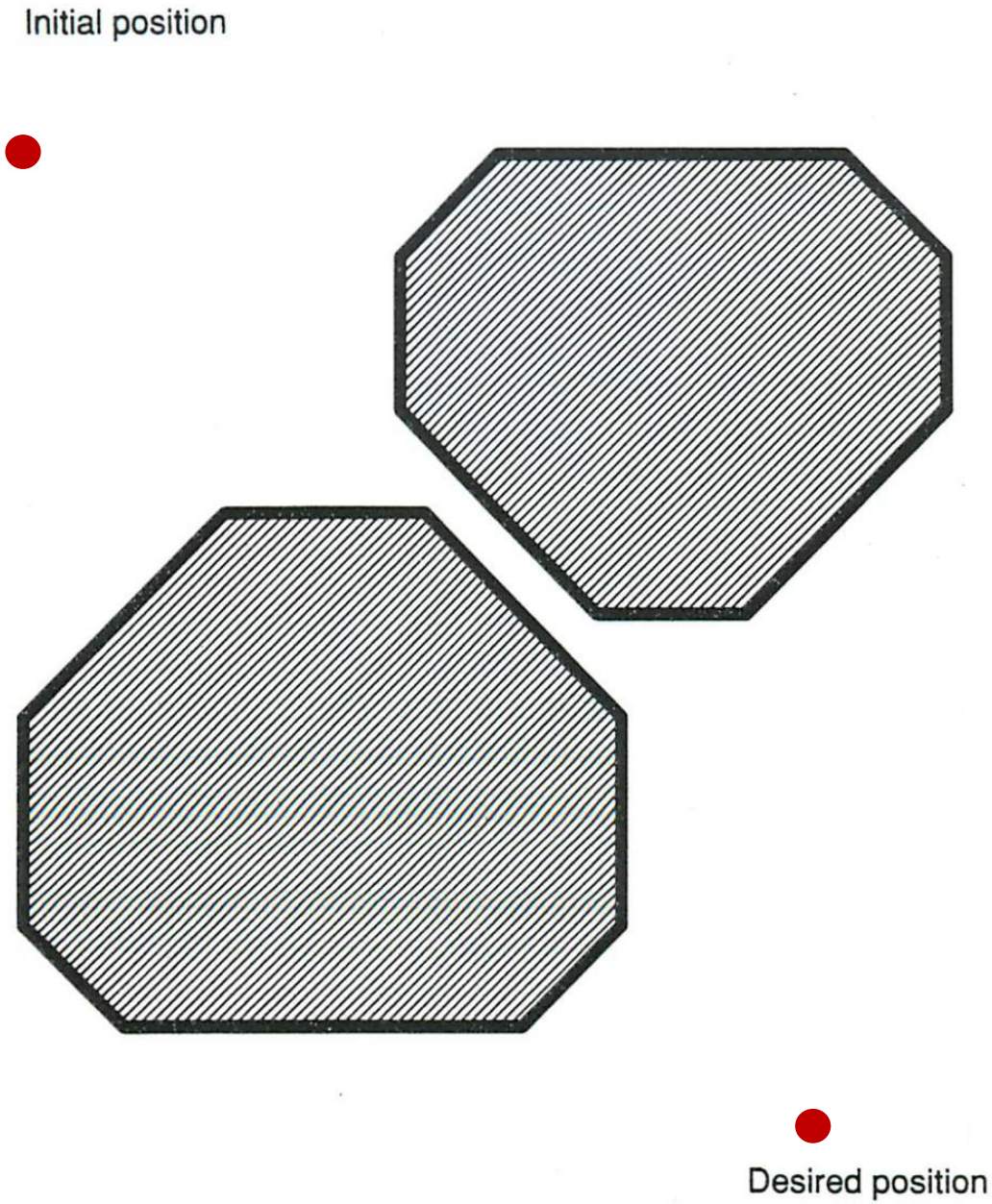
1. Reference point:

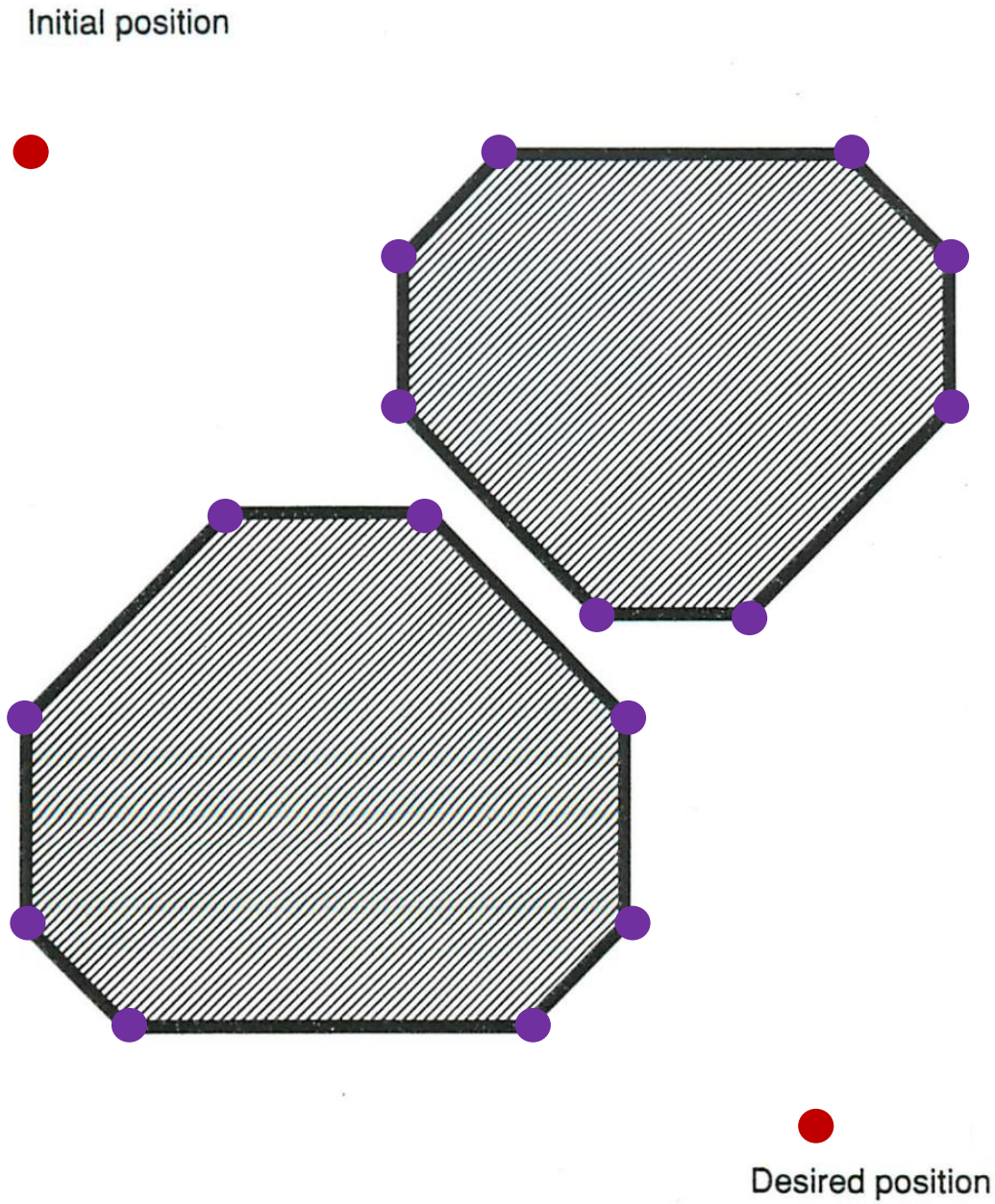
2. Eight unique shape positions

3. Fence: Thick black line

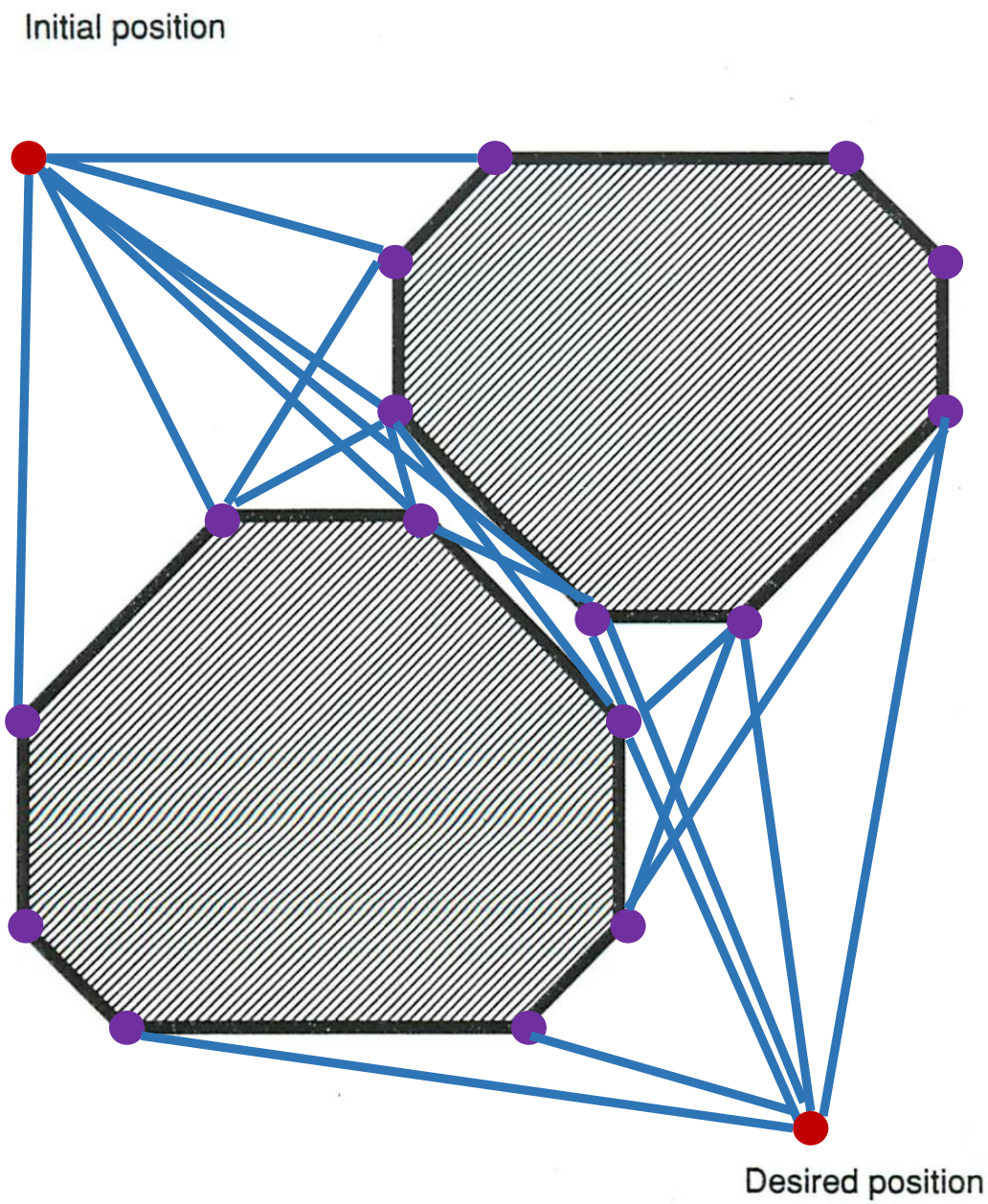



Initial position
Desired position

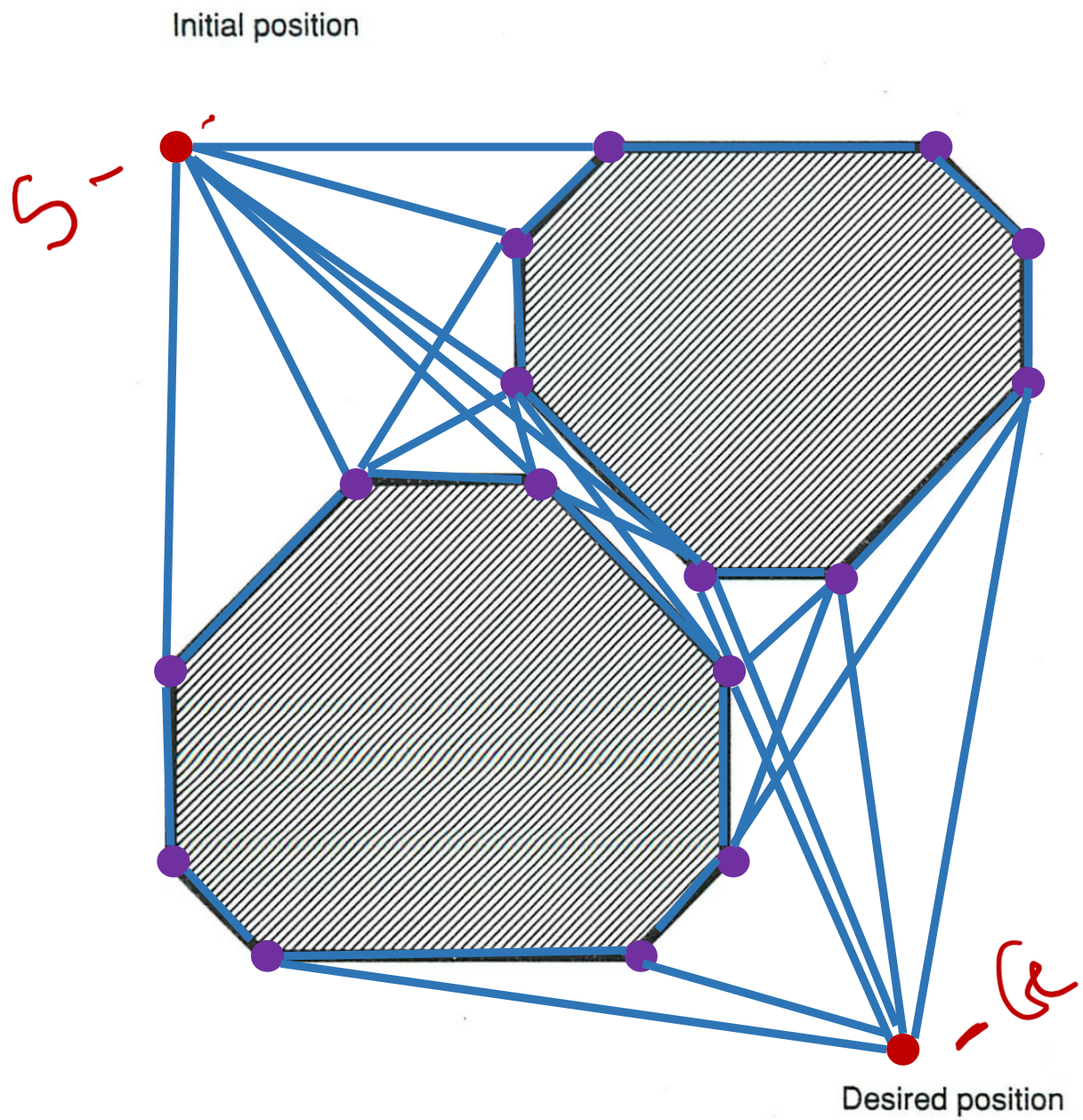

Initial position
Desired position


Initial position
Desired position

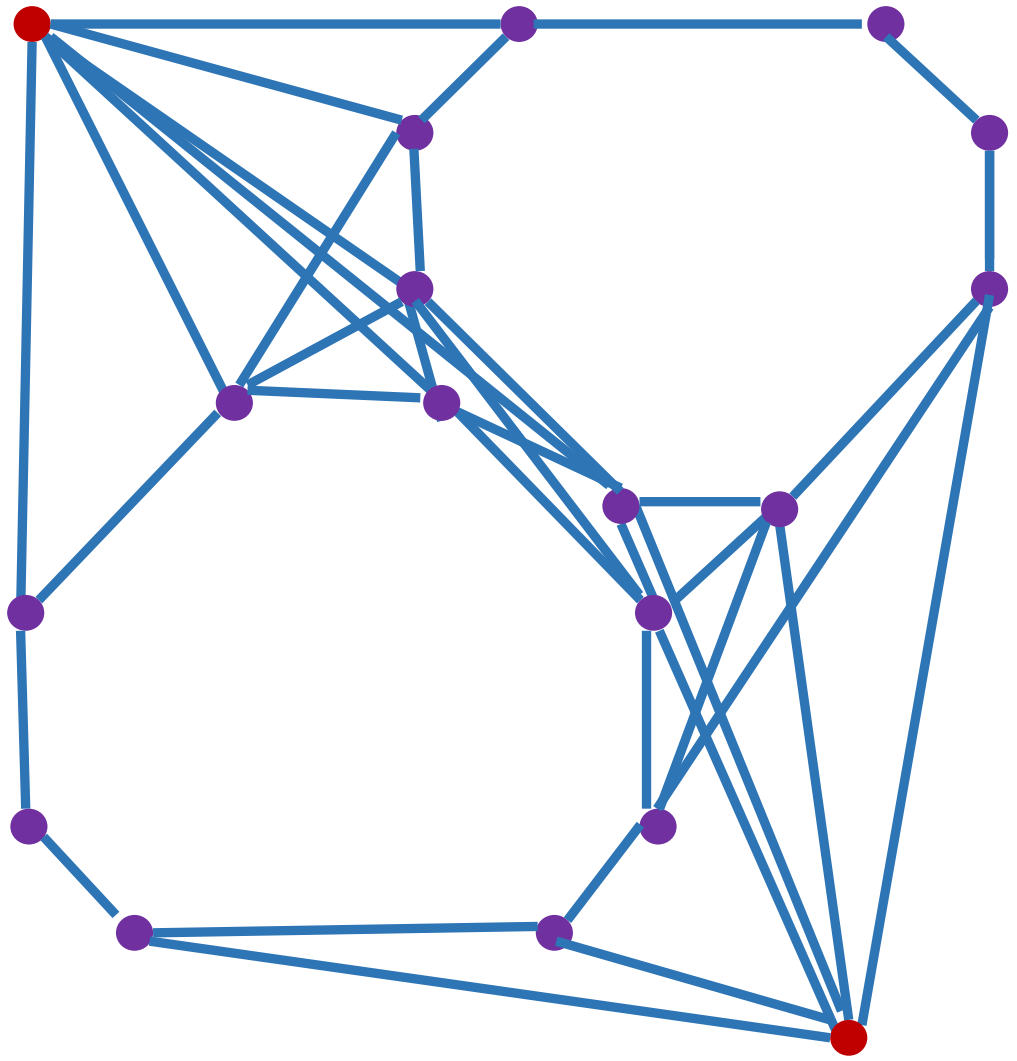
Visibility Graph



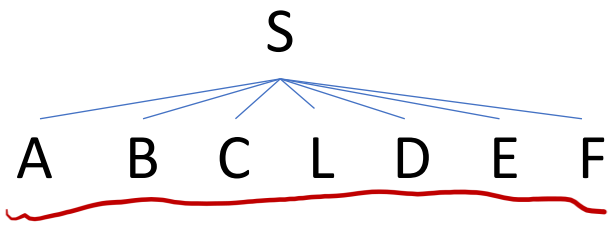
Visibility Graph



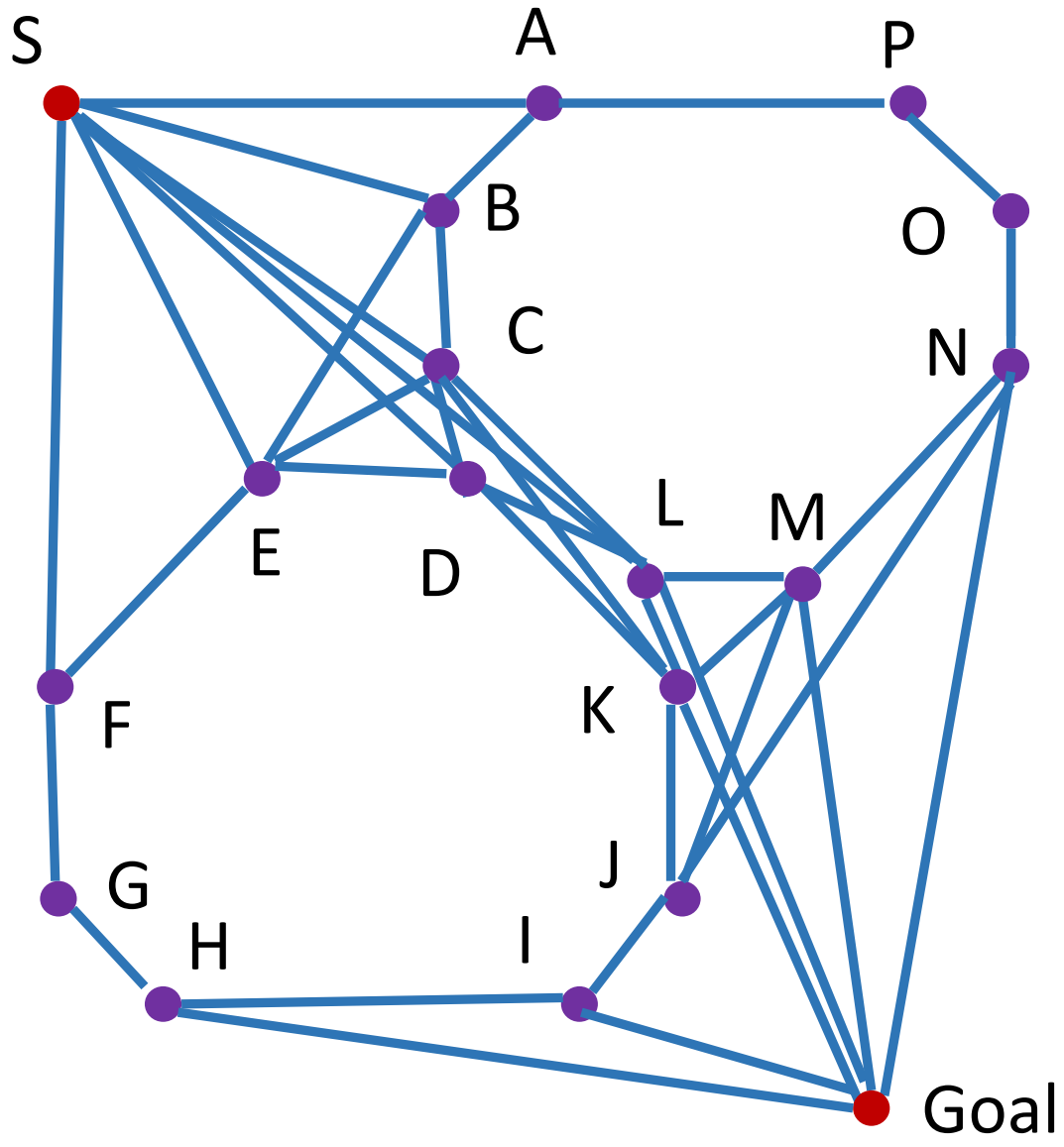
Visibility Graph



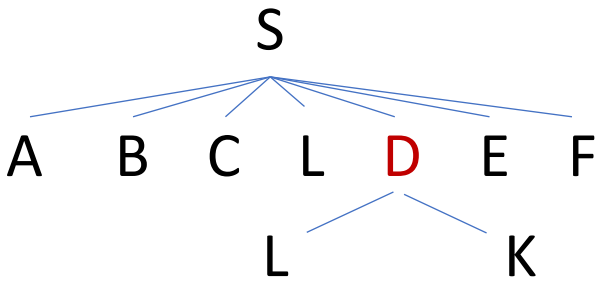
Run A* Algorithm on Visibility Graph:

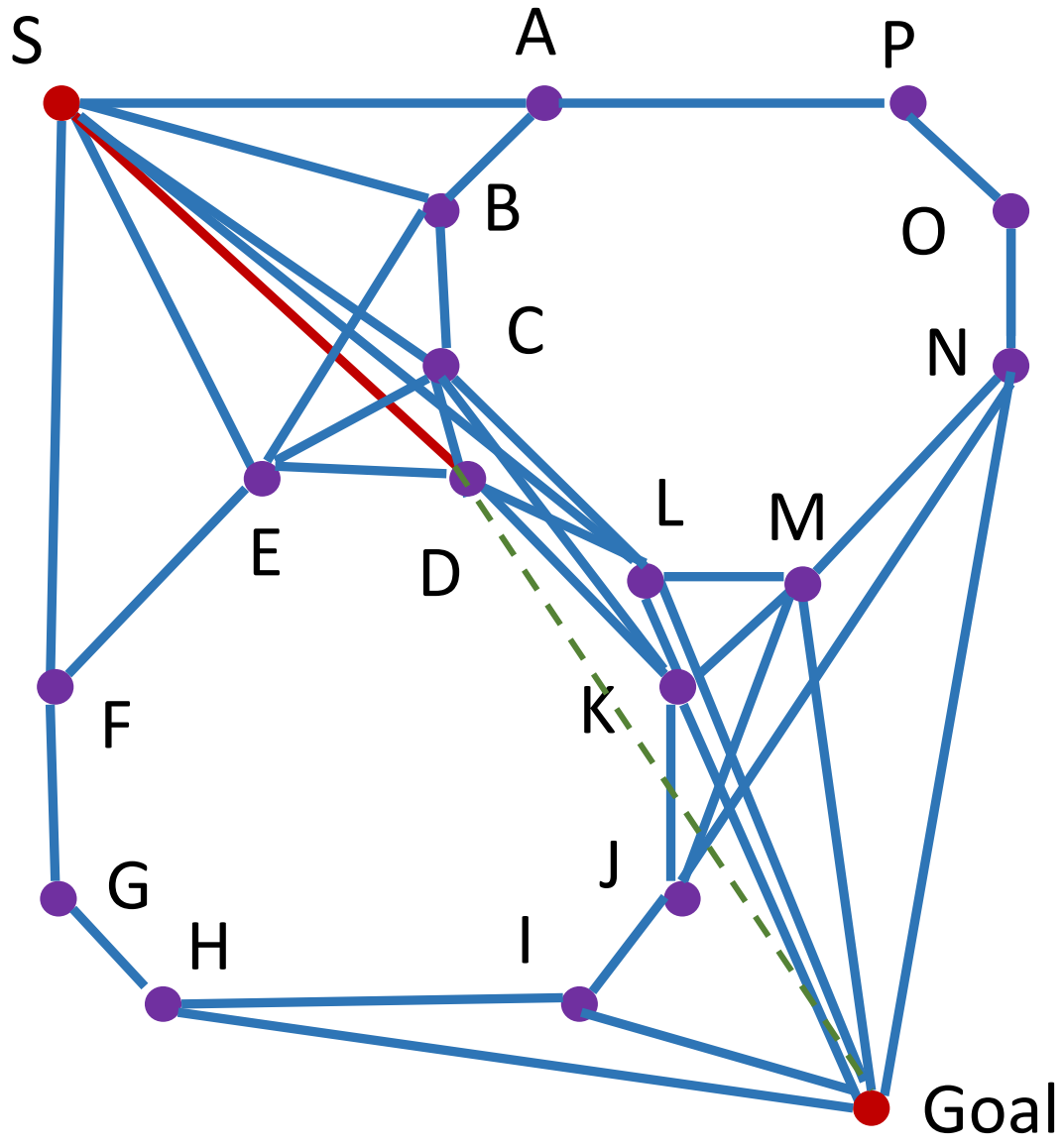


Node-to-node distance
+
Underestimate to Goal



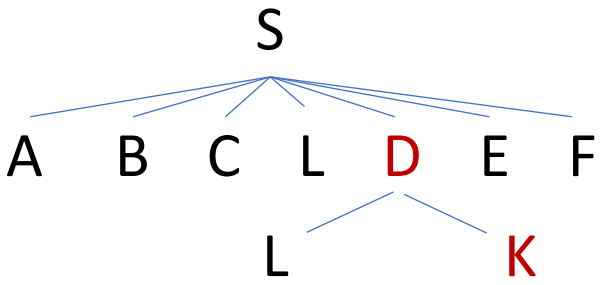
Run A* Algorithm on Visibility Graph:

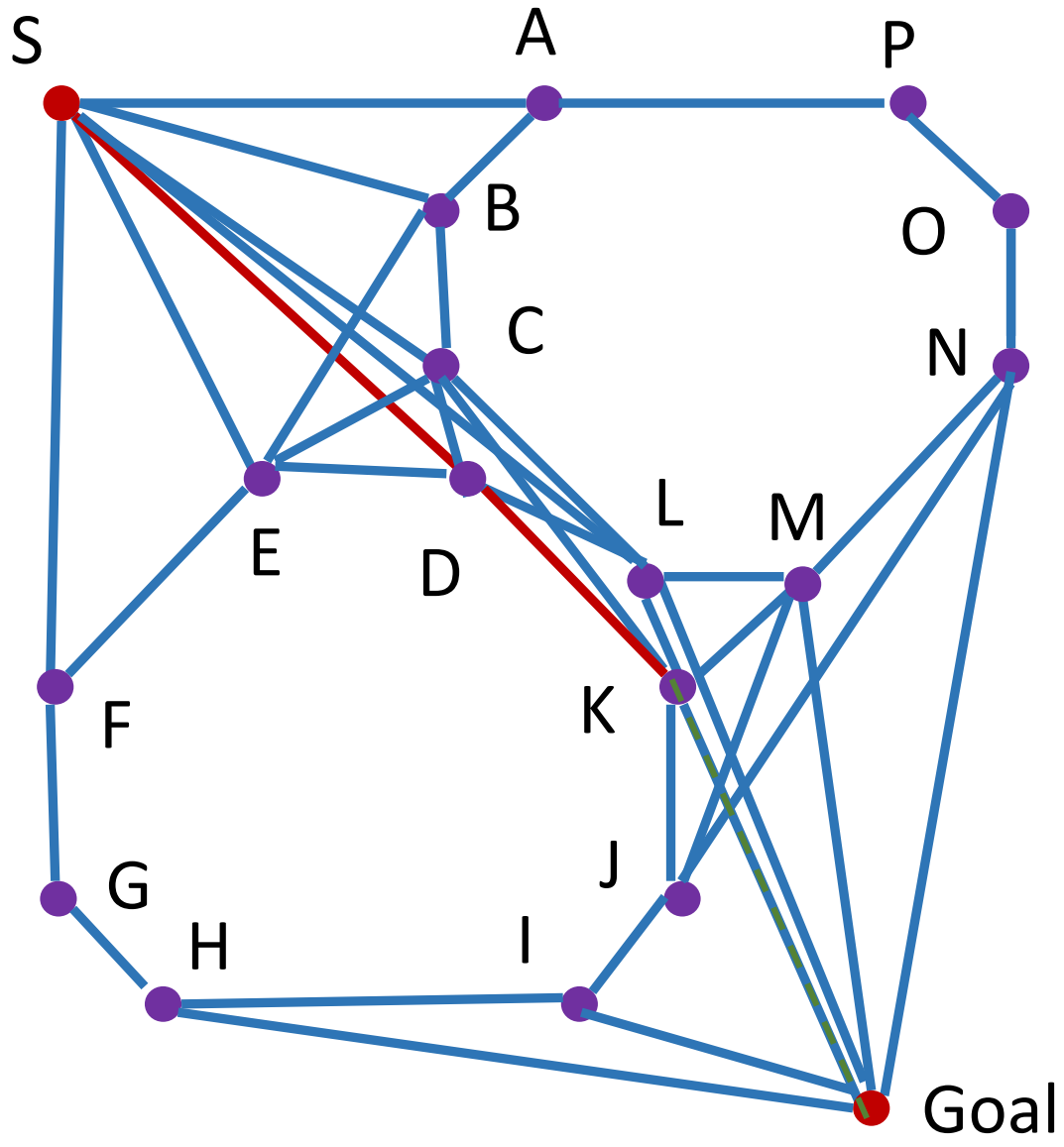




Node-to-node distance
+
Underestimate to Goal

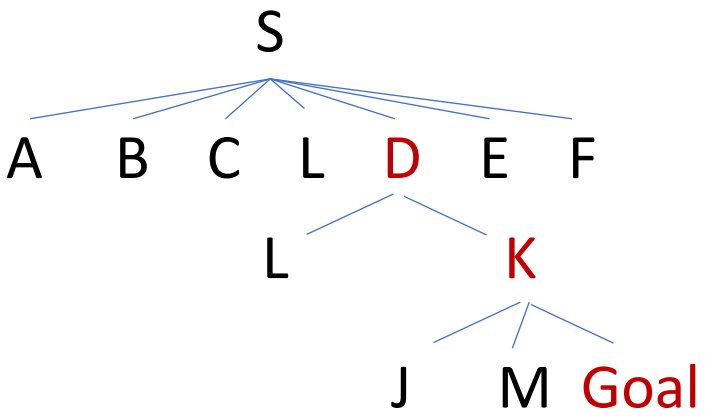
Run A* Algorithm on Visibility Graph:

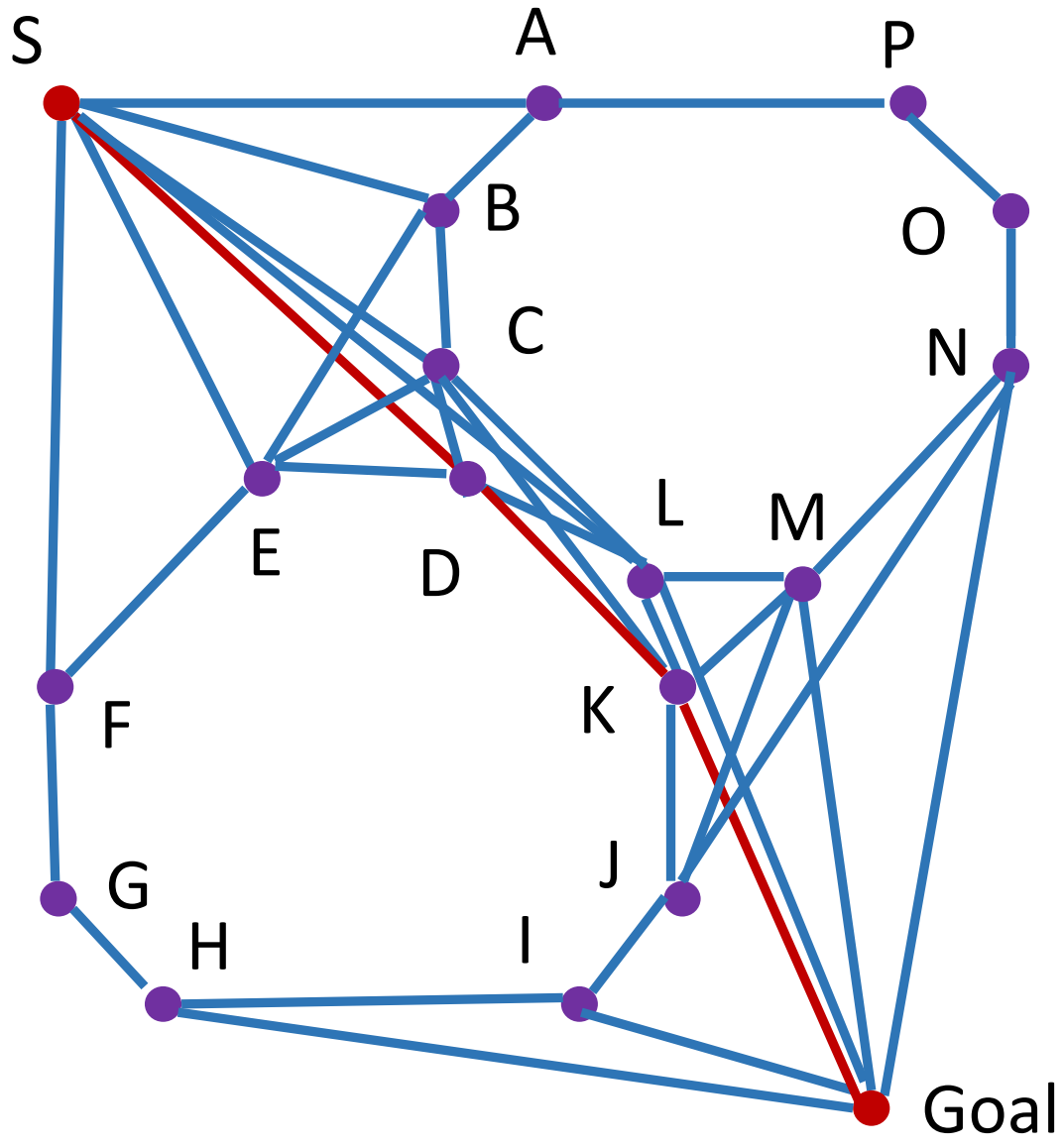




Node-to-node distance
+
Underestimate to Goal

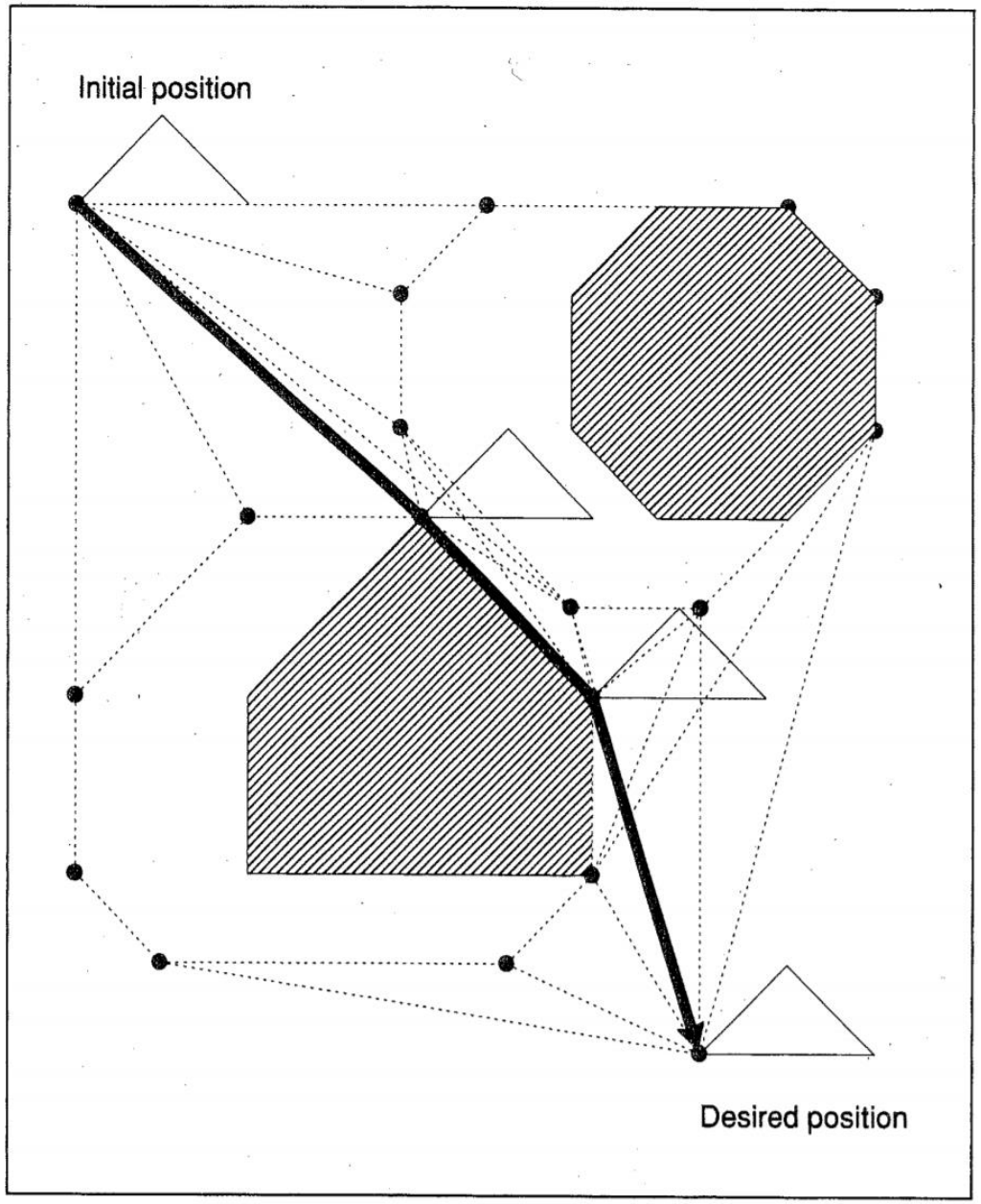
Run A* Algorithm on Visibility Graph:



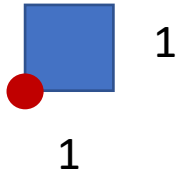


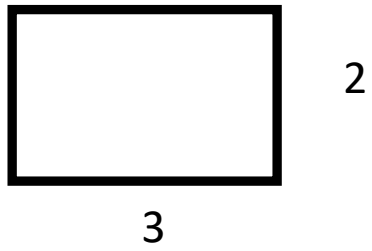
Node-to-node distance
+
Underestimate to Goal

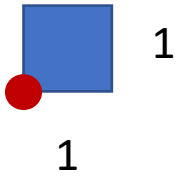
Shortest path computed by A*

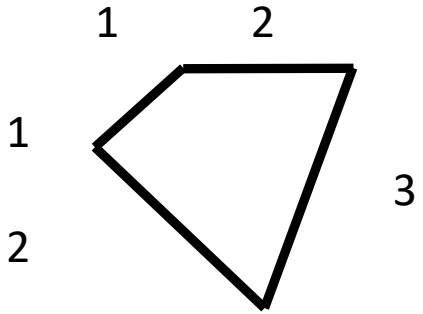


Your task:
Define Fence for the following robots & obstacles

1. Robot: 

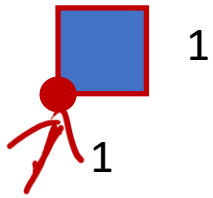
Obstacle: 

2. Robot: 

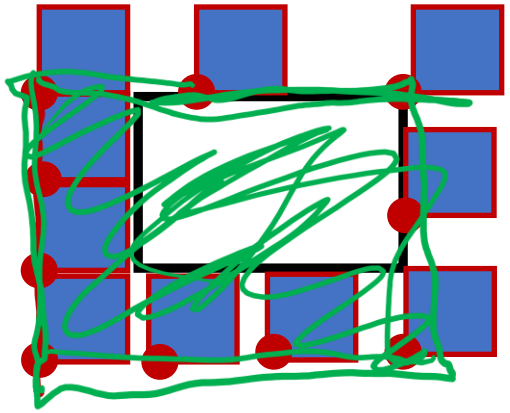
Obstacle: 

Define Fence for the following robots & obstacles

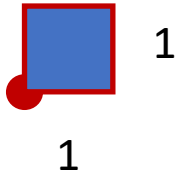
1. Robot:



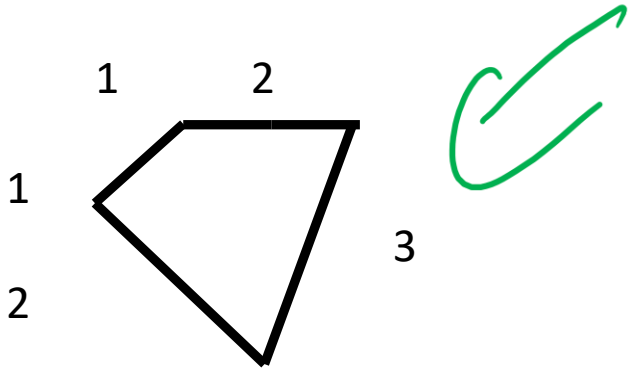
Obstacle:



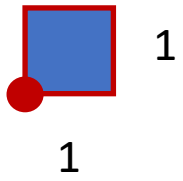
2. Robot:

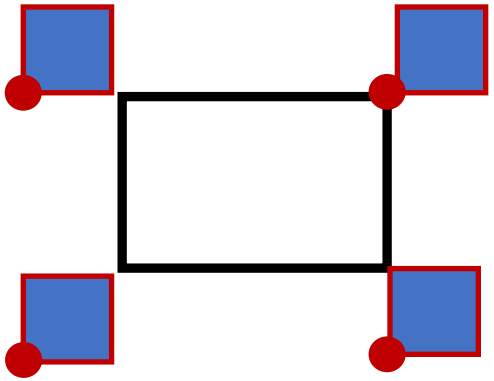


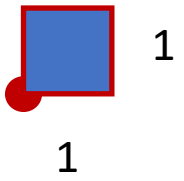
Obstacle:

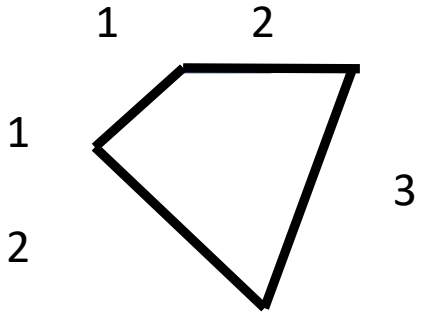


Define Fence for the following robots & obstacles

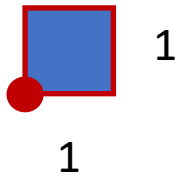
1. Robot: 

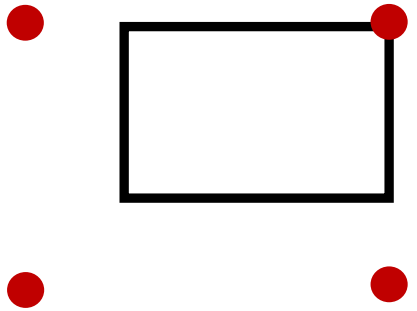
Obstacle: 

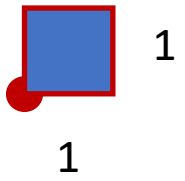
2. Robot: 

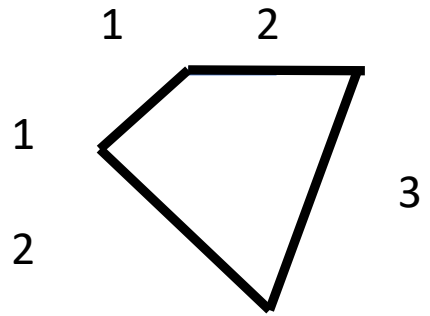
Obstacle: 

Define Fence for the following robots & obstacles

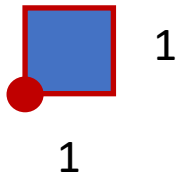
1. Robot: 

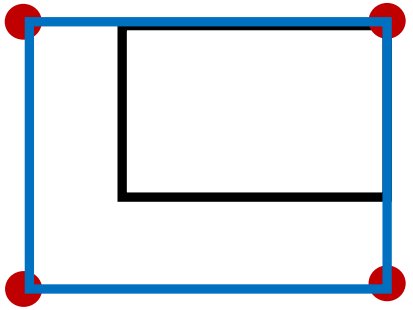
Obstacle: 

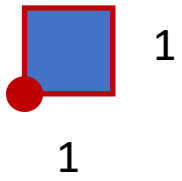
2. Robot: 

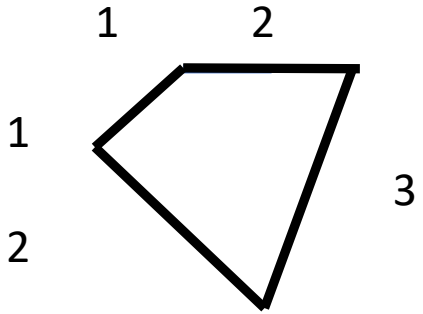
Obstacle: 

Define Fence for the following robots & obstacles

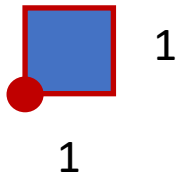
1. Robot: 

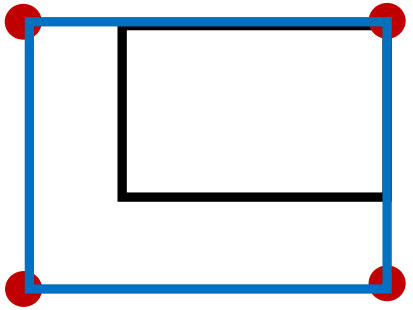
Obstacle: 

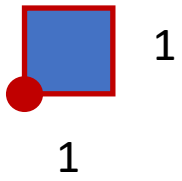
2. Robot: 

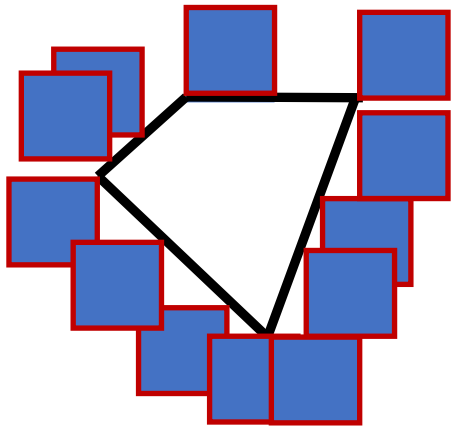
Obstacle: 

Define Fence for the following robots & obstacles

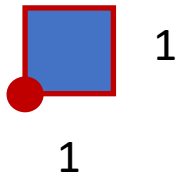
1. Robot: 

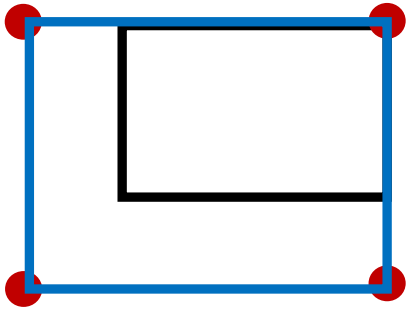
Obstacle: 

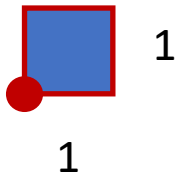
2. Robot: 

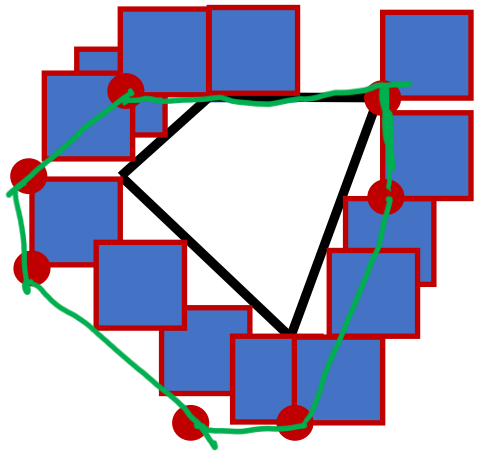
Obstacle: 

Define Fence for the following robots & obstacles

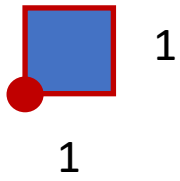
1. Robot: 

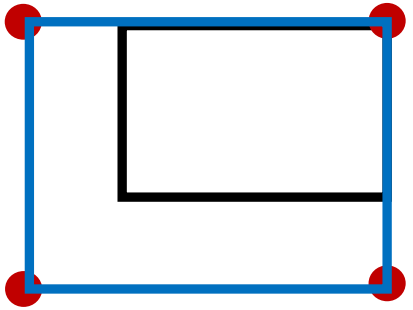
Obstacle: 

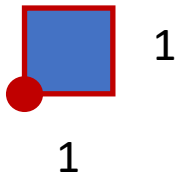
2. Robot: 

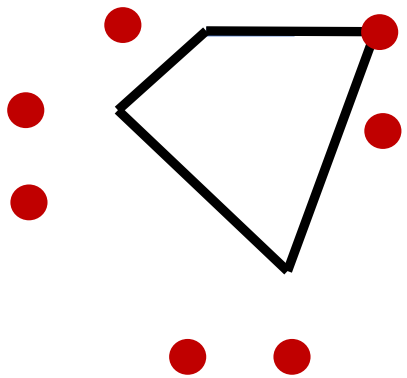
Obstacle: 

Define Fence for the following robots & obstacles

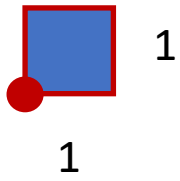
1. Robot: 

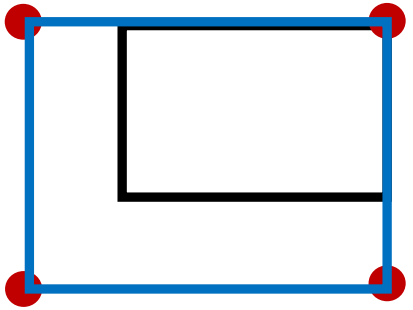
Obstacle: 

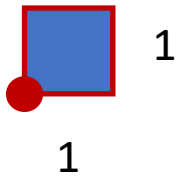
2. Robot: 

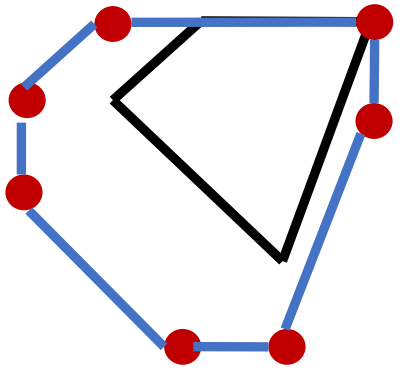
Obstacle: 

Fence for the following robots & obstacles:

1. Robot: 

Obstacle: 

2. Robot: 

Obstacle: 

General Problem of Robot Motion Planning

- Robot with k degrees of freedom:

  State or configuration of robot: $(q_1, q_2, \ldots, q_k)$

- So far (q_1, q_2) for two-dimensional position

- PUMA robot: 6 joint angles: $(q_1, q_2, \ldots, q_6)$

  6D configuration space

General Problem of Robot Motion Planning

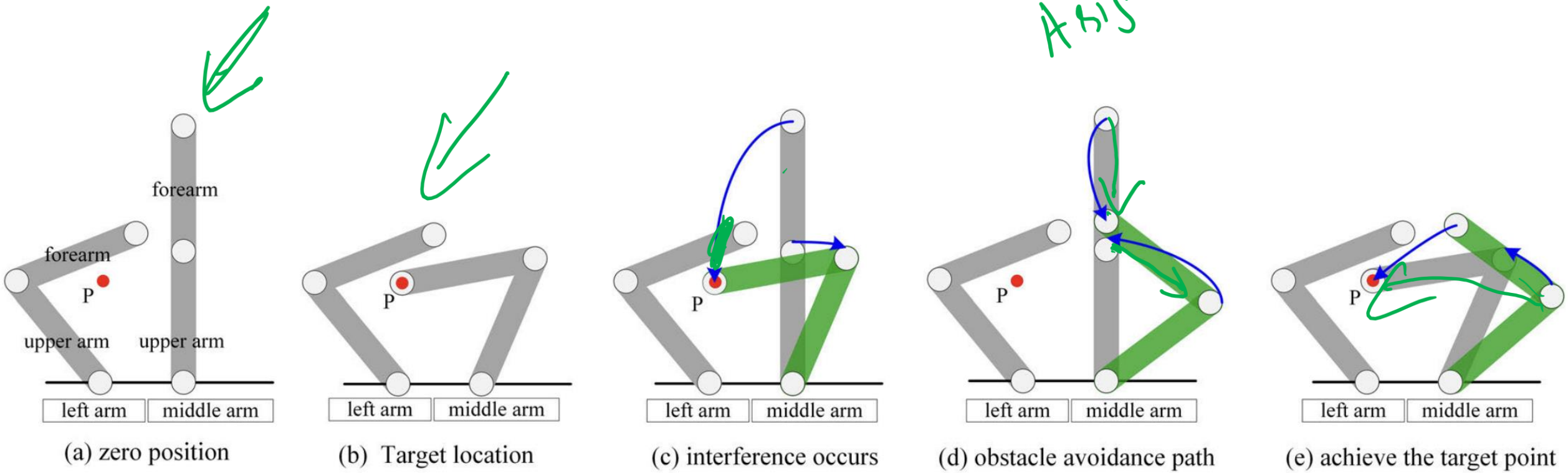
Given initial point c_1 and destination point c_2, in configuration space C:

Robot can safely move between corresponding points in physical space

if and only if

There exists a continuous path between c_1 and c_2 that lies entirely in the free space.

Robot Obstacle Avoidance



Zhao et al., 2020

Learning Outcomes of this Lecture

- Understand how search algorithms are evaluated
- Understand the unique properties of AI searching tasks (versus general search algorithms)
- Can explain 11 path-based search algorithms and run them on an example
- Can explain the dynamic programming principle
- Know what an admissible and a monotone heuristic function is for the A* algorithm
- Can design a configuration space from a 2D obstacle map & a translating robot
- Can design a visibility graph in free space
- Can run A* on a visibility graph for robot path planning
- Understand configuration spaces of robot arms