Benjamin Fenster
CS-104 (Lee)
18 December 2004

Graphs

I. Concepts
  a. A graph is a pair (V, E) of a set of vertices V and a set of edges E
  b. An edge is a pair of vertices (V_i and V_J) for some V_i and $V_J \in$ V.
  c. Can be the *empty* set. Vertices in an edge can be the same.
  d. Two distinct vertices V_i , V_J, i ≠ J are said to be adjacent if there exists an edge (V_i, V_J)
  e. Two distinct vertices V_i , V_J, i ≠ J are connected if there exists a path between them
  f. Path
    i. Sequence of vertices V_1, V_2, …, V_N such that (V_i, V_{i+1}) $\in$ E for i = 1, 2, …, N – 1
    ii. The length of a path is the number of *edges* in the path (i.e. N – 1)
    iii. The path is a *cycle* if $V_1 = V_N$
    iv. The path is *simple* of all V_i, i = 1, 2, …, N – 1 are distinct (NB: $V_1 = V_N$ is allowed!)
    v. Given a path V_1, V_2, …, V_{i-1}, V_i, V_{i+1}, …, V_N, we have V_1, …, V_{i-1} are V_i's predecessors. V_{i+1}, …, V_N are v_i's successors
  g. Edges
    i. An edge is directed if it is defined as an *ordered* pair of vertices (V_i, V_J)
    ii. Directed edges are denoted as V_i -> V_J
    iii. V_i is called the source, V_J the destination
    iv. The number of incoming edges to a vertex is the indegree
  h. Graph Types
    i. A graph is directed if each edge is directed
    ii. A graph is weighted if each edge has a weight
    iii. A graph is acyclic if there is no cycle in any path
    iv. A graph is connected if $\exists$ a path between any V_i, $V_J \in$ E, i ≠ J
    v. Completely Connected
      1. $| E | = | V | * (| V | – 1) / 2 = \theta(|V|^2)$
      2. Each vertex is connected to V – 1 vertices.
      3. Since it's undirected (A, B) = (B, A) so divide by 2.
    vi. Minimally Connected. $| E | = | V | – 1 = \theta(| V |)$
  i. Representation
    i. Adjacency Matrix
      1. 2D array
      2. bool type, shows whether [i][J] are connected or not
      3. By convention, array[x][x] all 1
      4. Typically done array[from][to] for directed graphs
    ii. Adjacency List
      1. Array of linked list
      2. Preferred for sparse graphs (fewer edges)
  j. Runtime

	Matrix	List
i. Adjacent(V_i, V_J)	θ(i)	O(|V|)
ii. Process / Visit All	$\theta (\|V\|^2)$	$\theta (\| V \| + \| E \|)$
iii. Storage Space	$\theta (\|V\|^2)$	$\theta (\| V \| + \| E \|)$

II. Graph Traversal
  a. First, tree traversal reminder
    i. Pre, in, post order (using stacks) (LIFO)
    ii. level order (using queue) (FIFO)
  b. Graph Traversal
    i. Breadth First (using queue) (FIFO)
      1. Visit nodes distance 1 first
      2. Then distance 2, 3, …
    ii. Depth First (using stack) (LIFO)
      1. Whichever direction It takes first, keep going in that direction
      2. Go as *deep* as possible, then switch to a new direction

c. Nonrecursive Breadth First

```
while Q not empty
    pop front
    if (not yet visited)
        mark as visited
        process
        add adjacent to queue
```

d. Nonrecursive Depth First – Same algorithm, but use LIFO stack
e. Recursive Breadth First
    i. Given vertex
    ii. Visit, mark as visited
    iii. For each adjacent, breadthFirstSearch(V_i)
    iv. Use a 'visited' 1D array to record which nodes have already been visited
f. Runtime
    i. The runtime is proportional to the number of elements put into the queue
    ii. Worst Case
        1. Completely connected graph
        2. How many vertices put into the queue / stack?
        3. $(|V| - 1) + (|V| - 2) + \ldots + 2 + 1 = |E|$
        4. First time, can pick $|V| - 1$ vertices to add
        5. Ultimately: $\theta(|V|^2)$
    iii. Best Case
        1. Minimally connected graph
        2. $(|V| - 1)$ total vertices put into the queue / stack
        3. $|E|$
    iv. Note that the runtime is proportional to $|E|$ in any case

III. Topological Sort
a. Given a directed acyclic graph (DAG) $G = (V, E)$, a topological sort orders the vertices in a topological order. For any V_i, V_J, $i \neq J$, V_i appears after V_J in the order iff there exists a path from V_i to V_J
b. Algorithm
    i. Find a vertex with no incoming edge.
    ii. Add to the result queue.
    iii. Remove V and all its outgoing edges.
    iv. Repeat until no vertices remain.
c. May want to store indegree field on each vertex to track the number of incoming edges
d. Runtime
    i. $O (|V| + |E|) = O(|E|)$
    ii. Same reasoning as for breadth-first and depth-first search times