22. Elementary Graph Algorithms
22.1 Representations of graphs

Given graph $G = (V, E)$.

- May be either directed or undirected.
- Two common ways to represent for algorithms:
  - Adjacency lists.
  - Adjacency matrix.
- When expressing the running time of an algorithm, it’s often in terms of both $|V|$ and $|E|$. In asymptotic notation—and only in asymptotic notation—we’ll drop the cardinality. Example: $O(V + E)$. 
Adjacency lists

Array $\text{Adj}$ of $|V|$ lists, one per vertex. Vertex $u$’s list has all vertices $v$ such that $(u, v) \in E$. (Works for both directed undirected graphs.)

**Example:** For an undirected graph:

If edges have *weights*, can put the weights in the lists.

Weight: $w : E \rightarrow \mathbb{R}$

We’ll use weights later on for spanning trees and shortest paths.

*Space:* $\Theta(V + E)$.

*Time:* to list all vertices adjacent to $u$: $\Theta(\text{degree}(u))$.

*Time:* to determine if $(u, v) \in E$: $O(\text{degree}(u))$. 
**Example:** For a directed graph:

![Directed graph diagram]

Same asymptotic space and time.
Adjacency matrix

$|V| \times |V|$ matrix $A = (a_{ij})$

$a_{ij} = \begin{cases} 
1 & \text{if } (i, j) \in E, \\
0 & \text{otherwise}.
\end{cases}$

$$
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\quad
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 1 \\
3 & 1 & 1 & 0 & 0 \\
4 & 0 & 0 & 1 & 1 \\
\end{array}
$$

**Space:** $\Theta(V^2)$.

**Time:** to list all vertices adjacent to $u$: $\Theta(V)$.

**Time:** to determine if $(u, v) \in E$: $\Theta(1)$.

Can store weights instead of bits for weighted graph.

We’ll use both representations in these lecture notes.
22.2 Breadth-first search

- Given a graph \( G = (V, E) \) and a distinguished source vertex \( s \), breadth-first search systematically explores the edges of \( G \) to "discover" every vertex that is reachable from \( s \).
- To keep track of progress, breadth-first search colors each vertex white, gray, or black.
- All vertices start out white.
- Gray and black vertices have been discovered:
  - all vertices adjacent to black vertices have been discovered
  - gray vertices may have some adjacent white vertices
22.2 Breadth-first search

- The color of each vertex \( u \in V \) is stored in the variable \( \text{color}[u] \)

- The predecessor of \( u \) is stored in the variable \( \pi[u] \)
  - If \( u \) has no predecessor (for example, if \( u = s \) or \( u \) has not been discovered), then \( \pi[u] = \text{NIL} \).

- The distance from the source \( s \) to vertex \( u \) computed by the algorithm is stored in \( d[u] \)
BFS algorithm

1. for each vertex \( u \in V[G] - \{s\} \)
   - do \( \text{color}[u] \leftarrow \text{WHITE} \)
   - \( d[u] \leftarrow \infty \)
   - \( \pi[u] \leftarrow \text{NIL} \)

2. \( \text{color}[s] \leftarrow \text{GRAY} \)
3. \( d[s] \leftarrow 0 \)
4. \( \pi[s] \leftarrow \text{NIL} \)
5. \( Q \leftarrow \emptyset \)
6. \( \text{ENQUEUE}(Q, s) \)

while \( Q \neq \emptyset \)
- do \( u \leftarrow \text{DEQUEUE}(Q) \)
- for each \( v \in \text{Adj}[u] \)
  - do if \( \text{color}[v] = \text{WHITE} \)
    - then \( \text{color}[v] \leftarrow \text{GRAY} \)
    - \( d[v] \leftarrow d[u] + 1 \)
    - \( \pi[v] \leftarrow u \)
    - \( \text{ENQUEUE}(Q, v) \)

- \( \text{color}[u] \leftarrow \text{BLACK} \)

Analysis: \( O(V+E) \)

- Initialize each node, starting at the source \( s \)
- Source gets color grey, and is added to the queue \( Q \)
- If the node adjacent to the node being examined is white, then it hasn’t been visited yet. So, color it grey and add it to \( Q \) to be processed later.
- When done processing all the nodes adjacent to \( u \), color \( u \) black.
The operation of BFS

(a)

(b)

(c)

(d)

Q

\[ Q \begin{bmatrix} s \end{bmatrix} 0 \]

\[ Q \begin{bmatrix} w & r \end{bmatrix} \]

\[ Q \begin{bmatrix} r & t & x \end{bmatrix} 1 2 2 \]

\[ Q \begin{bmatrix} t & x & v \end{bmatrix} 2 2 2 \]
The operation of BFS
The operation of BFS

Time = $O(V + E)$.

- $O(V)$ because every vertex enqueued at most once.
- $O(E)$ because every vertex dequeued at most once and we examine $(u, v)$ only when $u$ is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected.
Shortest paths

\( \delta(s,v) \): shortest path from \( s \) to \( v \) (minimum number of edges in any path from vertex \( s \) to vertex \( v \))

**Lemma 22.1.** Let \( G = (V, E) \) be a directed or undirected graph, and let \( s \in V \) be an arbitrary vertex. Then for any edge \( (u,v) \in E \)

\[ \delta(s,v) \leq \delta(s,u) + 1. \]
Lemma 22.2. Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on $G$ from a given source $s \in V$. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \geq \delta(s, v)$.

Proof. (Induction on the number of times a vertex is placed in the queue)

Lemma 22.3. Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue $Q$ contains the vertices $<v_1, v_2, \ldots, v_r>$, where $v_1$ is the head of $Q$ and $v_r$ is the tail. Then $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i=1, 2, \ldots, r-1$.

Proof. (induction on the number of queue operations)
**Corollary 22.4.** Suppose vertices $v_i$ and $v_j$ are enqueued during the execution of BFS, and that $v_i$ is enqueued before $v_j$ is enqueued. Then 

$$d[v_i] \leq d[v_j]$$

at the time that $v_j$ is enqueued.

**proof** Immediate form Lemma 22.3 and the property that each vertex receives a finite $d$ value at most once during the course of BFS.

We can now prove that breadth-first search correctly finds shortest-path distances.
Theorem 22.5 (Correctness of BFS)

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on $G$ from a given source $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source, and upon termination $d[v] = \delta(s, v)$ for all $v \in V$. Moreover, for any $v \neq s$ that is reachable from $s$, one of the shortest paths from $s$ to $v$ is the shortest path from $s$ to $\pi[v]$ followed by the edge $(\pi[v], v)$.

Proof. In pp.537
For a graph \( G = (V, E) \) with source \( s \), we define the predecessor subgraph of \( G \) as \( G_{\pi} = (V_{\pi}, E_{\pi}) \) where \( V_{\pi} = \{ v \in V | \pi[v] \neq NIL \} \cup \{ s \} \), and \( E_{\pi} = \{ (\pi[v], v) \in E | v \in V_{\pi} - \{ s \} \} \). The edges in \( E_{\pi} \) are called tree edges.

**Lemma 22.6.** When applied to a directed or undirected graph \( G = (V, E) \) procedure BFS constructs \( \pi \) so that the predecessor subgraph \( G_{\pi} = (V_{\pi}, E_{\pi}) \) is a breadth-first tree.
PRINT_PATH(G, s, v)

1 if v = s
2 then print s
3 else if \( \pi[v] = \text{NIL} \)
4 then print "no path from" s "to" v "exist"
5 else PRINT-PATH(G, s, \( \pi[v] \))
6 print v
22.3 Depth-First Search

**DFS(G)**

1. for each vertex $u \in V[G]$
2. do $color[u] \leftarrow \text{white}$
3. $\pi[u] \leftarrow \text{NIL}$
4. $time \leftarrow 0$
5. for each vertex $u \in V[G]$
6. do if $color[u] = \text{white}$
7. then DFS-VISIT($u$)

**DFS-VISIT($u$)**

1. $color[u] = \text{gray}$
2. $d[u] \leftarrow time \leftarrow time + 1$
3. for each $v \in \text{adj}[u]$ do if $\text{color}[v] = \text{white}$ then $\pi[v] \leftarrow u$
4. DFS-VISIT($v$)
5. $color[u] = \text{black}$
6. $f[u] \leftarrow time \leftarrow time + 1$
predecessor subgraph:

depth-first forest, depth-first tree

Time stamps: $d(u)$ discovered

$f(u)$ finished

Complexity: $O(V+E)$
The progress of DFS
The progress of DFS
The progress of DFS
The progress of DFS
Properties of depth-first search

Theorem 22.6. (Parenthesis theorem)

In any depth-first search of a (directed or undirected) graph $G = (V, E)$ for any two vertices $u$ and $v$, exactly one of the following conditions holds:

- the intervals $[d(u), f(u)]$ and $[d(v), f(v)]$ are entirely disjoint.
- the interval $[d(u), f(u)]$ is contained entirely within the interval $[d(v), f(v)]$, and $u$ is a descendant of $v$ in the depth-first tree, or
- the interval $[d(v), f(v)]$ is contained entirely within the interval $[d(u), f(u)]$, and $v$ is a descendant of $u$ in the depth-first tree.
Corollary 22.8. (Nesting of descendants’ interval)
Vertex \( v \) is a proper descendant of a vertex \( u \) in the depth-first forest for a (directed or undirected) graph \( G \) if and only if \( d(u) < d(v) < f(v) < f(u) \).
Property of DFS
Theorem 22.9 (white path theorem)
In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex $v$ is a descendant of vertex $u$ if and only if at time $d[u]$ that the search discover $u$, vertex $v$ can be reached from $u$ along a path consisting entirely of white vertices.

Classification of edges:
**Tree edges**: edges in the depth-first forest. Edge $(u,v)$ is a tree edge if $v$ was first discovered by exploring edge $(u,v)$
**Back edges**: $(u,v)$ connecting a vertex $u$ to an ancestor $v$ in a DFS tree. Self-loops are considered back edges.
**Forward edges**: those nontree edges $(u,v)$ connecting a vertex $u$ to a descendant $v$ in a DFS tree
**Cross edges**: all other edges.

Theorem 22.10. In a depth-first search of an undirected graph $G$, every edge of $G$ is either a tree edge or a back edge.
22.4 Topological sort

A topological sort of a dag \( G = (V, E) \) is a linear ordering of all its vertices such that if \( G \) contains an edge \((u, v)\), then \( u \) appears before \( v \) in the ordering.

DAG : Directed Acyclic Graph
(a)  

- 11/16: undershorts  
- 12/15: pants  
- 6/7: belt  
- 1/8: shirt  
- 2/5: tie  
- 3/4: jacket  
- 17/18: socks  
- 13/14: shoes  
- 9/10: watch

(b) 

- 17/18: socks  
- 11/16: undershorts  
- 12/15: pants  
- 13/14: shoes  
- 9/10: watch  
- 1/8: shirt  
- 6/7: belt  
- 2/5: tie  
- 3/4: jacket
TOPOLOGICAL_SORT(G)

- Call DFS(G) to compute finishing time f(v) for each vertex v.
- As each vertex is finished, insert it onto the front of a link list.
- Return the link list of vertices
Lemma 22.11. A directed graph $G$ is acyclic if and only if a depth first search of $G$ yields no back edge.

Theorem 22.12. $\text{TOPOLOGICAL\_SORT}(G)$ produces a topological sort of a directed acyclic graph $G$. 

$\rightarrow f[u] > f[v]$
22.5 Strongly connected components

STRONGLY_CONNECTED_COMPONENT(G)

1. call DFS(G) to compute finishing time $f[u]$ for each vertex $u$
2. compute $G^T$
3. call DFS($G^T$), but in the main loop of DFS, consider the vertices in the order of decreasing $f[u]$.
4. Output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected components

Complexity: $O(V + E)$
Lemma 22.13. Let $C$ and $C'$ be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that there is a path $u \xrightarrow{} u'$ in $G$. Then there cannot also be a path $v' \xrightarrow{} v$ in $G$. 
Lemma 22.14. Let $C$ and $C'$ be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E$, where $u \in C$ and $v \in C'$. Then $f(C) > f(C')$. 

\[ f(C) > f(C') \]
Corollary 22.15. Let $C$ and $C'$ be distinct strongly connected components in directed graph $G = (V, E)$. Suppose that there is an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$.

Theorem 22.16. Strongly-connected-component($G$) correctly computes the strongly connected components of a directed graph $G$. 