CS 430 Lecture 24 Activities

Relationships between items:

$1-1$ List, Stack, Queue

$1-N$ Tree, Hierarchical

$N-N$ Any graph item can be related to any other item.

Opening Questions

1. Give an example NOT discussed in the video lecture of a problem that can be represented by a graph. The connections between airports.
2. If there is a path in a graph from a vertex back to itself, that is called a cycle[1].
3. Which representation of a graph, adjacency-list and adjacency-matrix, usually uses more memory and why?

Graphs

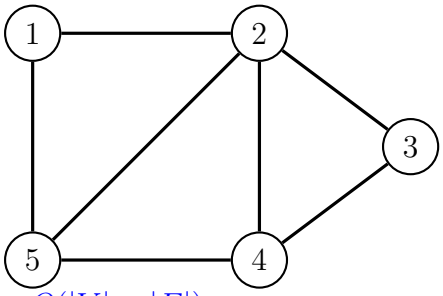
1. Draw the graph: A directed graph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. What is the edge $(2, 2)$ called? The edge $(2, 2)$ is a self-loop.
2. Draw the graph: An undirected graph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}\}$. What is vertex 4 called? What is the difference about how an edge set E is denoted for an undirected graph? Are self-loops allowed in an undirected graph?
3. Define these terms:
   - Vertex v is adjacent to vertex u in an undirected graph.
   - Vertex v is adjacent to vertex u in a directed graph.
   - The degree of a vertex in an undirected graph.
   - The degree of a vertex in a directed graph.
   - A path in an undirected graph.
   - A path in a directed graph.
   - The length of a path.
   - v is reachable from u.
   - A simple path.
   - A cycle in an undirected graph. What about a simple cycle?

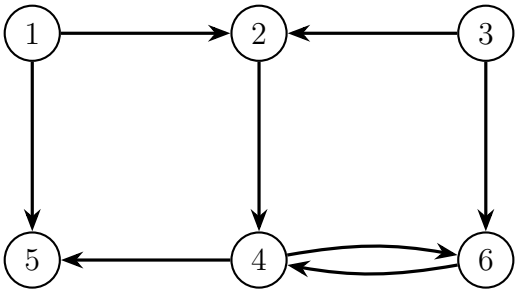
[1] A path of one edge is called a self-loop.

- A cycle in a directed graph. What about a simple cycle?
- Acyclic graph.
- Connected undirected graph.
- Connected directed graph.
- Bipartite Graph.

Graph Implementations

4. What is the adjacency list implementation of these two graphs?





Memory: $O(|V| + |E|)$

5. What is the adjacency matrix implementation of the above two graphs? Memory: $O(|V|^2)$

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

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

6. How do the two implementations handle a weighted graph? Edges have numbers, add weights to the adj list and adj matrix.

7. Two different representations of the graph data structure are discussed in the book, adjacency-list and adjacency-matrix. Please briefly discuss the runtime (in terms of $|V|$ and $|E$ of these graph operations/algorithms using each implementation.) Assume vertices are labeled as integers.

   - What is the worst-case big-O runtime for checking to see if an edge from vertex u to vertex v exists? Adj list: $O(V)$, Adj matrix: $O(1)$
   - How long does it take to compute the out-degree of every vertex of a directed graph?
   - How long does it take to compute the in-degree of every vertex of a directed graph?

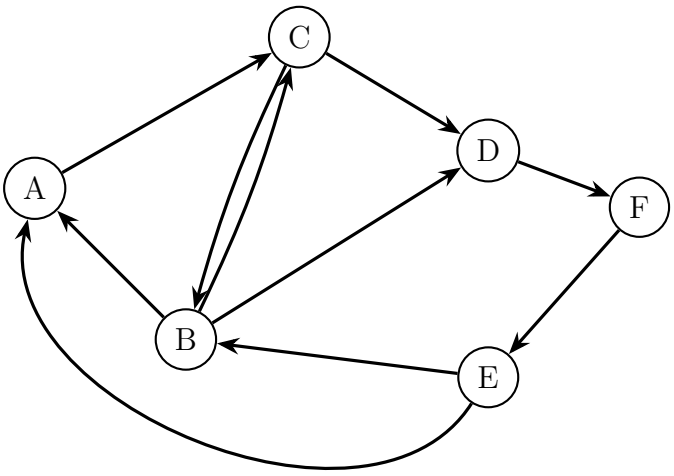
Graph Traversals

A way to search/visit all the vertices in a graph. There is not a unique answer usually.

- Undirected graph - if connected, all vertices will be visited.
- Directed graph - Must be strongly connected to be able to visit all vertices.

Breadth first - visit vertices one edge from a given (or random) source, two edges from source, etc. Uses a queue and some way to make a vertex (white initially, gray when first visited and put in queue, black when out of queue), label a vertex how far from the source, and label a vertex with how its predecessor vertex was during the traversal.

8. Perform a breadth first search on this graph.



Algorithm 24.1 Breadth-First Search for Graphs

```
1: function BFS(G, s)
2:     for all vertex u ∈ V[G] − {s} do
3:         Color(u) ← WHITE                          ▷ Unvisited
4:         d[u] ← ∞
5:         π[u] ← NIL
6:     end for
7:     Color(s) ← GRAY                               ▷ First time seen, put in queue
8:     d[s] ← 0                                      ▷ d is the distance from the start
9:     π[s] ← NIL                                    ▷ π is the predecessor
10:    Q ← ∅
11:    Enqueue(Q, s)
12:    while Q ≠ ∅ do
13:        u ← Dequeue(Q)
14:        for all v ∈ Adj(u) do
15:            if Color(v) == WHITE then
16:                Color(v) ← GRAY
17:                d[v] ← d[u] + 1
18:                π[v] ← u
19:                Enqueue(Q, v)
20:            end if
21:        end for
22:        Color(u) ← BLACK                          ▷ Last time seen, out of queue
23:    end while
24: end function
```