Lecture 8: Overview

- Graphs (Chapter 22)
Graphs

• $G=(V,E)$
• $V$: vertices (or nodes). Notation $u,v \in V$
• $E$: edges. Each connection is a pair of vertices $E \subseteq V \times V; (u,v) \in E$
Variations

- Undirected or Directed
- Weighted (notation $w(u, v)$)
- Can also have weighted vertices
Undirected Graphs

- degree of a vertex: # of edges connected to the vertex
- complete graph aka clique:
  - graph where all pairs of vertices are connected
- bipartite graph
  - undirected graph whose $V=V_1 \cup V_2$ and $E \subseteq V_1 \times V_2$
- multigraph: can have multiple edges between the same pair of vertices (including self edges)
- hypergraph: can have edges connect more than two vertices
Directed Graphs

- **in-degree**: # incoming edges
- **out-degree**: # outgoing edges
- **path**: \(<v_0, v_1, ..., v_k>\) where \(<v_i, v_{i+1}> \in E\)
- **simple path**: path where all \(v_i\) are distinct
- **cycle**: a non-trivial simple path plus \(<v_k, v_0>\) to close the path
- **DAG**: directed acyclic graph (contains no cycles)
- **strongly connect**: digraph whose vertices are all reachable from each other
Representations

• Adjacency list:
  • For each node, list its neighbors on outgoing edges
  • good for sparse graphs
  • for weighted graphs, store weight in linked list

• Adjacency matrix:
  • bit matrix to represent presence of edge
  • weighted - can store weight in matrix

• Tradeoffs
  • adjacency lists - more space efficient, easier to grow
  • adjacency matrix - fast access, but $O(V^2)$ space
Breadth First Search

- “Distance” refers to number of vertices
- Visit vertices in increasing order of distance from starting point
- Not necessarily unique
- Explored breadth and then depth
BFS(G,s)

1  for each vertex u in V[G] -[s]
2       do color[u] <- WHITE
3           d[u] <- Infinity
4           pr[u] <- NIL
5 color[s] <- GRAY
6  d[s] <- 0
7 pr[s] <- NIL
8 Q <- {}
9 ENQUEUE(Q,s)
10 while Q ≠ {}
11   do u <- DEQUEUE(Q)
12       for each v in Adj[u]
13           do if color[v]=WHITE
14           then color[v]<-GRAY
15           d[v] <- d[u] + 1
16           pr[v] <- u
17           ENQUEUE(Q, v)
18  color[u] <- BLACK
Depth first search

- Explores depth then neighbors
- Depth isn’t necessarily the same as distance in BFS
- discover vertices before we visit
- push vertices on stack
- DFS order may not be unique!
DFS(G)

1 for each u in V[G]
2 do color[u] <- WHITE
3 pr[u] <- NIL
4 time <- 0
5 for each u in V[G]
6 do if color[u] = WHITE
7 then DFS-VISIT(u)
DFS-VISIT(u)

1. color[u] <- GRAY (discovered u)
2. time <- time +1
3. d[u] <- time
4. for each v in Adj[u]
5. do if color[v] = WHITE
6. then pr[v] <- u
7. DFS-VISIT(v)
8. color[u] <- BLACK
9. f[u] <- time <- time +1
Time stamps

• Time stamp each vertex with two time stamps
  • discover time d[u]
  • finish time f[u]
• Can use to detect cycles
Topological Sorting

- Directed Acyclic Graph
- Sorted order in which all of a node’s predecessors appear before the node
- Useful if computing some property on a graph that requires computations of predecessors
- Useful for scheduling tasks that depend on other tasks
Algorithm

1 call DFS to compute finishing times for each vertex v
2 as each vertex is finished insert it onto the front of a list
3 return the list of vertices