CS2040 – Data Structures and Algorithms

Revision – Graphs

Putting It All Together ☺

axgopala@comp.nus.edu.sg

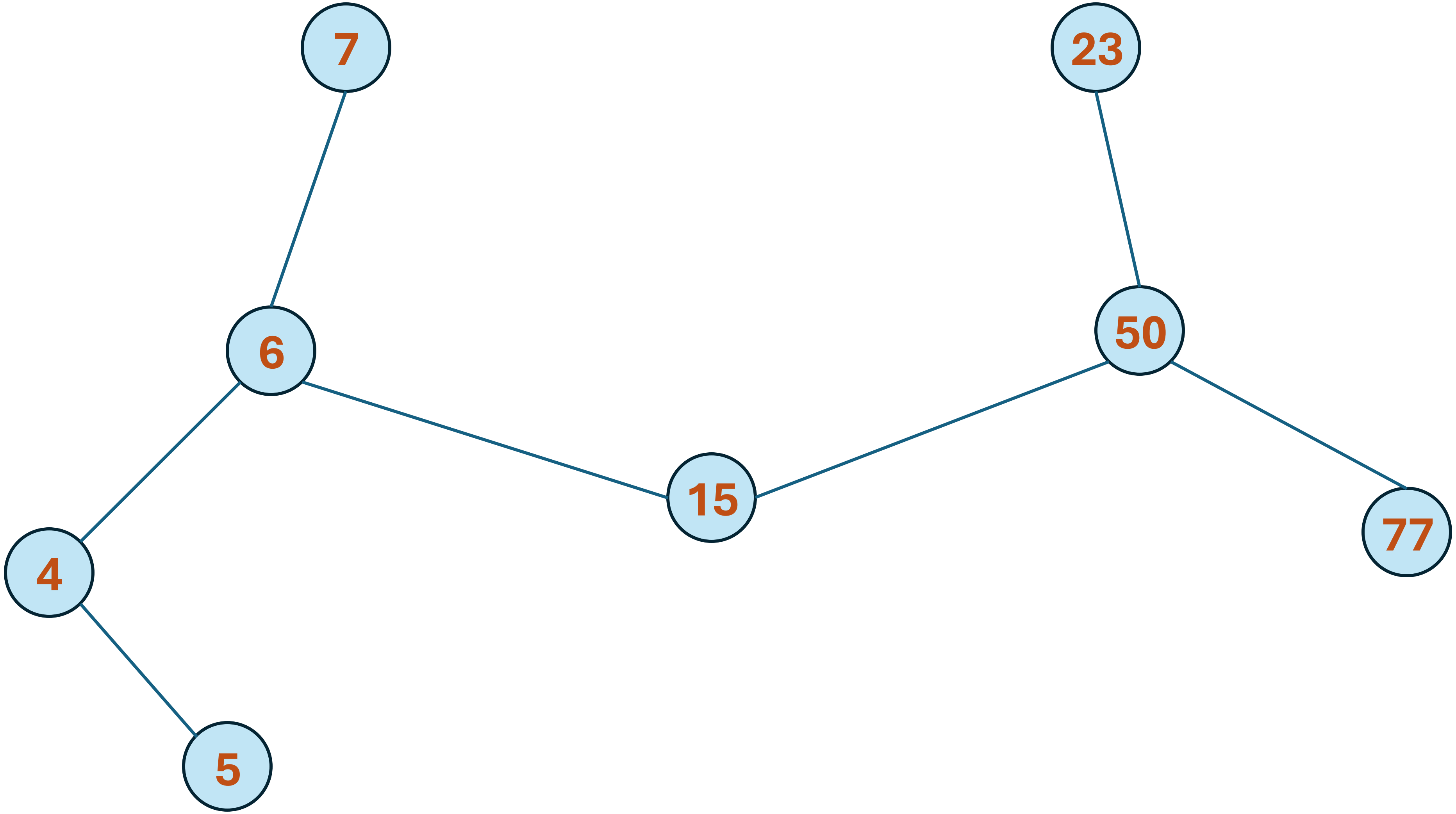
NUS National University of Singapore

School of Computing

Outline

- Basics of Graphs
- Graph Algorithms
- Bellman-Ford Algorithm
- Exam Tips ☺

Graph is ...



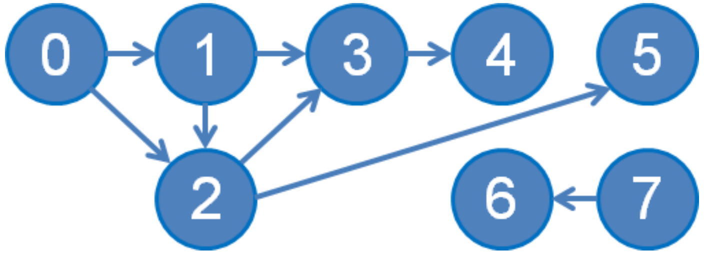
Graph is …

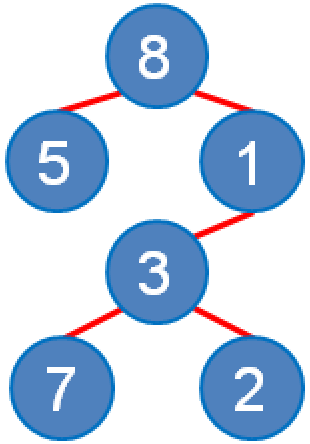
- Set of vertices V

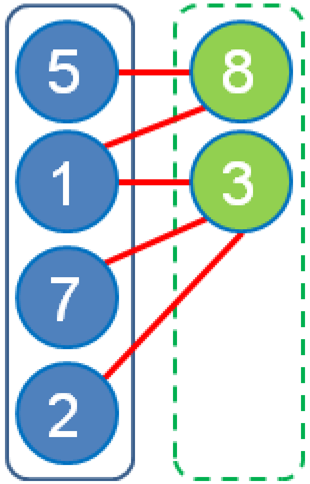
- Set of edges E
  - Edges may be undirected, directed, or bi-directed
  - Edges may have weights, no weights, or all have the same weight

Graph Terminologies

- Directed Acyclic Graph (DAG)
    - **Directed** graph that has no cycle
- Tree (bottom left)
    - Connected graph – one unique path between any pair of vertices
- Bipartite Graph (bottom right)
    - **Undirected** graph where we can partition the vertices into two sets so that there are no edges between members of the same set







Graph Data Structures

- Adjacency Matrix
  - 2D array (AdjMatrix)
  - AdjMatrix[i][j] = 1 or weight of edge (in weighted graph), if there exist an edge i ➔ j in G, otherwise 0
- Adjacency List
  - Array of V lists (AdjList) – one element for each vertex
  - For each vertex i, AdjList[i] = list of i’s neighbours
  - For weighted graph, stores pair (neighbour, weight), 1 if connected for unweighted graph
- Edge List
  - Array of E edges (EdgeList) – one element for each edge
  - For each edge i, EdgeList[i] = integer triple {u, v, w(u, v)}
  - For unweighted graph, the weight can be stored as 0 (or 1), or simply store an (integer) pair

Space Complexity

Adjacency Matrix	**O (V^2)**
Adjacency List	**O (V + E)**
Edge List	**O (E)**

GRAPH TRAVERSAL

Breadth First Search (BFS)

```
for all v in V
  visited[v] ← 0
  p[v] ← -1
Q ← {s} // start from s
visited[s] ← 1
```

```
while Q is not empty
  u ← Q.dequeue()
  for all v adjacent to u // order of neighbour
    if visited[v] = 0      // influences BFS
      visited[v] ← 1       // visitation sequence
      p[v] ← u
      Q.enqueue(v)
```

Depth First Search (DFS) – Recursive Version

```
DFSrec(u)
  visited[u] ← 1 // to avoid cycle
  for all v adjacent to u // order of neighbour
    if visited[v] = 0 //  influences DFS
     p[v] ←    u // visitation sequence
     DFSrec(v) // recursive (implicit stack)
```

```
for all v in V
  visited[v] ← 0
  p[v] ←  -1
DFSrec(s) // start the recursive call from s
```

Some Applications of BFS and DFS ☺

1. Reachability Test (BFS/DFS)
2. Find Shortest Path (BFS with **O (V + E)** for **unweighted** graphs)
3. Identifying/Counting Component(s) (DFSrec in **O (V + E)**)
4. Topological Sort (Modified BFS/DFSrec ‘post-order’ in **O (V + E)**)
5. Identifying/Counting Strongly Connected Component(s) (DFSrec ‘post-order’ in **O (V + E)**)

MINIMUM SPANNING TREE

Definition

- **Minimum Spanning Tree (MST)** of connected undirected weighted graph **G**
  - **MST** of **G** is a **ST** of **G** with the minimum possible **w(ST)**

Algorithms

- Jarnik's/Prim's greedy algorithm
  - Uses PriorityQueue Data Structure
  - O(**E** log **V**)

- Kruskal's greedy algorithm
  - Uses Union-Find Data Structure
  - O(**E** log **V**)

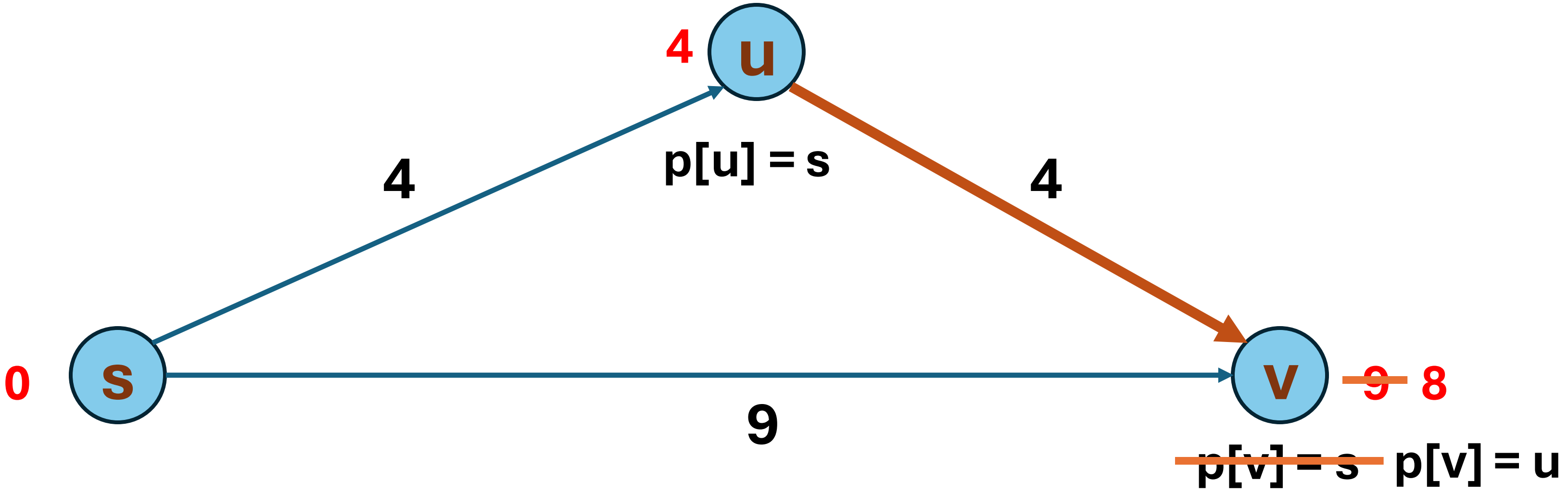
- Both use the **cut property** of graphs

SINGLE SOURCE SHORTEST PATHS (SSSP)

Find $\delta(s, b)$ from source vertex **s** to each vertex **b** (in V) together with the corresponding shortest path

‘Relaxation’ Operation

```
relax(u, v, w(u,v))
  if D[v] > D[u] + w(u,v) // if SP can be shortened
    D[v] ← D[u] + w(u,v) // relax this edge
    p[v] ←u  // remember/update the predecessor
    // if necessary, update some data structure
```



Bellman-Ford's Algorithm

```
initSSSP(s)

// Simple Bellman-Ford's algorithm runs in O(VE)
for i = 1 to |V|-1 // O(V) here
  for each edge(u, v) ∈  E // O(E) here
    relax(u, v, w(u,v)) // O(1) here

// At the end of Bellman-Ford's algorithm,
// D[v] = δ(s, v) if no negative weight cycle exist

// Q: Why “relaxing all edges V-1 times” works?
```

SSSP – Algorithms

- General case: weighted graph
  - Use O(**VE**) Bellman Ford's algorithm
- Special case 1: Tree
  - Use O(**V**) BFS or DFS ☺
- Special case 2: unweighted graph
  - Use O(**V**+**E**) BFS ☺
- Special case 3: DAG
  - Use O(**V**+**E**) DFS to get the topological sort, then relax the vertices using this topological order
- Special case 4ab: graph has no negative weight/negative cycle
  - Use O((**V**+**E**) log **V**) original/O(**E** log **E**) modified Dijkstra's, respectively

All-Pairs Shortest Paths (APSP)

*Find the shortest paths between **<u>any pair</u>** of vertices in the given directed weighted graph*

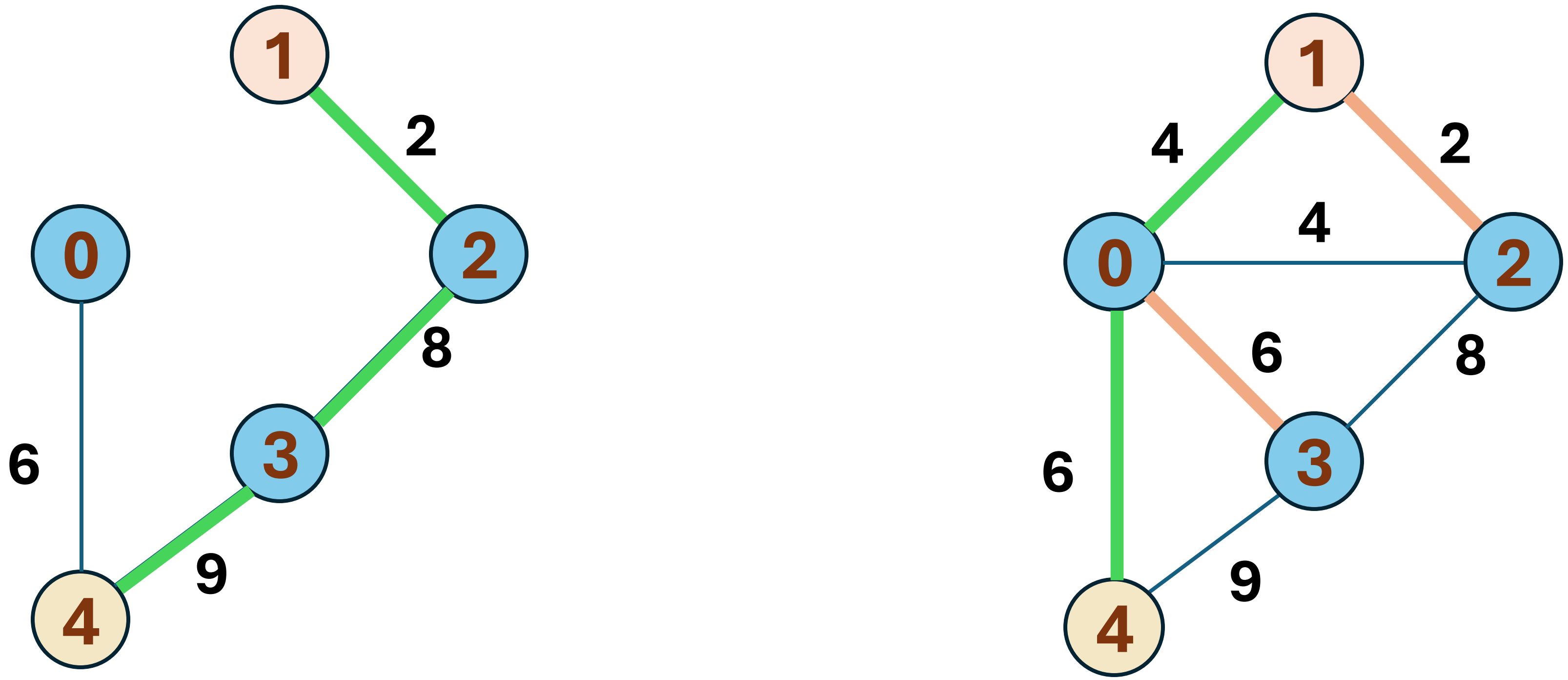
APSP Solutions with SSSP Algorithms

- On unweighted graph
  - Call BFS **V** times, once from each vertex
    - Time complexity: O(**V** * (**V**+**E**)) = **O(V^3)** if **E** = O(V^2)

- On weighted graph, for simplicity, non (-ve) weighted graph
  - Call Bellman Ford's **V** times, once from each vertex
    - Time complexity: O(**V** * **VE**) = **O(V^4)** if **E** = O(V^2)
  - Call Original/Modified Dijkstra's **V** times, once from each vertex
    - Time complexity: O(**V** * (**V**+**E**) * log **V**)/O(**V** * **E** * log **V**) = **O(V^3 log V)** if **E** = O(V^2)
  - Floyd-Warshall's Algorithm
    - Time complexity: O(V^3)

Minimax Problem

- Finding the path that **minimizes** the **maximum** edge from vertex **i** to vertex **j**



Exam Tips ☺

- Work through **past papers** and time yourself
- Group study and support each other
- Ask on Piazza/Email/drop by the office/etc.
- Start exam by reading all questions
- Start by answering the ones you know best!

Continuous Feedback

https://forms.office.com/r/KsNwmTUD0q