Graphs Algorithms

- A GRAPH G = (V, E)
  - V : set of vertices (nodes)
  - E : set of edges (links)
- Complete graph
  - There is an edge between every pair of vertices
  - Tow kinds of graph
    - undirected
    - directed (digraph)
- Undirected graph
  - E consists of sets of two elements each: Edge (u,v) is the same as {v,u}

Terminology

- Adjacency
  - Vertex w is adjacent to v if and only if (v, w) is in E
- Weight
  - A const parameter associated with each edge
- Path
  - Sequence of vertices where there is an edge for each pair of consecutive vertices
- Length of path
  - Number of edges along path
  - Length of a path of n vertices is n-1

Cycles

- A path is simple if all its vertices are distinct (first and last may be equal)
- A cycle path is a path $w_1, w_2, \dots w_n = w_1$
  - A cycle is simple if the path is simple
  - It has a loop if a node repeats
- An undirected graph is connected if
  - Each pair of vertices u,v there is a path that starts at u and ends at v
- A digraph H that satisfies the above condition is strongly connected
- Otherwise if H is not strongly connected, but the undirected graph G with the same set of vertices and edges is connected, H is said to be weakly connected
- BFS Finds all nodes

Representation of Graphs

- To store graph information, we need to store the connectivity (link) information.
- Two popular representations
  - Adjacency matrix
    - Use a 2d array to store the connectivity: `A[u][v]` is true if there is an edge from u to v, false otherwise
  - \

Adjacency matrix

- `A[u][v]` is true if there is an edge from u to v
- False otherwise

- For a weighted graph, assign weight instead of t or f
- O(1) time to decide whether (u,v) is an edge
- `A[N][N]` - O(|V|^2) space
  - Wasteful if the graph is sparse, not too many edges
- *Adjacency list*
  - Each node maintains a list of neighbors
  - Need to go through the list to decide if u, v is an edge

Topological Sorting

- Let G be a directed acyclic graph (DAG)
- an ordering the vertices of G such that if there is an edge from v_i to v_j appears after v_i
- In a DAG, there must be a vertex with no incoming edges
- Have each vertex maintain its indegree, indegree of v = number of edges (u, v)
- Repeat
  - Find a vertex of current indegree 0
  - assign it a rank
  - reduce the indegrees of the vertices in its adjacency list

```C++
void Graph()::topsort{
	for(int counter = 0; counter < NUM_VERT; counter++){
		Vertex v - findNewVertexOfIndegreeZero();
		if(v == NOT_A_VERTEX){
			throw CycleFoundException()
		}
		v.topNum = counter;
		for each Vertex w adjacent to v
			w.indegree--;
	}
}

void Graph::topsort(){
	Queue<Vertex> q;
	int counter = 0;
	q.makeEmpty();
	for each Vertex v
		if( v.indegree == 0){
			q.dequeue(v);
		}
	while(!q.isEmpty){
		Vertex v = q.dequeue();
		v.topNum ++counter;
	for each Vertex w adjacent to v
		if(--w.indegree == 0)
			// Etc
	}
}
```

Single Source Shortest Path Problem

- Unweighted shortest paths
  - BFS
- Weighted
  - Dijkstras algorithm
  1. Pick one node with the shortest distance

2. Update the distance for all nodes that are adjacent to the picket node
3. repeat till all nodes are picked