CS112: Data Structures

Lecture 12:

Heaps
Graphs
Shortest path

Final Exam

- **Wednesday, August 16**
  - **2 weeks from today**
- **6:00 PM**
- **In our normal lecture room**
- **More cumulative than Exam 2 was**
- **Topic list and practice exam will be on Sakai**

- **Today's topics**
  - **Heaps**
  - **Graphs**
  - **shortest path**

Review: Priority Queues

- Each data item has a priority
- Add items to queue in any order
- Remove items in priority order
  - add A:5, B:6, C:3
  - remove B
  - add D:4
  - remove A, remove D

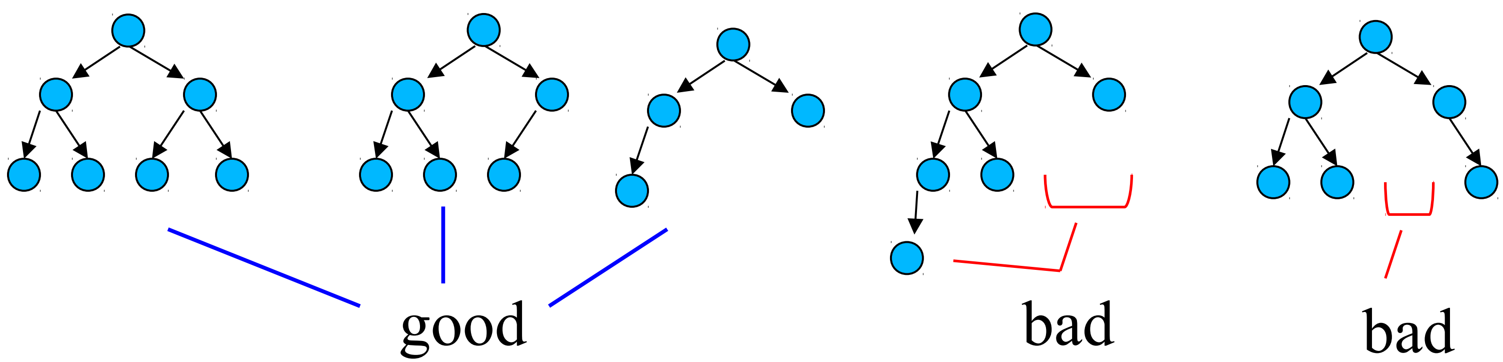
Priority Queue as an Array

- **Either as sorted or unsorted, one of**
  - **add item to queue, or**
  - **remove highest priority item**

  is O(n)
- **Can we add and remove in less than O(n)?**

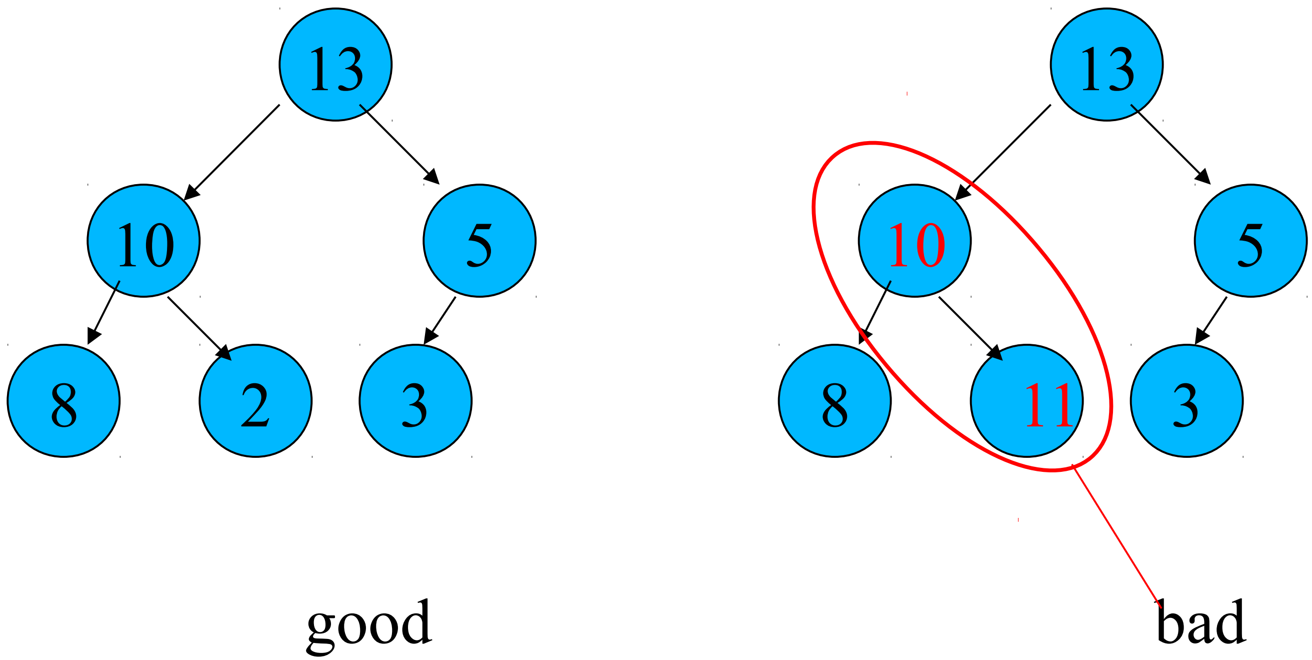
Heap

- **A heap is a way to implement a priority queue with O(log n) complexity**
- **A heap is a complete binary tree**
  - **all levels except maybe the last are full**
  - **last level filled from left to right**



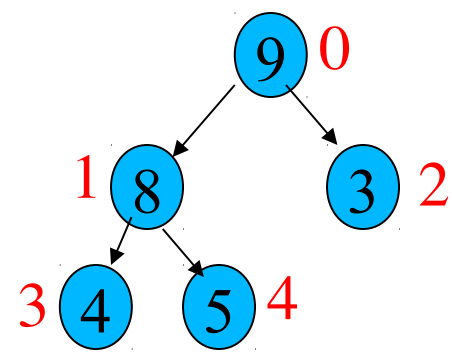
New: Heap

- **And the number at a node is greater than the number at any descendant**



Heap Representation

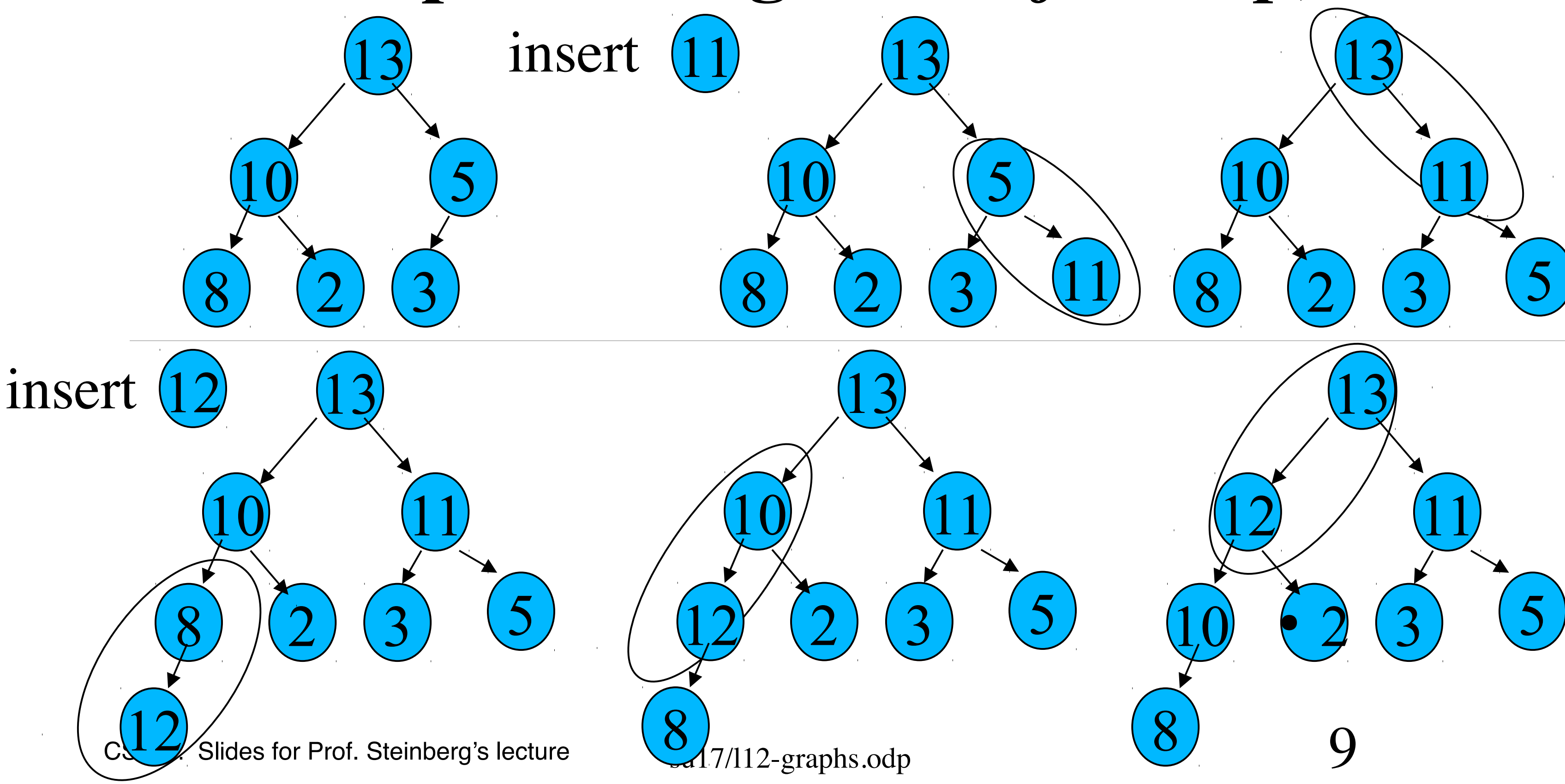
- Store heap in an array
  - For node at index j, children are at 2j+1 and 2j+2
  - Parent at floor((j-1)/2)
  - Root at index **0**



0	1	2	3	4
9	8	3	4	5

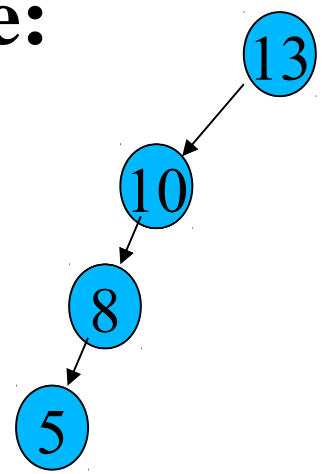
Heap Insert

- **Add node at end of last level**
- **Move up restoring order (*filter up*)**



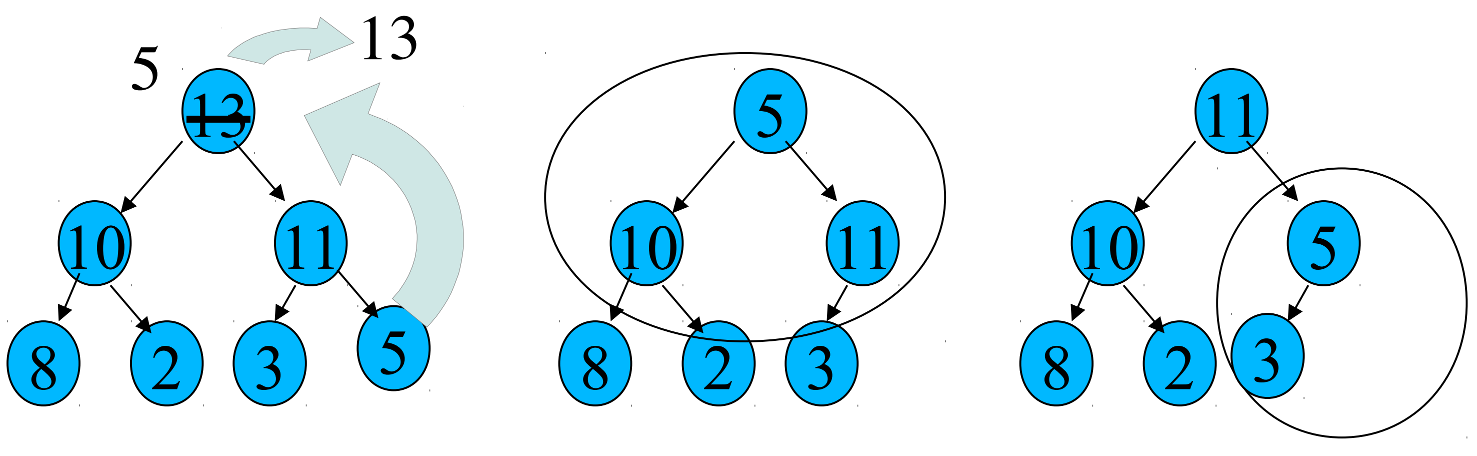
Big O of insertion

- O(H) where H is height of whole tree
  = O(log(n)) where n is number of nodes
- Heap is a complete binary tree so this is impossible:



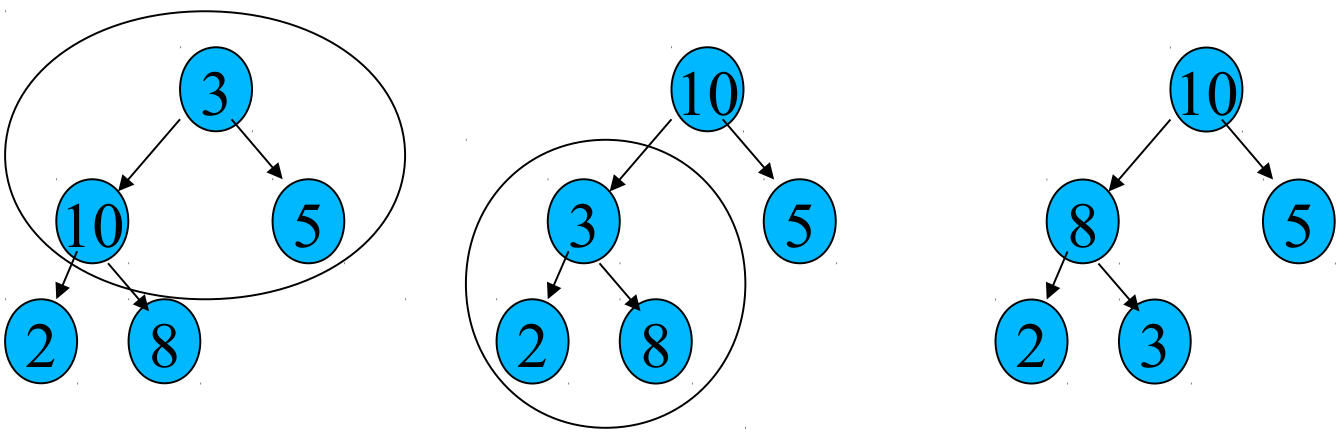
Heap Deletion

- **Copy out data at root**
- **Delete last node on last row & put data in root**
- **Move down restoring order (*filter down*)**



Filter Down

- Compare current node and its two children
  - if current node largest, stop
  - if left child is largest
    swap data of current and left, current ← left
  - similarly if right child is largest



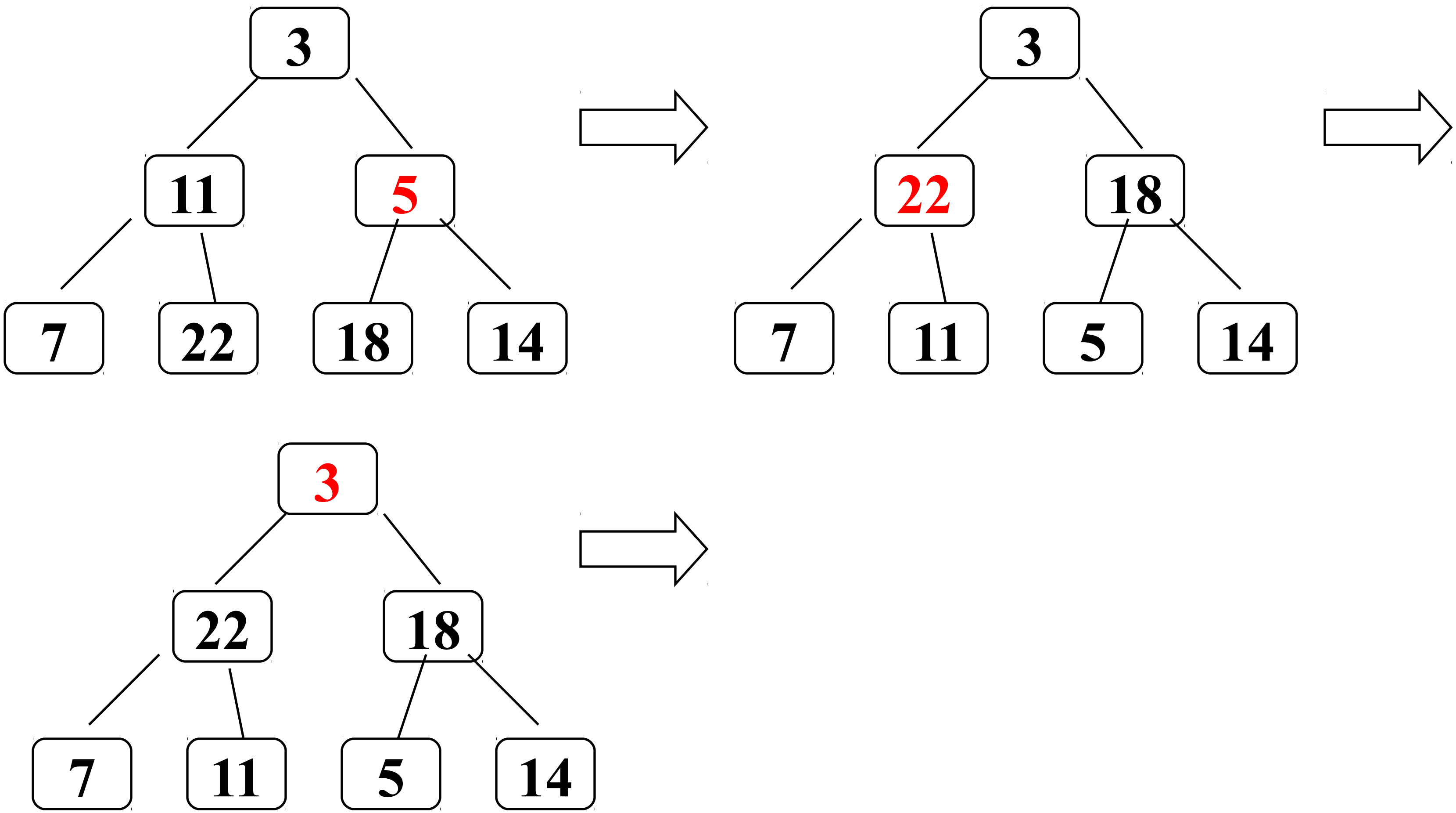
Big O of deletion

- **O(H) where H is height of whole tree**
  = O(log(n)) where n is number of nodes

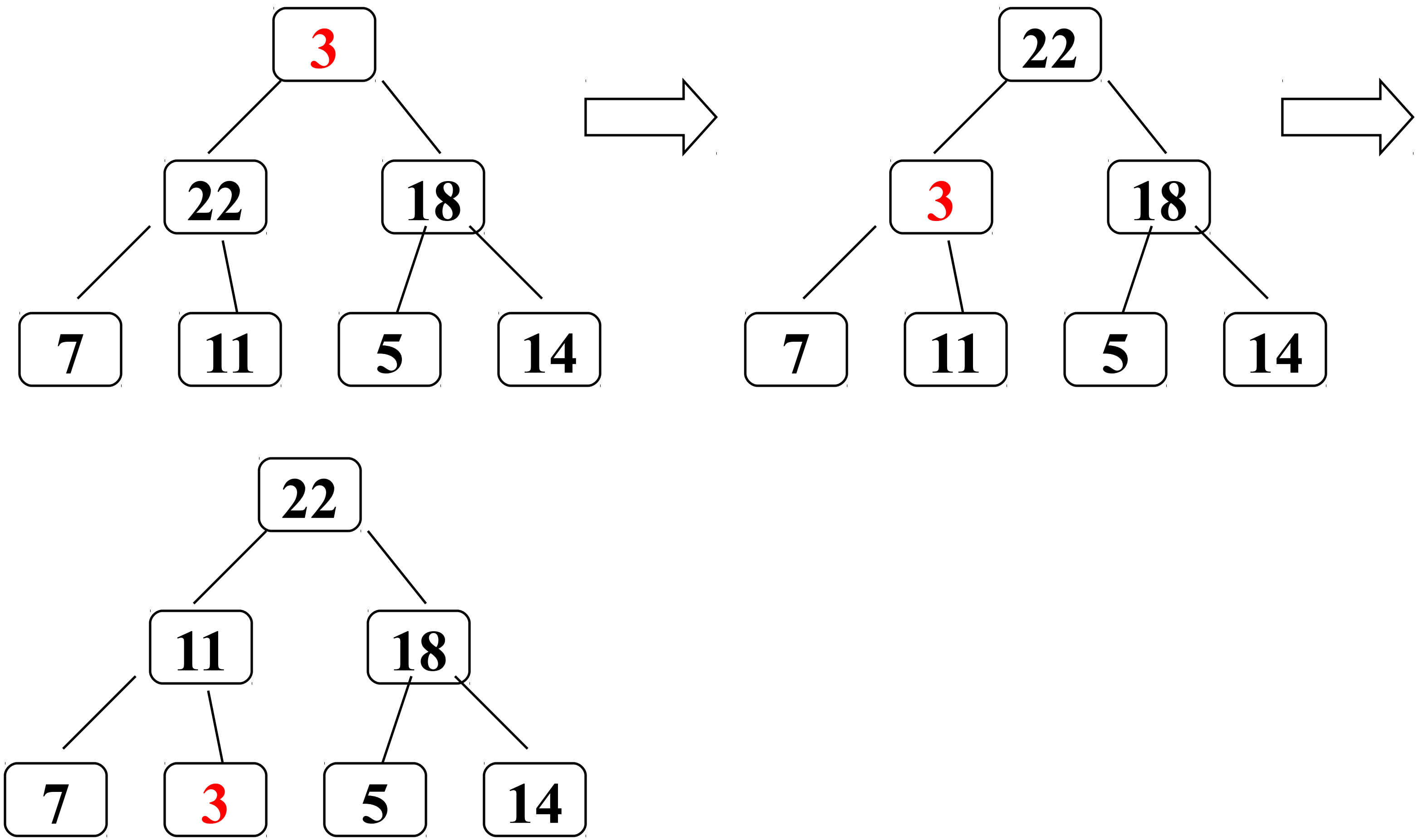
Building a Heap from an Array

- Go through the array in reverse order
  - from parent of the last leaf
  - to index 0
- At each node, do filter-down

Building a Heap from an Array



Building a Heap from an Array



Building a Heap from an Array

- **Work at a node is O(h) where h is height of subtree rooted at that node**
- **In a complete binary tree, majority of nodes close to bottom, so adds up to O(n)**

Implementing a heap

- **See tree/Heap.java**

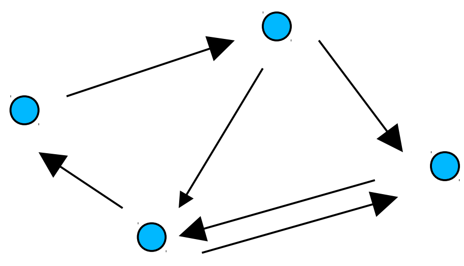
Heap Sort

- **Heapify unsorted array: O(n)**
- **Remove all n items from the heap, one by one: n*O(log n) = O(n log n)**
- **Total: O(n)+O(n log n) = O(n log n)**

Review: Graphs

Generalization of trees

- **Digraph (Directed Graph)**
  - **Like a tree but any vertex can point to any other**

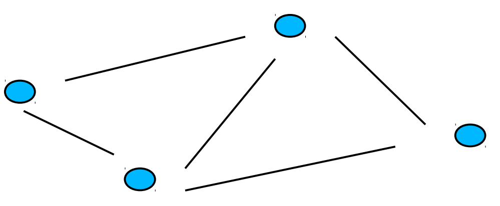


  - **E.g., Twitter follows relationship**

Graphs

Generalization of trees

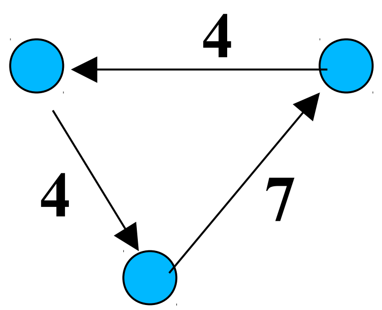
- **Graph**
  - **like digraph but arcs have no direction**

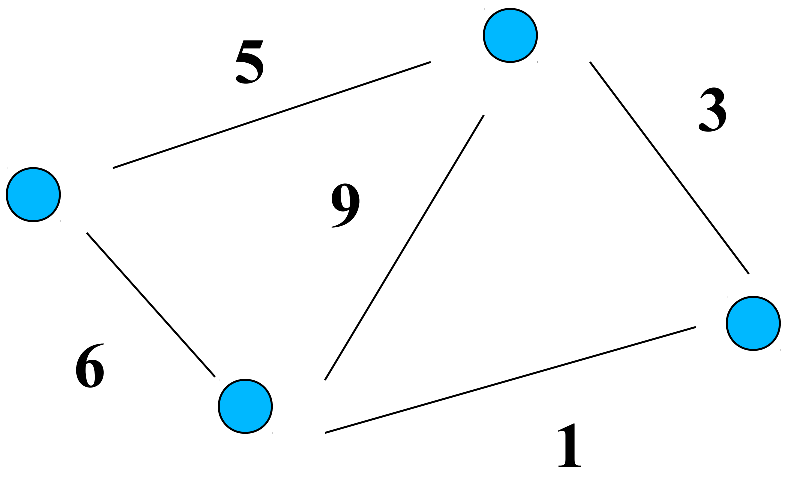


  - **E.g., Facebook friends relationship**

Graphs

- **Weighted Graph**
  - Positive integer weights on each edge



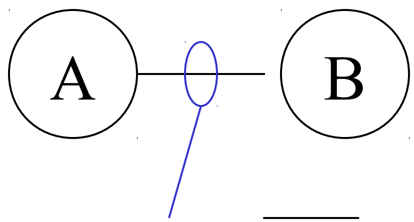


Applications

- **Paths**
  - **On streets (eg Google Maps)**
- **Electrical networks**
  - **Power lines**
  - **Printed Circuits**
- **Constraints**
  - **Ordering constraints on building steps**
    eg counters before sinks
- **Many more**

Notation

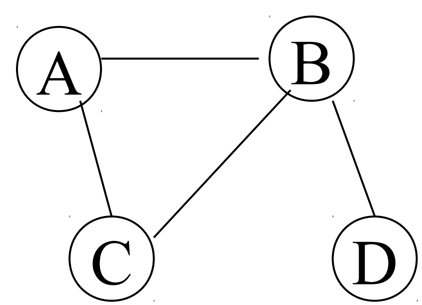
- **Arcs are named by the vertices they connect**



(B,A) or $\overline{\mathbf{BA}}$

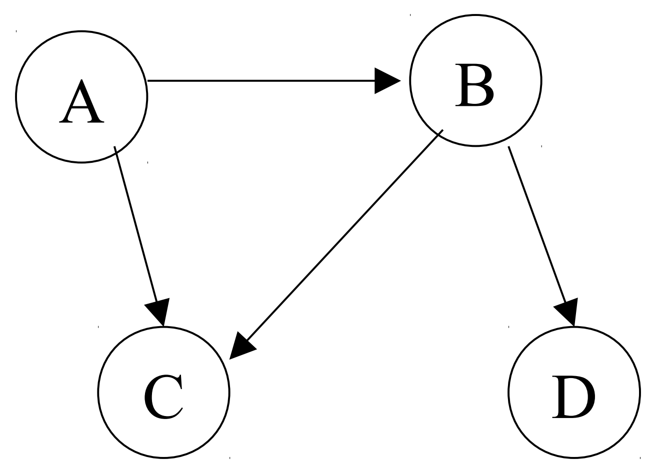
Graph Concepts

- **Neighbors of a vertex: vertices that it shares an arc with**
  - **Neighbors of A are B and C**
- **Degree of a vertex: number of neighbors**
  - **Degree of A is 2, degree of B is 3**



Graph Concepts

- **In degree (in a digraph): number of vertices that have arcs to this vertex**
  - **In degree of B is 1**
- **Out degree (in a digraph): number of vertices that have arcs from this vertex**
  - **Out degree of B is 2**



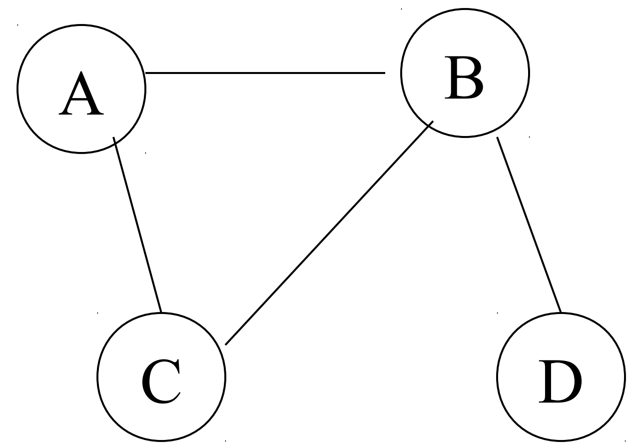
Graph Concepts

- **(Simple) Path**
  - **Sequence of arcs**

    (A,B),(B,C)
  - **May not revisit a vertex**

    ~~**(B,A),(A,C),(C,B),(B,D)**~~
  - **Except last vertex may = first**

    (B,A),(A,C),(C,B)
- **Vertex A is** reachable **from B if there is a path from B to A**

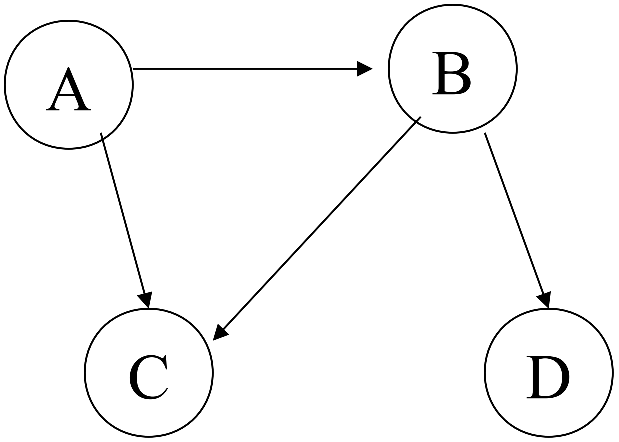


Graph Concepts

- **Path**
  - **On digraph must follow arc directions**

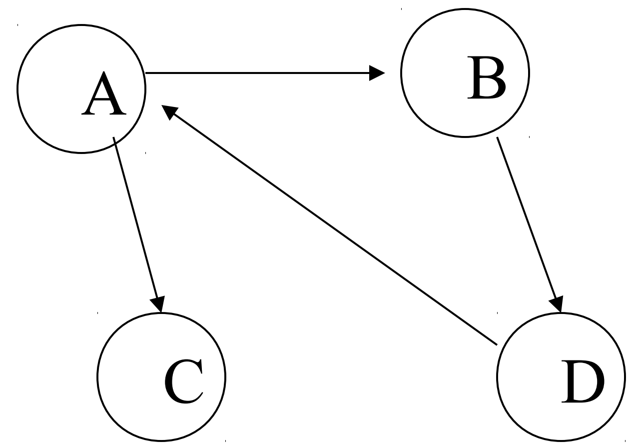
    (A,B),(B,D)

    ~~(A,C),(C,B)~~



Graph Concepts

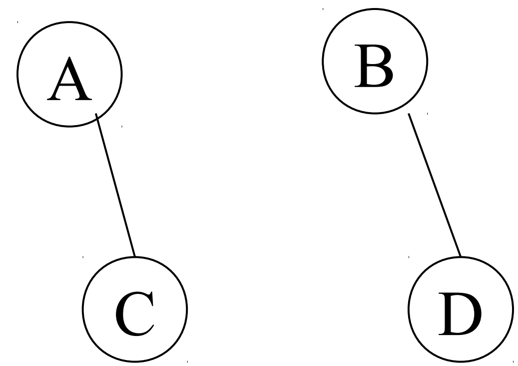
- **A cycle is a path from a node back to itself**
  - **(A, B)(B, D)(D, A)**
- **A graph with no cycles is called acyclic**



Graph Concepts

- **Connected Graph**

  For any two vertices X and Y there is a path from X to Y.

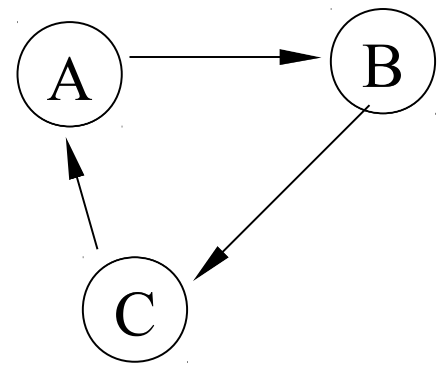


not connected

Graph Concepts

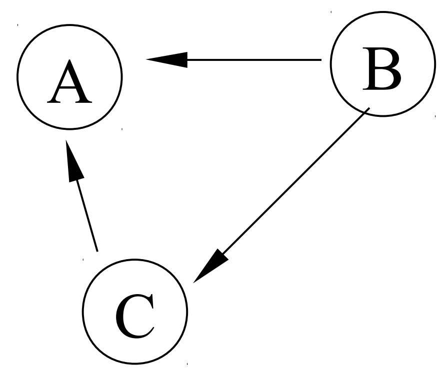
- **Strongly Connected Digraph**

  For any two vertices X and Y there is a path from X to Y. (Paths must follow arc directions)



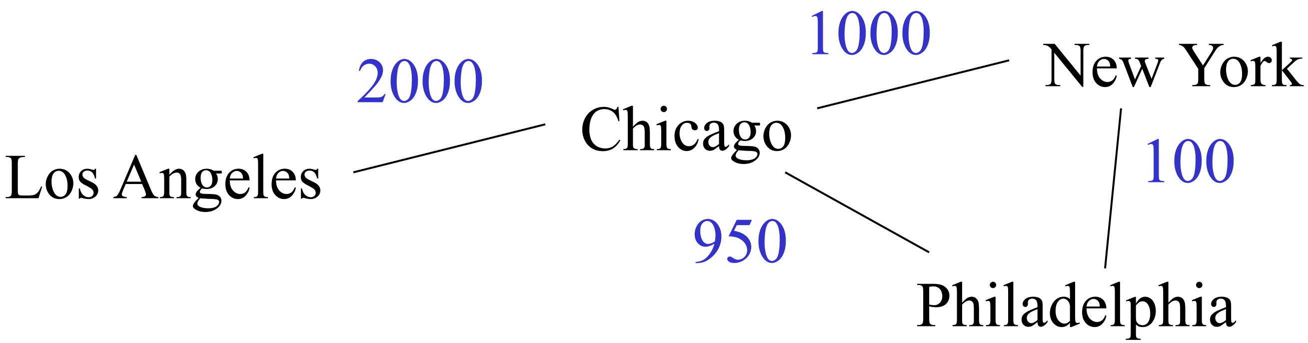
- **Weakly Connected Digraph**

  Corresponding graph is connected (i.e., ignoring arc direction)



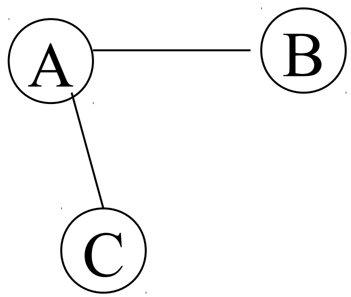
Graph Concepts

- **Weighted graph: each arc has a numerical weight**



Representing Graphs

- **Adjacency matrix**
  - **n x n boolean matrix: is there an arc?**

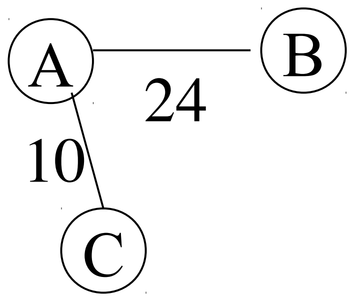


	name
0	**A**
1	**B**
2	**C**

	0	**1**	**2**
0		**T**	**T**
1	**T**		
2	**T**		

Representing Graphs

- **Adjacency matrix**
  - **n x n boolean matrix: weight of the arc or -1**

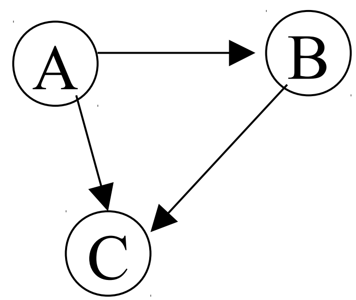


	name
0	**A**
1	**B**
2	**C**

	0	1	2
0	**-1**	**24**	**10**
1	**24**	**-1**	**-1**
2	**10**	**-1**	**-1**

Representing Graphs

- **Adjacency matrix**
  - **n x n boolean matrix: is there an arc?**



	name
0	**A**
1	**B**
2	**C**

	0	**1**	**2**
0		**T**	**T**
1			**T**
2			

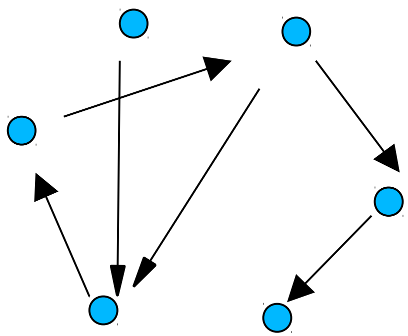
Adjacency Matrix

- Space cost: v^2 booleans where v is number of vertices
- If v is large, v^2 is huge
  - Facebook: $v = 10^9$, $v^2 = 10^{18}$

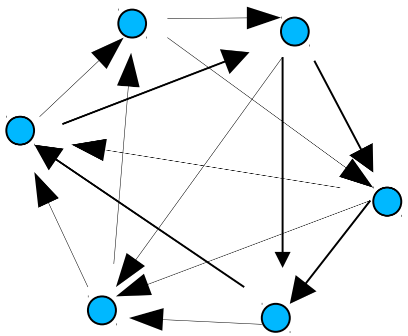
    1,000,000,000,000,000,000
  - An average Facebook user has about 350 friends
  - if e is number of edges, $e = 10^9 * 175$
  - Fraction of Trues in matrix = $10^9 * 175 / 10^{18} = 1.75 * 10^{-7}$

    $\approx 1 / 5{,}000{,}000$

Sparse vs Dense Graphs



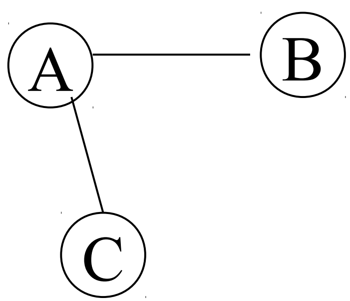
Sparse

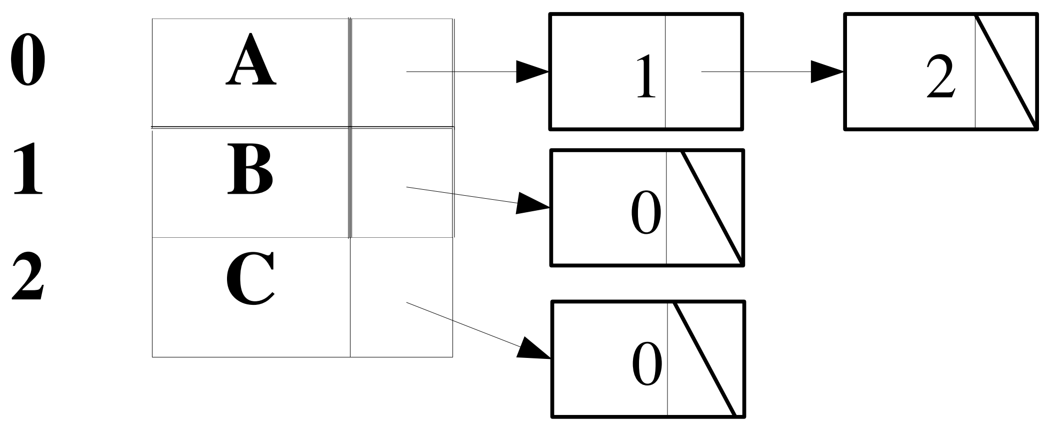


Dense

Representing Graphs

- **Adjacency list**
  - **for each node, a linked list of edges that touch it**



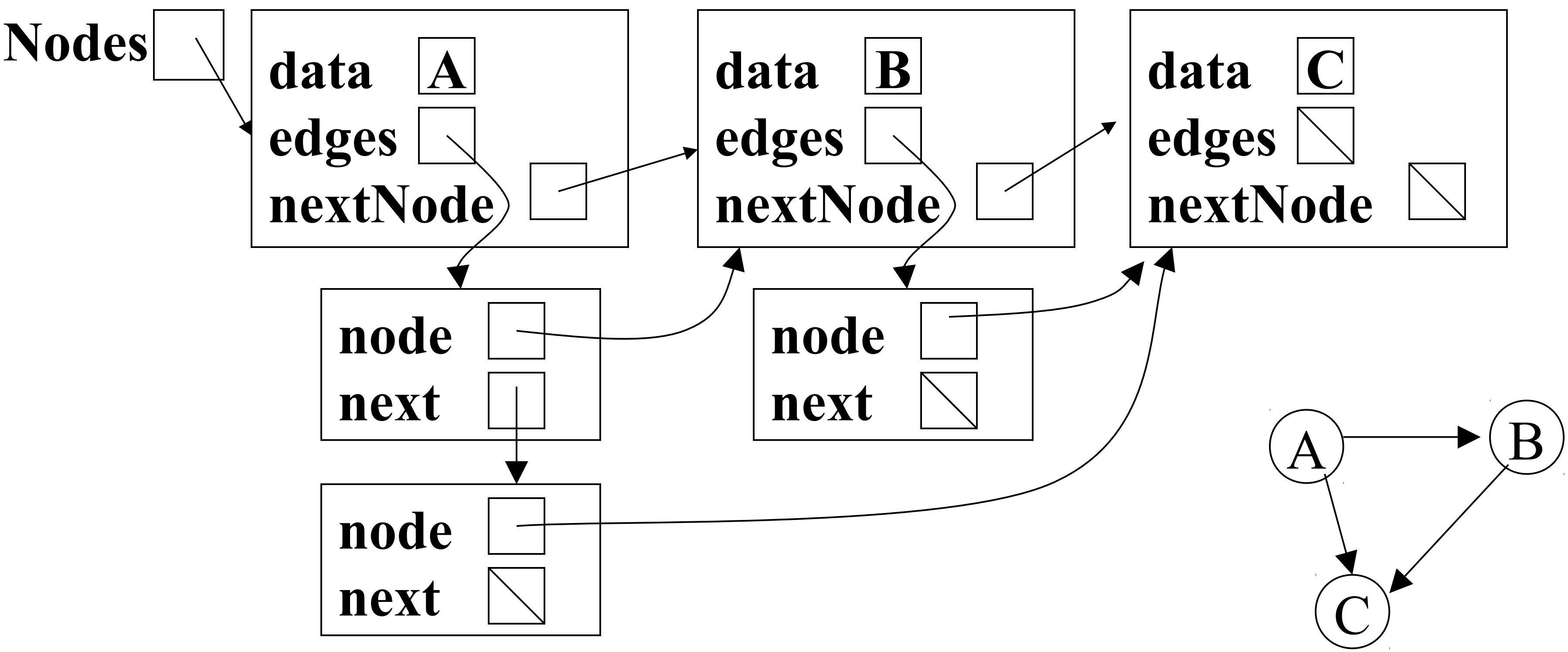


Representing Graphs

- **Adjacency list**
  - **for each node, a linked list of edges that touch it**
  - **Space cost: 2 * (v + e) = O(v+e)**
  - **For Facebook: $2*(10^9 + 175 * 10^9)$ = $350*10^9$**

Representing Graphs

- **Adjacency list**
  - for each node, linked list of edges



Time costs, Worst case

	Is there and edge from i to j	List the neighbors of i
Adjacency matrix	O(1)	O(v)
Adjacency list	O(d)	O(d)

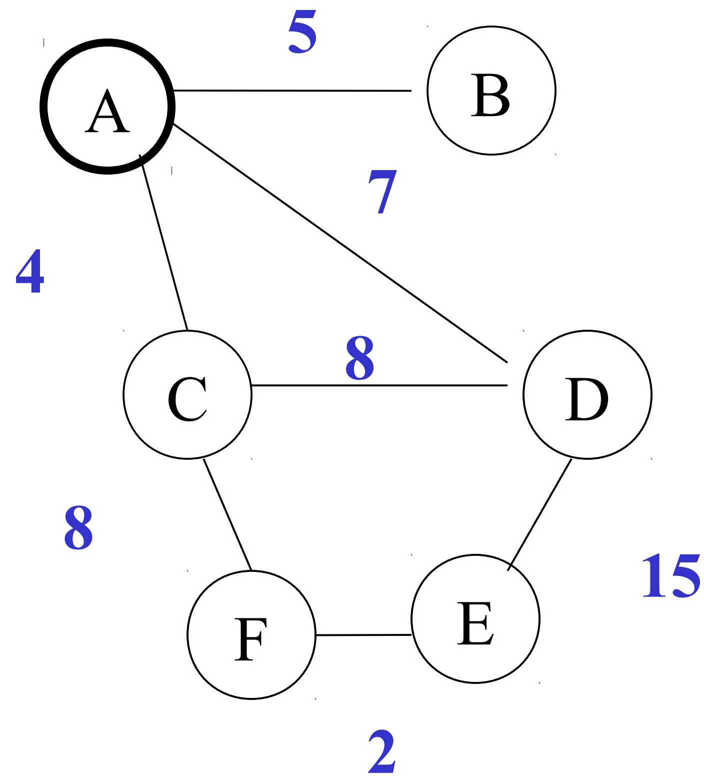
d is degree of vertex i, d<v
(usually d << v)

New: Shortest Path

- weighted digraph
  - weights are all > 0
- "length" of a path = sum of weights of arcs on path
- given start vertex, end vertex, find shortest path from start to end

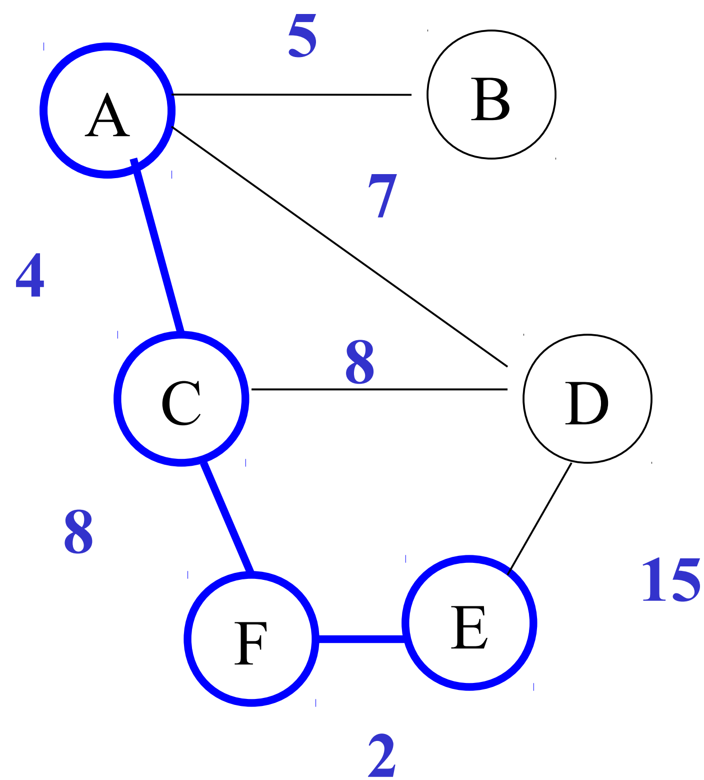
Shortest Paths

- **What is the shortest path**
  - **from A to E?**
  - **from A to F?**



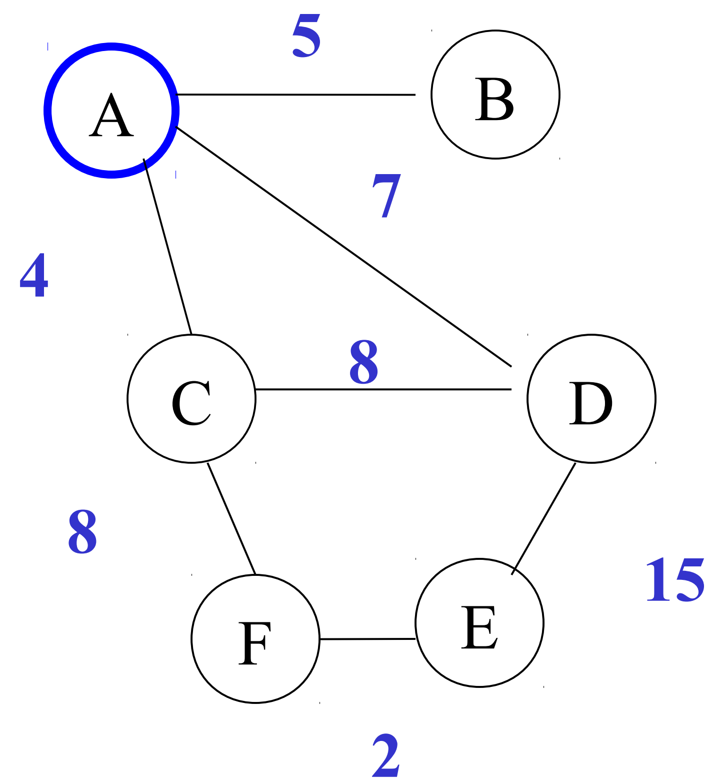
Shortest Paths

- **If a shortest path from A to E runs through F, the part from A to F is a shortest path from A to F**



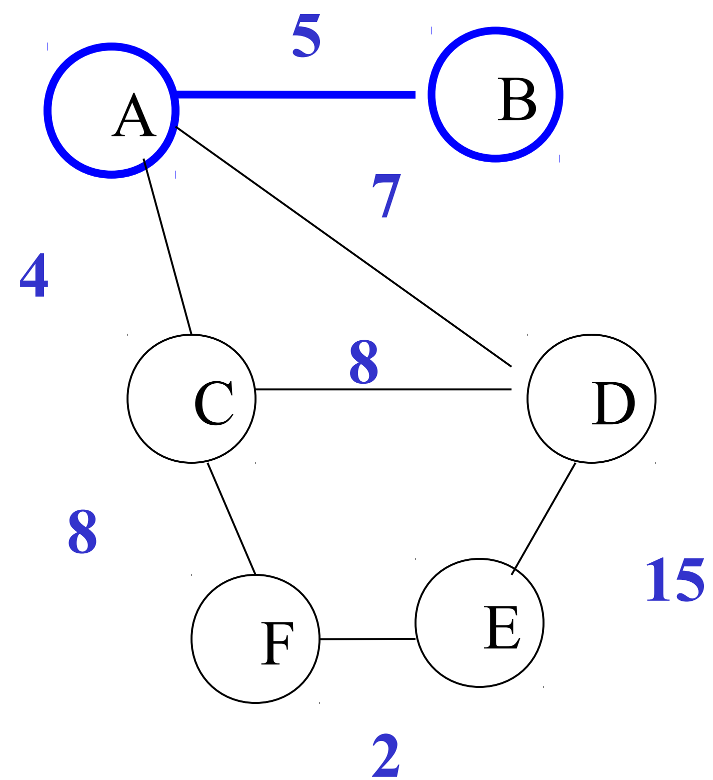
Dijkstra’s Algorithm

- **Consider the shortest paths from A to each other vertex.**



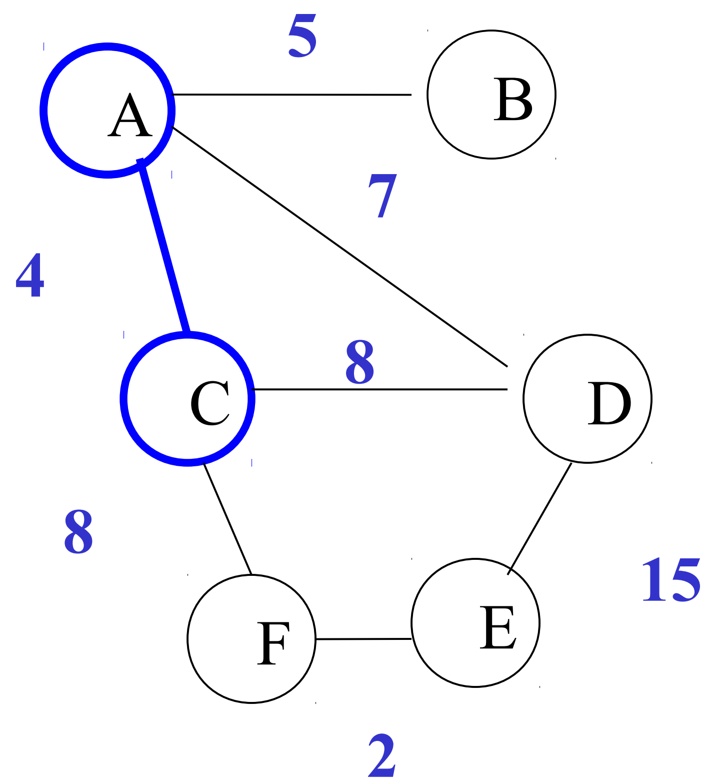
Dijkstra's Algorithm

- **Consider the shortest paths from A to each other vertex.**



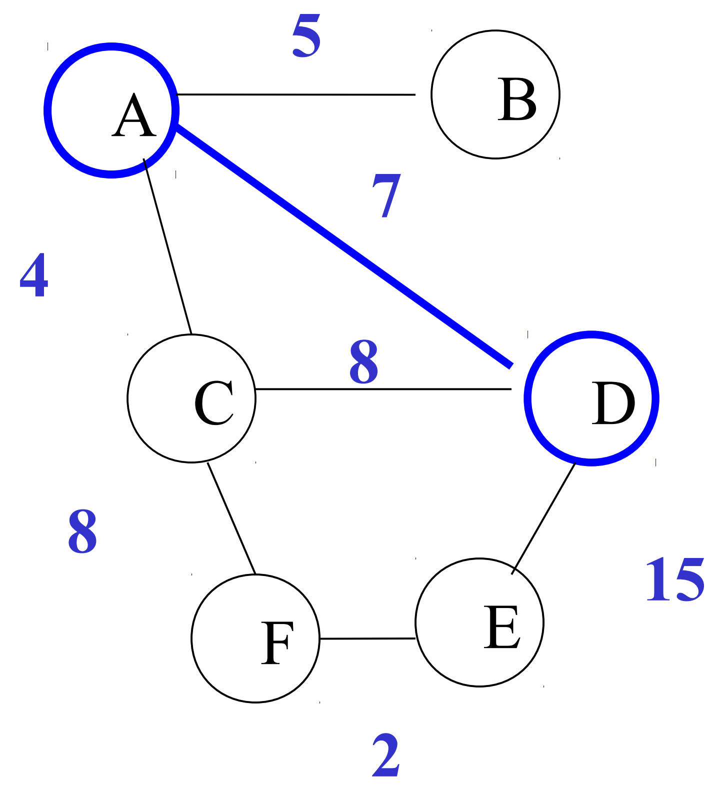
Dijkstra's Algorithm

- **Consider the shortest paths from A to each other vertex.**



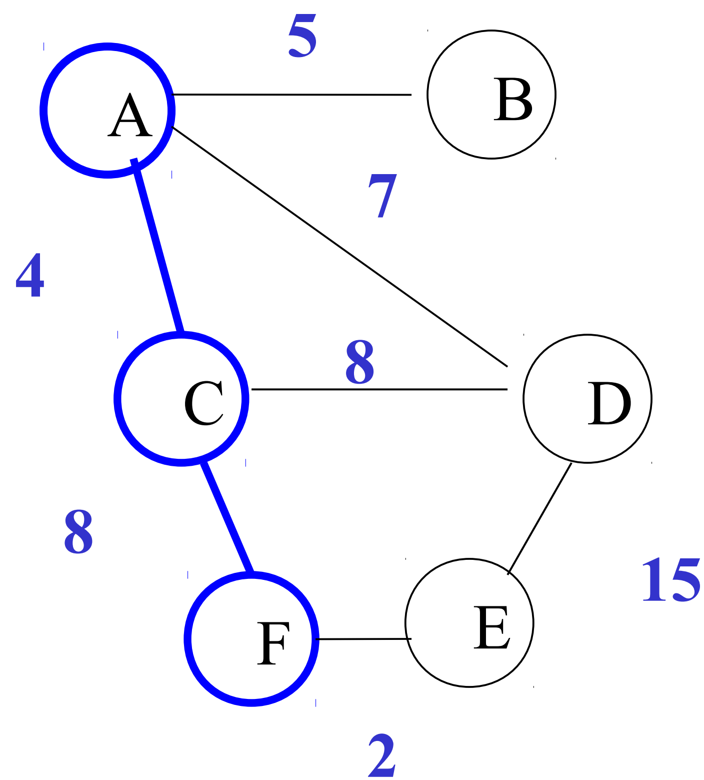
Dijkstra's Algorithm

- **Consider the shortest paths from A to each other vertex.**



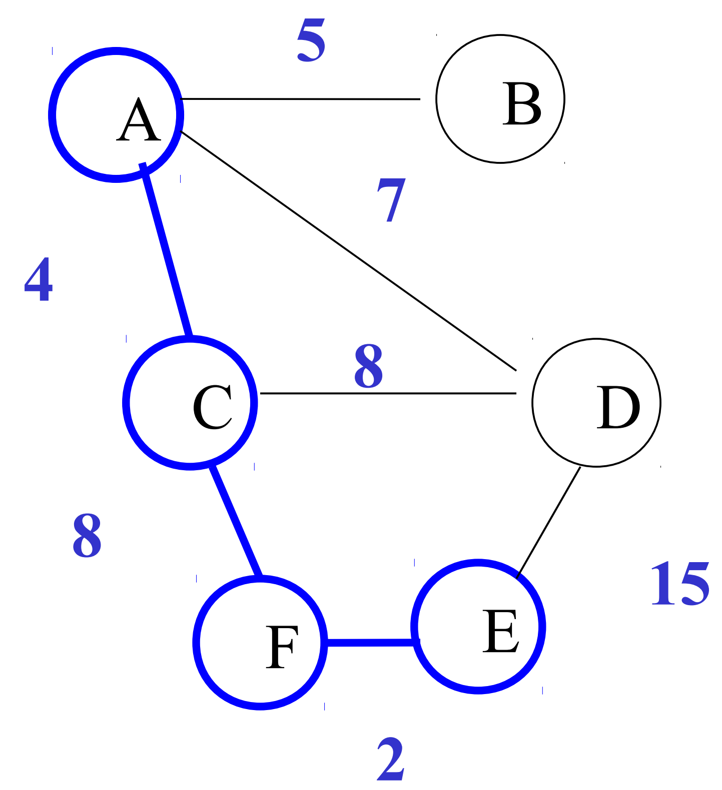
Dijkstra's Algorithm

- **Consider the shortest paths from A to each other vertex.**



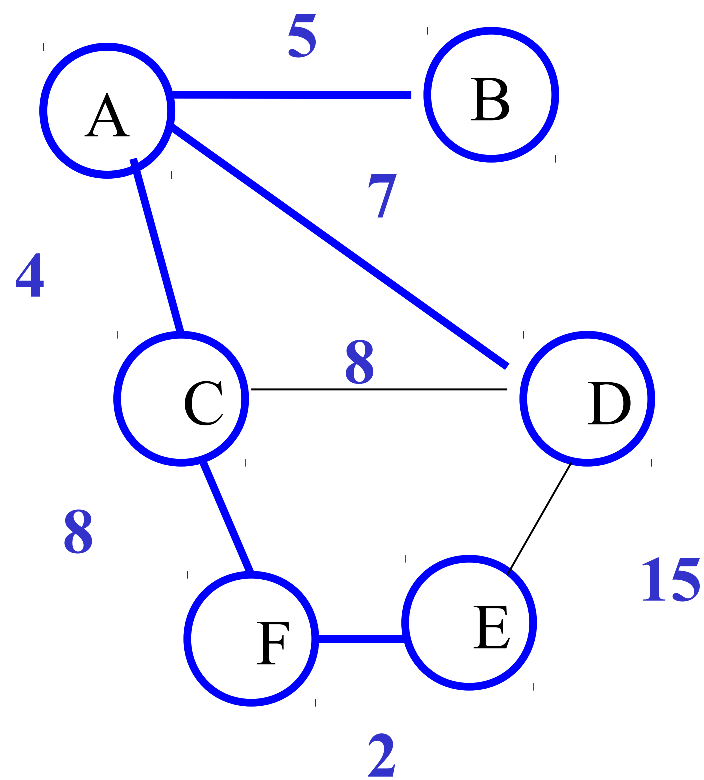
Dijkstra's Algorithm

- **Consider the shortest paths from A to each other vertex.**



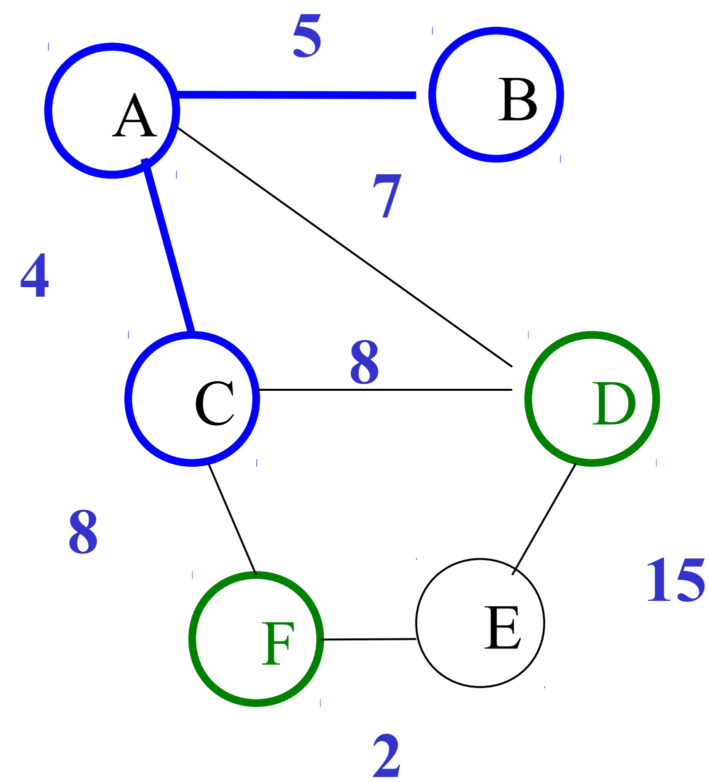
Dijkstra's Algorithm

- **These can be put together to form a tree**
- **A *Shortest Path Tree***



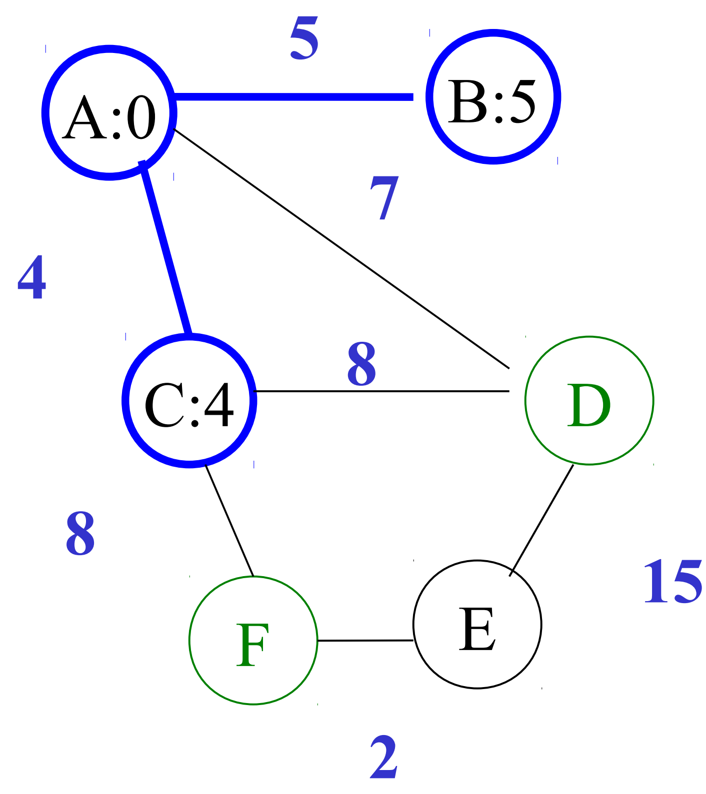
Dijkstra's Algorithm

- Grow a tree of shortest paths from start
  - grow it one vertex at a time, closest to farthest
- Fringe: nodes that are not in the tree yet but have a neighbor in the tree



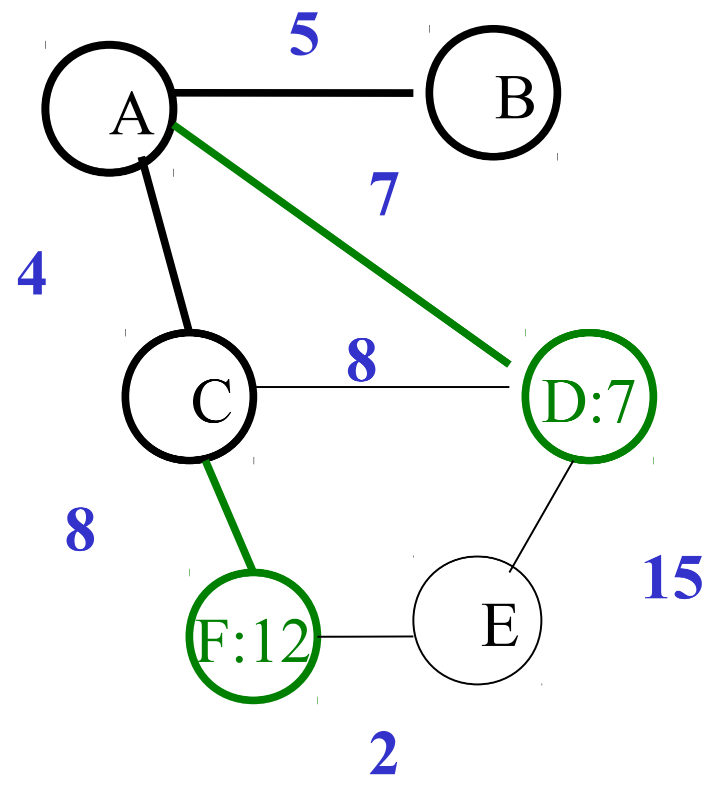
Dijkstra's Algorithm

- Vertices in the tree have
  - a link: first step on the shortest path back to start
  - a distance: the length of that whole path back to start



Dijkstra's Algorithm

- Vertices in the fringe have
  - a link: an arc to the tree if > 1 of these, use the arc that gives the shortest path back to start
  - a distance: the length of the path using link



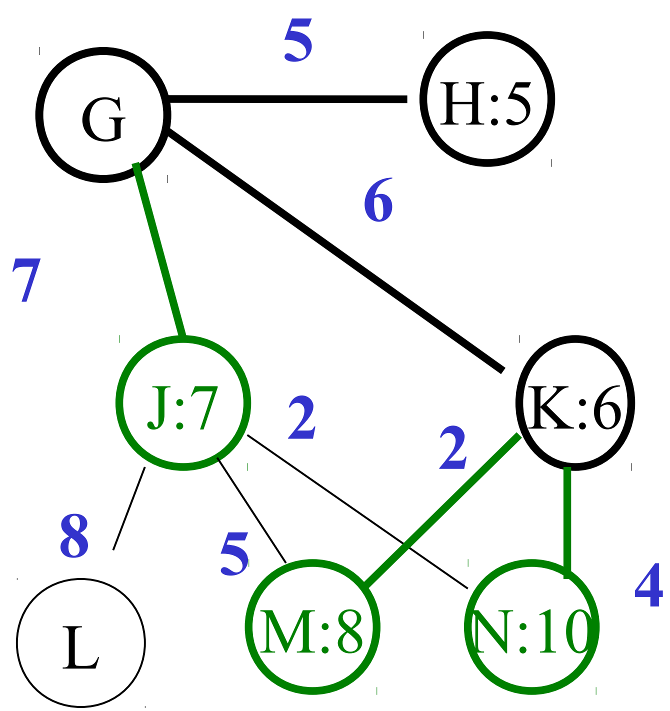
Algorithm:

- **Put start vertex in the tree**
- **While there are any vertices in fringe**
  - **Let v be vertex in fringe with smallest distance-from-start.**
  - **Put v in the tree.**
  - **Update fringe**

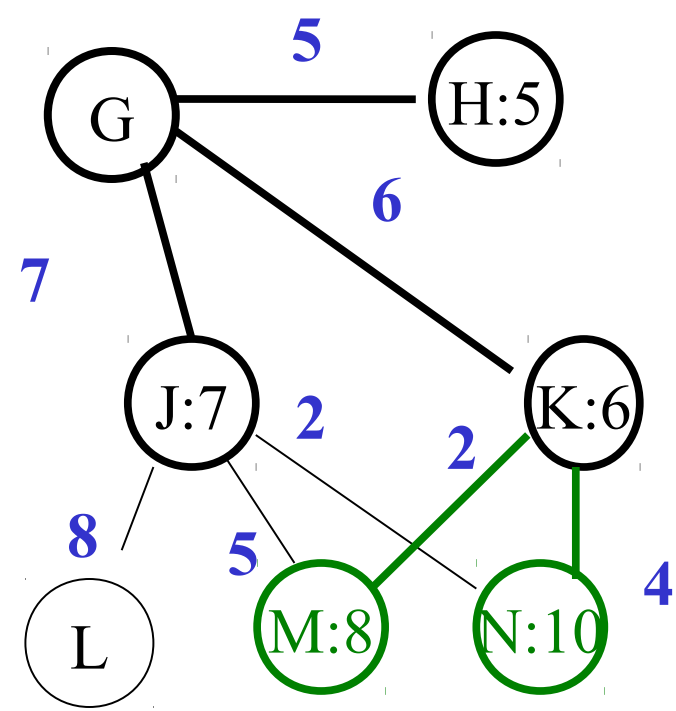
Update fringe

- **Neighbors of v that are not in tree or fringe get added to fringe**
- **Neighbors of v that are in the fringe get checked: would changing link to be v result in a smaller distance? If so, change link and distance**

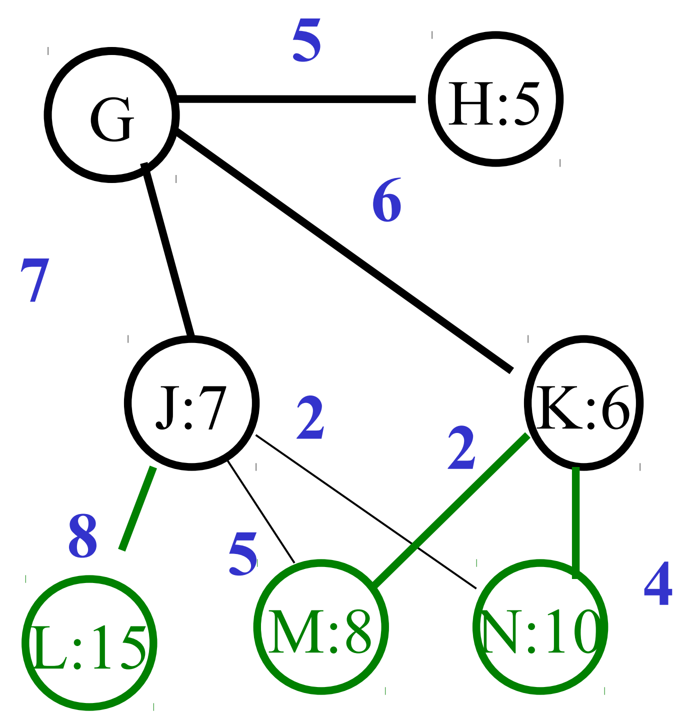
J → tree, Update fringe



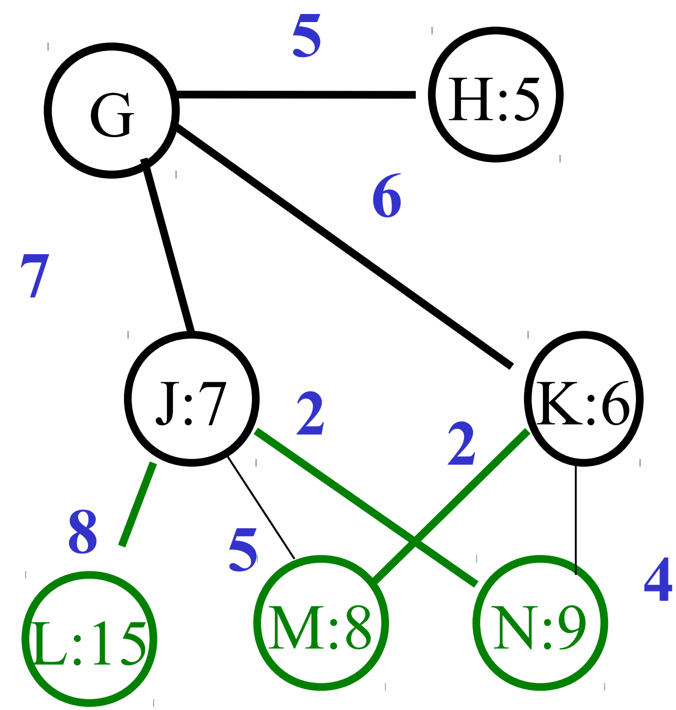
J → tree



Update fringe: neighbors → fringe



Update fringe:
Check neighbors' links



Example

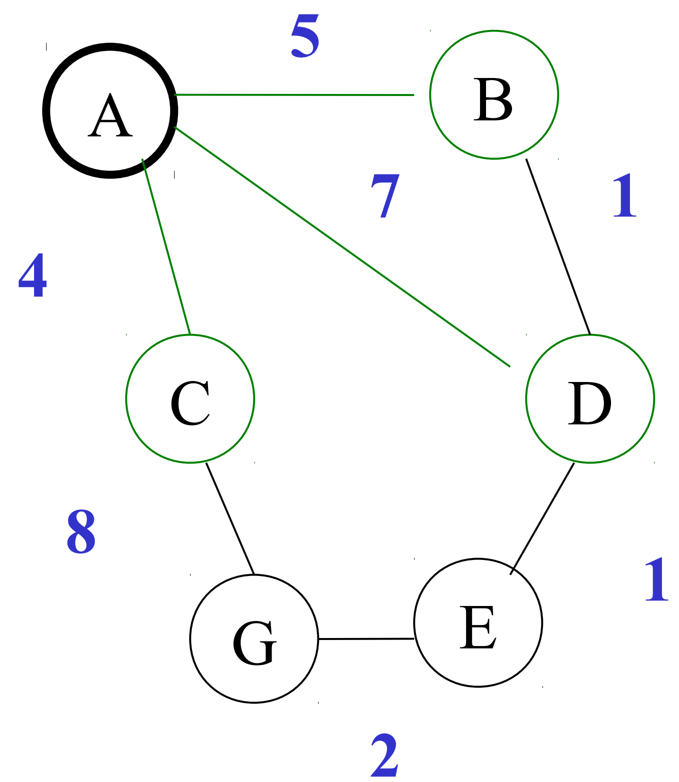
Step:	1
Node	SLD
A	Tx0
B	FA5
C	FA4
D	FA7
E	Nxx
G	Nxx

State:

T: in Tree

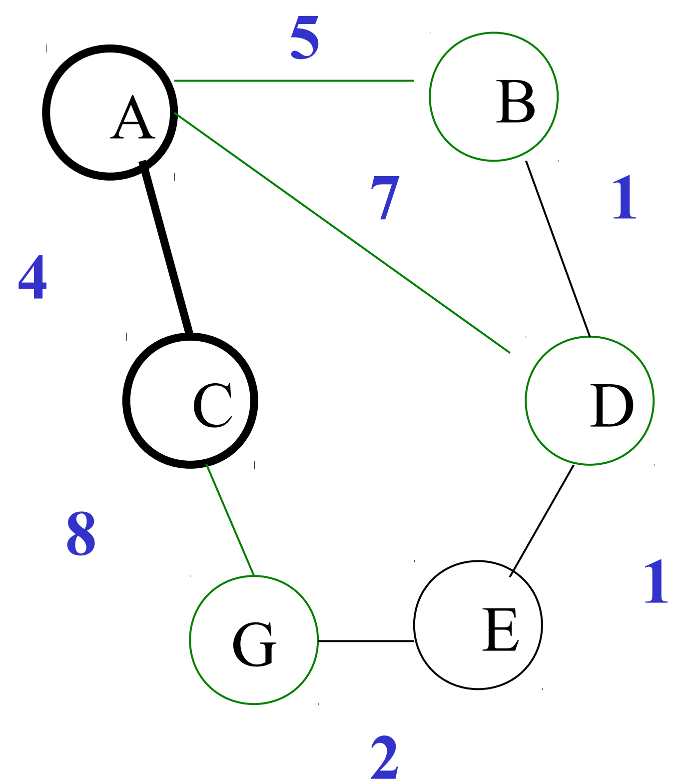
F: in Fringe

N: Neither



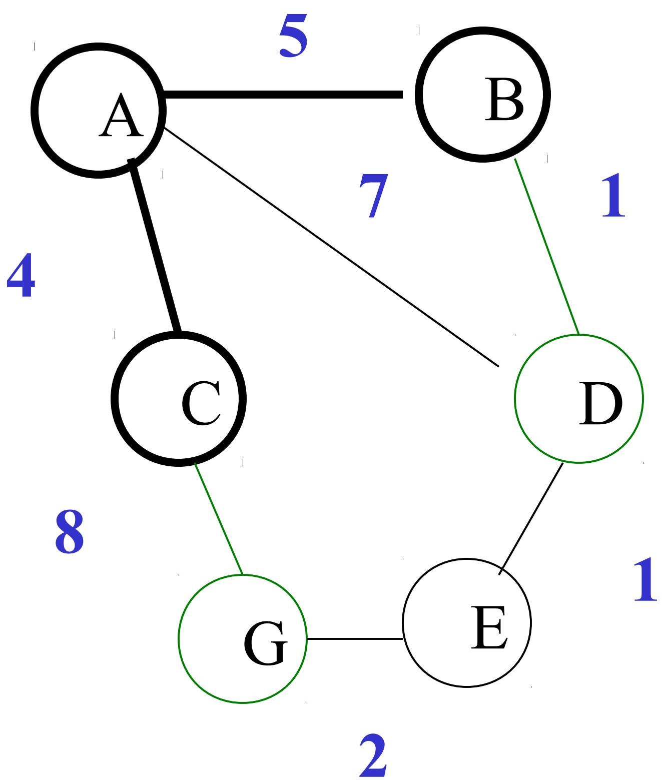
Example

Step:	1	2
Node	SLD	SLD
A	Tx0	
B	FA5	FA5
C	FA4	TA4
D	FA7	FA7
E	Nxx	Nxx
G	Nxx	FC12



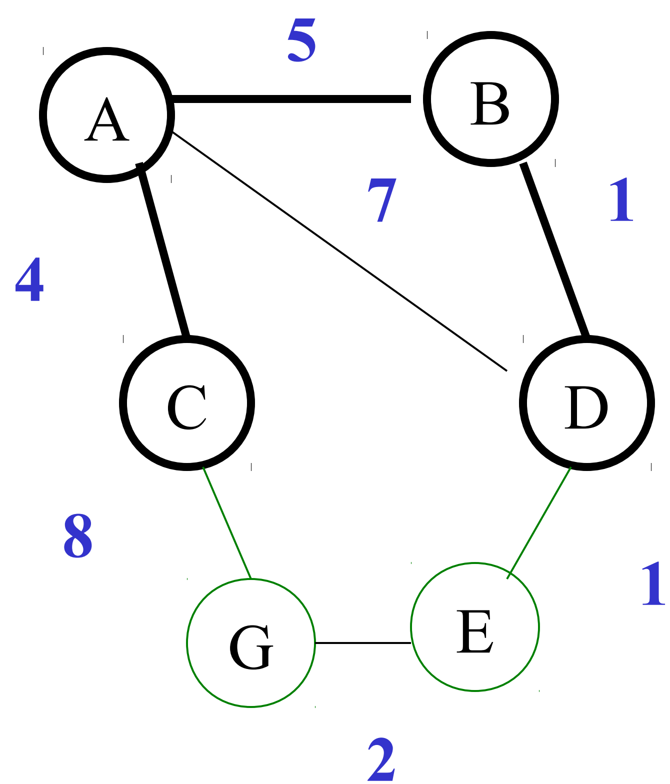
Example

Step:	1	2	3
Node	SLD	SLD	SLD
A	Tx0		
B	FA5	FA5	TA5
C	FA4	TA4	
D	FA7	FA7	FB6
E	Nxx	Nxx	Nxx
G	Nxx	FC12	FC12



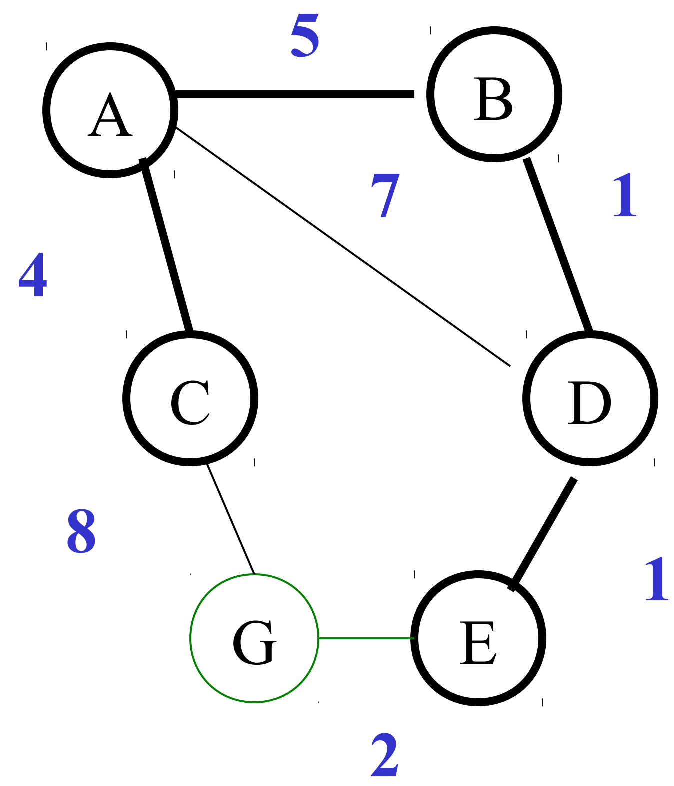
Example

Step:	1	2	3	4
Node	SLD	SLD	SLD	SLD
A	Tx0			
B	FA5	FA5	TA5	
C	FA4	TA4		
D	FA7	FA7	FB6	TB6
E	Nxx	Nxx	Nxx	FD7
G	Nxx	FC12	FC12	FC12



Example

Step:	1	2	3	4	5
Node	SLD	SLD	SLD	SLD	SLD
A	Tx0				
B	FA5	FA5	TA5		
C	FA4	TA4			
D	FA7	FA7	FB6	TB6	
E	Nxx	Nxx	Nxx	FD7	TD7
G	Nxx	RC12	FC12	FC12	FE9



Example

Step:	1	2	3	4	5	6
Node	SLD	SLD	SLD	SLD	SLD	SLD
A	Tx0					
B	FA5	FA5	TA5			
C	FA4	TA4				
D	FA7	FA7	FB6	TB6		
E	Nxx	Nxx	Nxx	FD7	TD7	
G	Nxx	FC12	FC12	FC12	FE9	TE9