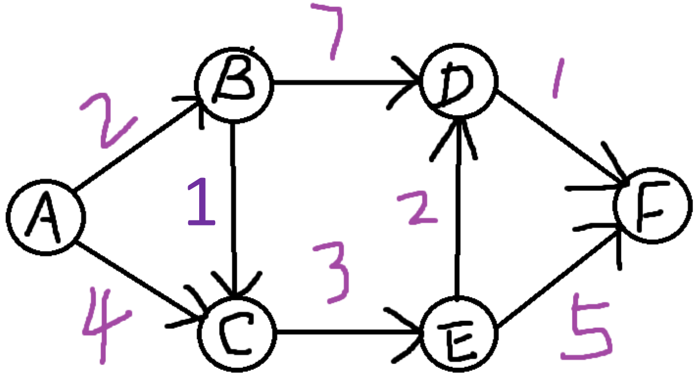
CS 180 Discussion 1A/E

Week 4

Haoxin Zheng

10/27/2023

Shortest path



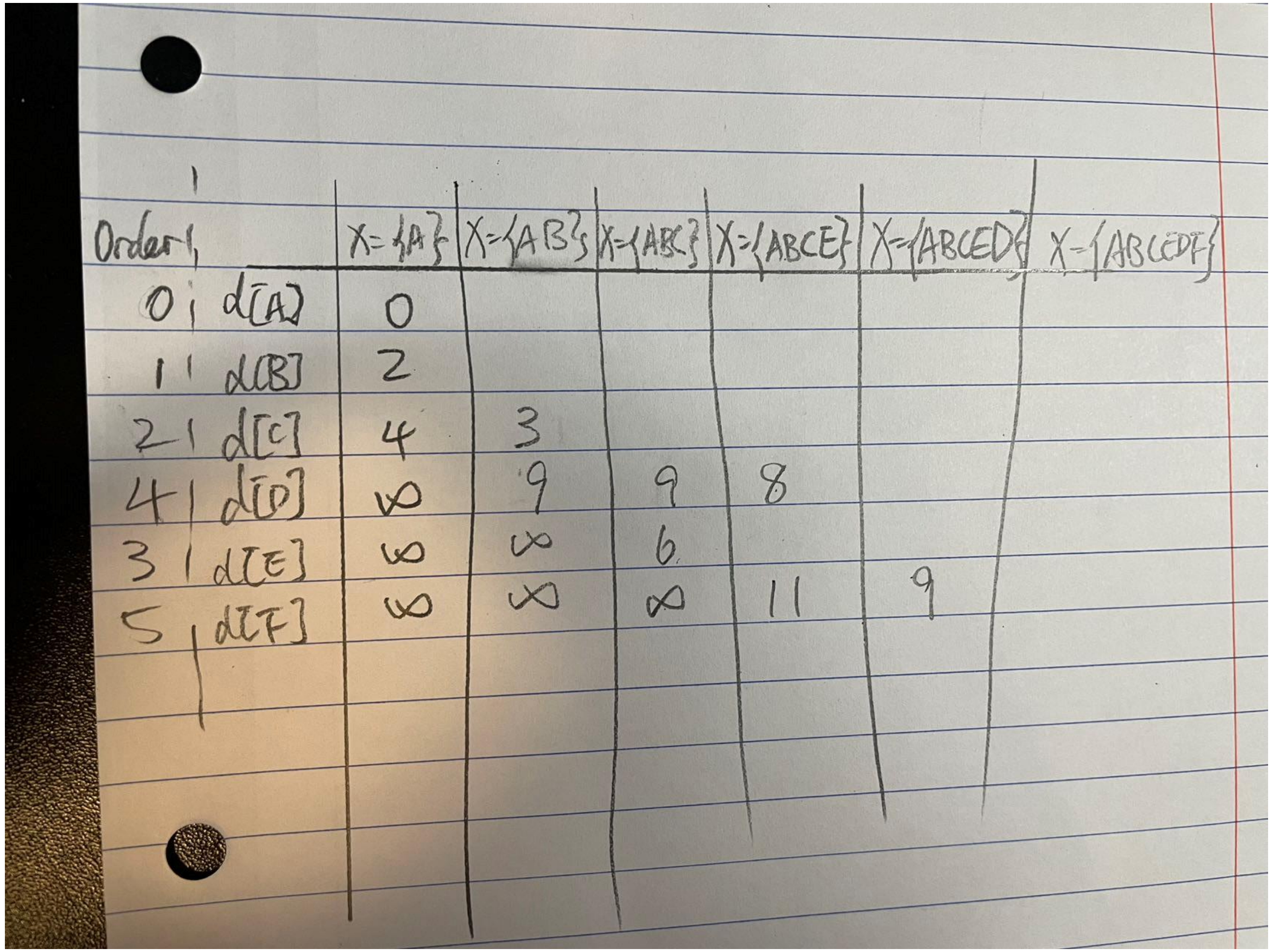
- Problem:
  - Directed/Undirected graph, with non-neg edges
  - Given node S, find shortest paths from S to all other nodes
- Algorithm: Dijkstra's algorithm
  - Initialization:
    - X:{s}
    - d[u] = $\begin{cases} l(s,u), & if (s,u)\in E \\ \infty, & otherwise \end{cases}$
  - For i=1, … n-1
    - Select u s.t. d[u] is the min among V-X
    - X = X + {u}
    - For each v s.t. (u, v)∈E:
      - If d[u] + l(u, v) < d[v]:
        - d[v] = d[u] + l(u, v)
        - pre[v]=u

- Great instruction:
  - https://www.youtube.com/watch?v=CerlT7tTZfY

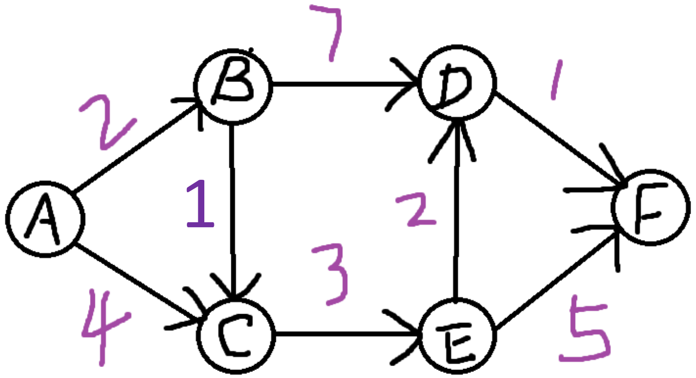
Shortest path

- Initialization:
  - X:{s}
  - d[u] = $\begin{cases} l(s,u), & if(s,u)\in E \\ \infty, & otherwise \end{cases}$
- For i=1, … n-1
  - Select u s.t. d[u] is the min among V-X
  - X = X + {u}
  - For each v s.t. (u, v)∈E:
    - If d[u] + l(u, v) < d[v]:
      - d[v] = d[u] + l(u, v)
      - pre[v]=u

- Time complexity using Heap?
  - O((m+n)logn) -> O(mlogn)



Order		X={A}	X={AB}	X={ABC}	X={ABCE}	X={ABCED}	X={ABCEDF}
0	d[A]	0					
1	d[B]	2					
2	d[C]	4	3				
4	d[D]	∞	9	9	8		
3	d[E]	∞	∞	6			
5	d[F]	∞	∞	∞	11	9	

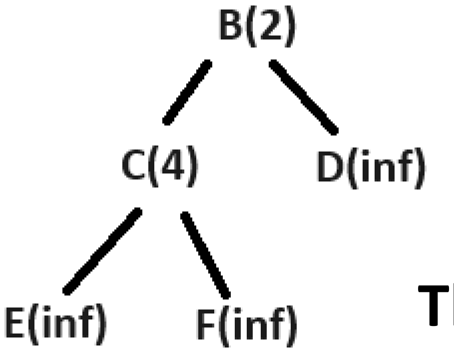


Shortest path

- Initialization:
  - X:{s}
  - d[u] = $\begin{cases} l(s,u), & if (s,u) \in E \\ \infty, & otherwise \end{cases}$
- For i=1, … n-1
  - Select u s.t. d[u] is the min among V-X
  - X = X + {u}
  - For each v s.t. (u, v)∈E:
    - If d[u] + l(u, v) < d[v]:
      - d[v] = d[u] + l(u, v)
      - pre[v]=u
- Time complexity using Heap?
  - O((m+n)logn) -> O(mlogn)

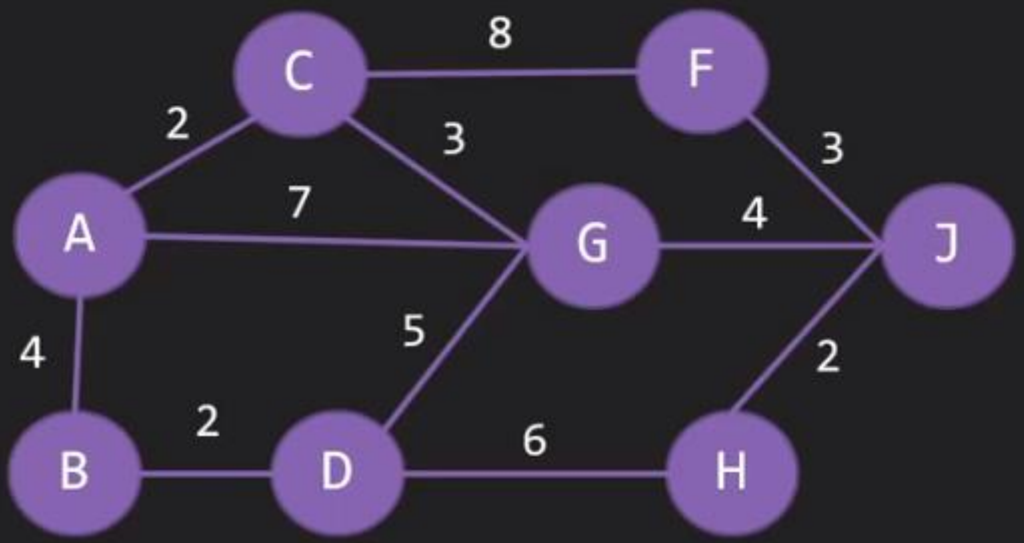
Order		X={A}	X={AB}	X={ABC}	X={ABCE}	X={ABCED}	X={ABCEDF}
0	d[A]	0					
1	d[B]	2					
2	d[C]	4	3				
4	d[D]	∞	9	9	8		
3	d[E]	∞	∞	6			
5	d[F]	∞	∞	∞	11	9	

Initialization:



Then: White Board

Initial State



Priority Queue

From

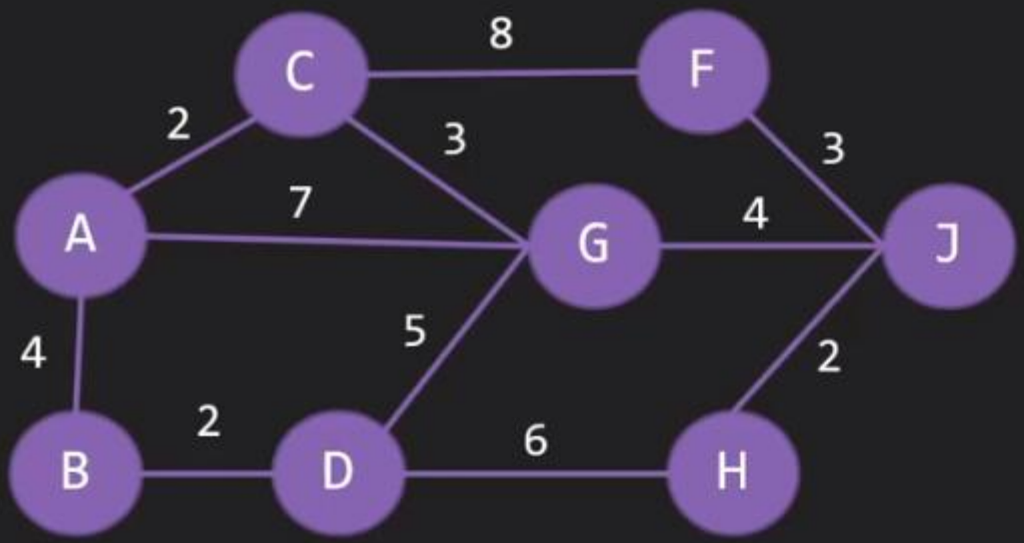
To

Cost

Path and Cost Arrays

A	-	-	✓
B			
C			
D			
F			
G			
H			
J			

Check Loop Conditions



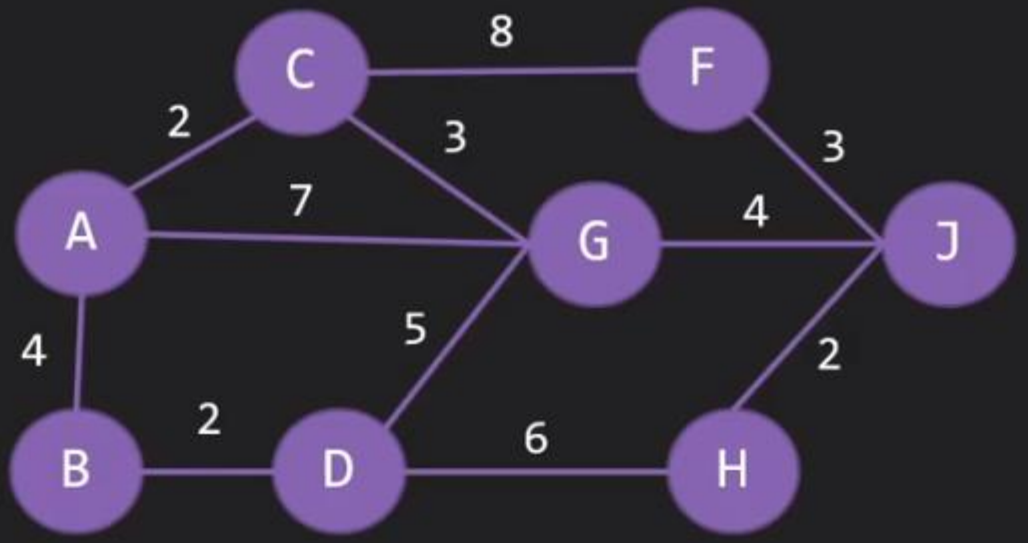
Priority Queue

From	A	A	A
To	C	B	G
Cost	2	4	7

Path and Cost Arrays

A	-	-
B		
C		
D		
F		
G		
H		
J		

Handle Min Item from Priority Queue



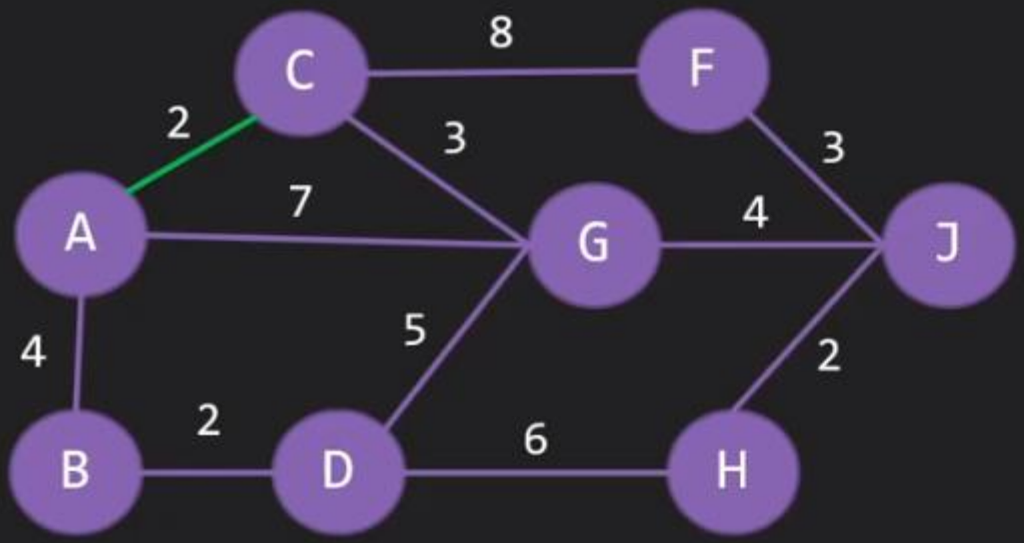
Priority Queue

From	A	A	A
To	C	B	G
Cost	2	4	7

Path and Cost Arrays

A	-	-	✓
B			
C	A	2	✓
D			
F			
G			
H			
J			

Handle Edge from C to A—No Action



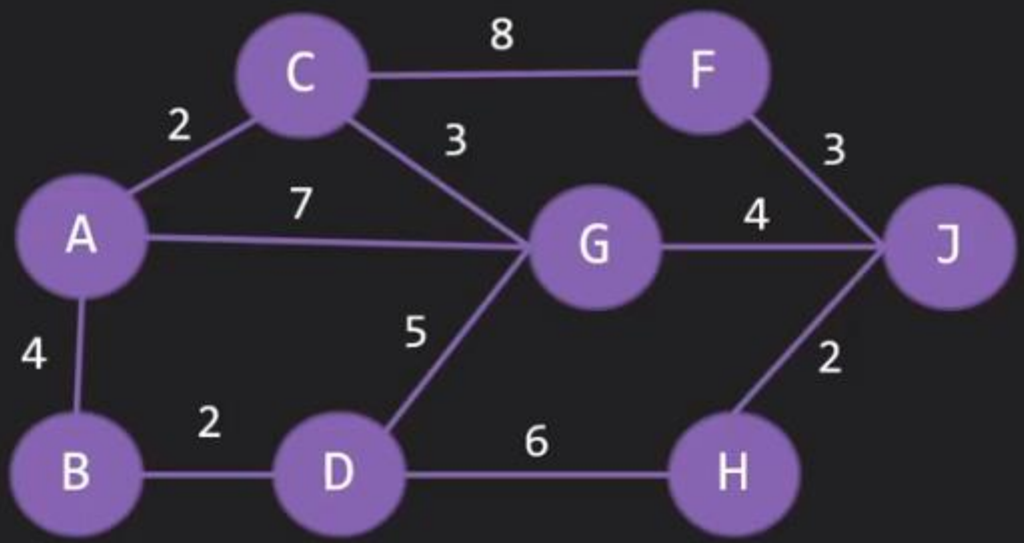
Priority Queue

From	A	A
To	B	G
Cost	4	7

Path and Cost Arrays

A	-	-	✓
B			
C	A	2	✓
D			
F			
G			
H			
J			

Check Loop Conditions



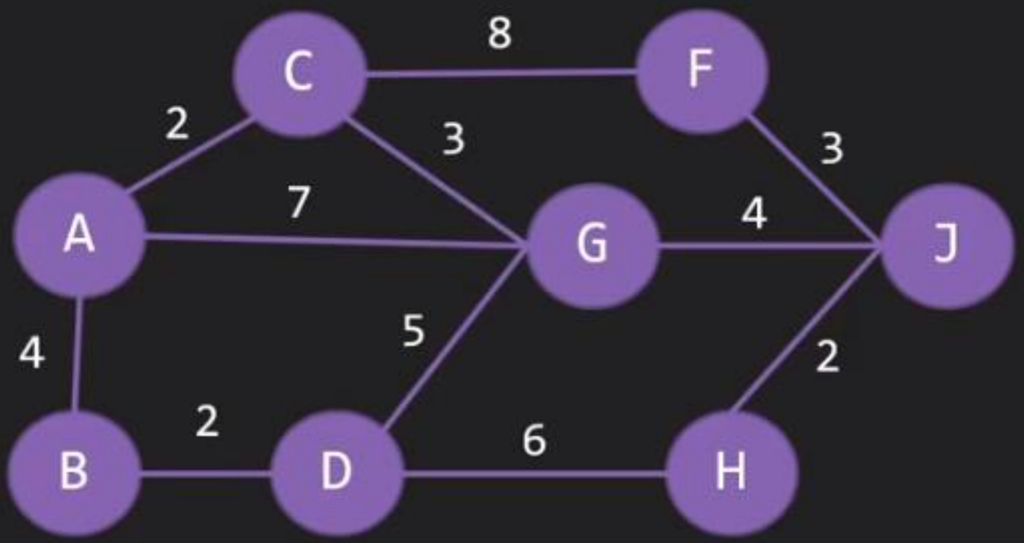
Priority Queue

From	A	C	A	C
To	B	G	G	F
Cost	4	5	7	10

Path and Cost Arrays

A	-	-	✓
B			
C	A	2	✓
D			
F			
G			
H			
J			

Handle Min Item from Priority Queue



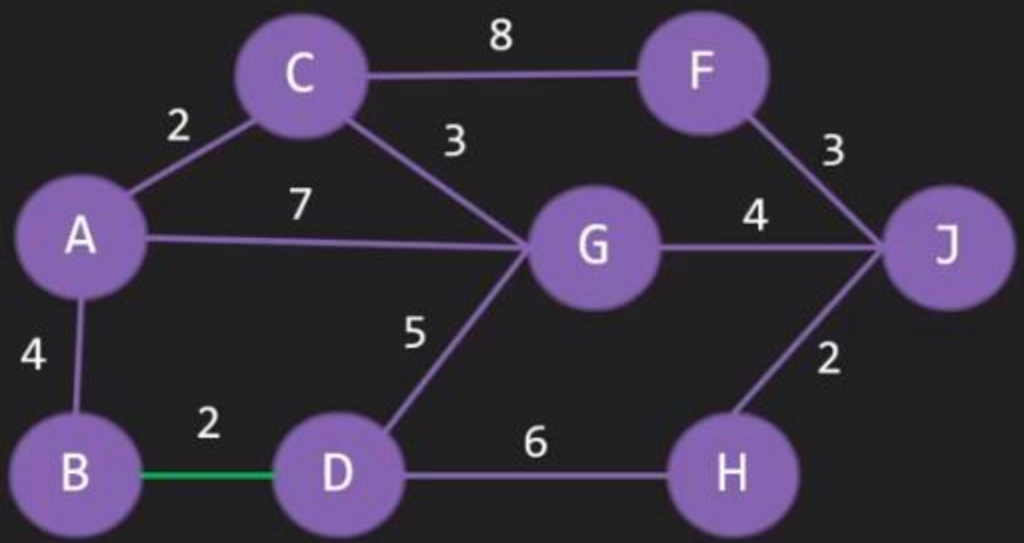
Priority Queue

From	A	C	A	C
To	B	G	G	F
Cost	4	5	7	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G			
H			
J			

Handle Edge from B to D



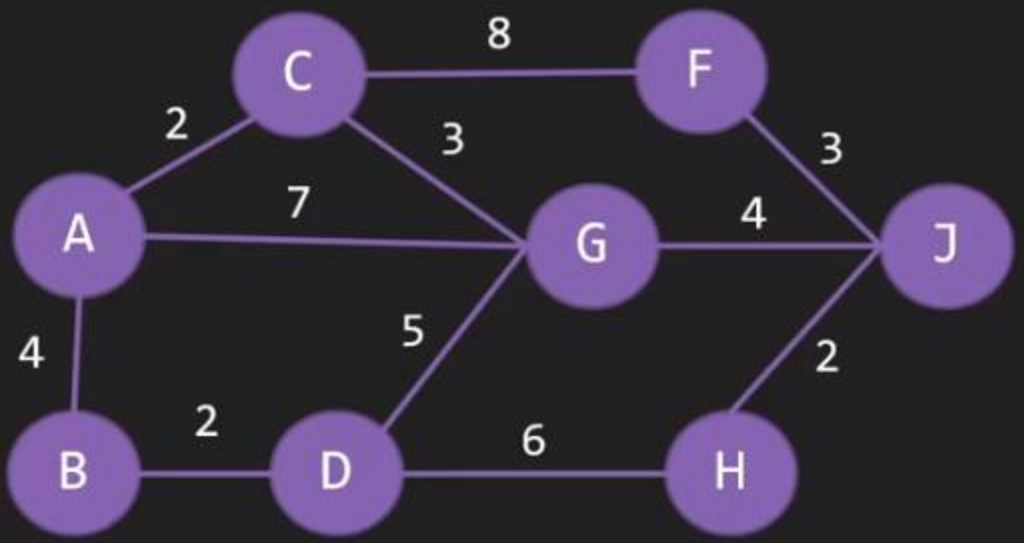
Priority Queue

From	C	B	A	C
To	G	D	G	F
Cost	5	6	7	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G			
H			
J			

Handle Min Item from Priority Queue



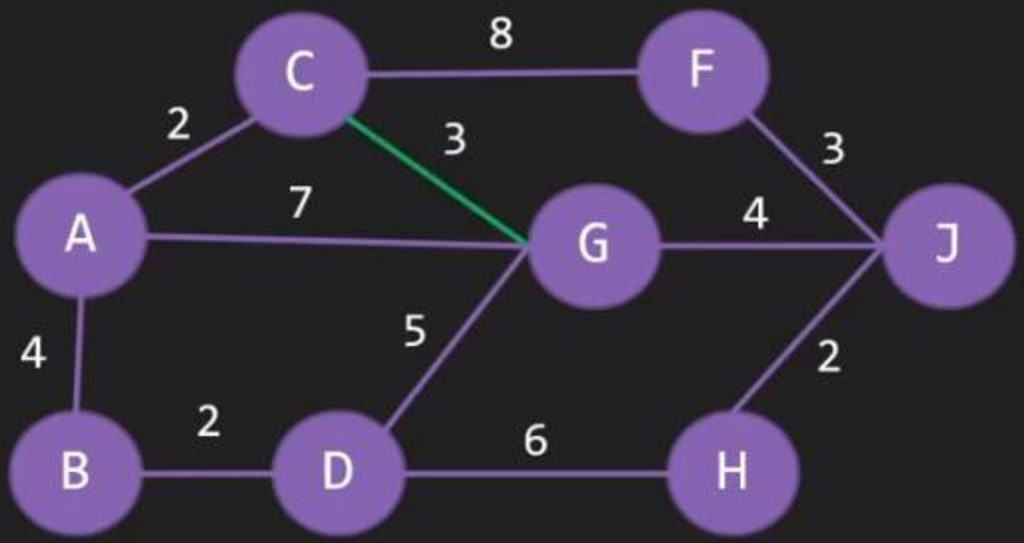
Priority Queue

From	C	B	A	C
To	G	D	G	F
Cost	5	6	7	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G	C	5	✓
H			
J			

Handle Edge from G to C—No Action



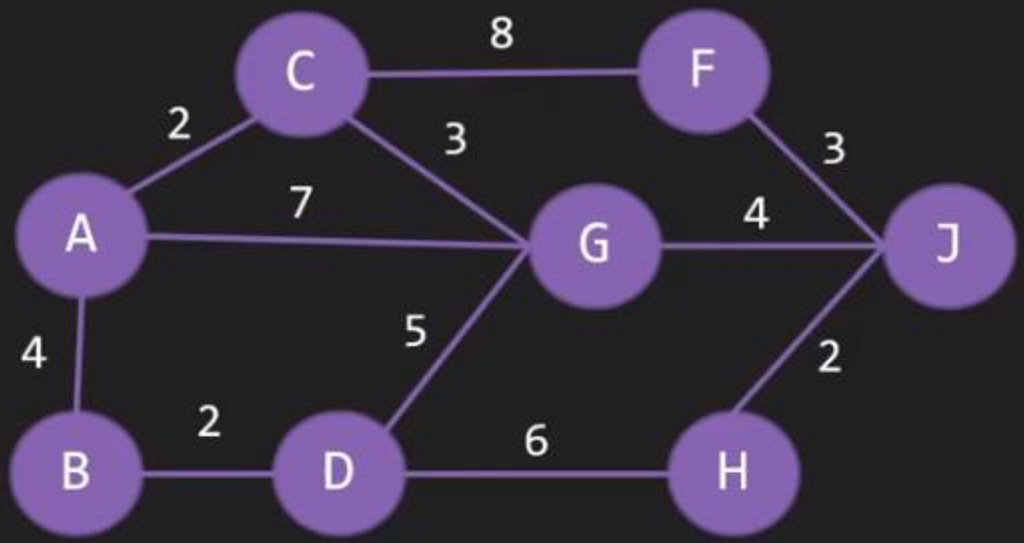
Priority Queue

From	B	A	C
To	D	G	F
Cost	6	7	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G	C	5	✓
H			
J			

Check Loop Conditions



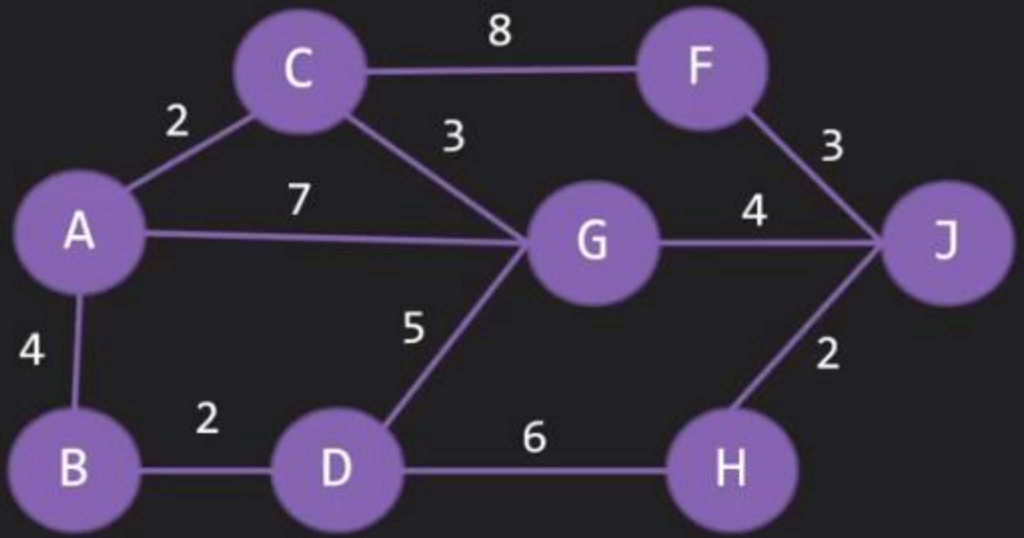
Priority Queue

From	B	A	G	C	G
To	D	G	J	F	D
Cost	6	7	9	10	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D			
F			
G	C	5	✓
H			
J			

Handle Min from Priority Queue



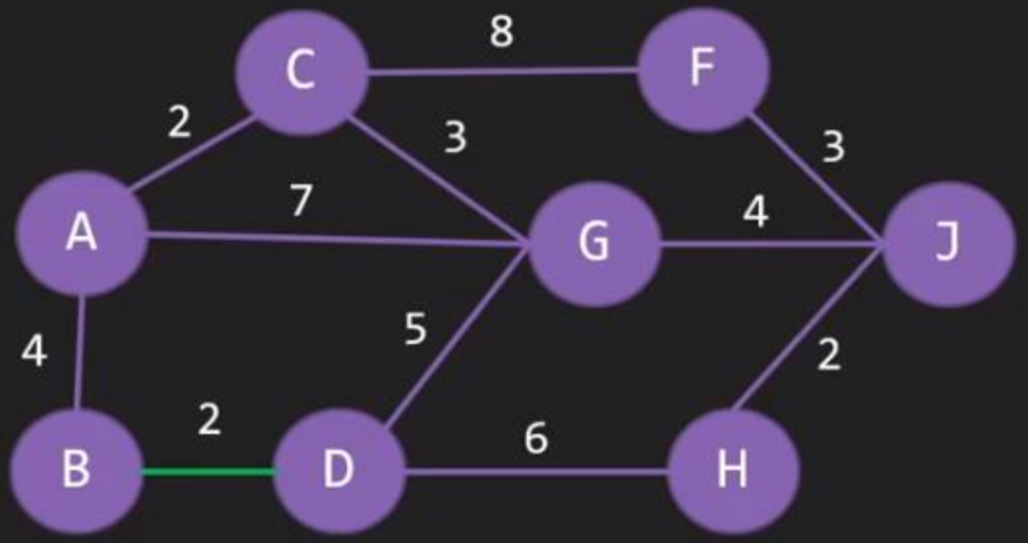
Priority Queue

From	B	A	G	C	G
To	D	G	J	F	D
Cost	6	7	9	10	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F			
G	C	5	✓
H			
J			

Handle Edge from D to B—No Action



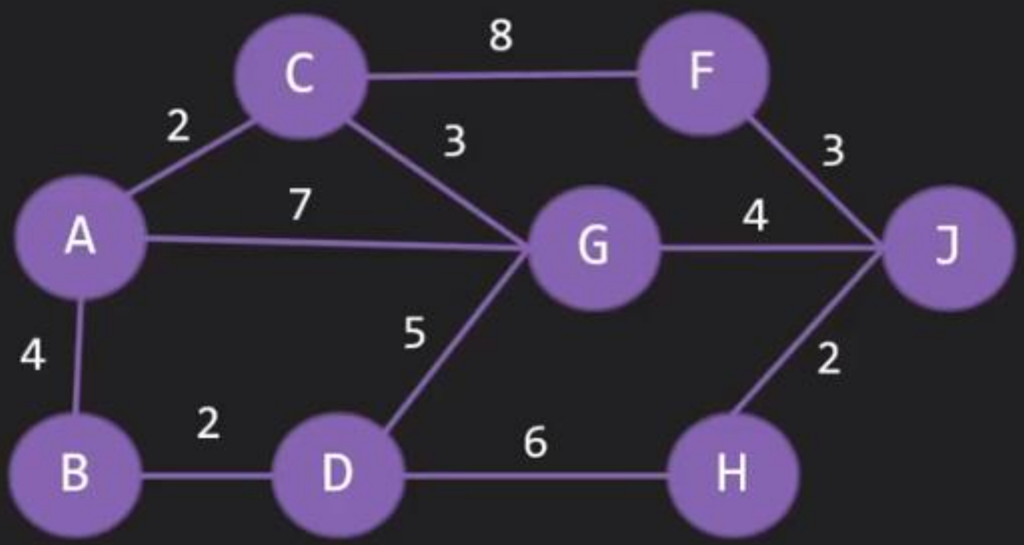
Priority Queue

From	A	G	C	G
To	G	J	F	D
Cost	7	9	10	10

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F			
G	C	5	✓
H			
J			

Check Loop Conditions



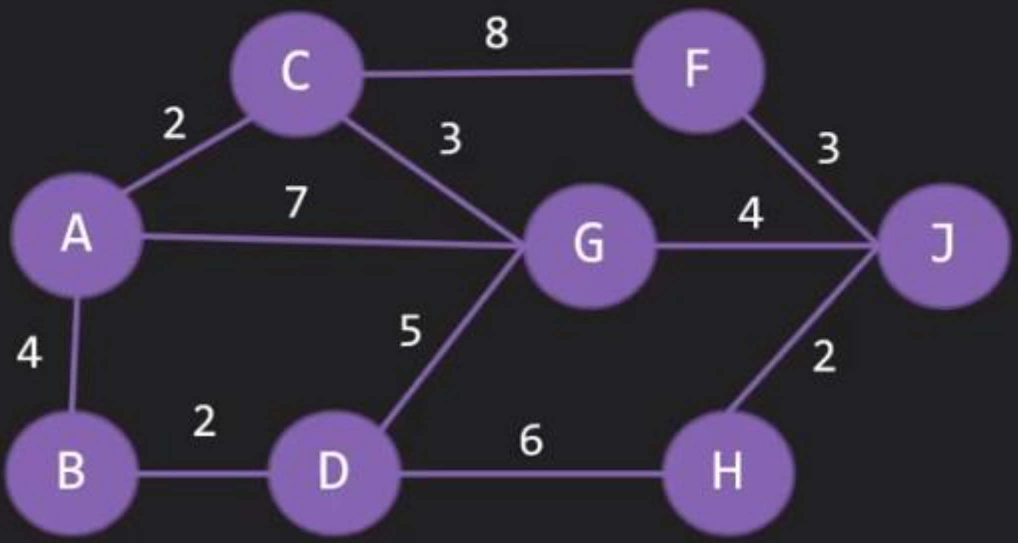
Priority Queue

From	A	G	C	G	D
To	G	J	F	D	H
Cost	7	9	10	10	12

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F			
G	C	5	✓
H			
J			

Handle Min from Priority Queue—No Action



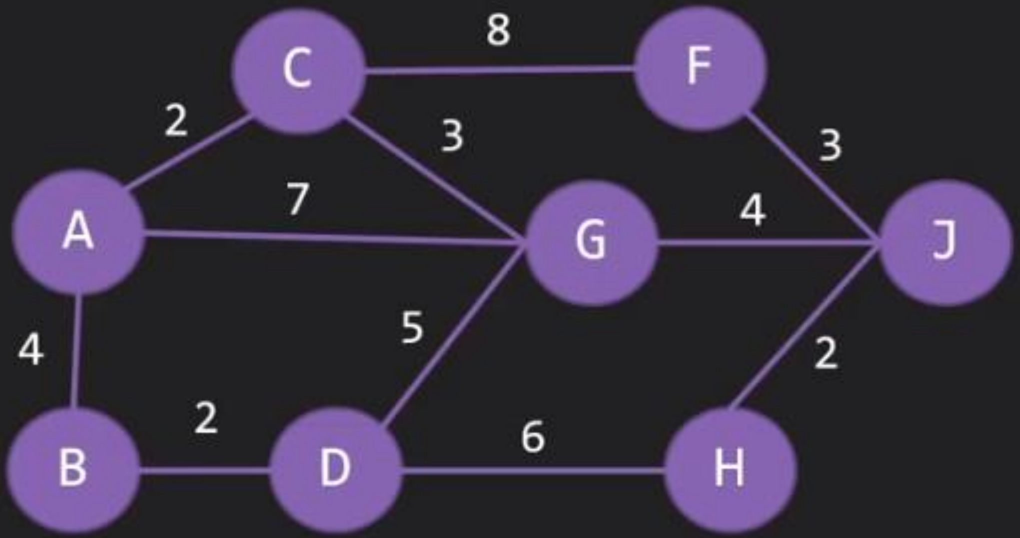
Priority Queue

From	A	G	C	G	D
To	G	J	F	D	H
Cost	7	9	10	10	12

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F			
G	C	5	✓
H			
J			

Handle Min from Priority Queue



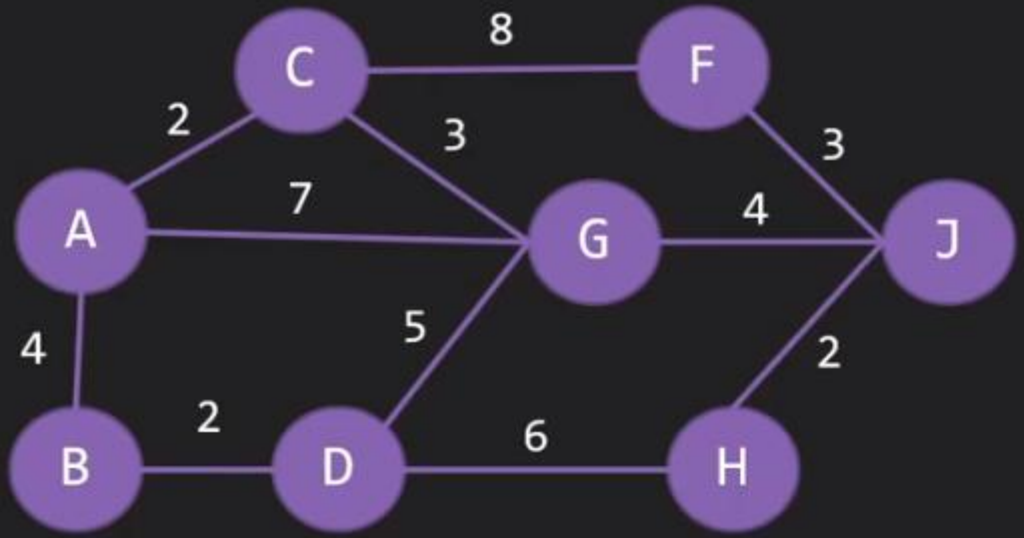
Priority Queue

From	G	C	G	D
To	J	F	D	H
Cost	9	10	10	12

Path and Cost Arrays

A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F			
G	C	5	✓
H			
J	G	9	✓

Handle Min from Priority Queue



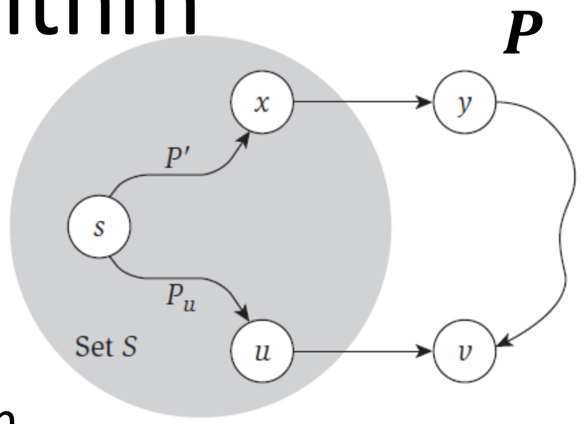
Priority Queue

From	J	D	J
To	H	H	F
Cost	11	12	12

Path and Cost Arrays

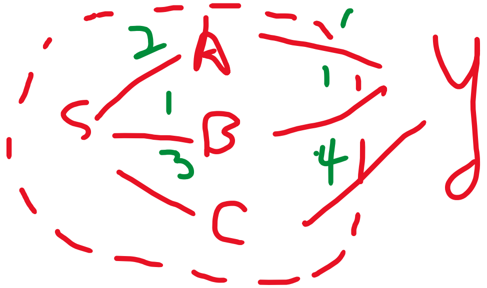
A	-	-	✓
B	A	4	✓
C	A	2	✓
D	B	6	✓
F	C	10	✓
G	C	5	✓
H	J	11	✓
J	G	9	✓

Shortest path – Proof of the greedy algorithm



(4.14) *Consider the set S at any point in the algorithm's execution. For each $u \in S$, the path P_u is a shortest s-u path.*

- Proof: (Induction)
  - |S| == 1 works, we assume |S|==k also works.
  - Now we want to expand |S| -> k+1 by adding v using the Dijkstra Algorithm.
  - We assign (u, v) as the final edge on our shortest path from s to v, pointing out from S to $V - S$
  - Assume $s \rightarrow v$ isn't the shortest path to v, then there has to be another path $\boldsymbol{P}$.
  - In the path $\boldsymbol{P}$, we assign y as the first node pointed by edge from S to $V - S$, using edge (x, y)
  - We then have $l(P) = l(s, x) + l(x, y) + l(y, v)$
  - Define $\boldsymbol{d(x)}$: the length of the shortest path P_x from s to node x. Therefore, we have $l(s, x) \geq d(x)$
  - Define $\boldsymbol{d'(y)} = \min_{e=(u,y):u \in S} \boldsymbol{d(u)} + l_e$ for given y.
  - $l(P) \geq l(s, x) + l(x, y) \geq d(x) + l(x, y) \geq \min_{e=(u,y):u \in S} d(u) + l_e = d'(y)$
  - By Dijkstra algorithm, $d'(v) = \min_{a \in V-S} \left(\min_{e=(u,a):u \in S} d(u) + l_e \right)$
  - Therefore, $d'(v) \leq d'(y)$ # $\because$ both $v \& y$ $are \in V - S$, v is the one we choose.
  - Therefore, $l(P) \geq d'(y) \geq d'(v)$. $d'(v)$ is the shortest path starts from s.
  - Proved.



Shortest path

- Algorithm: Dijkstra's algorithm
  - Initialization:
    - X:{s}
    - d[u] = $\begin{cases} l(s,u), & if (s,u) \in E \\ \infty, & otherwise \end{cases}$
  - For i=1, … n-1
    - Select u s.t. d[u] is the min among V-X
    - X = X + {u}
    - For each v s.t. (u, v)∈E:
      - If d[u] + l(u, v) < d[v]:
        - d[v] = d[u] + l(u, v)
        - pre[v]=u

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes

    For each $u \in S$, we store a distance $d(u)$

Initially $S = \{s\}$ and $d(s) = 0$

While $S \neq V$

    Select a node $v \notin S$ with at least one edge from S for which

        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible

    Add v to S and define $d(v) = d'(v)$

EndWhile

Tree

- Definitions:
  - An undirected graph that
    - 1) connected
    - 2) Don’t have any cycles

- Properties:
  - 1) Adding an edge to the tree will create a cycle
  - 2) If the original tree has a cycle by adding an edge, then by removing any one of the edges in that cycle will result in another tree

Spanning Tree

- Definitions:
  - A *spanning tree* is a subset of graph that
    - 1) Spans to every vertex (‘connected’ in undirected G, ‘reachable’ in directed G)
    - 2) Don’t have any cycles

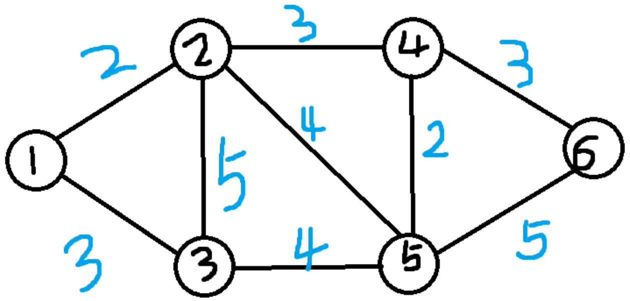
- How to find a spanning tree?
  - Run BFS / DFS

Minimum Spanning Tree (MST)

- Definitions:
  - A *minimum spanning tree* is the tree with minimum total weight among all trees given a positive weighted graph G.

- How to find a MST?
  - Run Prim's/Kruskal's algorithm

Minimum Spanning Tree - Prim



- **Given:** A connected undirected graph, G=(V, E), with edge length l(e)>0
- **Goal:** Find a set of edges $T^* \subseteq E, s.t.$
  - 1) G' = (V, T^*) is connected
  - 2) Minimize total cost $\sum_{e \in T^*} l(e)$

	X={1}	X={1,2}	X={1,2,3}	1,2,3,4	1,2,3,4,5
a[1]					
a[2]	2				
a[3]	3	3			
a[4]	∞	3	3		
a[5]	∞	4	4	2	
a[6]	∞	∞	∞	3	3

- **Prim's algorithm**:
  - Initialization:
    - X = {s} # nodes connected to s
    - pre[u] = $\begin{cases} s, & if (s,u) \in E \\ \infty, & otherwise \end{cases}$ # previous node of u
    - a[u] = $\begin{cases} l(s,u), & if (s,u) \in E \\ \infty, & otherwise \end{cases}$ # cost of adding u to X
  - For i = 1,2, …, n-1:
    - Find u, which is the node in V-X with min a[u]
    - Add u to X
    - For each v s.t. (u, v)∈E:
      - If l(u, v)<a[v]:
        - a[v] = l(u, v)
        - Pre[v] = u

Dijkstra & Prim Differences

- 1. Any Graph vs Undirected Graph
- 2. Find the shortest path vs Find the minimum spanning tree.
- 3. Calculate the **accumulated min** distance vs the **current min** weighted edge

- **Prim's algorithm**:

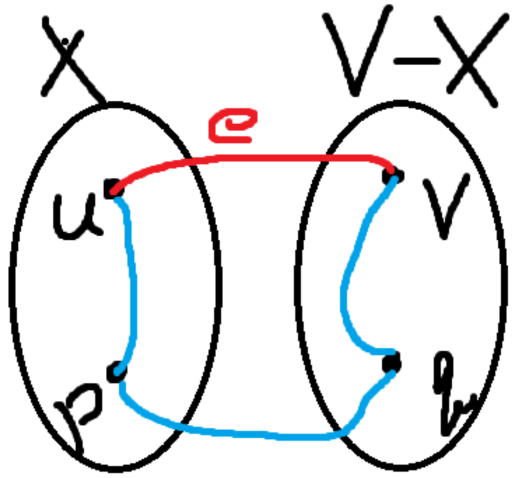
- Initialization:
  - ....
- For i = 1,2, …, n-1:
  - Find u, which is the node in V-X with min A[u]
  - Add u to X
  - For each v s.t. (u, v)∈E:
    - If l(u, v)<A[v]:
      - A[v] = l(u, v)
      - Pre[v] = u

- **Dijkstra's algorithm:**

- Initialization:
  - ....
- For i=1, … n-1
  - Select u s.t. d[u] is the min among V-X
  - X = X + {u}
  - For each v s.t. (u, v)∈E:
    - If d[u] + l(u, v) < d[v]:
      - d[v] = d[u] + l(u, v)
      - Pre[v]=u

Proof of correctness for the Prim's algorithm

- Define "cut" of two node sets cut(A, B): All (u, v)∈E s.t. u∈ A, v∈ B
- Then the Prim's algorithm adds the min-cost edge in cut (X, V-X) in each iteration
- **Cut Property:** if edge e is the min-cost edge in cut(X, V-X) for any node set X, then e must be in the MST.
- **Proof:** Proof by contradiction.
  - Assume MST T^*, e(=(u, v)) is the min-cost edge of cut (X, V-X) but e ∉ MST T^*
  - In T^*, u, v are connected by another path. Then we know u ----> p -> q ----> v.
  - p: final node of X, q: first node of V-X of this u---------->v path
  - Define another tree T'=T^*-(p, q)+(u, v)
  - $T'$is still a connected graph since we have u ----> p -> q ----> v
  - Since l(e) = l(u, v) < l(p, q), we know l(T')<l(T^*).
  - Contradictive to the statement T^* is a MST
  - Proved.



Exercises

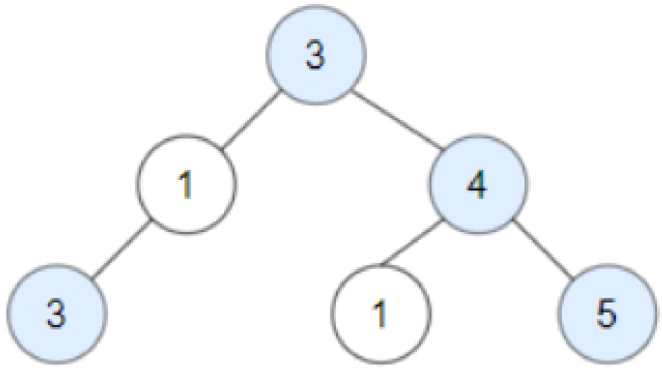
1448. Count Good Nodes in Binary Tree

Medium 1693 59 Add to List Share

Given a binary tree `root`, a node *X* in the tree is named **good** if in the path from root to *X* there are no nodes with a value *greater than* X.

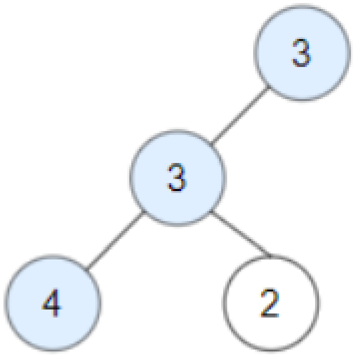
Return the number of **good** nodes in the binary tree.

Example 1:



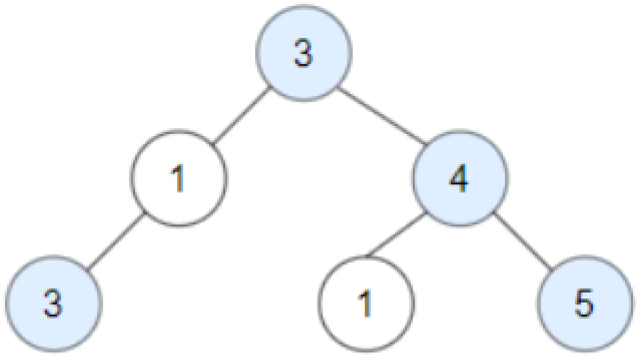

```
Input: root = [3,1,4,3,null,1,5]
Output: 4
Explanation: Nodes in blue are good.
Root Node (3) is always a good node.
Node 4 -> (3,4) is the maximum value in the path starting from the root.
Node 5 -> (3,4,5) is the maximum value in the path
Node 3 -> (3,1,3) is the maximum value in the path.
```

Example 2:




```
Input: root = [3,3,null,4,2]
Output: 3
Explanation: Node 2 -> (3, 3, 2) is not good, because "3" is higher than it.
```

Exercises - BFS




```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
```

```
class Solution:
    def goodNodes(self, root: TreeNode) -> int:
        num_good_nodes = 0

        # Use collections.deque for efficient popping
        queue = deque([(root, float("-inf"))])
        while queue:
            node, max_so_far = queue.popleft()
            if max_so_far <= node.val:
                num_good_nodes += 1
            if node.right:
                queue.append((node.right, max(node.val, max_so_far)))
            if node.left:
                queue.append((node.left, max(node.val, max_so_far)))

        return num_good_nodes
```