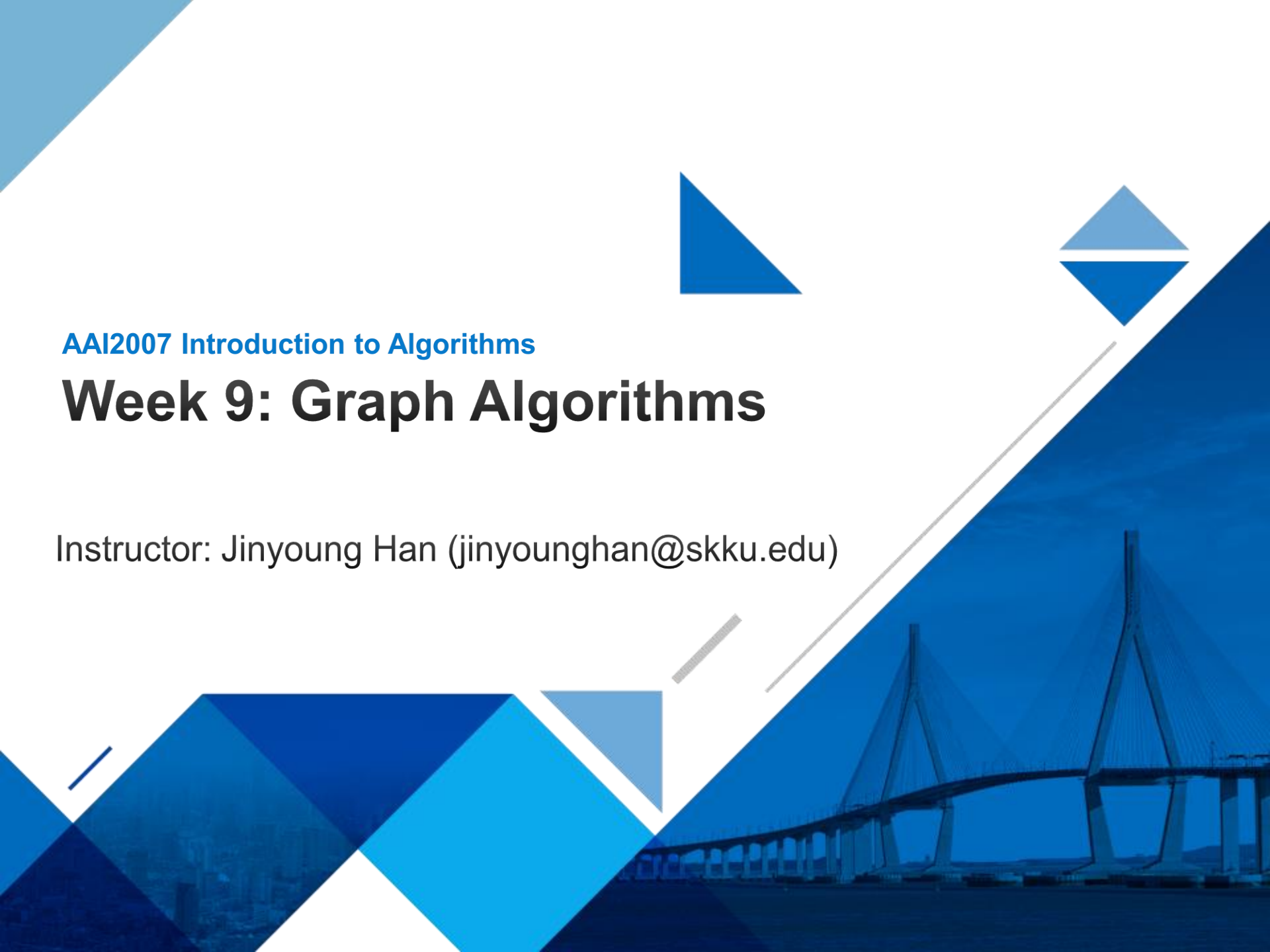
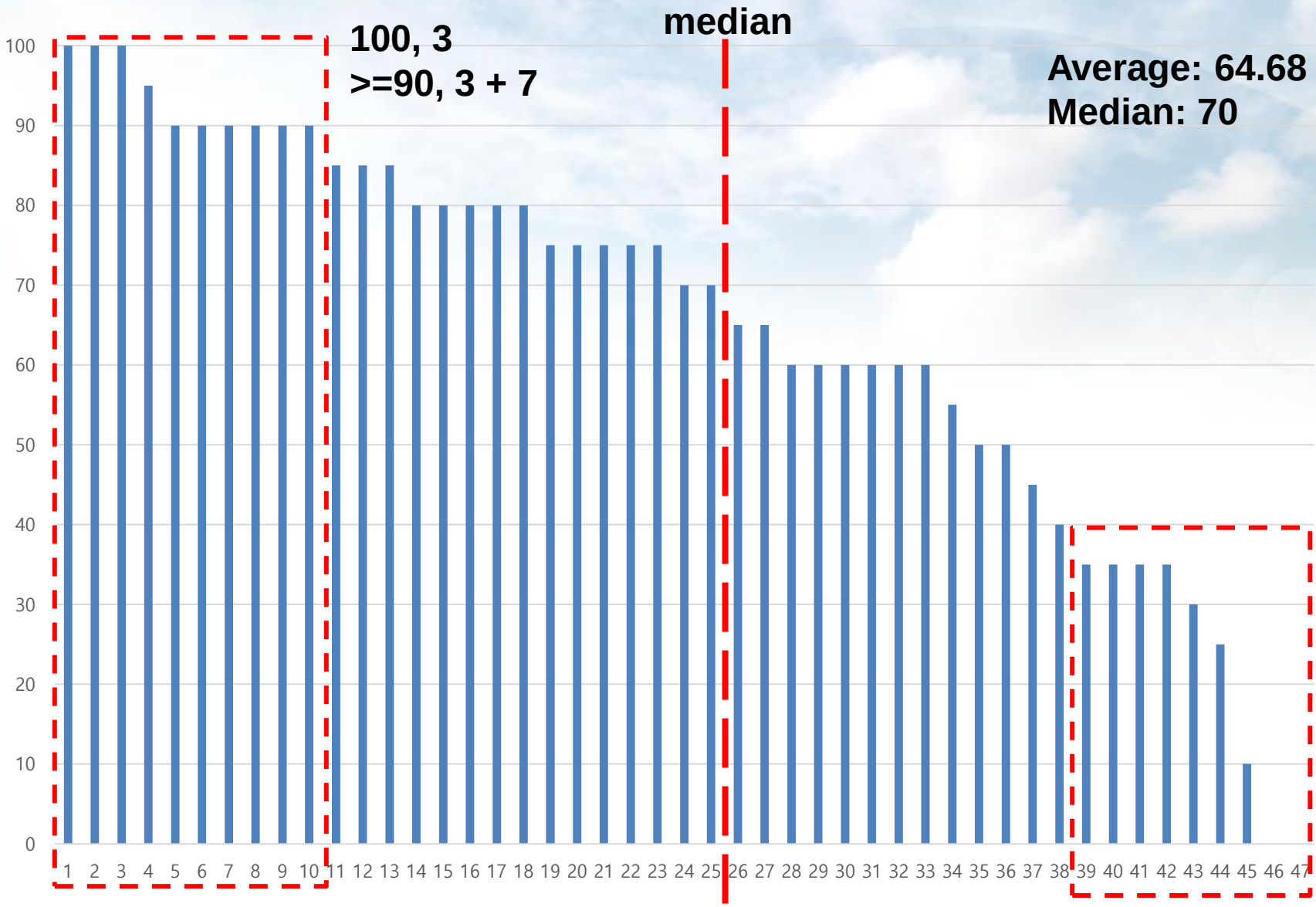
AAI2007 Introduction to Algorithms

Week 9: Graph Algorithms

Instructor: Jinyoung Han (jinyounghan@skku.edu)



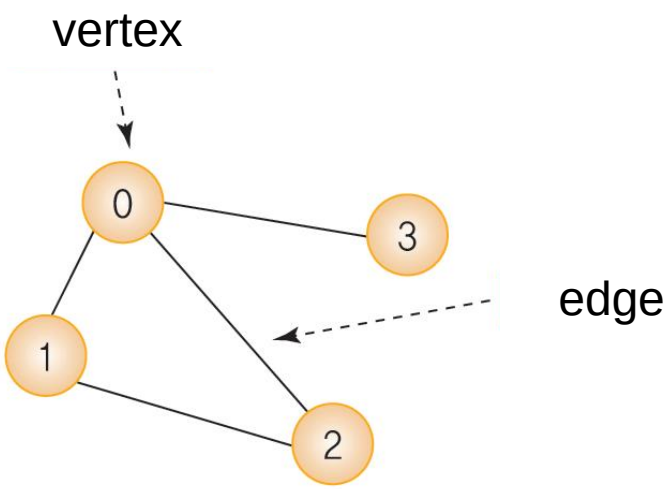
Midterm


100, 3
>=90, 3 + 7
median
Average: 64.68
Median: 70
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

Graph Revisited

Definition

- Graph G = (V, E)
  - V: a set of vertices
  - E: a set of edges
- Vertex (or node)
  - Any object such as human, web page, …
- Edge (or link)
  - Relation between two nodes

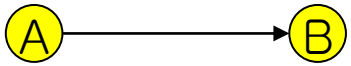


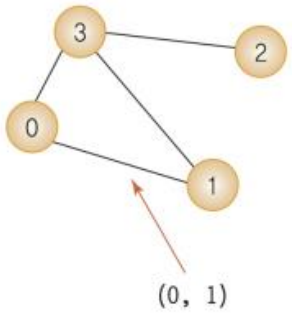
Graph Revisited

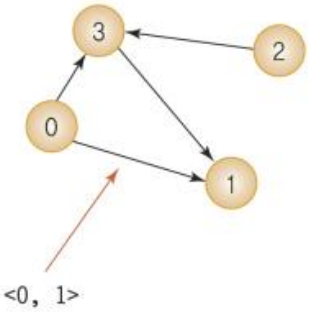
Graph Type

- Undirected graph
  - Uses only undirected edges
  - Bidirectional
  - Represented by (A,B)
  - (A,B) = (B,A)

- Directed graph
  - Uses directed edges
  - One-way
  - Represented by <A,B>
  - <A,B> ≠ <B,A>





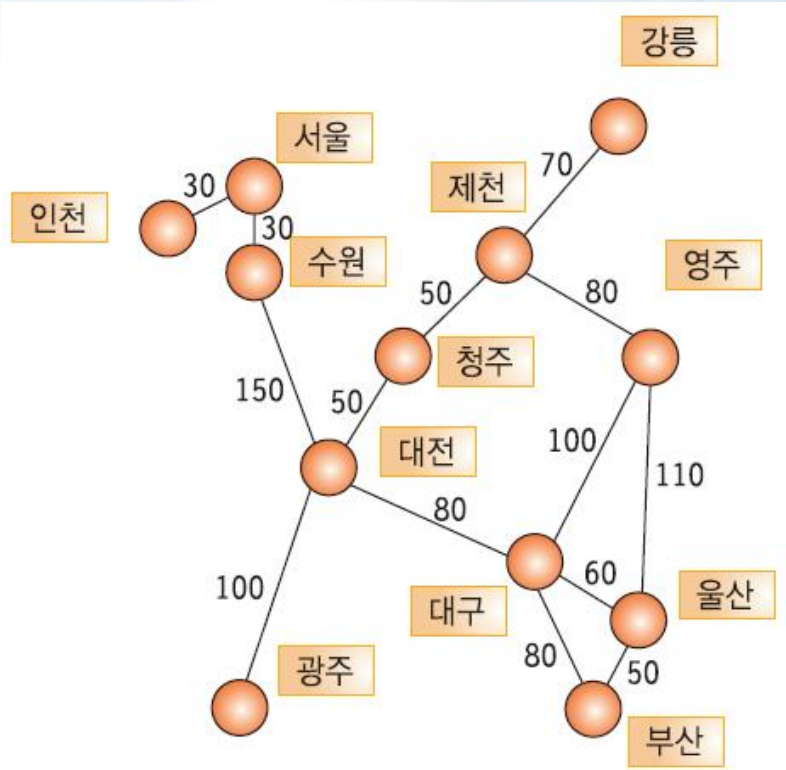


Graph Revisited

Graph Type

- Weighted graph
  - Cost or weight on an edge

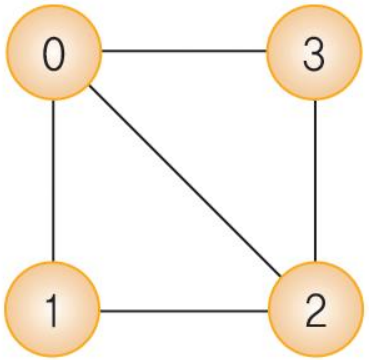
- Examples
  - Vertex: city
  - Edge: road between two cities
  - Weight: distance or time



Graph Revisited

Degree

- Adjacent vertex
  - Directly connected edges from a vertex
  - Vertex 0's adjacent vertex in G1: vertices 1, 2, and 3
- Degree in undirected graph
  - Number of adjacent vertices of a vertex
  - 0's degree in G1: 3
  - Sum of degree = 2 * number of edges
    - Sum of degree in G1 = 10
    - # edges in G1 = 5

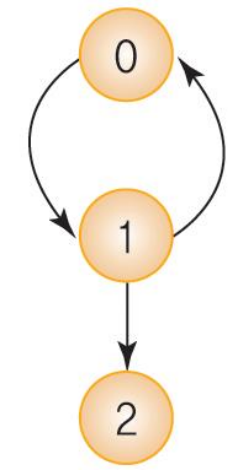


G1

Graph Revisited

Degree

- Degree in directed graph
  - In-degree: edges coming from outside
  - Out-degree: edges going for outside
  - Vertex 1's degree in G3
    - In-degree: 1
    - Out-degree: 2
  - Sum of in-(or out-)degree = # edges
    - Sum of in-degree in G3 = 3
    - Sum of out-degree in G3 = 3
    - # edges in G3 = 3



G3

Graph Revisited

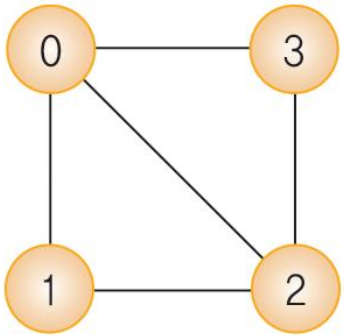
Graph Path

- A path from vertex s to vertex e in an undirected graph
  - s, v1, v2, ..., vk, e
  - Edges (s, v1), (v1, v2), ... , (vk, e) EXIST
- A path from vertex s to vertex e in a directed graph
  - s, v1, v2, ..., vk, e
  - Edges <s, v1>, <v1, v2>, ... ,<vk, e> EXIST
- Path length
  - Number of edges consisting of a path
- Simple path
  - Path with out repeating edges
- Cycle
  - Simple path where starting and ending vertices are identical

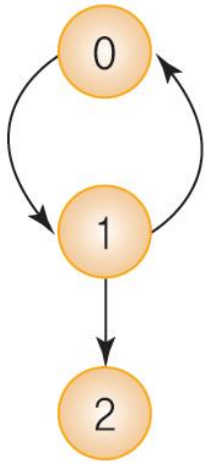
Graph Revisited

Example

- G1's 0, 1, 2, 3 is a path, but 0, 1, 3, 2 is not a path
- G1's 1, 0, 2, 3 is a simple path, but 1, 0, 2, 0 is not
- G1's 0, 1, 2, 0 and G3's 0, 1, 0 are cycles



G1

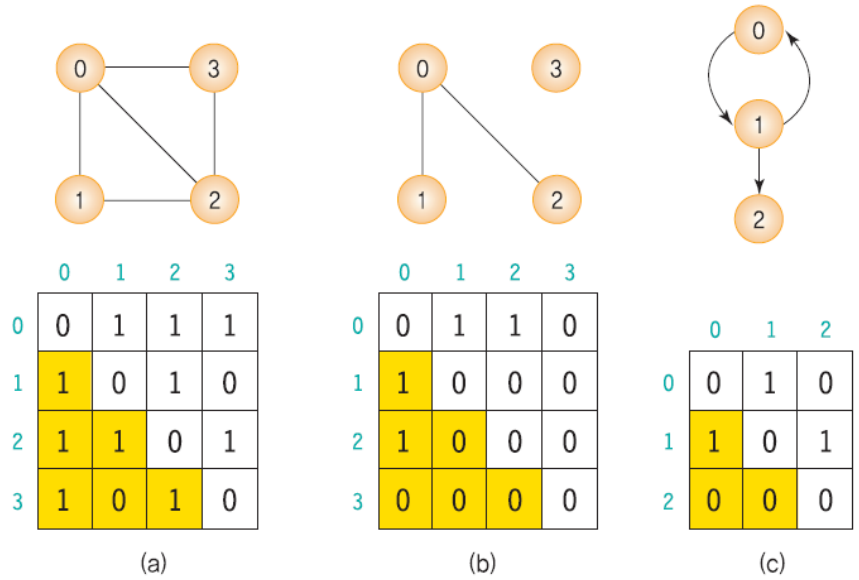


G3

Graph Revisited

Representation

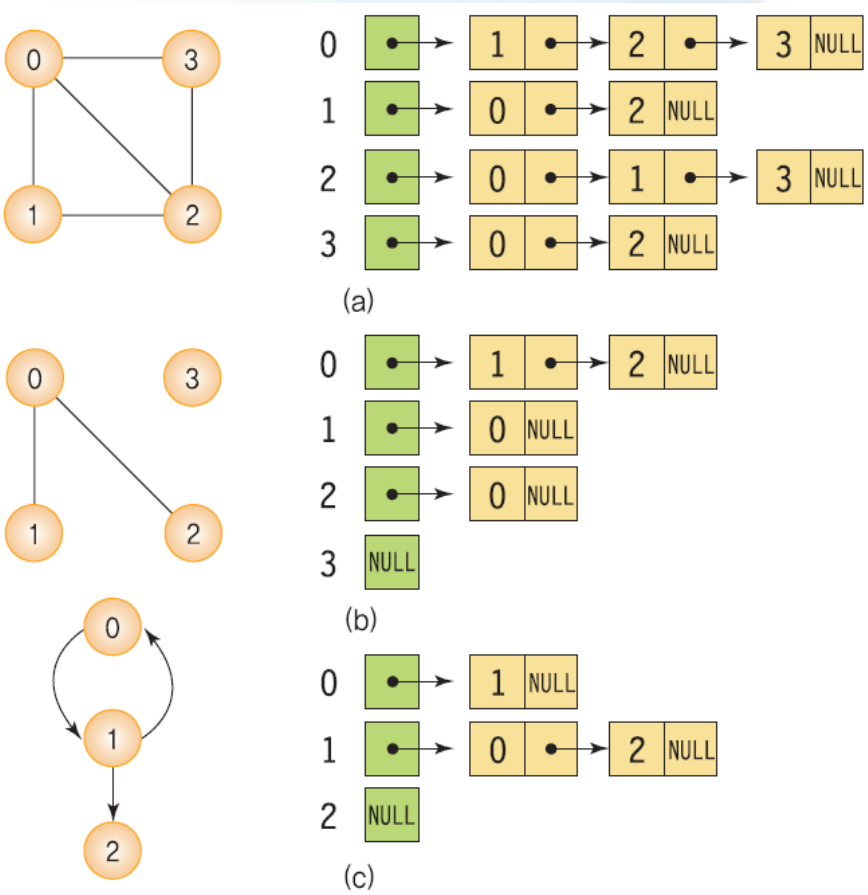
- Adjacency Matrix
  - If there exists edge (i, j) in a graph, M[i][j] = 1
  - Otherwise, M[i][j] = 0
- Diagonal elements in an adjacency matrix is 0
- Adjacency matrix is symmetric in an undirected graph



Graph Revisited

Representation

- Adjacency list
  - Use a linked list of adjacency vertices for each vertex



In This Lecture

Outline

1. Minimum Spanning Tree
2. Shortest Paths

01. Minimum Spanning Tree

Spanning Tree

- A tree includes all the nodes in a graph
- All the nodes in a tree should be connected, and no cycle exists
- A spanning tree with n nodes has n-1 edges
- Used in constructing a network with minimum links
  - Computer network, road network, distribution network

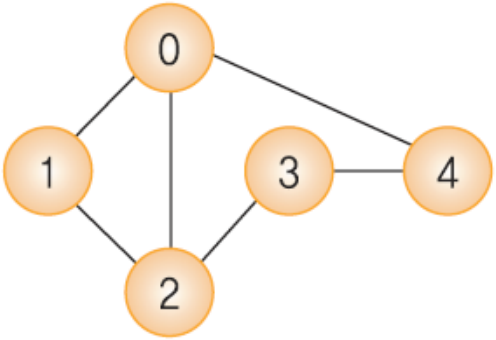
- A spanning tree algorithm

```
depth_first_search(v)

mark v as visited;
for all u ∈ (v's adjacent vertices) do
        if (u is not visited)
                then mark (v,u) as an edge of a spanning tree;
                        depth_first_search(u);
```

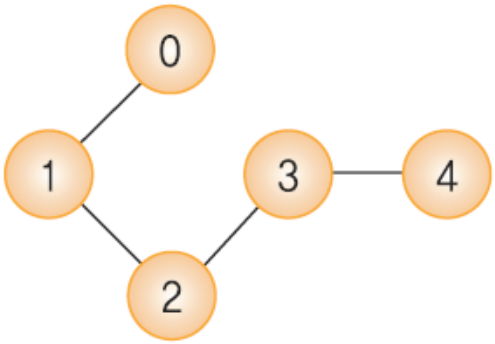
01. Minimum Spanning Tree

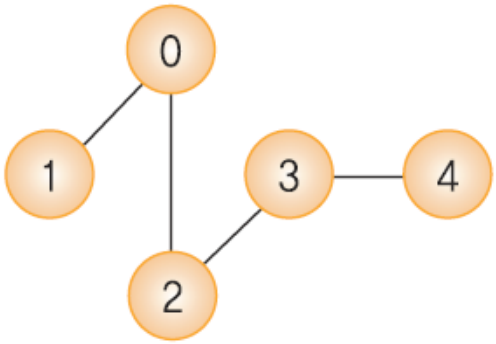
Spanning Tree

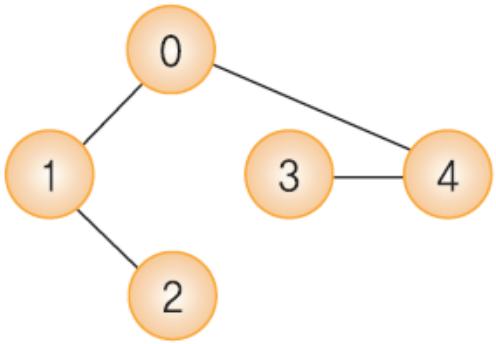
- Spanning tree example



connected graph







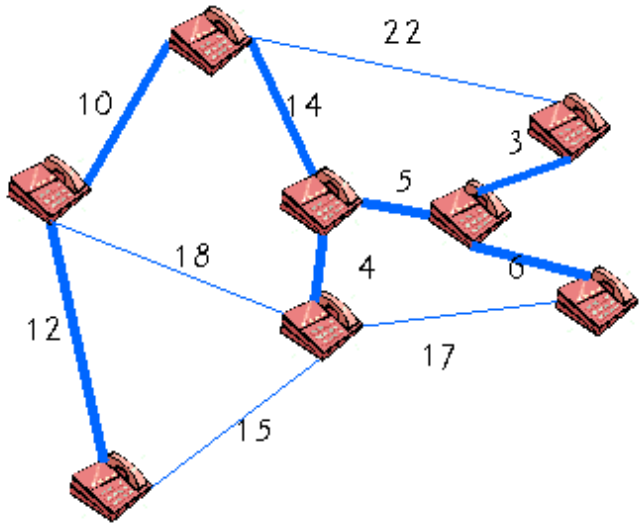
spanning trees

01. Minimum Spanning Tree

MST

- Minimum spanning tree (MST)
  - All the nodes in a network are connected with minimum edges/costs

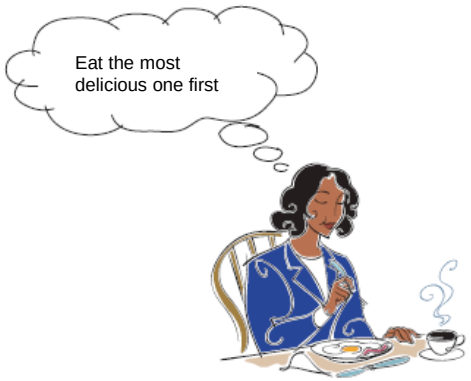
- MST applications
  - Road construction: cities are connected and the lengths of the roads become minimum
  - Electrical circuit: devices are all connected with minimum wires
  - Networks: with minimum telephone lines, all the telephones are connected
  - Pipes: all the pipes are connected with minimum length of pipes



01. Minimum Spanning Tree

Kruskal MST

- A greedy method
  - One of the well-known algorithm designs
  - For each step, choose the best answer iteratively, and finally reach to the final solution
  - Need to verify whether the algorithm gives the optimal solution
  - Kruskal's MST
    - Proved as optimal



01. Minimum Spanning Tree

Kruskal MST

- Algorithm

Algorithm Kruskal(G):
Input: A simple connected weighted graph G=(V,E) with n vertices
Output: A minimum spanning tree T for G

1. Sort E in non-decreasing order in terms of weight.

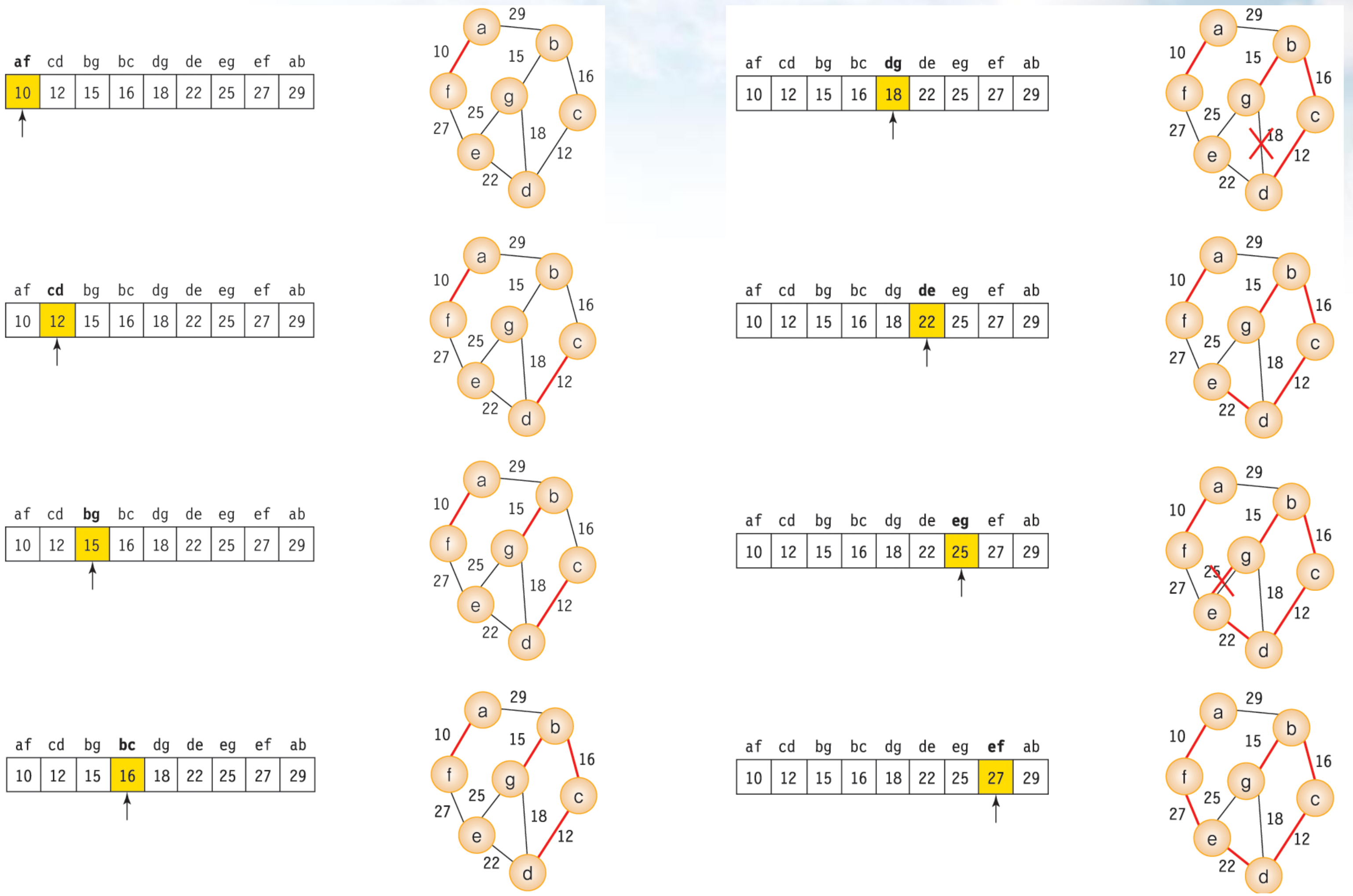
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

01. Minimum Spanning Tree

Kruskal MST

- Example



01. Minimum Spanning Tree

Kruskal MST

- Complexity
  - Mostly depends on sorting edges
    - Other jobs, such as testing a cycle, etc., can be computed quickly compared to sorting jobs
  - If ‘e’ edges are sorted by efficient algorithms such as quick sorting, the time complexity is O(e * log(e))

01. Minimum Spanning Tree

Prim MST

- Prim's MST algorithm

1. Create a set mstSet that keeps track of vertices already included in MST.

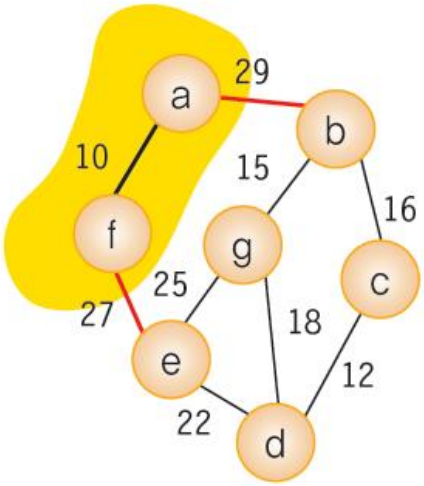
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3. While mstSet doesn't include all vertices
   ① Pick a vertex u which is not there in mstSet and has minimum key value.
   ② Include u to mstSet.
   ③ Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

01. Minimum Spanning Tree

Prim MST

- Prim's MST algorithm



edge (a, b)=29
edge (f, e)=27

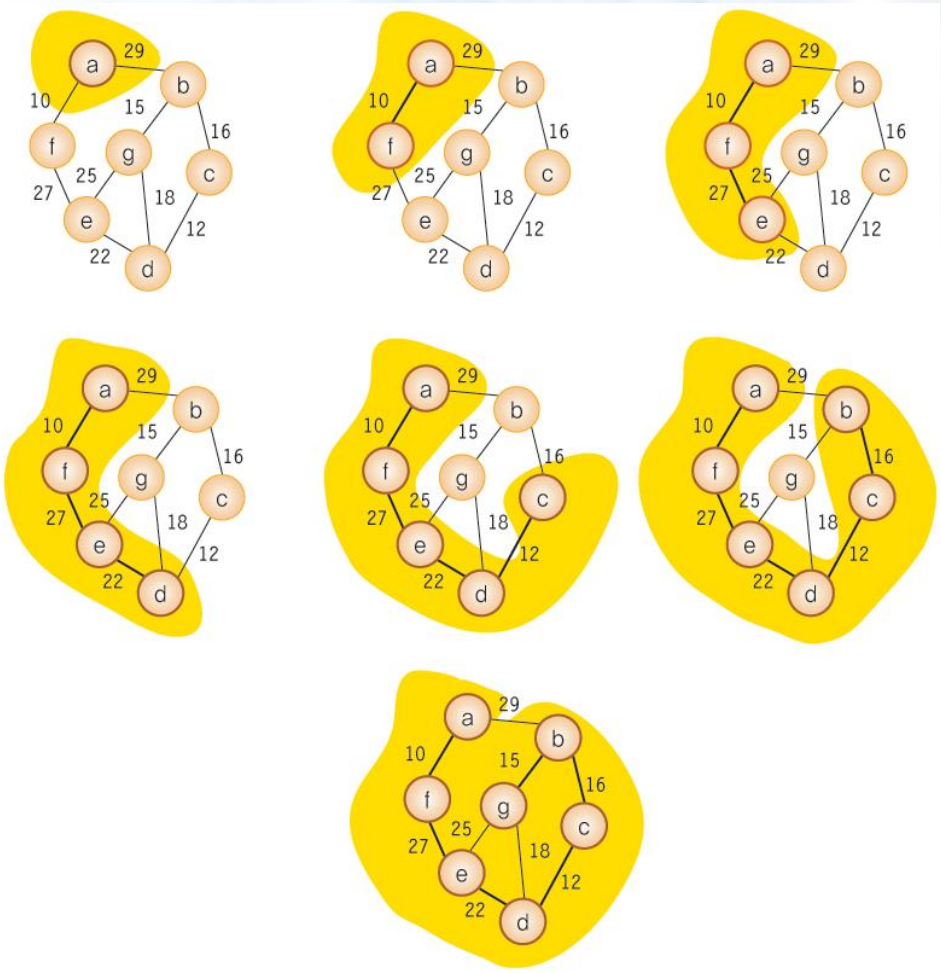
select edge (f, e)

vertex e is added
to mstSet

01. Minimum Spanning Tree

Prim MST

- Prim's MST algorithm



01. Minimum Spanning Tree

Prim MST

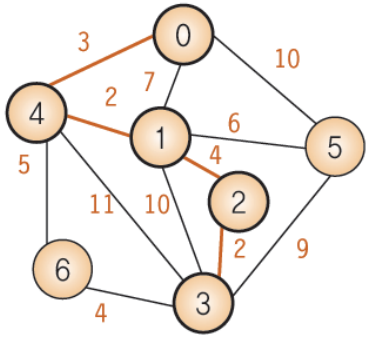
- Complexity
  - For each node, iterate # its adjacent nodes, hence, $O(n^2)$
- For a sparse graph,
  - Kruskal may be favorable
    - O(e*log(e))
- For a dense graph,
  - Prim may be favorable
    - $O(n^2)$

Shortest Paths

02. Shortest Path

Shortest Path

- Among the paths that connect node u and node v in the given network, the path where sum of weights on the edges is minimum
  - Weight can be cost, distance, time, etc.
- A problem: to find the shortest path from node 0 to node 3
  - Adjacent matrix
    - If there is no direct edge, its weight is ∞
  - 0, 4, 1, 2, 3 is the shortest path
    - Length of the shortest path = 3 + 2 + 4 + 2 = 11

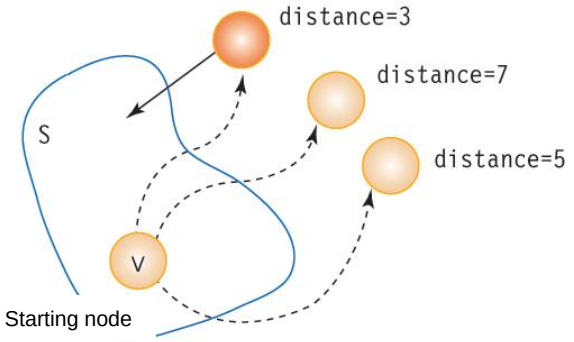


	0	1	2	3	4	5	6
0	0	7	∞	∞	3	10	∞
1	7	0	4	10	2	6	∞
2	∞	4	0	2	∞	∞	∞
3	∞	10	2	0	11	9	4
4	3	2	∞	11	0	∞	5
5	10	6	∞	9	∞	0	∞
6	∞	∞	∞	4	5	∞	0

02. Shortest Path

Dijkstra

- Algorithm
  - Search the shortest paths from “a starting node” to all the other nodes
  - Set S
    - A set of nodes (already) included in the shortest path starting from node v
  - Distance variables
    - The shortest paths from node v to other nodes
    - Initialization (starting node v)
      - distance[v] = 0
      - distance[n] = w(v, w) if an edge exists,
        ∞ otherwise

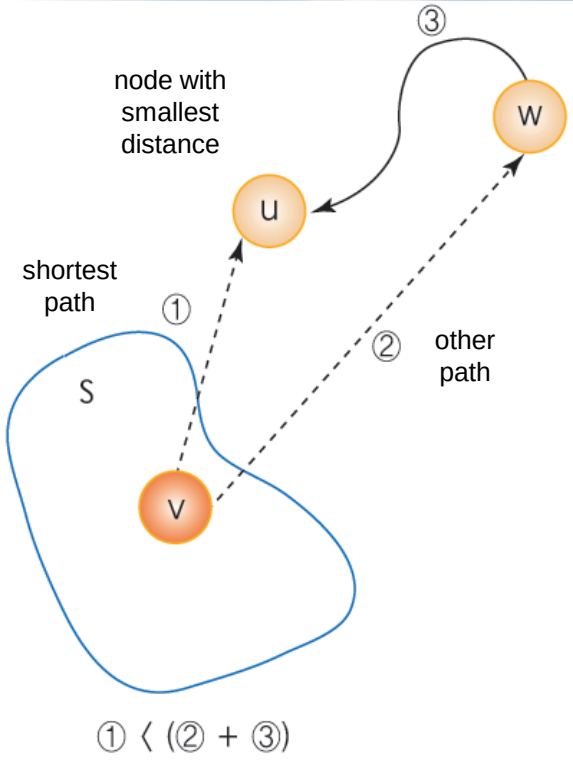


  - For each step, a node with the smallest distance is added to S

02. Shortest Path

Dijkstra

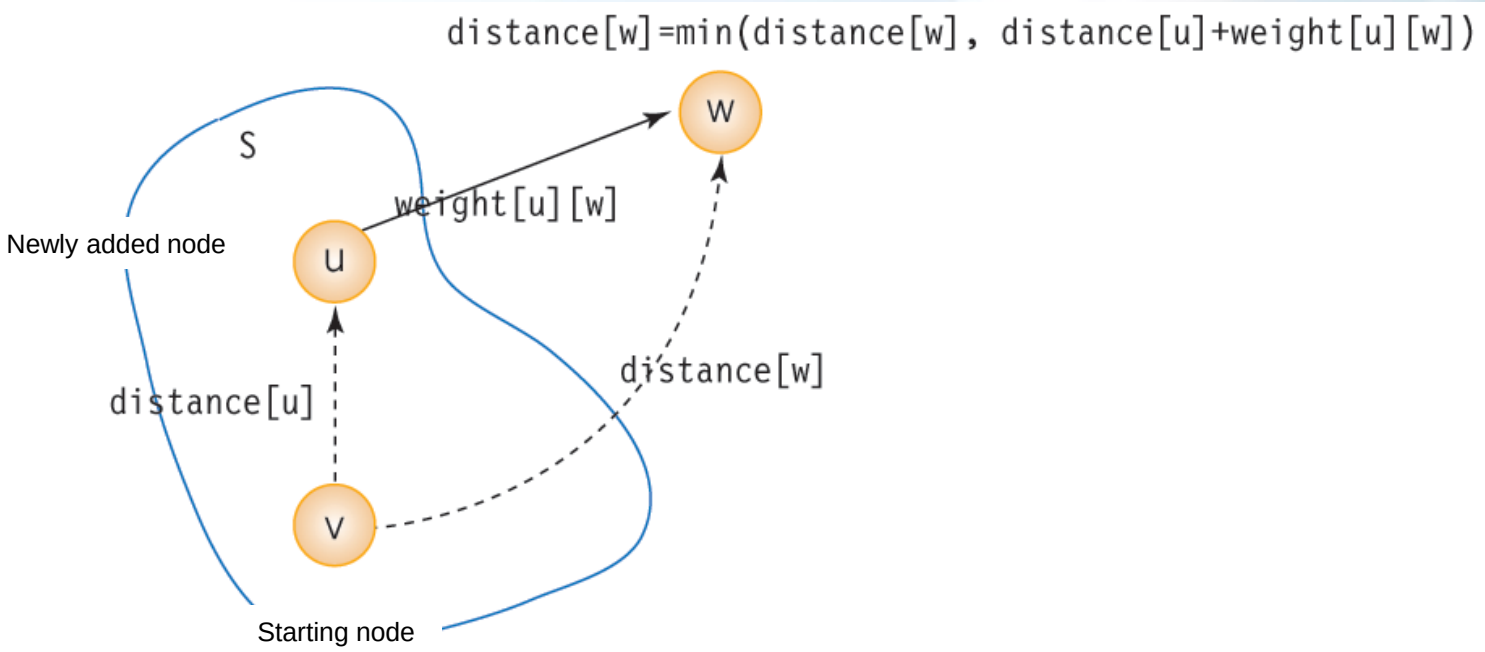
- Algorithm
  - For node u with smallest distance, the shortest path from v to u is ①
  - If there is a shorter path through w, the shortest path from v to u is (v,w)(②) + (w,u)(③)
  - But always, ② > ①, by definition
  - So, for each step, add a node with smallest distance, and then finally obtain all the shortest paths to other nodes



02. Shortest Path

Dijkstra

- Algorithm
  - Update distance variables if a node is added to S



02. Shortest Path

Dijkstra

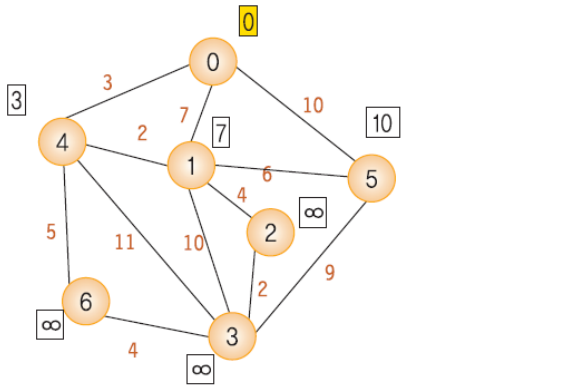
- Pseudo code

```
// input: a (non-negative) weighted graph G
// output: distance array, distance[u] is the shortest path from v to u

shortest_path(G, v)

S←{v}
for each node w∈G do
        distance[w]←weight[v][w];
while all the nodes are not contained in S do
        u← node with smallest distance, which are not included in S;
        S←S∪{u}
        for u's adjacent and member of S "z" do
                if distance[u]+weight[u][z] < distance[z]
                        then distance[z]←distance[u]+weight[u][z];
```

02. Shortest Path

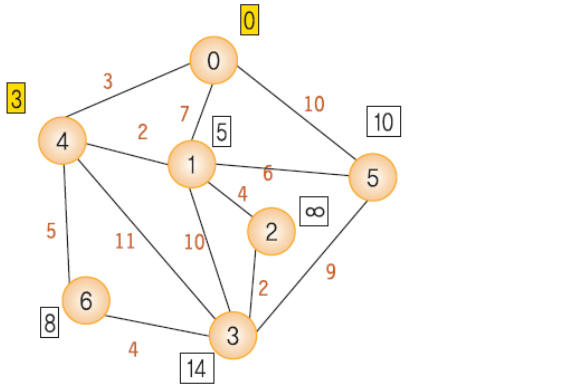
Dijkstra

- Example



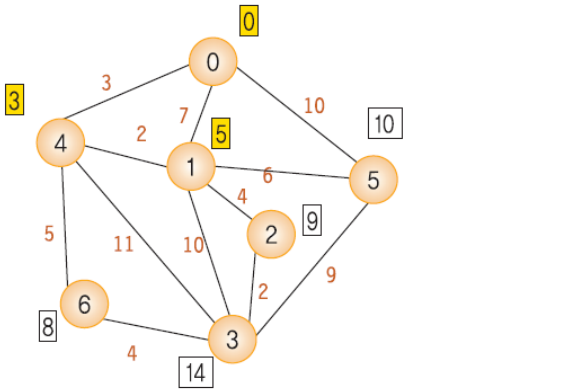
S={0}

	0	1	2	3	4	5	6
distance[]=	0	7	∞	∞	3	10	∞



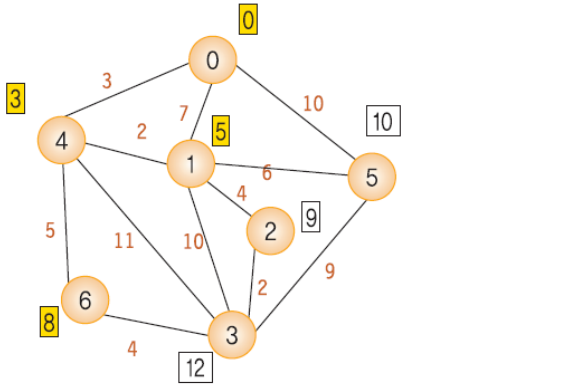
S={0, 4}

	0	1	2	3	4	5	6
distance[]=	0	5	∞	14	3	10	8



S={0, 4, 1}

	0	1	2	3	4	5	6
distance[]=	0	5	9	14	3	10	8



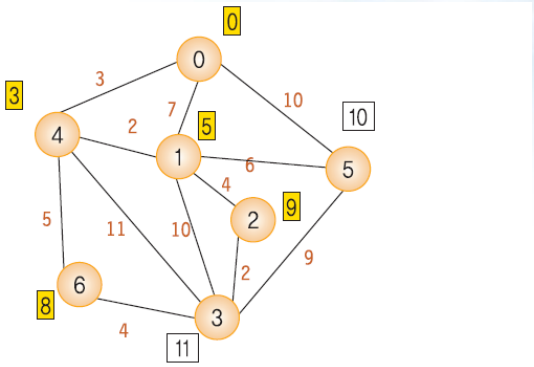
S={0, 4, 1, 6}

	0	1	2	3	4	5	6
distance[]=	0	5	9	12	3	10	8

02. Shortest Path

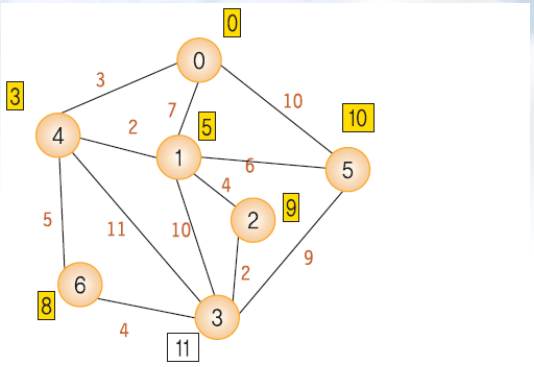
Dijkstra

- Example (cont'd)



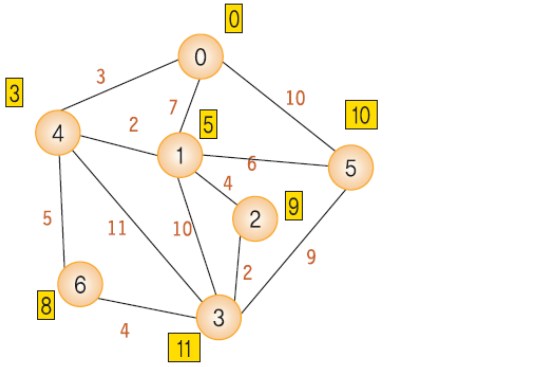
S={0, 4, 1, 6, 2}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8



S={0, 4, 1, 6, 2, 5, 3}

	0	1	2	3	4	5	6
distance[]=	0	5	9	11	3	10	8

02. Shortest Path

Dijkstra

- Complexity
  - Given n nodes in G, complexity of Dijkstra is $O(n^2)$

02. Shortest Path

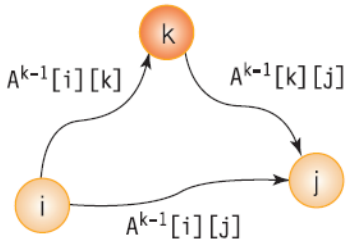
Floyd

- Algorithm
  - Find shortest paths among all the nodes
    - Cf) Dijkstra?
  - Adjacent weight array A
    - If i == j, weight[i][j] = 0
    - If no edge exists between i and j, weight[i][j] = ∞
    - If an edge exists between i and j, weight[i][j] = weight between i and j
    - Initialization: weight matrix

02. Shortest Path

Floyd

- $A^K[i][j]$
  - Shortest path from i to j with 0, …, k nodes
- Get shortest path in an order, $A^{-1} \rightarrow A^0 \rightarrow A^1 \rightarrow \dots \rightarrow A^{n-1}$
- Consider the k^{th} node with A^{k-1}



- The shortest path from i to j with 0, …, k nodes can be calculated with two ways
  - Pass through k: $A^{K-1}[i][k] + A^{K-1}[k][j]$
  - Otherwise: $A^{K-1}[i][j]$

02. Shortest Path

Floyd

- Algorithm

```
floyd(G)

for k ← 0 to n - 1
    for i ← 0 to  n - 1
        for j ← 0 to n - 1
            A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

02. Shortest Path

Floyd

- Example

	0	1	2	3	4	5	6
0	0	7	INF	INF	3	10	INF
1	7	0	4	10	2	6	INF
2	INF	4	0	2	INF	INF	INF
3	INF	10	2	0	11	9	4
4	3	2	INF	11	0	13	5
5	10	6	INF	9	13	0	INF
6	INF	INF	INF	4	5	INF	0

	0	1	2	3	4	5	6
0	0	7	11	17	3	10	INF
1	7	0	4	10	2	6	INF
2	11	4	0	2	6	10	INF
3	17	10	2	0	11	9	4
4	3	2	6	11	0	8	5
5	10	6	10	9	8	0	INF
6	INF	INF	INF	4	5	INF	0

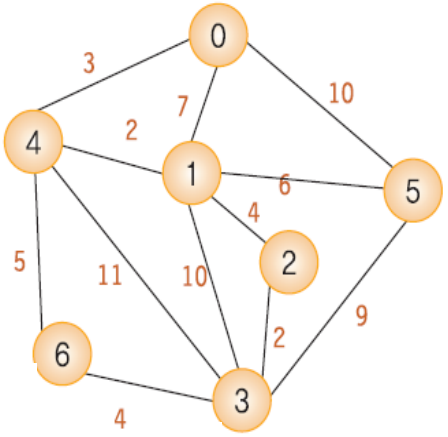
	0	1	2	3	4	5	6
0	0	7	11	13	3	10	INF
1	7	0	4	6	2	6	INF
2	11	4	0	2	6	10	INF
3	13	6	2	0	8	9	4
4	3	2	6	8	0	8	5
5	10	6	10	9	8	0	INF
6	INF	INF	INF	4	5	INF	0

	0	1	2	3	4	5	6
0	0	7	11	13	3	10	17
1	7	0	4	6	2	6	10
2	11	4	0	2	6	10	6
3	13	6	2	0	8	9	4
4	3	2	6	8	0	8	5
5	10	6	10	9	8	0	3
6	17	10	6	4	5	13	0

	0	1	2	3	4	5	6
0	0	5	9	11	3	10	8
1	5	0	4	6	2	6	7
2	9	4	0	2	6	10	6
3	11	6	2	0	8	9	4
4	3	2	6	8	0	8	5
5	10	6	10	9	8	0	13
6	8	7	6	4	5	13	0

	0	1	2	3	4	5	6
0	0	5	9	11	3	10	8
1	5	0	4	6	2	6	7
2	9	4	0	2	6	10	6
3	11	6	2	0	8	9	4
4	3	2	6	8	0	8	5
5	10	6	10	9	8	0	13
6	8	7	6	4	5	13	10

	0	1	2	3	4	5	6
0	0	5	9	11	3	10	8
1	5	0	4	6	2	6	7
2	9	4	0	2	6	10	6
3	11	6	2	0	8	9	4
4	3	2	6	8	0	8	5
5	10	6	10	9	8	0	13
6	8	7	6	4	5	13	0



02. Shortest Path

Floyd

- Complexity
  - Given n nodes in G, the complexity is $O(n^3)$

- To get all the shortest paths among all the nodes using Dijkstra,

  $n * O(n^2) = O(n^3)$
  - Floyd may have a simpler structure

What You Need to Know

Summary

- Graph
  - A data structure that represents the relations among connected objects
  - Graph terminologies: degree, path, cycle
- Minimum spanning tree
  - Kruskal
  - Prim
- Shortest path
  - Dijkstra
  - Floyd

Thanks

Week 9: Graph Algorithms

Instructor: Jinyoung Han (jinyounghan@skku.edu)