

Graphs & Graph Algorithms 2



Nelson Padua-Perez
Chau-Wen Tseng

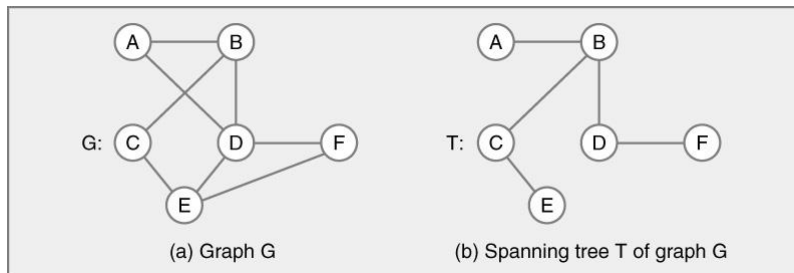
Department of Computer Science
University of Maryland, College Park

Overview

- **Spanning trees**
- **Minimum spanning tree**
 - **Kruskal's algorithm**
- **Shortest path**
 - **Dijkstra's algorithm**
- **Graph implementation**
 - **Adjacency list / matrix**

Spanning Tree

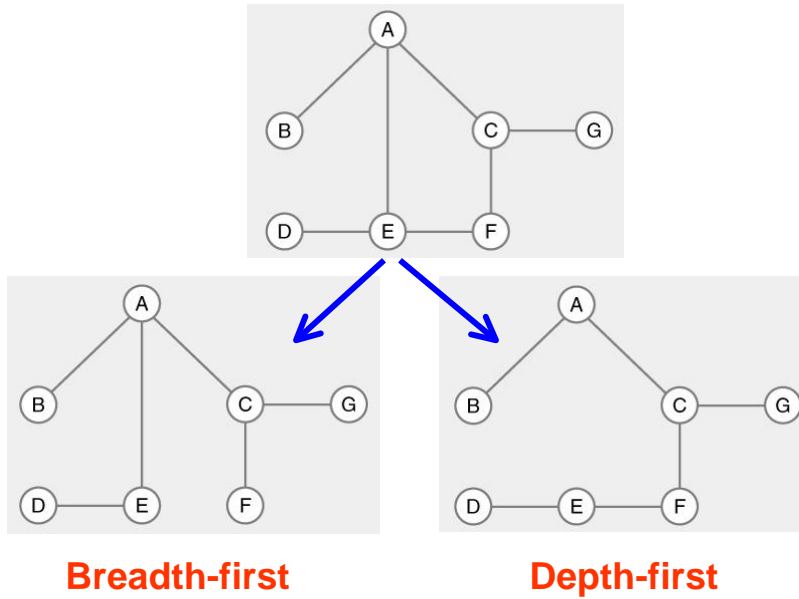
- Tree connecting all nodes in graph
- N-1 edges for N nodes
- Can build tree during traversal



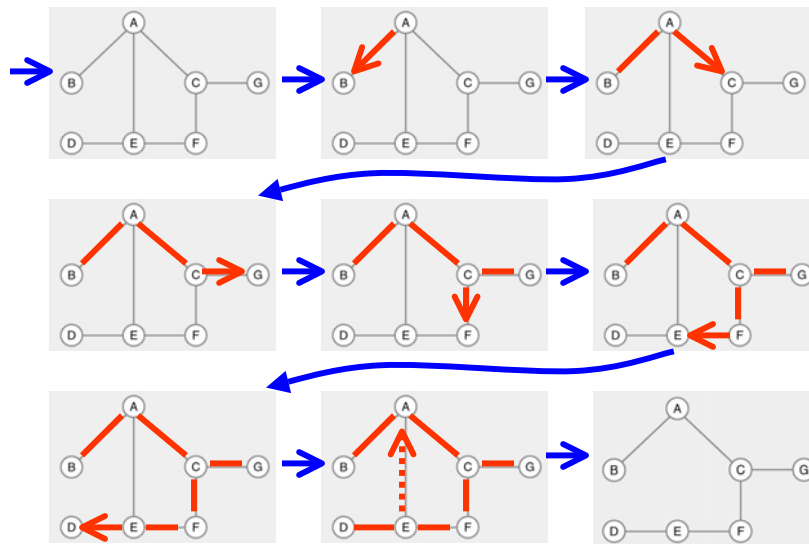
Spanning Tree Construction

```
for all nodes X
  set X.tag = False
  set X.parent = Null
{ Discovered } = { 1st node }
while ( { Discovered } ≠ ∅ )
  take node X out of { Discovered }
  if (X.tag = False)
    set X.tag = True
    for each successor Y of X
      if (Y.tag = False)
        set Y.parent = X // add (X,Y) to tree
        add Y to { Discovered }
```

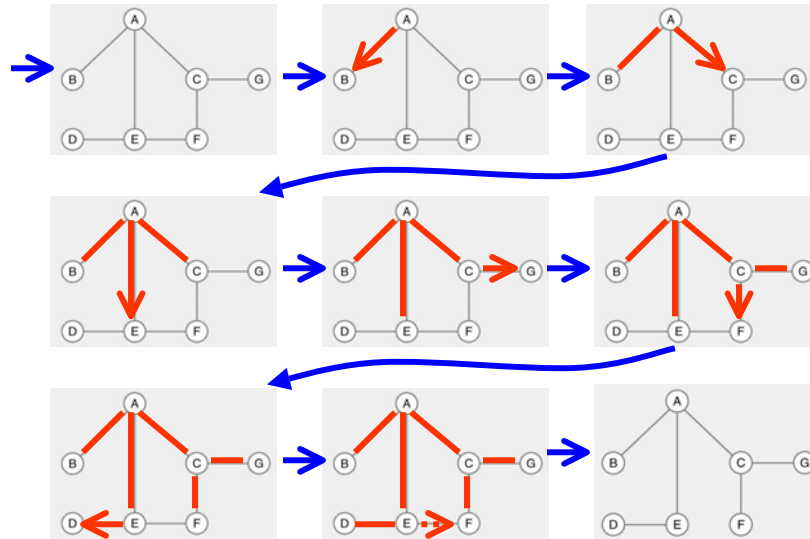
Breadth & Depth First Spanning Trees



Depth-First Spanning Tree Example



Breadth-First Spanning Tree Example

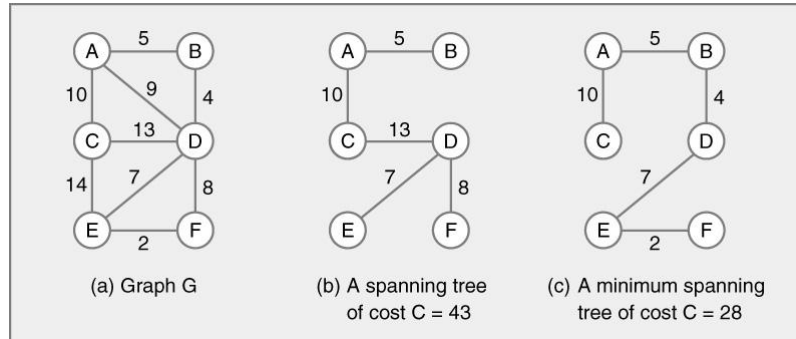


Spanning Tree Construction

- **Multiple spanning trees possible**
 - **Different breadth-first traversals**
 - Nodes same distance visited in different order
 - **Different depth-first traversals**
 - Neighbors of node visited in different order
 - **Different traversals yield different spanning trees**

Minimum Spanning Tree (MST)

- Spanning tree with minimum total edge weight
- Multiple MSTs possible (with same weight)



MST – Kruskal's Algorithm

sort edges by weight (from least to most)

tree = \emptyset

for each edge (X,Y) in order

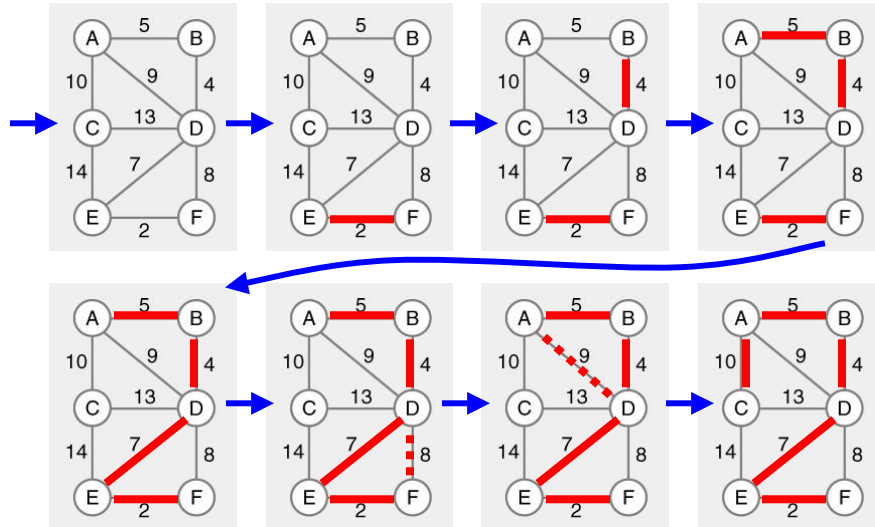
if it does not create a cycle

add (X,Y) to tree

stop when tree has $N-1$ edges

Optimal solution computed with **greedy** algorithm

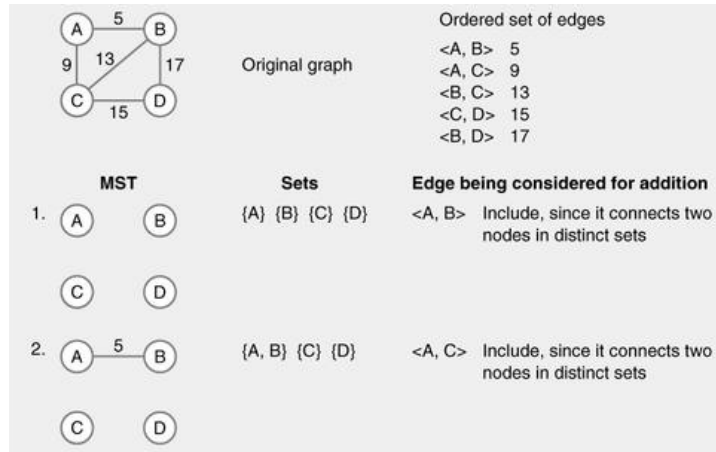
MST – Kruskal's Algorithm Example



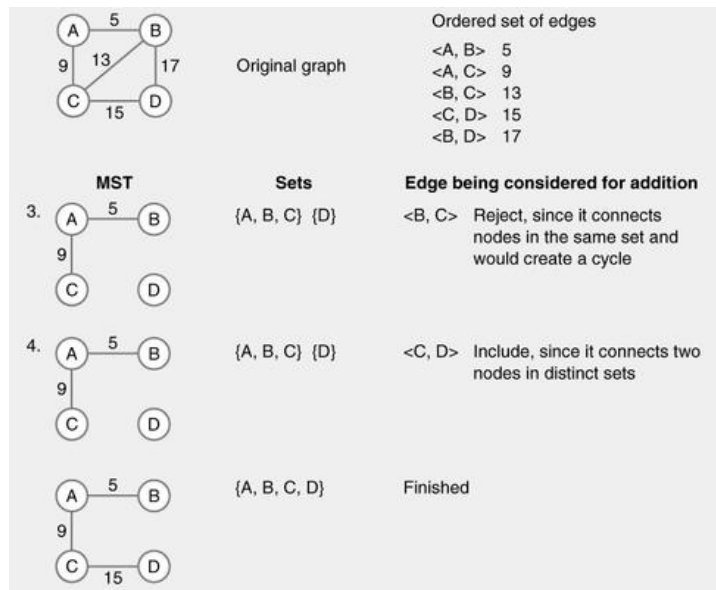
MST – Kruskal's Algorithm

- When does adding (X,Y) to tree create cycle?
 1. Traversal approach
 1. Traverse tree starting at X
 2. If we can reach Y, adding (X,Y) would create cycle
 2. Connected subgraph approach
 1. Maintain set of nodes for each connected subgraph
 2. Initialize one connected subgraph for each node
 3. If X, Y in same set, adding (X,Y) would create cycle
 4. Otherwise
 - We can add edge (X,Y) to spanning tree
 - Merge sets containing X, Y (single subgraph)

MST – Connected Subgraph Example



MST – Connected Subgraph Example



Single Source Shortest Path

- **Common graph problem**
 - Find path from X to Y with lowest edge weight
 - Find path from X to **any** Y with lowest edge weight
- **Useful for many applications**
 - Shortest route in map
 - Lowest cost trip
 - Most efficient internet route
- **Can solve both problems with same algorithm**

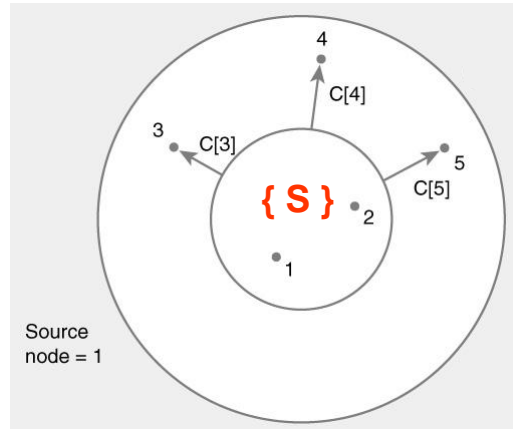
Shortest Path – Dijkstra's Algorithm

- **Maintain**
 - Nodes with known shortest path from start $\Rightarrow \{ S \}$
 - Cost of shortest path to node K from start $\Rightarrow C[K]$
 - Only for paths through nodes in $\{ S \}$
 - Predecessor to K on shortest path $\Rightarrow P[K]$
 - Updated whenever new (lower) $C[K]$ discovered
 - Remembers actual path with lowest cost
 - **Extension to algorithm in book**

Shortest Path – Intuition for Dijkstra's

■ At each step in the algorithm

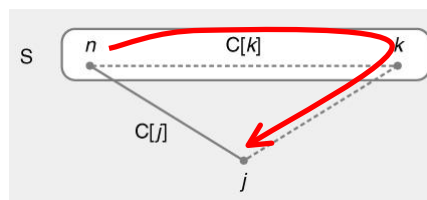
- Shortest paths are known for nodes in $\{S\}$
- Store in $C[K]$ length of shortest path to node K (for all paths through nodes in $\{S\}$)
- Add to $\{S\}$ next closest node



Shortest Path – Intuition for Dijkstra's

■ Update distance to J after adding node K

- Previous shortest paths already in $C[K]$
- Possibly shorter path by going through node K
- Compare $C[J]$ to $C[K] + \text{weight of } (K,J)$



Shortest Path – Dijkstra’s Algorithm

■ Algorithm

- Add starting node to { S }
- Repeat until all nodes in { S }
 - Find node K not in { S } with smallest C[K]
 - Add K to { S }
 - Examine C[J] for all neighbors J of K not in { S }
 - If (C[K] + weight for edge (K,J)) < C[J]
 - New shortest path by first going to K, then J
 - Update C[J] ← C[K] + weight for edge (K,J)
 - Update P[J] ← K

Shortest Path – Dijkstra’s Algorithm

{ S } = \emptyset , P[] = none for all nodes
C[start] = 0, C[] = ∞ for all other nodes
while (not all nodes in { S })
 find node K not in { S } with smallest C[K]
 add K to { S }
 for each node J not in { S } adjacent to K
 if (C[K] + cost of (K,J) < C[J])
 C[J] = C[K] + cost of (K,J)
 P[J] = K

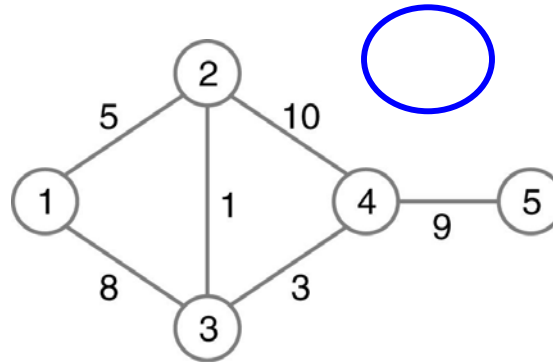
Optimal solution computed with **greedy** algorithm

Dijkstra's Shortest Path Example

■ Initial state

■ $\{ S \} = \emptyset$

	C	P
1	0	none
2	∞	none
3	∞	none
4	∞	none
5	∞	none

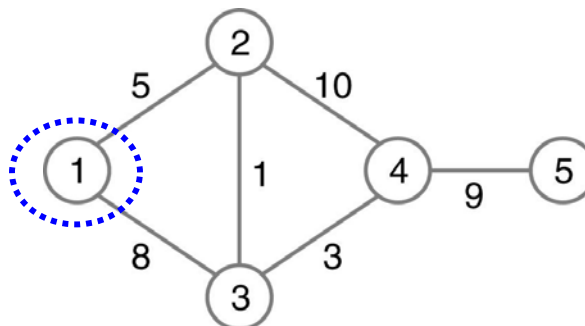


Dijkstra's Shortest Path Example

■ Find node K with smallest C[K] and add to $\{ S \}$

■ $\{ S \} = 1$

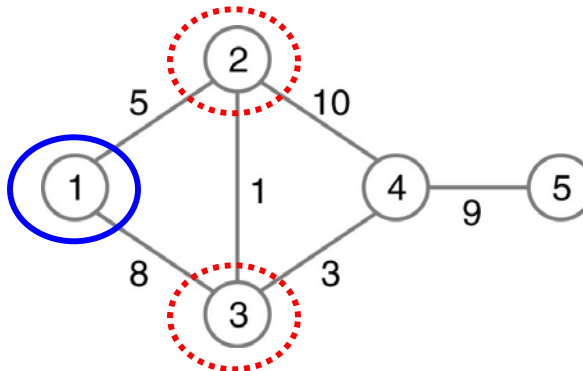
	C	P
1	0	none
2	∞	none
3	∞	none
4	∞	none
5	∞	none



Dijkstra's Shortest Path Example

- Update $C[K]$ for all neighbors of 1 not in $\{S\}$
- $\{S\} = 1$

	C	P
1	0	none
2	5	1
3	8	1
4	∞	none
5	∞	none



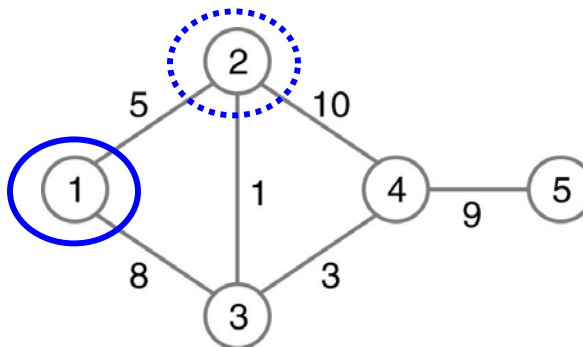
$$C[2] = \min(\infty, C[1] + (1,2)) = \min(\infty, 0 + 5) = 5$$

$$C[3] = \min(\infty, C[1] + (1,3)) = \min(\infty, 0 + 8) = 8$$

Dijkstra's Shortest Path Example

- Find node K with smallest $C[K]$ and add to $\{S\}$
- $\{S\} = 1, 2$

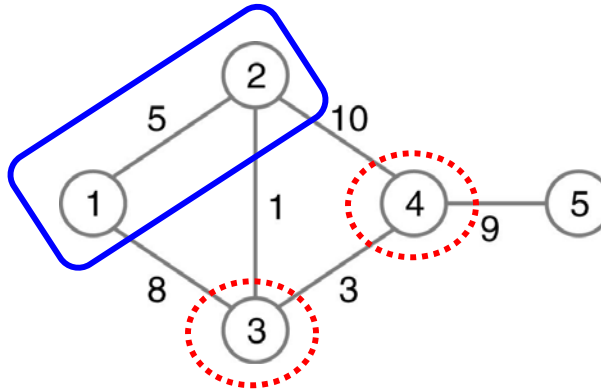
	C	P
1	0	none
2	5	1
3	8	1
4	∞	none
5	∞	none



Dijkstra's Shortest Path Example

- Update $C[K]$ for all neighbors of 2 not in $\{S\}$
- $\{S\} = 1, 2$

	C	P
1	0	none
2	5	1
3	6	2
4	15	2
5	∞	none



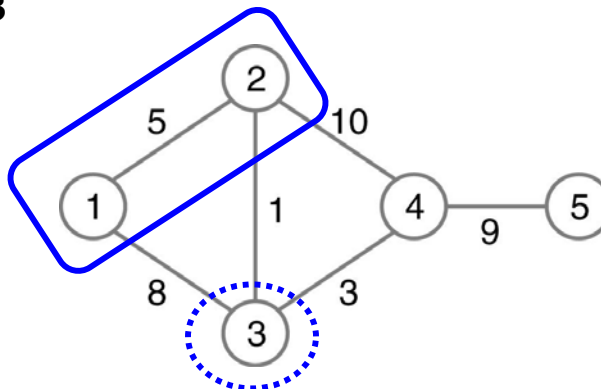
$$C[3] = \min(8, C[2] + (2,3)) = \min(8, 5 + 1) = 6$$

$$C[4] = \min(\infty, C[2] + (2,4)) = \min(\infty, 5 + 10) = 15$$

Dijkstra's Shortest Path Example

- Find node K with smallest $C[K]$ and add to $\{S\}$
- $\{S\} = 1, 2, 3$

	C	P
1	0	none
2	5	1
3	6	2
4	15	2
5	∞	none

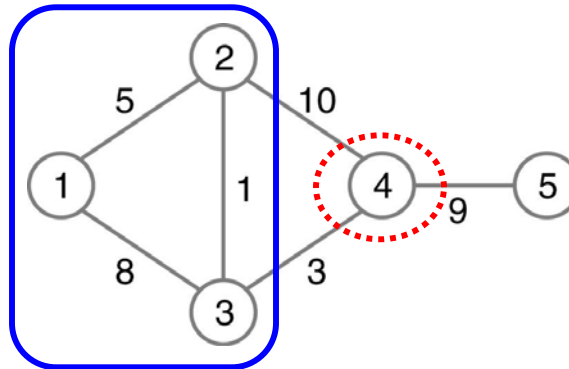


Dijkstra's Shortest Path Example

■ Update $C[K]$ for all neighbors of 3 not in $\{S\}$

■ $\{S\} = 1, 2, 3$

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	∞	none



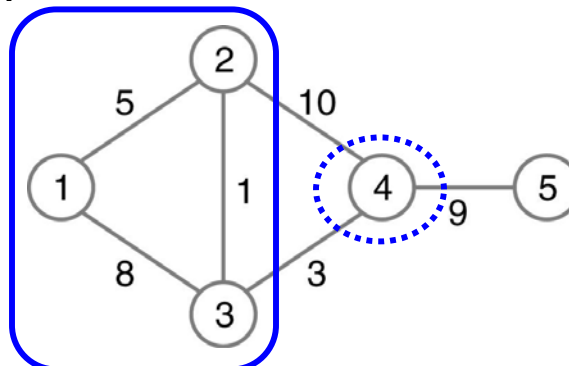
$$C[4] = \min(15, C[3] + (3,4)) = \min(15, 6 + 3) = 9$$

Dijkstra's Shortest Path Example

■ Find node K with smallest $C[K]$ and add to $\{S\}$

■ $\{S\} = 1, 2, 3, 4$

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	∞	none

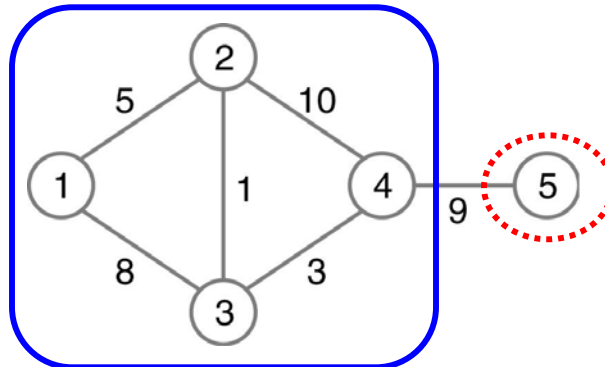


Dijkstra's Shortest Path Example

■ Update $C[K]$ for all neighbors of 4 not in $\{ S \}$

■ $\{ S \} = 1, 2, 3, 4$

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



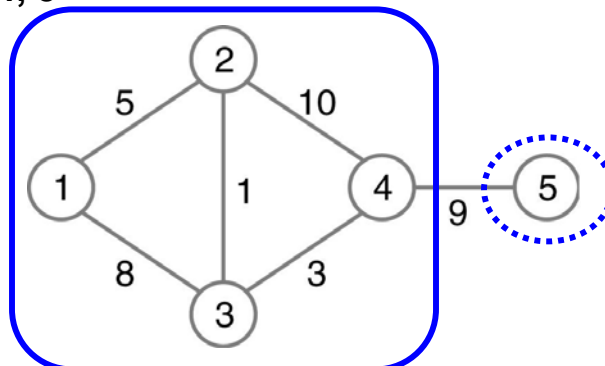
$$C[5] = \min(\infty, C[4] + (4,5)) = \min(\infty, 9 + 9) = 18$$

Dijkstra's Shortest Path Example

■ Find node K with smallest $C[K]$ and add to $\{ S \}$

■ $\{ S \} = 1, 2, 3, 4, 5$

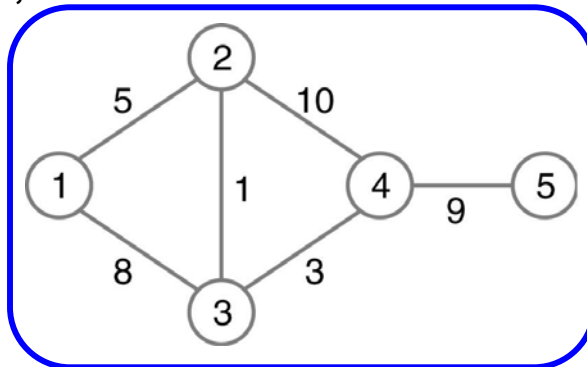
	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



Dijkstra's Shortest Path Example

- All nodes in { S }, algorithm is finished
- { S } = 1, 2, 3, 4, 5

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



Dijkstra's Shortest Path Example

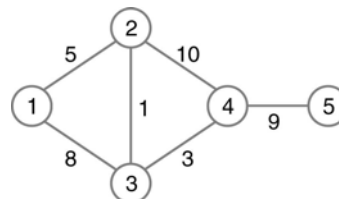
- Find shortest path from start to K

- Start at K
- Trace back predecessors in P[]

- Example paths (in reverse)

- 2 → 1
- 3 → 2 → 1
- 4 → 3 → 2 → 1
- 5 → 4 → 3 → 2 → 1

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



Graph Implementation

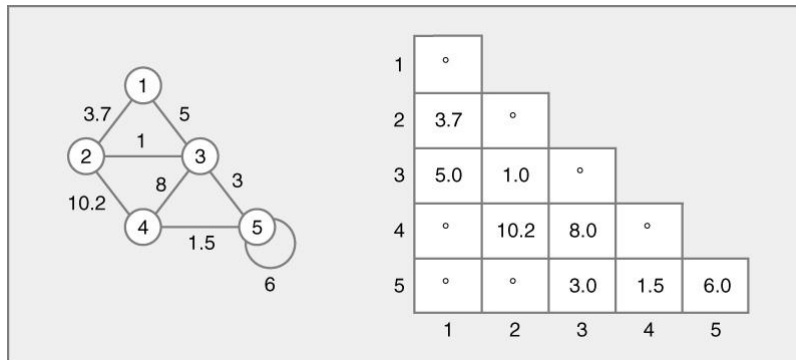
- **Representations**
 - **Explicit edges (a,b)**
 - Maintain set of edges for every node
 - **Adjacency matrix**
 - 2D array of neighbors
 - **Adjacency list**
 - Linked list of neighbors
- **Important for very large graphs**
 - Affects efficiency / storage

Adjacency Matrix

- **Representation**
 - 2D array
 - Position $j, k \Rightarrow$ edge between nodes n_j, n_k
 - **Unweighted graph**
 - Matrix elements \Rightarrow boolean
 - **Weighted graph**
 - Matrix elements \Rightarrow weight

Adjacency Matrix

■ Example



Adjacency Matrix

■ Properties

- Single array for entire graph
- Only upper / lower triangle matrix needed for undirected graph
 - Since n_j, n_k implies n_k, n_j

Adjacency List

■ Representation

- Linked list for each node
- Unweighted graph
 - store neighbor
- Weighted graph
 - store neighbor, weight

Adjacency List

■ Example

■ Unweighted graph

<i>Node</i>	<i>Neighbor List</i>
1	2 → 3
2	1 → 3 → 4
3	1 → 2 → 4 → 5
4	2 → 3 → 5
5	3 → 4 → 5

■ Weighted graph

<i>Node</i>	<i>Neighbor List</i>
1	(2, 3.7) → (3, 5.0)
2	(1, 3.7) → (3, 1.0) → (4, 10.2)
3	(1, 5.0) → (2, 1.0) → (4, 8.0) → (5, 3.0)
4	(2, 10.2) → (3, 8.0) → (5, 1.5)
5	(3, 3.0) → (4, 1.5) → (5, 6.0)

Graph Space Requirements

- **Adjacency matrix**
 - $\frac{1}{2} N^2$ entries (for graph with N nodes, E edges)
 - Many empty entries for large graphs
 - Can implement as sparse array
- **Adjacency list**
 - E edges
 - Each edge stores reference to node & next edge
- **Explicit edges**
 - E edges
 - Each edge stores reference to 2 nodes

Graph Time Requirements

- **Complexity of operations**
 - For graph with N nodes, E edges

Operation	Adj Matrix	Adj List
Find edge	$O(1)$	$O(E/N)$
Insert node	$O(1)$	$O(E/N)$
Insert edge	$O(1)$	$O(E/N)$
Delete node	$O(N)$	$O(E)$
Delete edge	$O(1)$	$O(E/N)$