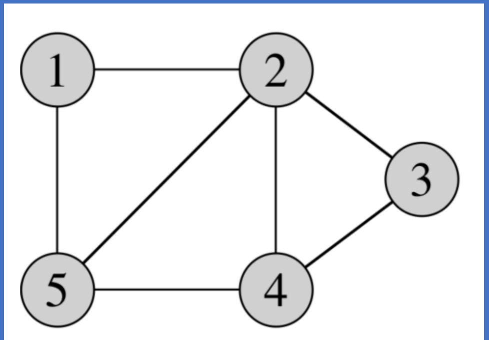
CS141: Intermediate Data Structures and Algorithms



Graph Representation and Review of Graph Algorithms

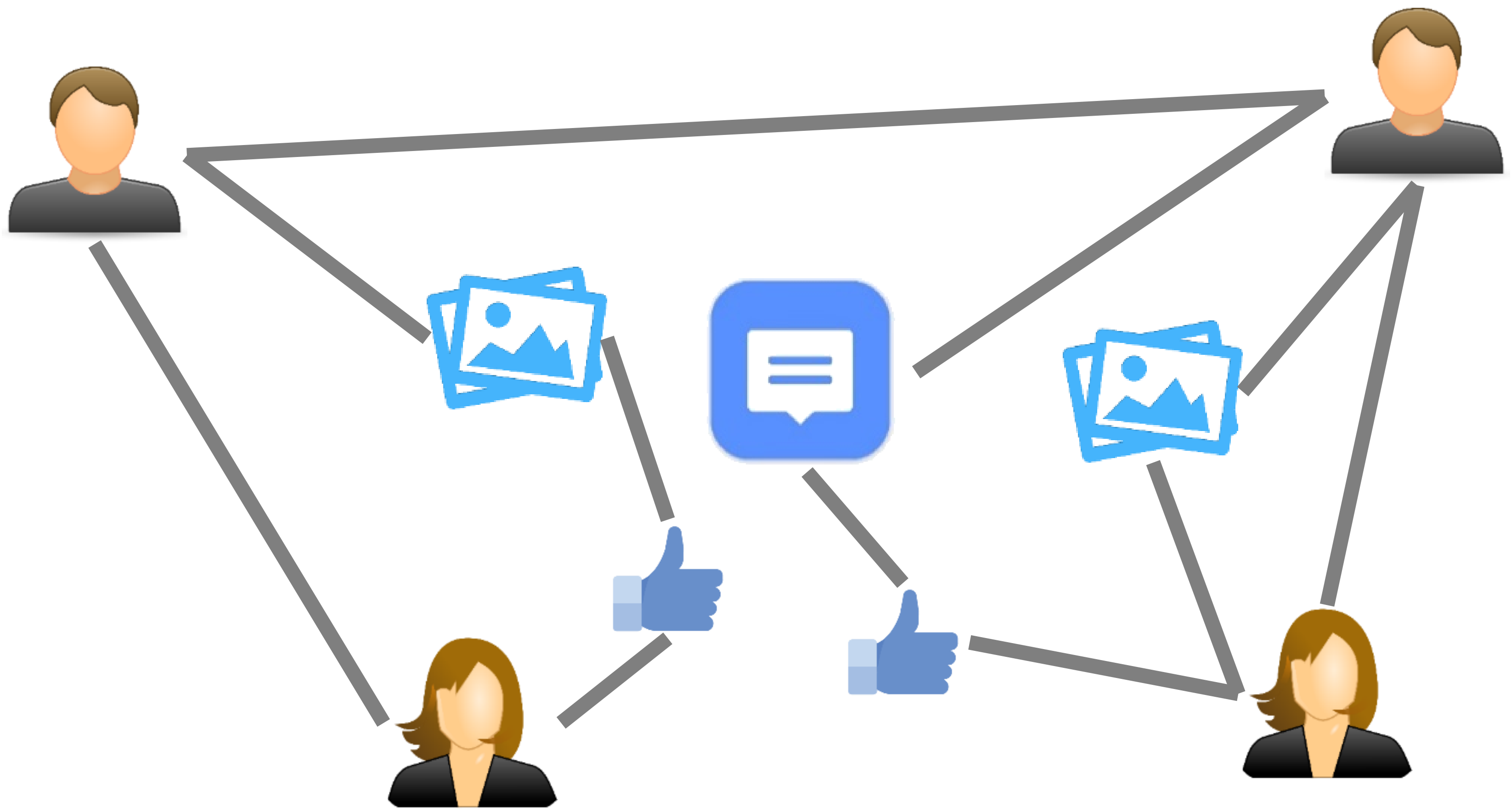
Yihan Sun

This lecture covers Section 22.1-22.3 of CLRS

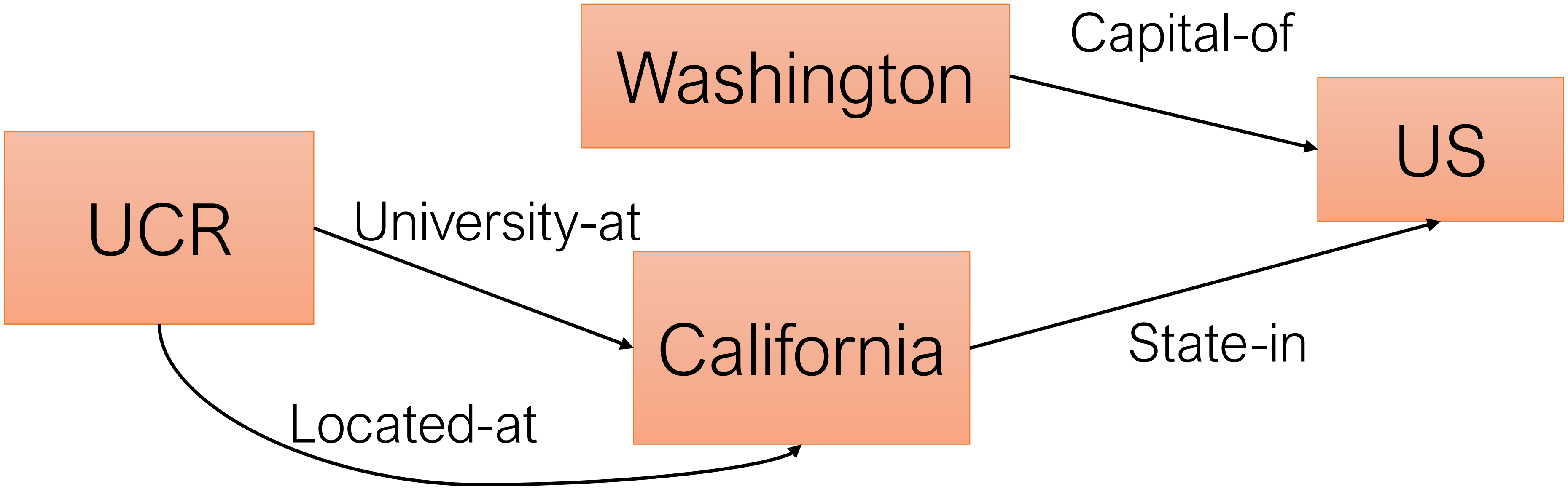
Graphs

- A good abstraction for a wide range of applications
- Consists mainly of <u>V</u>ertices (nodes) and <u>E</u>dges (arcs)
- Vertices and/or Edges can be annotated with further information

Social Network

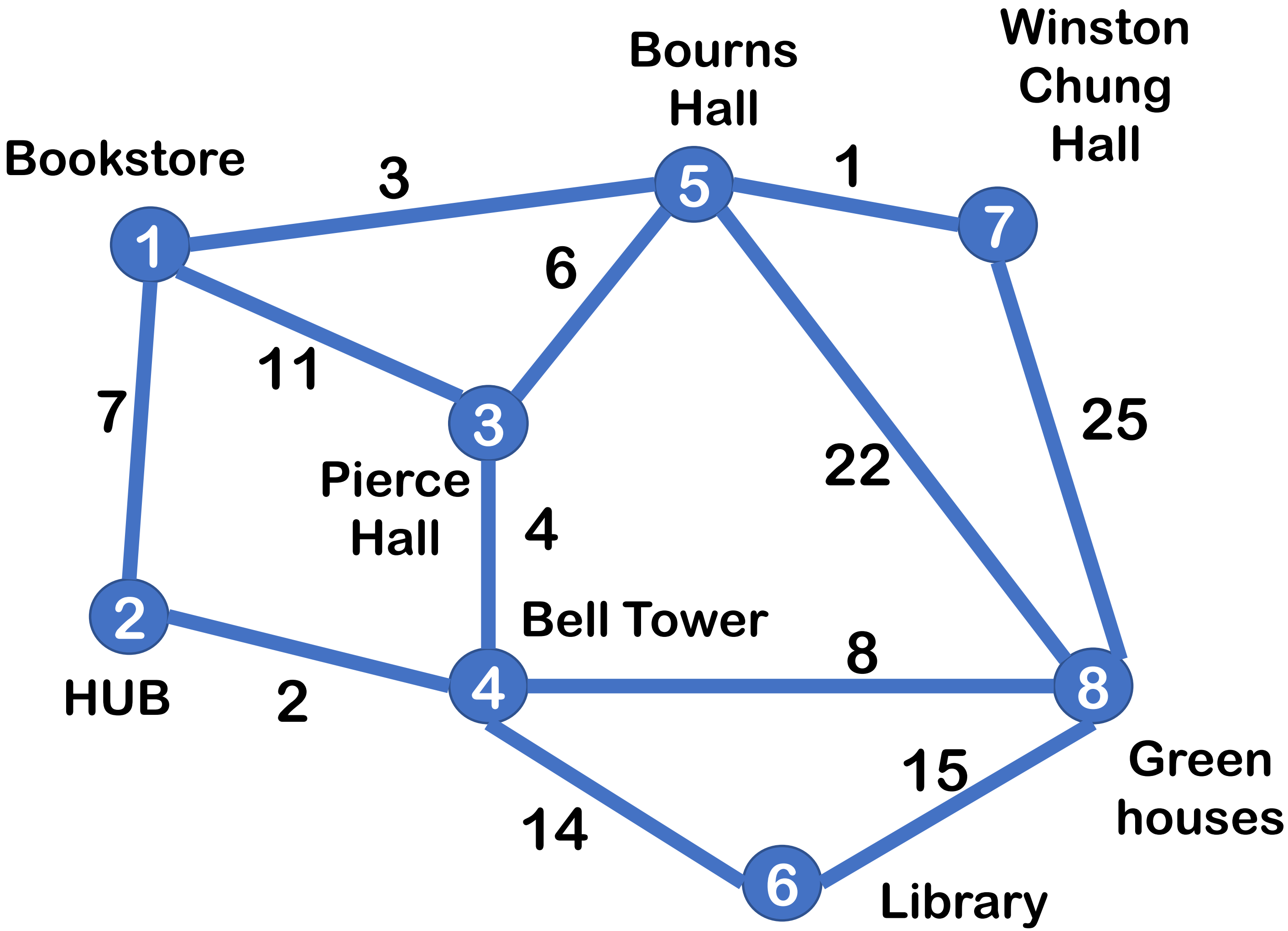


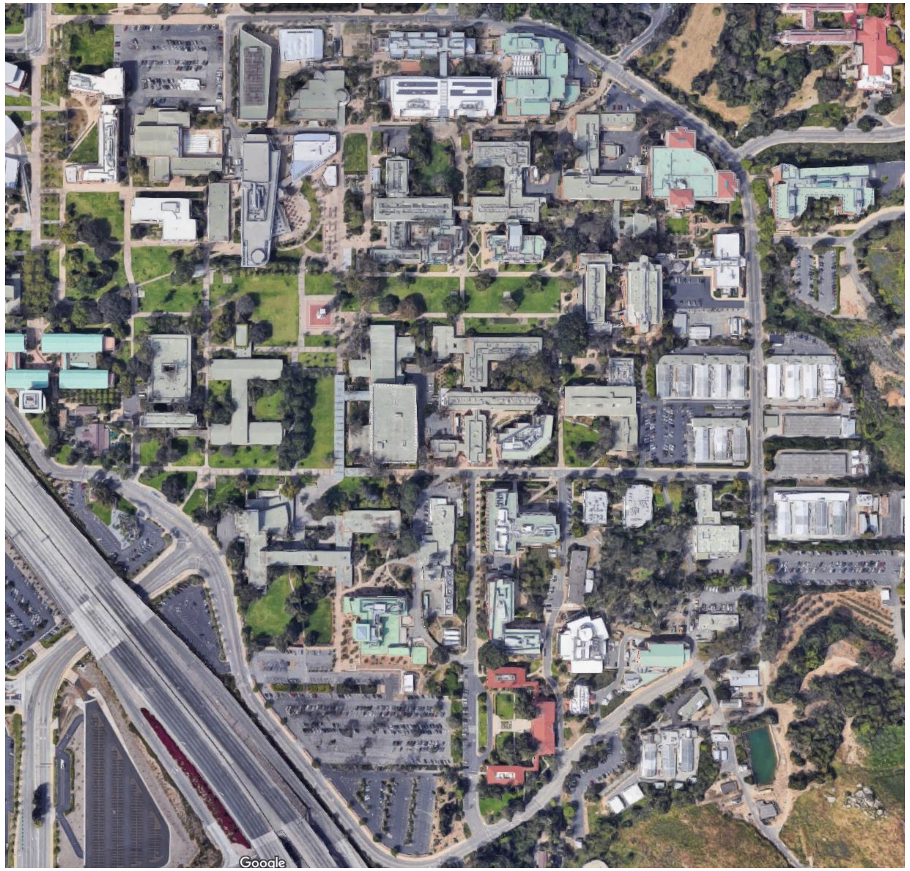
Knowledge



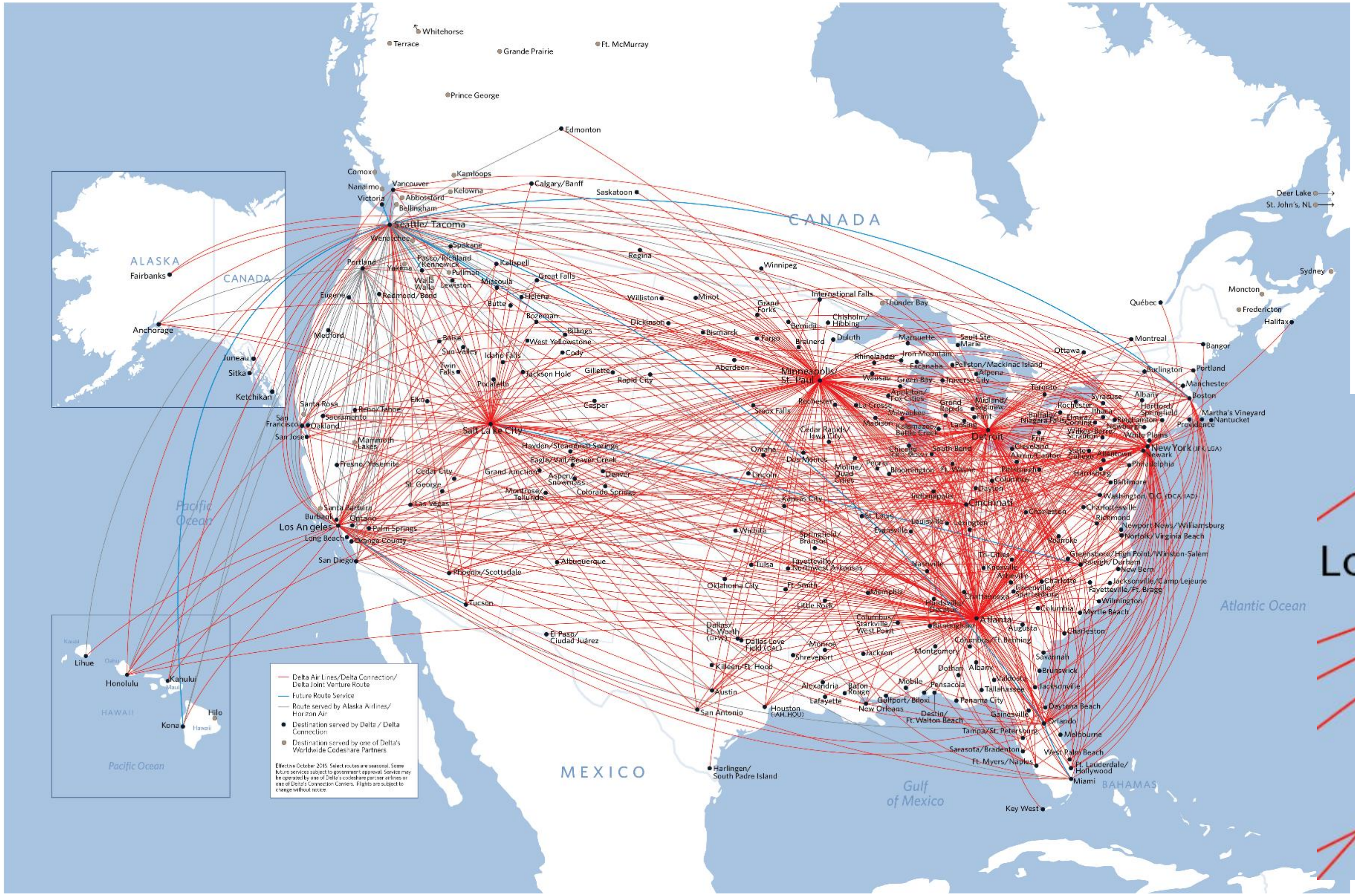
Washington
Capital-of
US
UCR
University-at
California
State-in
Located-at

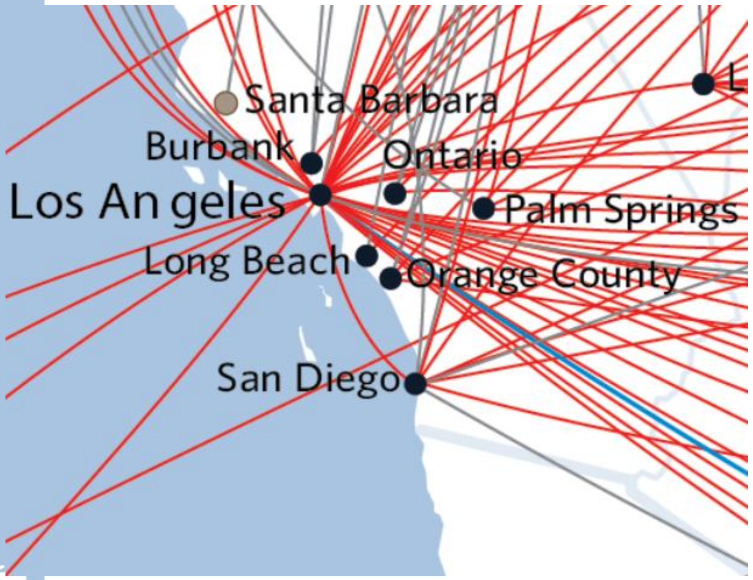
Map


Bookstore
Bourns Hall
Winston Chung Hall
Pierce Hall
Bell Tower
HUB
Library
Green houses
1
2
3
4
5
6
7
8
3
1
6
11
7
25
22
4
8
2
15
14

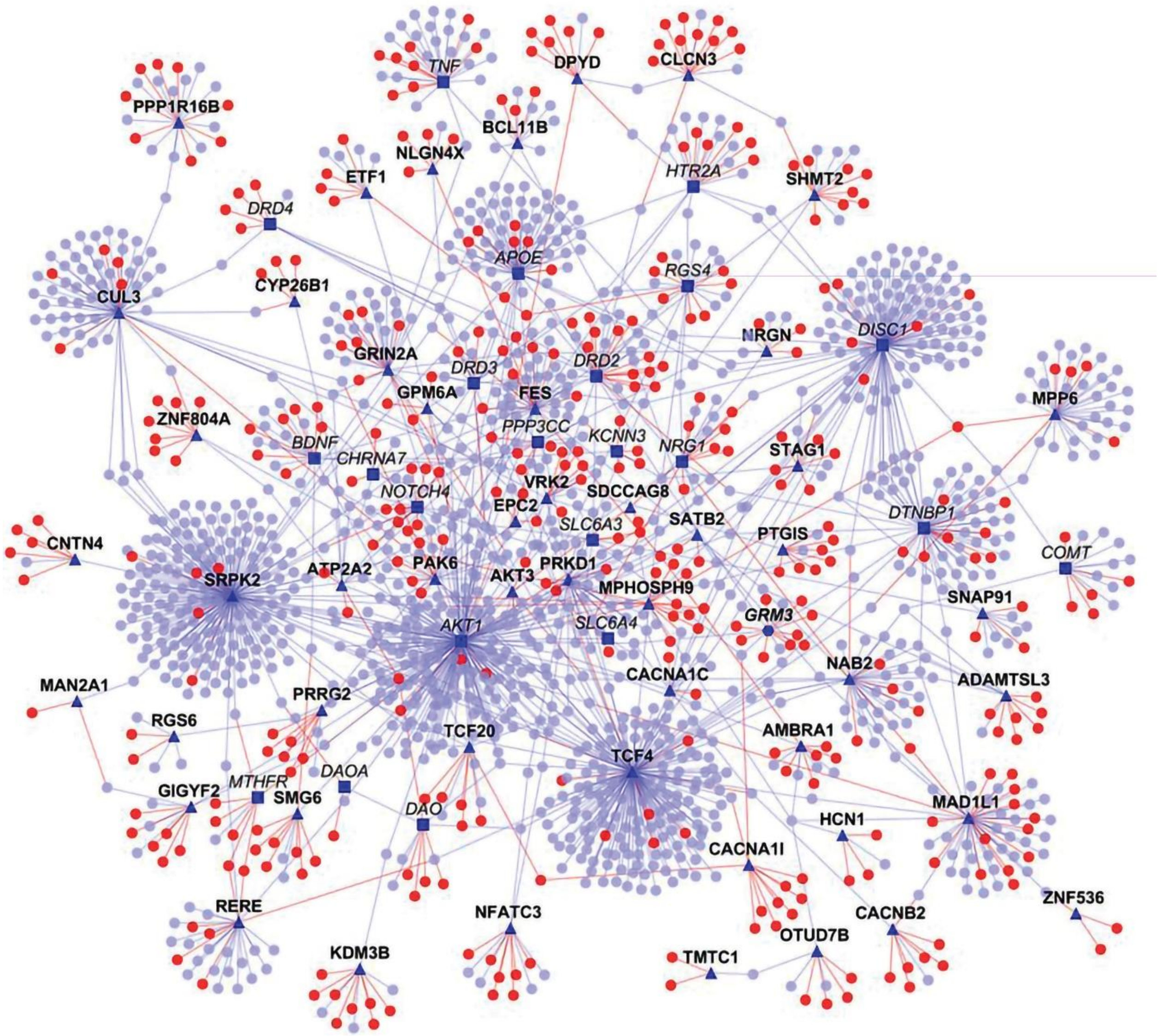


Delta’s route map for USA and Canada

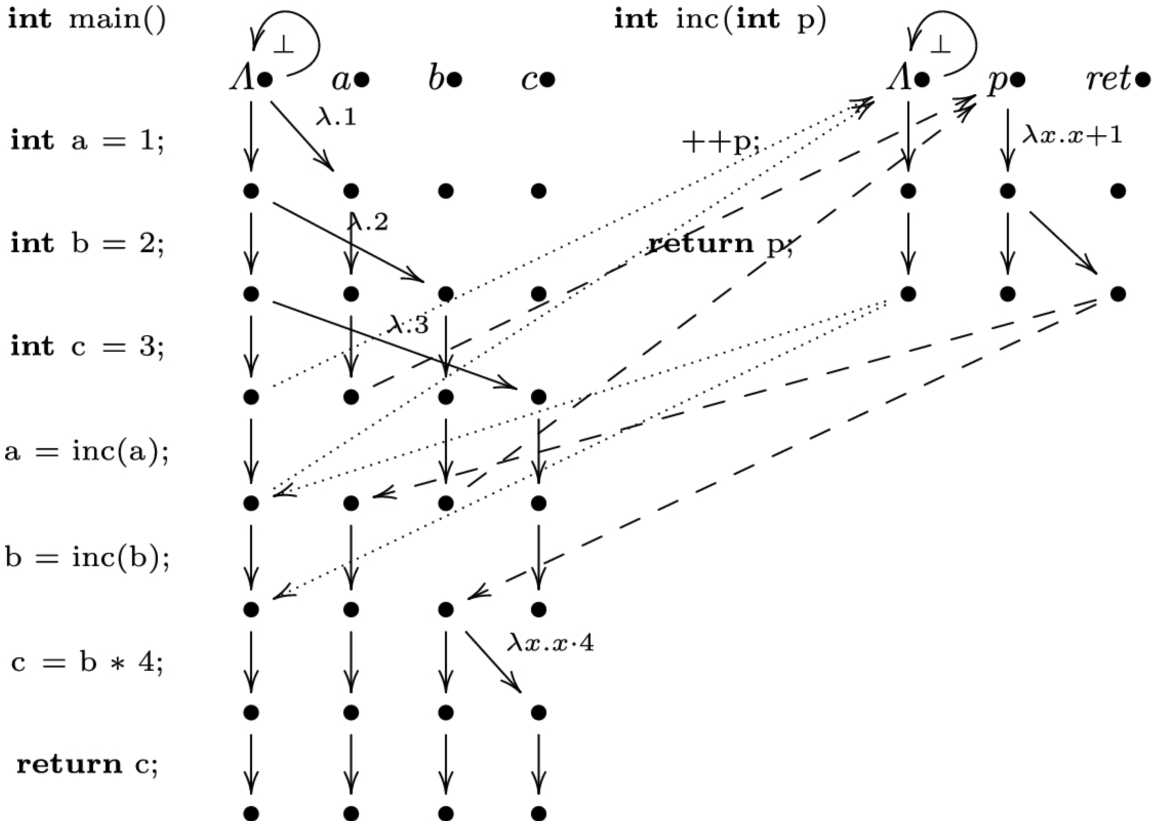




Graph of protein–protein interactions (PPI)



The graph for code analysis and security



Review graph knowledge in CS 14

- **Types of graphs**
- **Representations of graphs**
  - Adjacency list
  - Adjacency matrix
- **Elementary graph algorithms**
  - Bread-first Search (BFS)
  - Depth-first Search (DFS)
  - Connectivity
  - Cycle Detection

What is a graph G

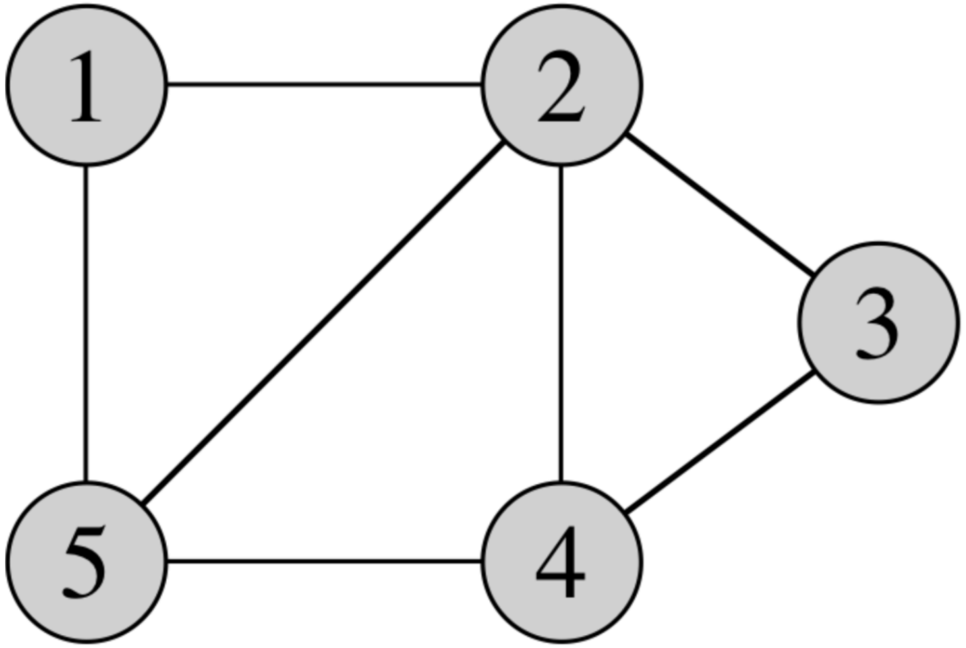
- A Graph $G = (V, E)$, where V is a vertex set, and E is the edge set
- Usually we say $n = |V|$, the number of vertices; $m = |E|$, the number of edges
- $V = \{v_1, v_2, \ldots, v_n\}$
- $E = \{e_1, e_2, \ldots, e_m\}$

Types of Graphs

- Directed and Undirected graphs
- Weighted and Unweighted graphs
- Connected graphs
- Bipartite graphs
- Acyclic graphs

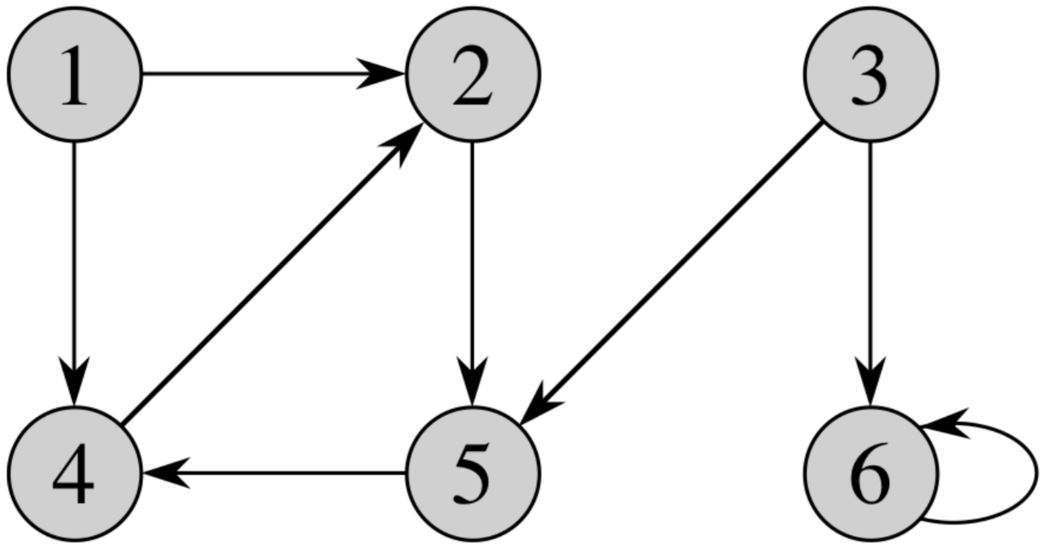
Undirected Graph

- No direction in edges
- An edge can be traversed in both ways
- E.g., Facebook friends, most roads, most flights



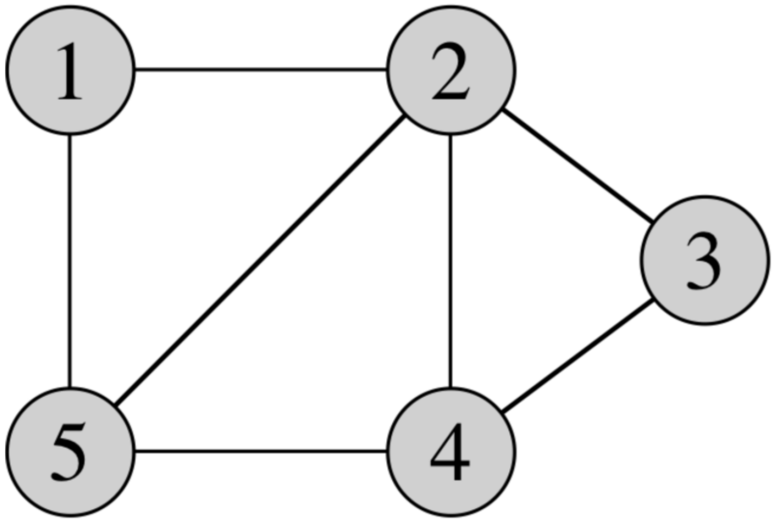
Directed Graph

- Direction on edges
- An edge can be traversed in one direction
- E.g., Twitter follows, code analysis

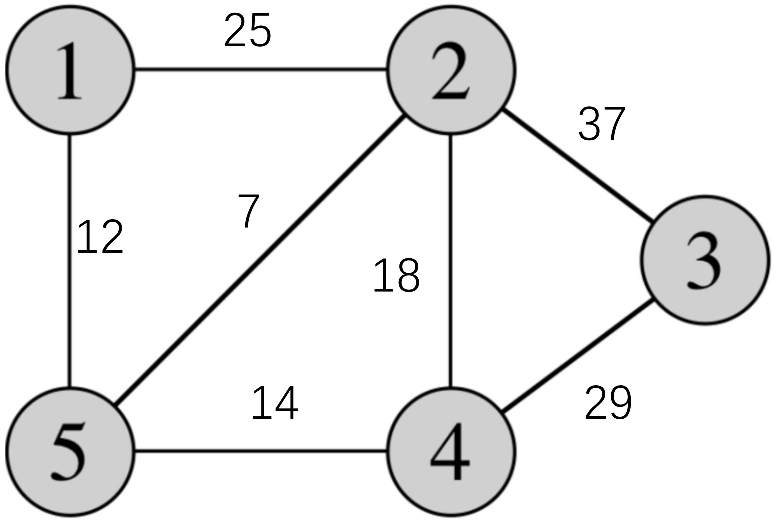


Weighted Graph

- Vertices and/or edges can be assigned weights
- Weights can be cost, capacity, etc.
- E.g., road network



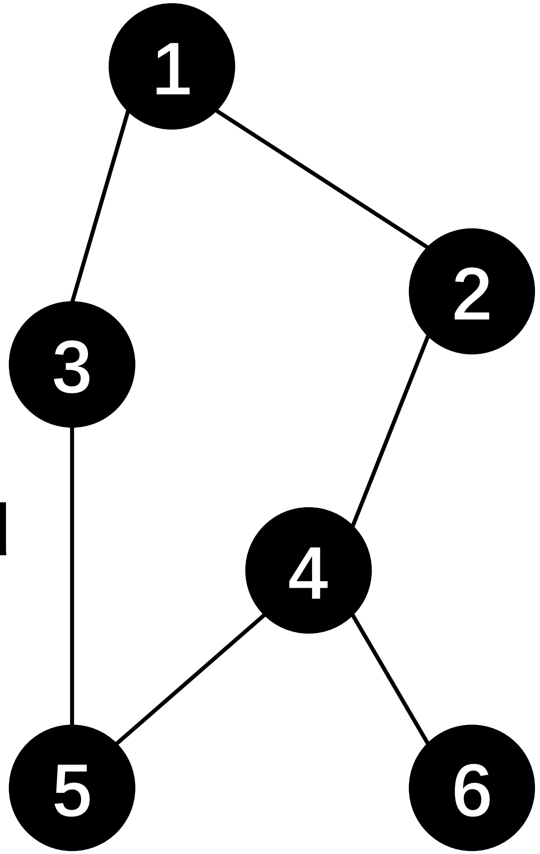
Unweighted Graph



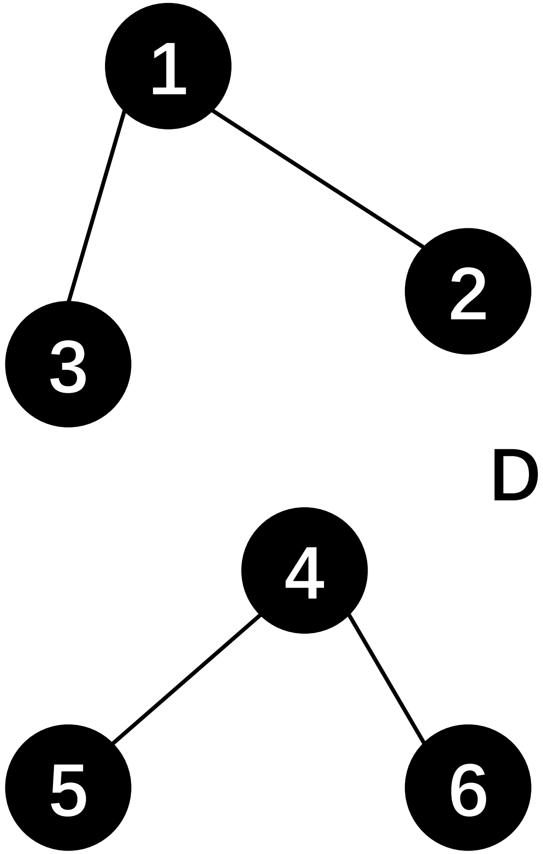
Weighted Graph

Connected Graphs

- For simplicity, most graph algorithms assume the graph is connected. Otherwise, we can run connectivity first, and work on each component.



Connected Graph

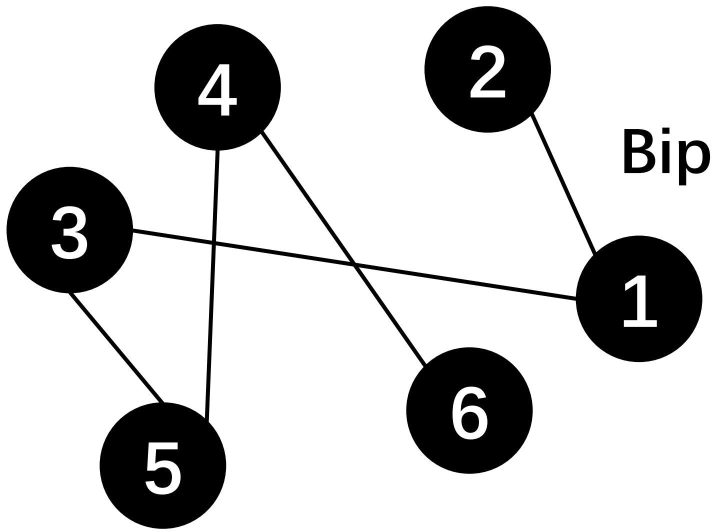


Disconnected Graph

Bipartite Graph

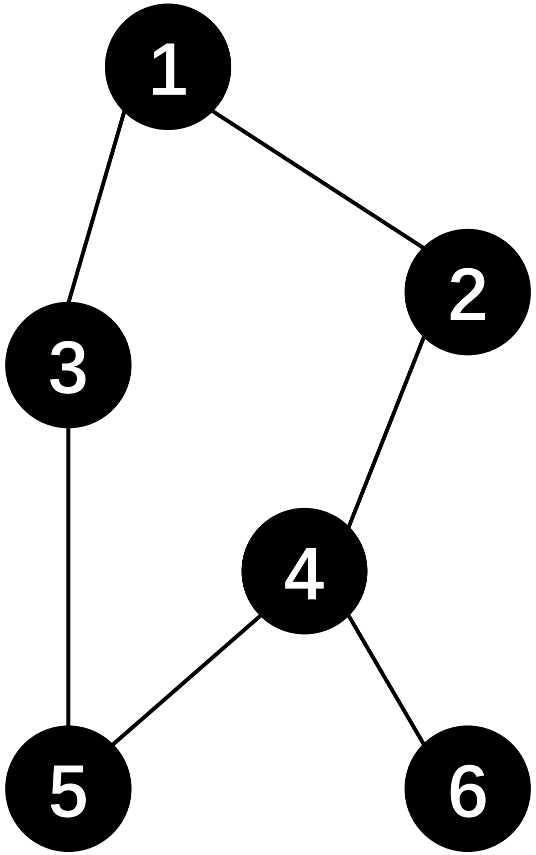
- **A graph where the vertices can be partitioned into two subsets: no edges within a subset and all the edges are between two subsets**

- **Usually, vertices in two subsets have different meanings**
  - E.g., students and courses, courses and classrooms, jobs and applicants

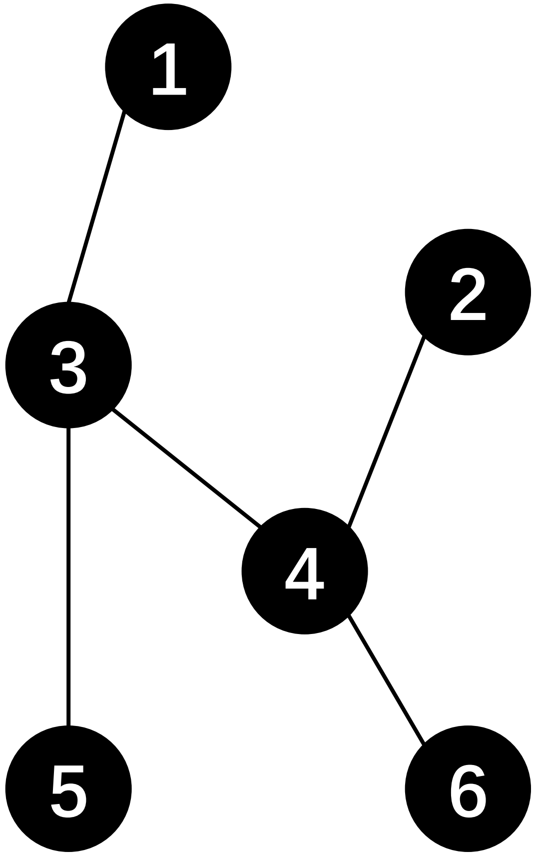


Bipartite Graph

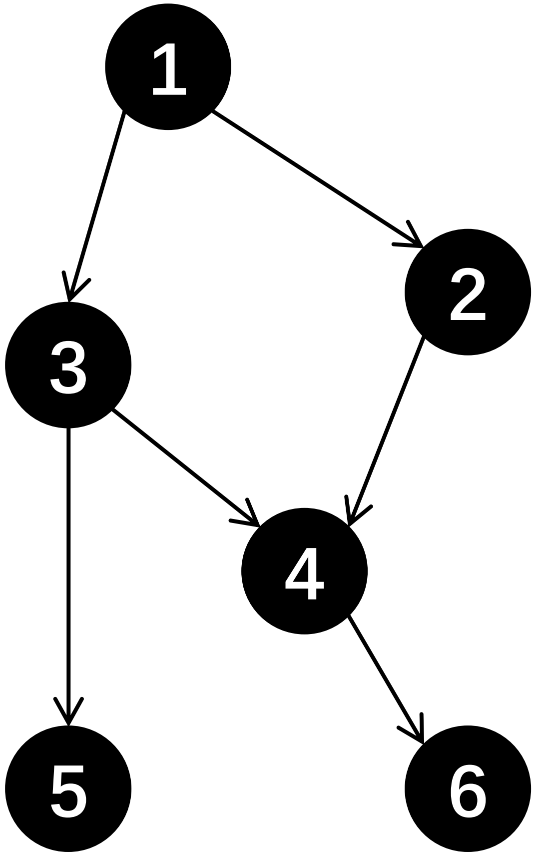
Cyclic Graph



Cyclic Graph



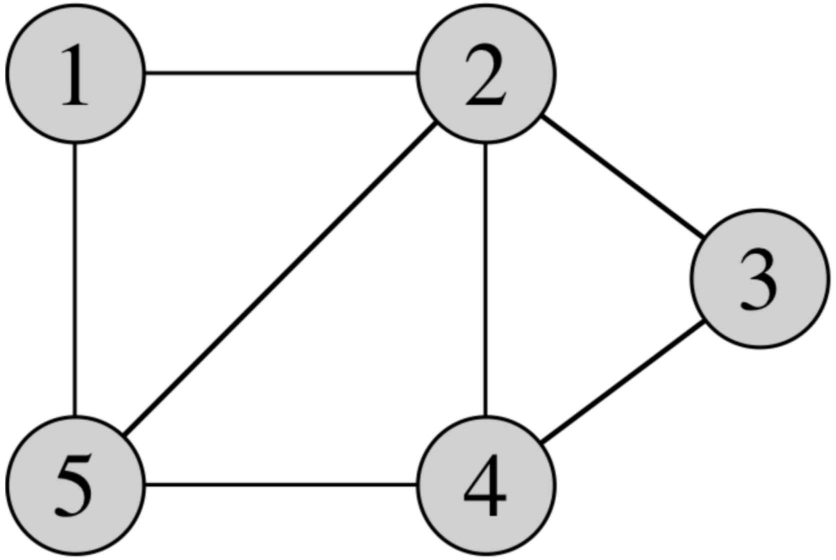
Acyclic Graph



Directed Acyclic Graph (DAG)

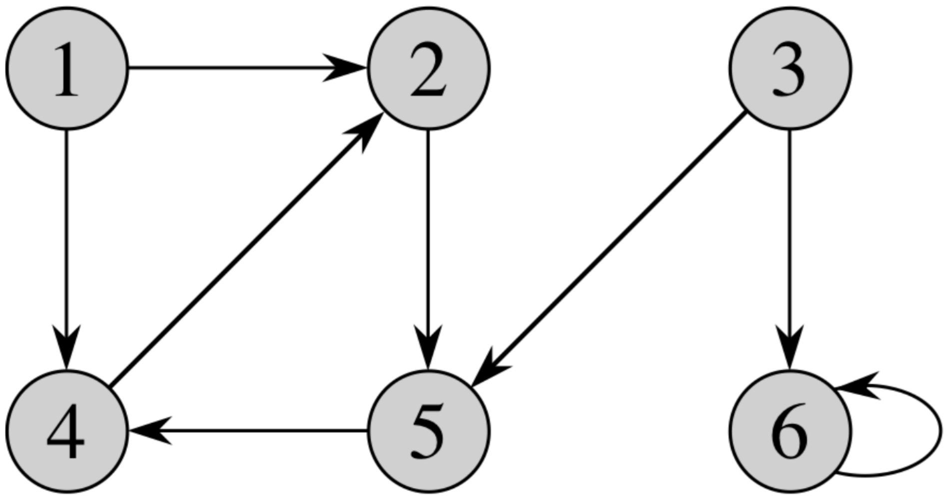
Graph Representations

Adjacency Matrix



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

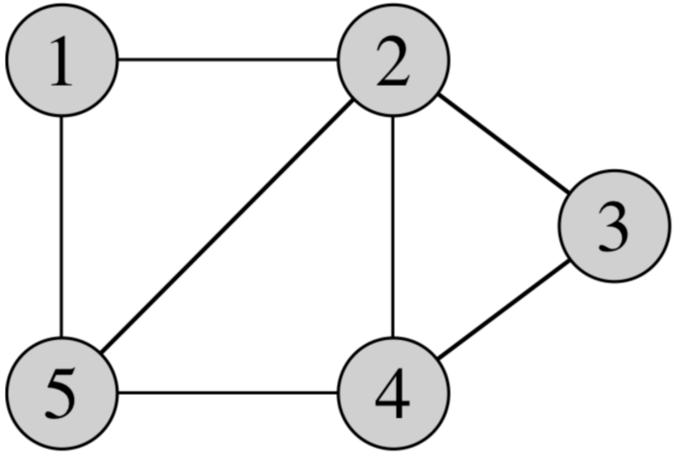
Adjacency Matrix

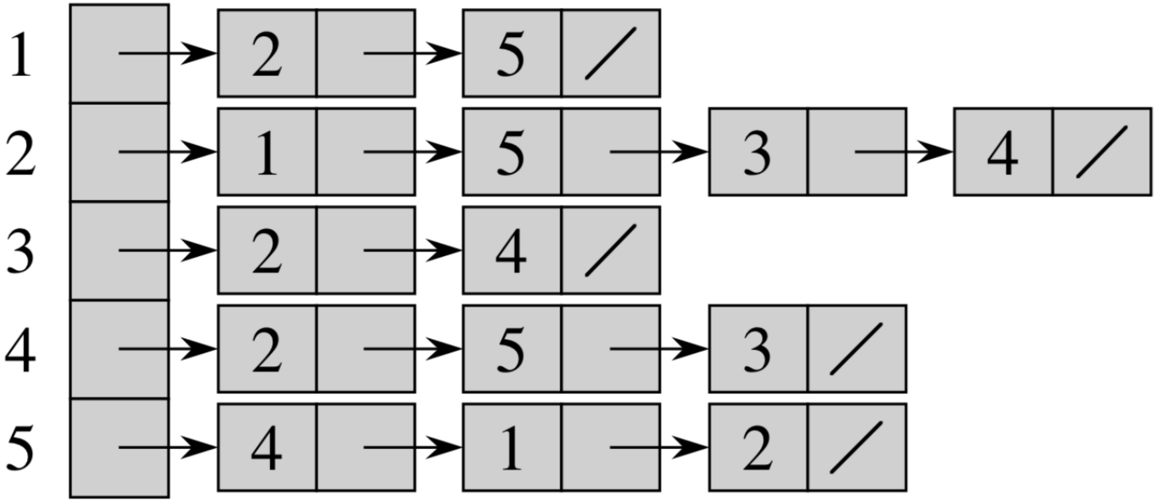


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

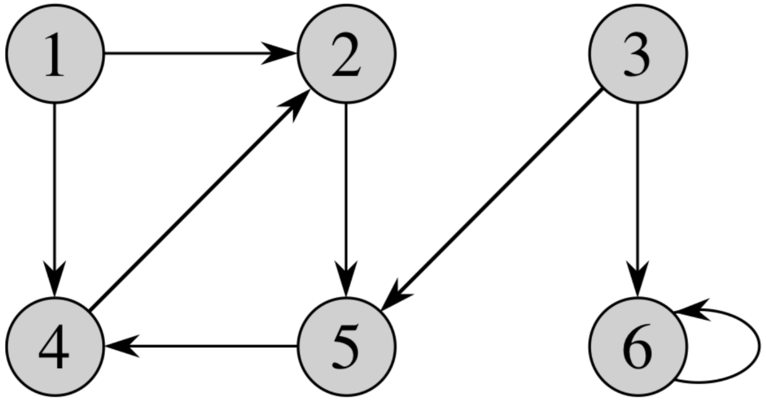
- **Problem: takes too much space $O(n^2)$**

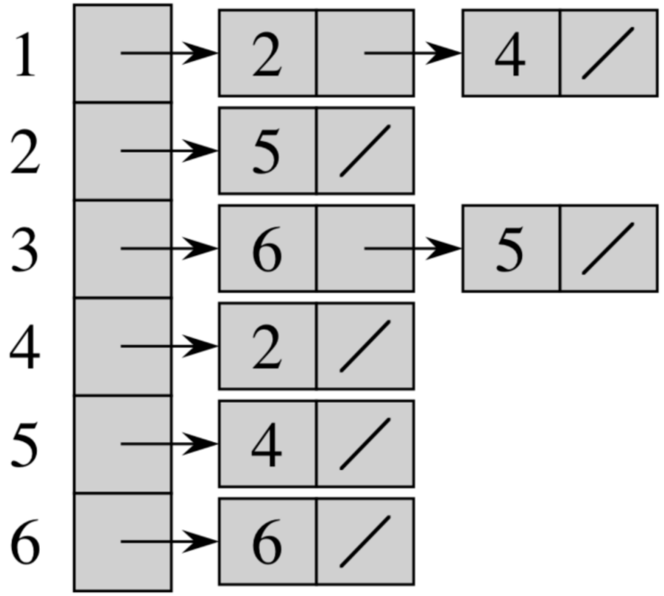
Adjacency List





Adjacency List



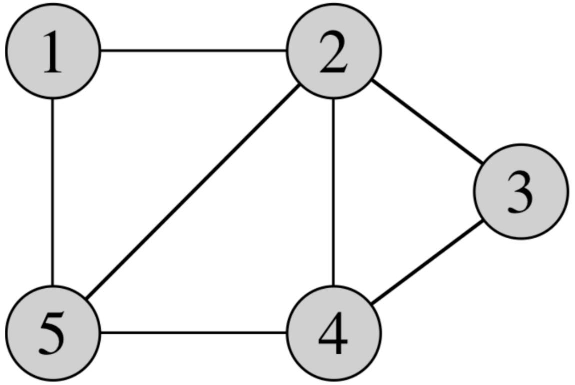


- **What do scientists use in practice?**

Compressed sparse row (CSR)

- Two arrays: Offsets and Edges
- Offsets[i] stores the offset of where vertex i's edges start in Edges

- Total space: $O(n+m)$



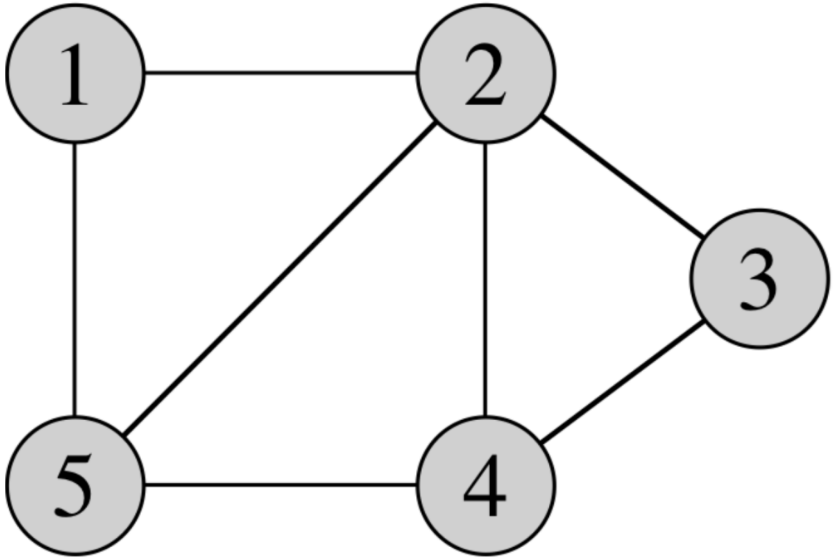
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Vertex IDs	1	2	3	4	
Offsets	1	3	7	9	…

Edges	2	5	1	3	4	5	2	4	2	…
Actual Edges	1,2	1,5	2,1	2,3	2,4	2,5	3,2	3,4	4,2	…

Edge list (for certain algorithms)

```
(1,2)
(1,5)
(2,3)
(2,4)
(2,5)
(3,4)
(4,5)
```



Summary for graph representation

n = # of vertices
m = # of edges

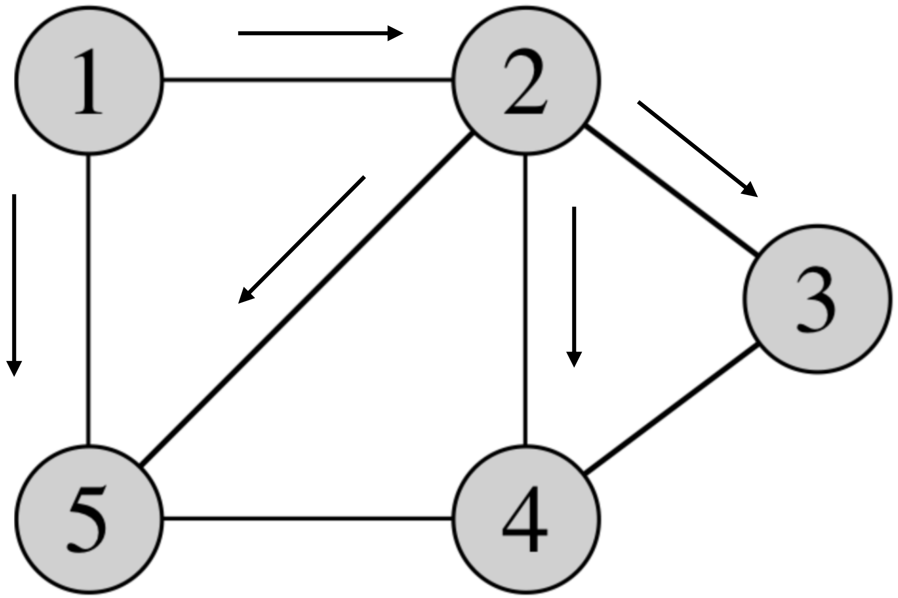
- What is the cost of different operations?

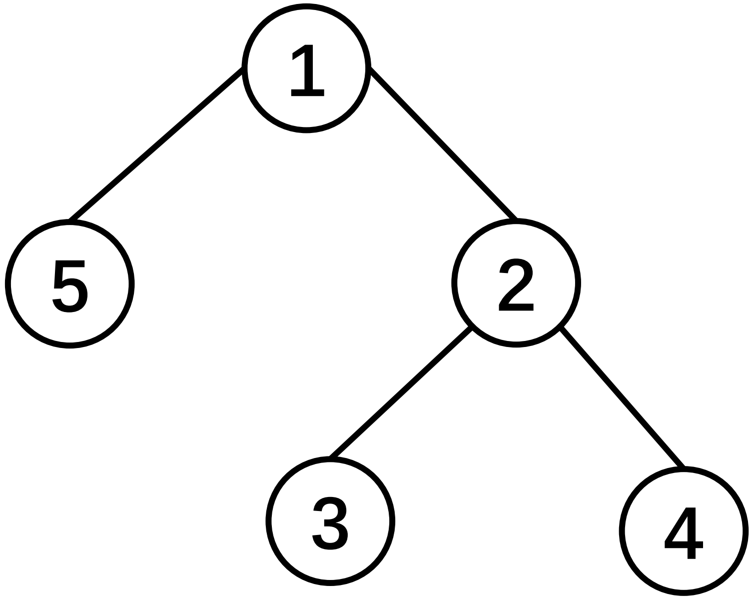
	0	1	2	3	4
0	0	1	0	0	0
1	1	0	0	1	1
2	0	0	0	1	0
3	0	1	1	0	0
4	0	1	0	0	0

	Adjacency matrix
Storage cost / scanning whole graph	$O(n^2)$
Add edge	O(1)
Delete edge from vertex v	O(1)
Finding all neighbors of a vertex v	O(n)
Finding if w is a neighbor of v	O(1)

Graph Traversals

Breadth-first Search (BFS)

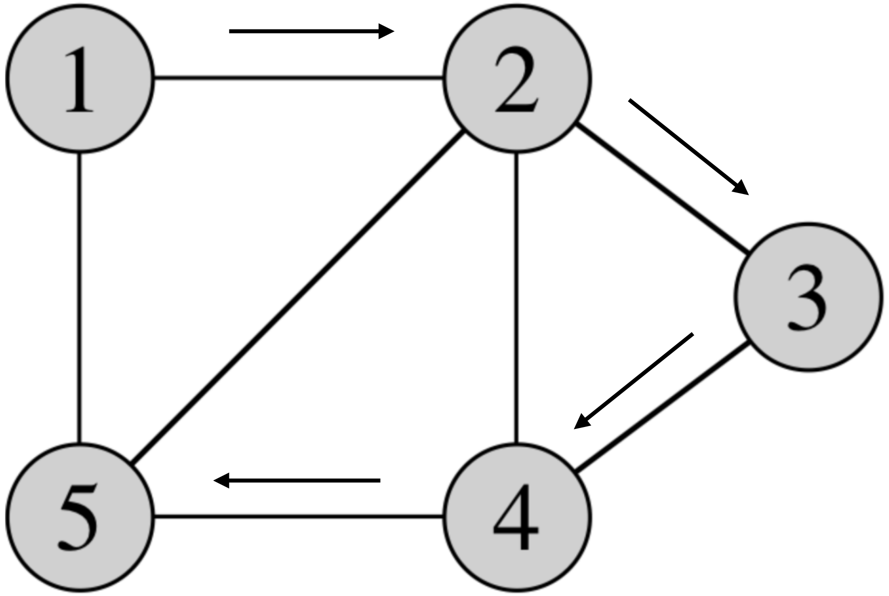


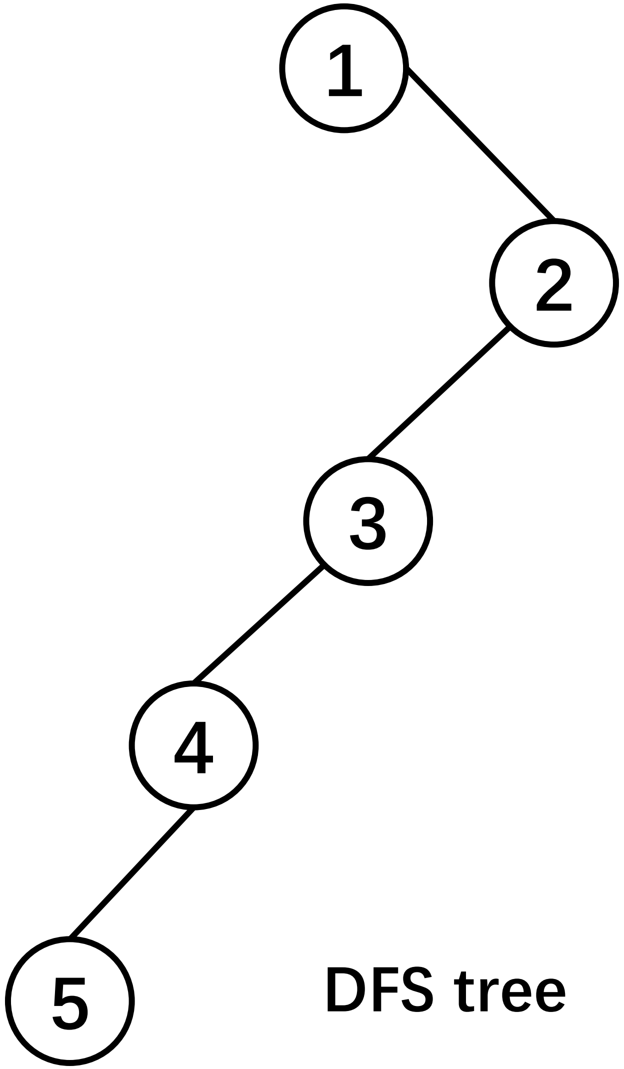


BFS tree

- Start from 1
- Visit 2, 5
- Visit 3, 4

Depth-first Search (DFS)

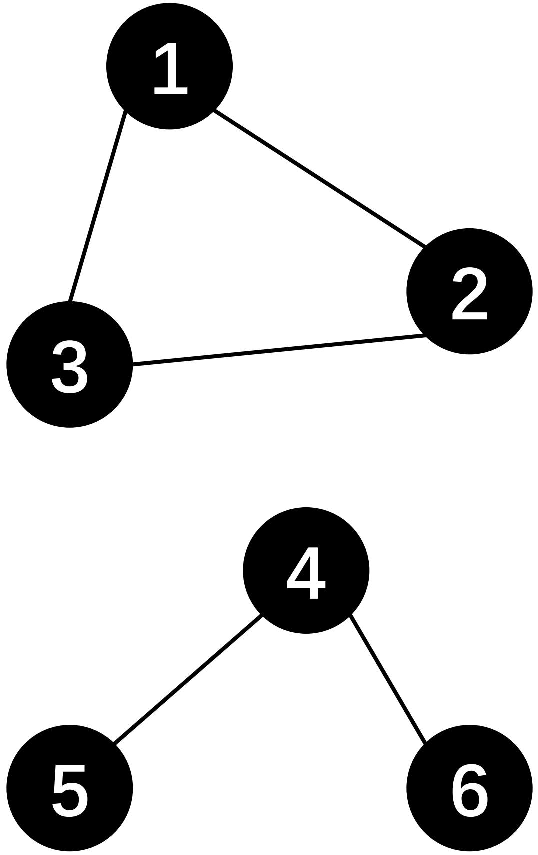




Algorithms based on Graph Traversals

Graph connectivity

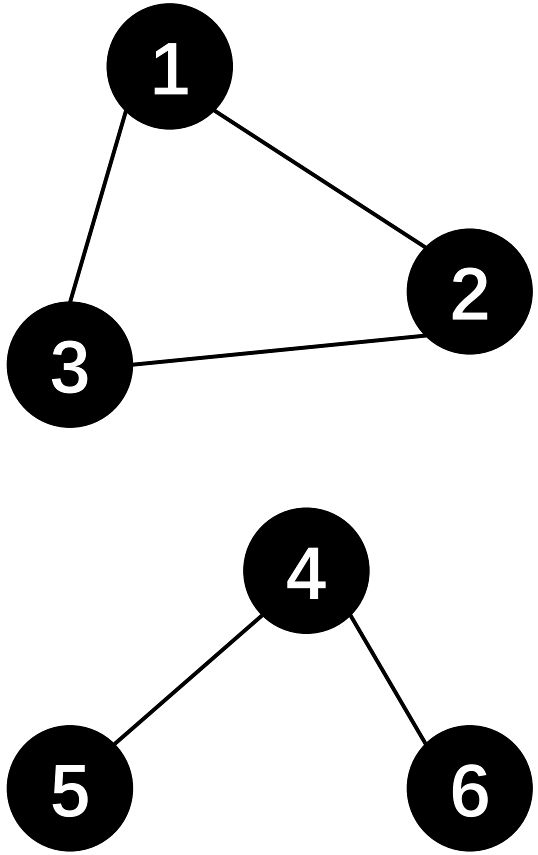
- For each vertex
  - If it is not visited
    - Run BFS/DFS on it, mark all visited nodes in the same connected components



Floodfill (based on DFS)

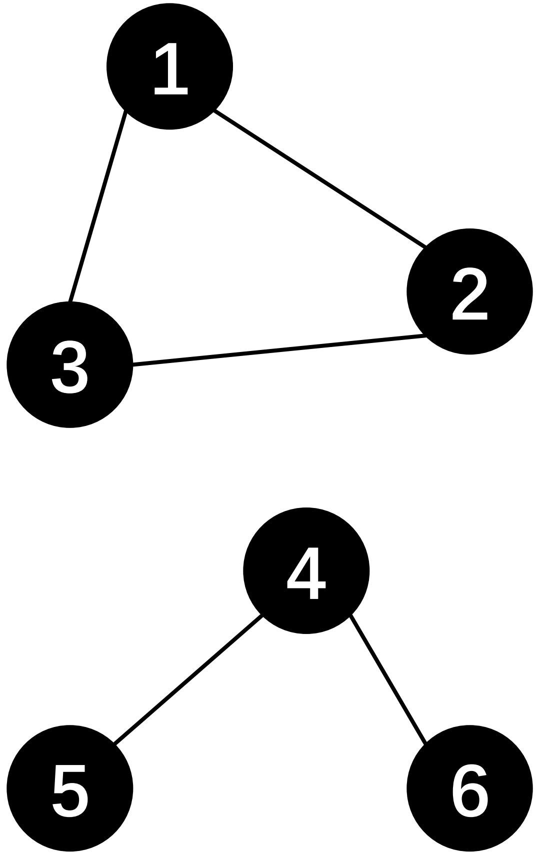
```
ff(vertex u)
    if visited[u] return;
    visited[u] = id;
    for (u,v) in E
      ff(v);

visited[:] = false; // 0
for u in V
    if !visited[u] { id++; ff(u); }
```

- **Time bound:** $O(n+m)$



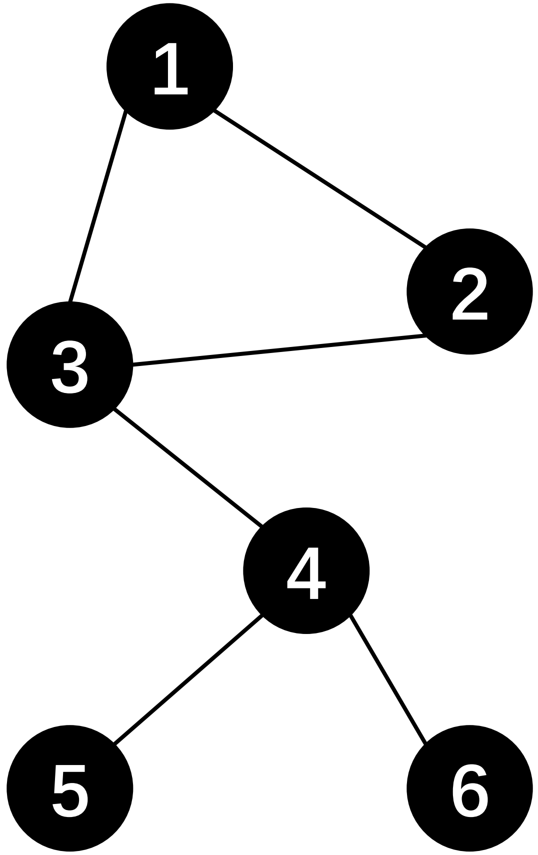
Some algorithms based on DFS

- Biconnectivity, articulation point, bridges (CLRS pp. 621-622)
- Topological sort (CLRS pp. 612-614)
- Strongly connected components (CLRS pp. 615-618)
- Will briefly mention them if we have time in the last lecture for graphs



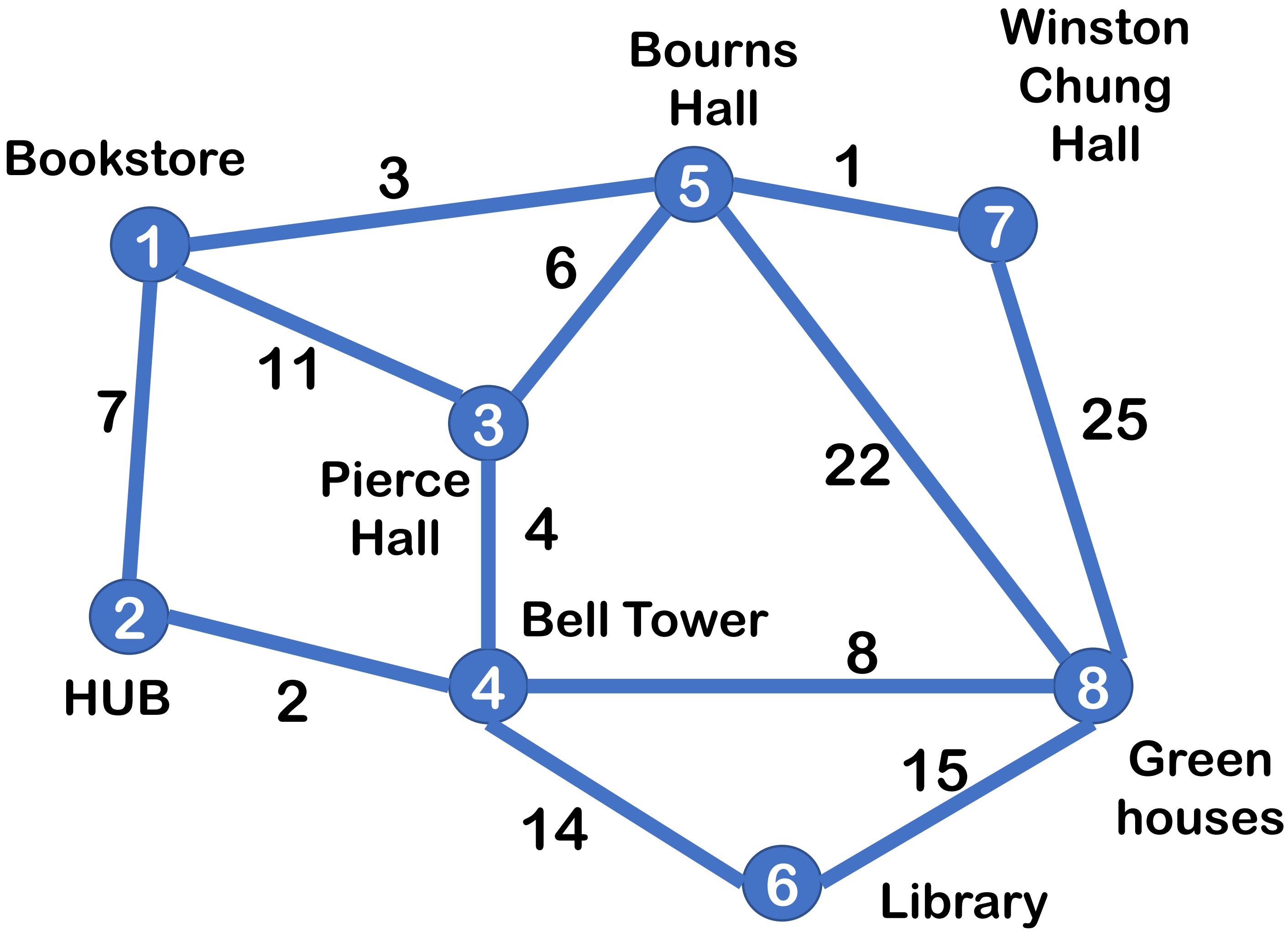
BFS

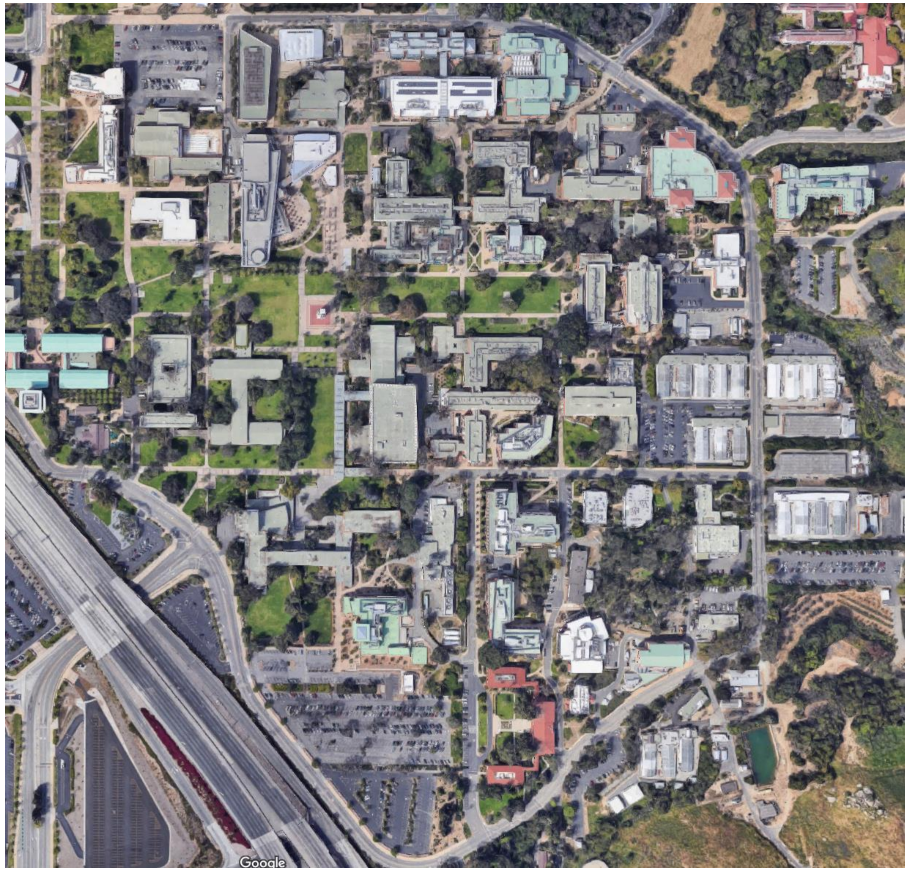
```
queue[n];
head = 0; tail = 1;
Visited[0..n-1] = {false};
visited[s] = true;
while (head != tail) {
  forall u of queue[head]'s neighbor:
    if (!visited[u]) {
      queue[tail++] = u;
      visited[u] = true;
    }
}
```



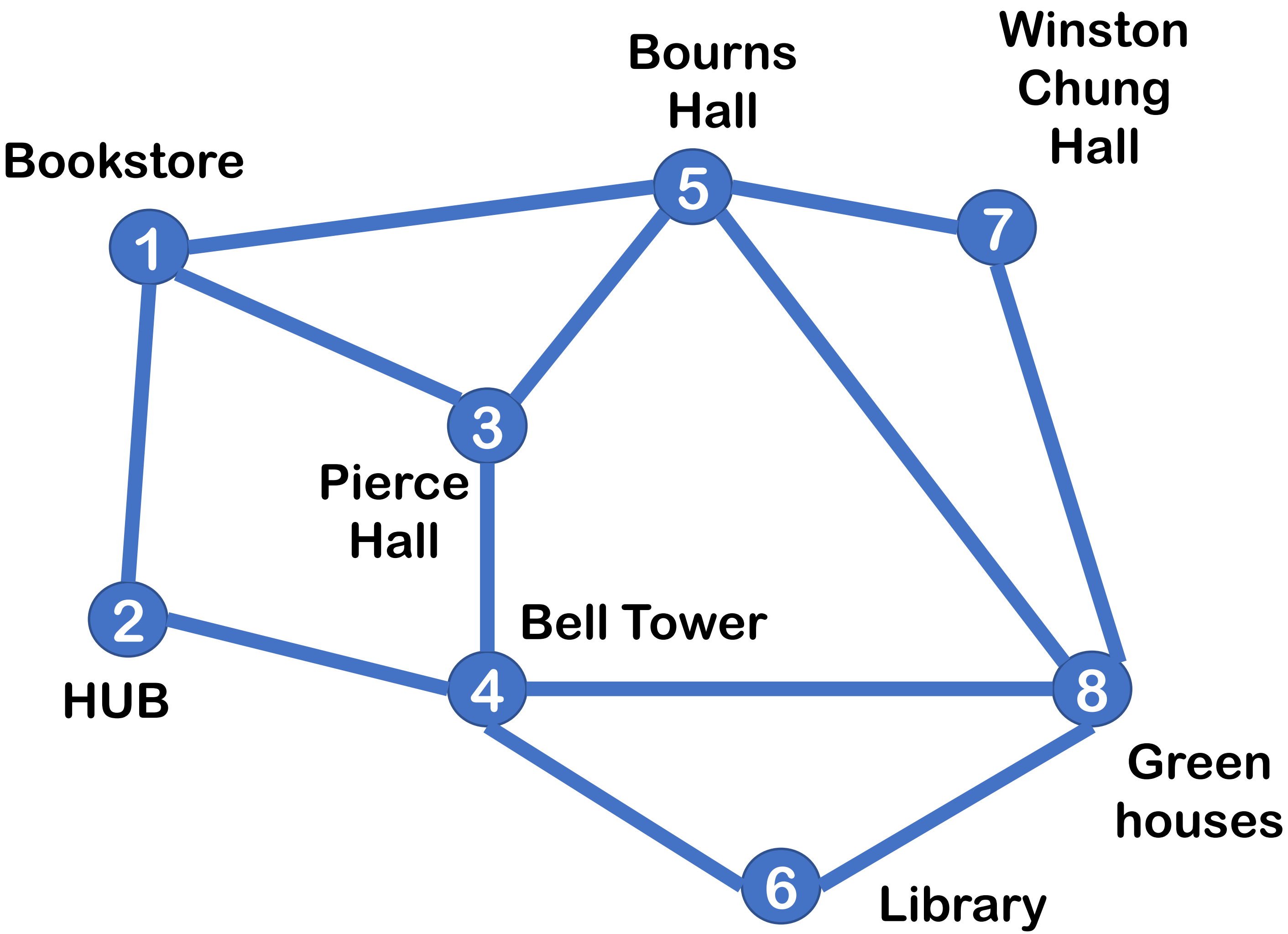
- **Time bound:** $O(m)$

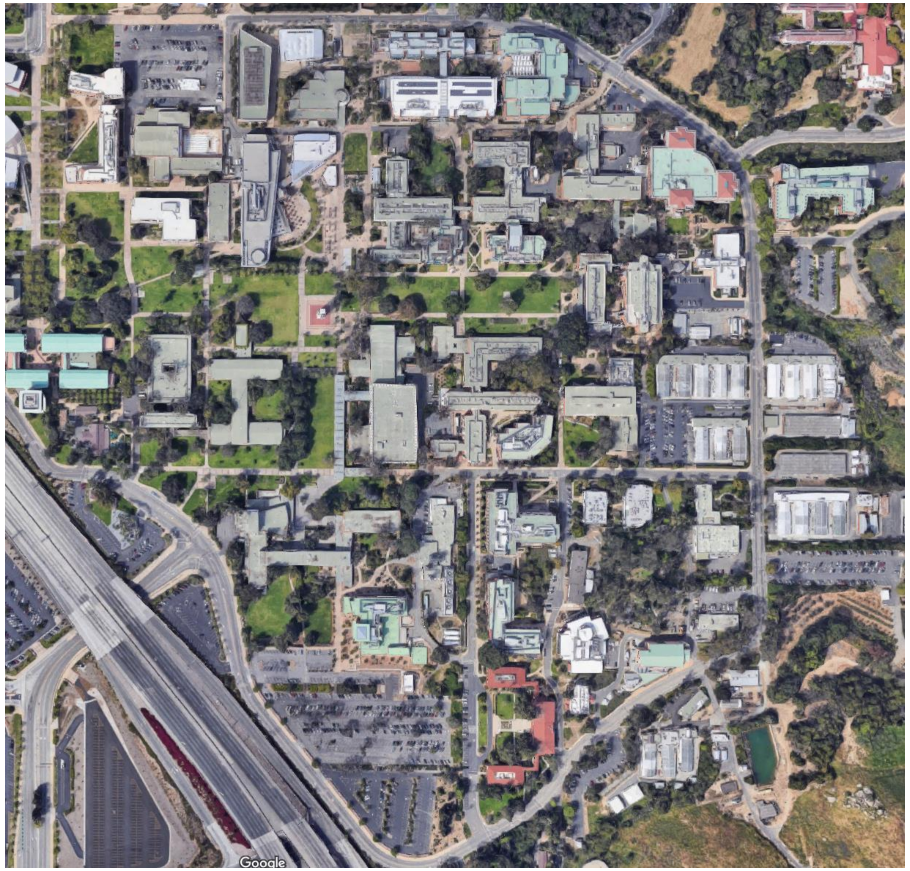
Unweighted shortest paths





Unweighted shortest paths





Next two lectures

- **Minimum spanning trees (MST)**
  - Prim's algorithm
  - Kruskal's algorithm

- **Single-source shortest-paths (SSSP)**
  - Dijkstra's algorithm
  - Bellman-Ford algorithm