Graph Algorithms: Chapters 22-25

Part 1: Introductory graph concepts
91.404 Graph Review

Elementary Graph Algorithms
Minimum Spanning Trees
Single-Source Shortest Paths
Introductory Graph Concepts

- \( G = (V, E) \)
- Vertex Degree
- Self-Loops

- Undirected Graph
  - No Self-Loops
  - Adjacency is symmetric

- Directed Graph (digraph)
  - Degree: in/out
  - Self-Loops allowed

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Introductory Graph Concepts: Representations – Undirected

- Undirected Graph
  - Adjacency-list
    - Compact for sparse graphs
      - $|E| < |V|^2$
  - Adjacency-matrix
    - For dense graph
      - $|E| \approx |V|^2$
    - Quickly tell if there’s edge connecting 2 vertices

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Undirected graph representation

- Alternative to vertices: pointers
  - Both for undirected and directed

Figure 22.1 Two representations of an undirected graph. (a) An undirected graph $G$ having five vertices and seven edges. (b) An adjacency-list representation of $G$. (c) The adjacency-matrix representation of $G$. 
Introductory Graph Concepts: Representations -- Directed

- Directed Graph (digraph)

![Directed Graph Diagram]

Adjacency Matrix

Adjacency List

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Figure 22.2  Two representations of a directed graph. (a) A directed graph $G$ having six vertices and eight edges. (b) An adjacency-list representation of $G$. (c) The adjacency-matrix representation of $G$. 
Adjacency list

- Memory requirements: $\Theta(V + E)$
- Time: to list all vertices adjacent to $u$:
  - $\Theta(\text{degree}(u))$
- Easily adapt to weighted graph
  - Weight function: $w : E \rightarrow \mathbb{R}$
  - E.g., $G = (V, E)$
  - $w(u, v), (u, v) \in G$
    - Stored $w/ v$ in $u$’s adjacency list
Adjacency list – disadvantage

- No quick way to determine if given edge \((u, v)\) is in graph
  - Must search for \(v\) in \(\text{Adj}[u]\)
    - \(O(\text{degree}(u))\)
  - Fixed by adjacency matrix

- Cost: asymptotically more memory
  - (Exercise 22.1-8: variations on adjacency lists w/faster edge lookup)
Adjacency matrix

- \( G = (V, E) \)
  - Vertices numbered 1, 2, ..., \(|V|\)
    - Arbitrary numbering
  - \(|V| \times |V|\) matrix \( A = (a_{ij}) \)
    \[
    a_{ij} = \begin{cases} 
      1 \text{ if } (i, j) \in E \\ 
      0 \text{ otherwise} 
    \end{cases}
    \]
- Can use w/ weighted
  - Stored \( w(u, v) \) in matrix entry
    - (instead of 1)
Adjacency matrix

- Space: $\Theta(V^2)$
- Time: to list all vertices adjacent to $u$: $\Theta(V)$
- Time: to determine if $(u, v) \in E$: $\Theta(1)$
Introductory Graph Concepts: Paths, Cycles

- Path:
  - length: number of edges
  - simple: all vertices distinct

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Introductory Graph Concepts: Paths, Cycles

- **Cycle:**
  - **Directed Graph:**
    - \(<v_0, v_1, \ldots, v_k>\) forms cycle if \(v_0 = v_k\) and \(k \geq 1\)
    - simple cycle: \(v_1, v_2, \ldots, v_k\) also distinct
    - self-loop is cycle of length 1

![Diagram of a graph with labeled vertices A, B, C, D, E, F showing a simple cycle <E, B, F, E>](image)

*This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.*
Introductory Graph Concepts:
Paths, Cycles

- Cycle:
  - Undirected Graph:
    - \( \langle v_0, v_1, \ldots, v_k \rangle \) forms (simple) cycle if \( v_0 = v_k \) and \( k \geq 3 \)
    - simple cycle: \( v_1, v_2, \ldots, v_k \) also distinct

simple cycle \( \langle A, B, C, A \rangle = \langle B, C, A, B \rangle \)

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Introductory Graph Concepts: Connectivity

- Undirected Graph:
  - connected
    - every pair of vertices connected by path
    - one connected component
  - connected components:
    - equivalence classes
    - under “is reachable from” relation

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Introductory Graph Concepts: Connectivity

- Directed Graph:
  - strongly connected
    - every pair of vertices reachable from each other
    - one strongly connected component
    - strongly connected components:
      - equivalence classes
      - under “mutually reachable” relation

This treatment follows 91.503 textbook Cormen et al. Some definitions differ slightly from other graph literature.
Elementary Graph Algorithms: SEARCHING: DFS, BFS

- for unweighted directed or undirected graph $G = (V,E)$
- Time:
  - $O(|V| + |E|)$ adj list
  - $O(|V|^2)$ adj matrix
Breadth-First-Search (BFS):

- Shortest Path Distance
  - From source to each reachable vertex
  - Record during traversal
- Foundation of many “shortest path” algorithms
- EdgeColor of vertex when first tested determines edge type

**Vertex color shows status:**
- not yet encountered
- encountered, but not yet finished
- finished
- **Input:**
  - Graph $G = (V, E)$, un/directed
  - *source vertex* $s \in V$

- **Output:**
  - $d[v] =$ distance (smallest # of edges) from $s$ to $v$, for all $v \in V$
  - $\pi[v] = u$ such that $(u, v)$ is last edge on shortest path $s \rightarrow v$
  - $u$ is $v$’s *predecessor*
  - set of edges $\{(\pi[v], v) : v = s\}$ forms tree

- Later, generalization of breadth-first search, with edge weights
Breadth-First-Search (BFS)

- **Idea:** Send wave out from $s$
  - First hit all vertices 1 edge from $s$
  - From there, hit all vertices 2 edges from $s$
  - Etc.

- Use FIFO queue $Q$ to maintain wavefront.
  - $v \in Q$ iff wave has hit $v$ but has not come out of $v$ yet
BFS\((G, s)\)

1. for each vertex \(u \in V[G] - \{s\}\)
2. \(\text{do } color[u] \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(color[s] \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \(\text{ENQUEUE}(Q, s)\)
10. while \(Q \neq \emptyset\)
11. \(\text{do } u \leftarrow \text{DEQUEUE}(Q)\)
12. \(\text{for each } v \in Adj[u]\)
13. \(\text{do if } color[v] = \text{WHITE}\)
14. \(\text{then } color[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \(\text{ENQUEUE}(Q, v)\)
18. \(color[u] \leftarrow \text{BLACK}\)
Figure 22.3  The operation of BFS on an undirected graph. Tree edges are shown shaded as they are produced by BFS. Within each vertex $u$ is shown $d[u]$. The queue $Q$ is shown at the beginning of each iteration of the while loop of lines 10–18. Vertex distances are shown next to vertices in the queue.
Example: directed graph

- Can show that $Q$ consists of vertices with $d$ values
  - $i \ i \ i \ . . . \ i \ i + 1 \ i + 1 \ . . . \ i + 1$
    - Only 1 or 2 values
    - If 2, differ by 1 and all smallest are first
- Since each vertex gets a finite $d$ value at most once
  - values assigned to vertices monotonically increasing over time
Analysis of BFS

- Aggregate analysis
  - After initialization no vertex ever whitened
  - \( \Rightarrow \) Test in line 13 (if \( \text{color}[v] = \text{WHITE} \))
    - Ensures ea. vertex enqueued at most once
    - \( \Rightarrow \) dequeued at most once
  - Enqueue / dequeue take \( O(1) \) time
    - \( \Rightarrow \) total time for queue op’s is \( O(V) \)
  - Adj-list of each vertex scanned only when vertex dequeued
  - \( \Rightarrow \) adj-list scanned at most once
BFS analysis, cont.

- Sum of lengths of all adj-lists = $\Theta(E)$
- $\Rightarrow$ total time spent scanning adj-lists = $O(E)$
- Initialization overhead = $O(V)$
- $\Rightarrow$ Total running time = $O(V + E)$
- $\Rightarrow$ BFS runs in time linear in size of adj-list rep in $G$
• BFS may not reach all vertices
Elementary Graph Algorithms: SEARCHING: DFS

- for unweighted directed or undirected graph $G = (V,E)$
- Time:
  - $O(|V| + |E|)$ adj list
  - $O(|V|^2)$ adj matrix
Depth-First-Search (DFS)

- **Input:** $G = (V, E)$, un/directed
  - No source vertex given!

- **Output:** 2 *timestamps* on each vertex:
  - $d[v] = \textit{discovery time}$
  - $f[v] = \textit{finishing time}$
    - Will be useful for other algorithms later on
  - Also compute $\pi[v]$
DFS

- Will methodically explore every edge
  - Start over from different vertices as necessary
- As soon as discover vertex, explore from it
  - Unlike BFS
    - puts vertex on queue
    - explore from it later
As DFS progresses, every vertex has color:

- **WHITE** = undiscovered
- **GRAY** = discovered, but not finished (not done exploring from it)
- **BLACK** = finished (have found everything reachable from it)
- Discovery and finish times:
  - Unique integers from 1 to 2 $|V|$
    - For all $v$, $d[v] < f[v]$.
  - In other words, $1 \leq d[v] < f[v] \leq 2 |V|$

**Vertex color shows status:**
- not yet encountered
- encountered, but not yet finished
- finished

Before $d[v]$
Between $d[v]$ and $f[v]$
After $f[v]$
Depth-First-Search (DFS)

- Discovery \( (d[u]) \), finishing \( (f[u]) \) times
  - \( \Rightarrow \) “well-formed” nested \( ((())()) \) structure (later)
- Every edge of undirected \( G \) is either
  - tree edge, or
  - back edge
- \text{EdgeColor} of vertex when first tested determines edge type
DFS

DFS($G$)

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

- 7: $u$ root of new tree in DF forest
- Return: every $u$ has $d[u] \& f[u]$
- Vertex visit order may affect results
DFS-Visit

DFS-Visit(*u*)

1. `color[u] ← GRAY`  \(\triangleright\) White vertex *u* has just been discovered.
2. `time ← time + 1`
3. `d[u] ← time`
4. `for each v ∈ Adj[u]`  \(\triangleright\) Explore edge (*u*, *v*).
   - `do if color[v] = WHITE`
   - `then π[v] ← u`
   - `DFS-Visit(v)`  \(\triangleright\) New tree
5. `color[u] ← BLACK`  \(\triangleright\) Blacken *u*; it is finished.
6. `f[u] ← time ← time + 1`

Exploration order may affect results

Explore only if not yet
Figure 22.4  The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Vertices are timestamped by discovery time/finishing time.
DFS running time

- Loops on lines 1-3 & 5-7 of DFS take $\Theta(V)$
  - Exclusive of DFS-Visit calls
- Aggregate analysis:
  - DFS-Visit called exactly once for ea. $v \in V$
    - Invoked only on white vertices
    - First thing: paint vertex gray
DFS running time, cont.

- During DFS-Visit(\(v\)) execution
  - Loop on 4-7 executed \(|\text{Adj}[v]|\) times

- \(\Rightarrow\) total cost of executing lines 4-7 = \(\Theta(E)\)
- \(\Rightarrow\) DFS running time = \(\Theta(V + E)\)
DFS properties

- Info about structure of graph
- Predecessor subgraph forms forest of trees
  - Mirror structure of recursive calls of DFS-Visit
DFS properties – parenthesis structure

- Discovery of vertex $u$ : “($u$)”
- Finishing : “($u$)”

→ history of discoveries and finishings makes well-formed expression
  - Parentheses properly nested
  - E.g., Fig. 22.5 (a), (b) …
  - Parenthesis theorem (Thm. 22.7 p. 543)
Figure 22.5 Properties of depth-first search. (a) The result of a depth-first search of a directed graph. Vertices are timestamped and edge types are indicated as in Figure 22.4. (b) Intervals for the discovery time and finishing time of each vertex correspond to the parenthesization shown. Each rectangle spans the interval given by the discovery and finishing times of the corresponding vertex. Tree edges are shown. If two intervals overlap, then one is nested within the other, and the vertex corresponding to the smaller interval is a descendant of the vertex corresponding to the larger. (c) The graph of part (a) redrawn with all tree and forward edges going down within a depth-first tree and all back edges going up from a descendant to an ancestor.
Parenthesis theorem (Thm. 22.7 p. 543)

- $G = (V, E), u, v$
- Exactly one of following three:
  - $[d[u], f[u]] \& [d[v], f[v]]$ disjoint
    - $u, v$ not descendants of each other in DF forest
  - $[d[u], f[u]]$ contained entirely in $[d[v], f[v]]$
    - $u$ descendant of $v$
  - $[d[v], f[v]]$ contained entirely in $[d[u], f[u]]$
    - $v$ descendant of $u$
Nesting of descendants’ intervals (Corollary 22.8 p. 545)

- Vertex $v$ is a proper descendant of vertex $u$ in the DF forest for a (directed or undirected) graph $G$ iff $d[u] < d[v] < f[v] < f[u]$
Theorem (White-path theorem)

- [Proof omitted]
- \( v \) is a descendant of \( u \) iff at time \( d[u] \), there is a path \( u \sim \rightarrow v \) consisting of only white vertices.
- (Except for \( u \), which was just colored gray.)
DFS properties – classification of edges

- Classify edges $\Rightarrow$ important info about graph
- E.g., directed graph acyclic iff DFS yields no “back” edges (Lemma 22.11)
- Edge types ….
Edge types

- Tree edges …
- Back edges …
- Forward edges …
- Cross edges …
(u, v) is tree edge
- if v was 1st discovered by exploring (u, v)
  - i.e., v was WHITE when reached from u
Back edges

- \((u, v)\) edges connecting \(u\) to ancestor \(v\) in DF tree
  - \(u\) is descendent of \(v\)
- Self loops (in directed graphs)
Forward edges

- \((u, v)\) edges connecting \(u\) to descendant \(v\) in DF tree
Cross edges: all other edges

- Go between vertices in same DF tree
  - as long as one is not ancestor of other
- Go between vertices in different DF trees
Edge type classification

- Can modify DFS alg to classify edges as encountered
  - By color of $v$ reached when edge first explored
  - But forward/cross not distinguished
- WHITE $\rightarrow$ tree
- GRAY $\rightarrow$ back
- BLACK $\rightarrow$ forward / cross
Elementary Graph Algorithms: DFS, BFS

- **Review problem: TRUE or FALSE?**
  - tree on right can be DFS tree
  - for some adjacency list representation of graph on left

![Graph Diagram]
Elementary Graph Algorithms: Topological Sort

- For Directed, Acyclic Graph (DAG) $G = (V, E)$
- Produce linear ordering of vertices
  - For edge $(u, v)$, $u$ ordered before $v$
  - Not possible if not acyclic
- Order vertices along horizontal line
  - All directed edges go L $\rightarrow$ R
- E.g., Fig. 22.7 (p. 550) …
  - order of getting dressed

source: 91.503 textbook Cormen et al.
Figure 22.7  (a) Professor Bumstead topologically sorts his clothing when getting dressed. Each directed edge \((u, v)\) means that garment \(u\) must be put on before garment \(v\). The discovery and finishing times from a depth-first search are shown next to each vertex. (b) The same graph shown topologically sorted. Its vertices are arranged from left to right in order of decreasing finishing time. Note that all directed edges go from left to right.
Topological-Sort($G$)

- Topological-Sort($G$)
  - call DFS($G$) to compute finishing times $f[v]$ for each vertex
  - as each vertex is finished, insert it onto the front of a linked list
  - return the linked list of vertices
Topological sort performance

- $\Theta(V + E)$
- DFS ($\Theta(V + E)$) + $O(1)$ to insert each of $|V|$ vertices onto front of linked list
- Correctness: Lemma 22.11 & Thm. 22.12 (p. 550-551)
- Lemma 22.11: A directed graph $G$ is acyclic iff a DFS of $G$ yields no back edges
  - Proof: p. 540 (must go ==> and <=)
- Thm. 22.12: Topological-Sort($G$) produces a topological sort of a directed graph $G$. 
Strongley connected components

- Example (Fig. 22.9) …
- The algorithm …
- Details …
Figure 22.9  (a) A directed graph $G$. The strongly connected components of $G$ are shown as shaded regions. Each vertex is labeled with its discovery and finishing times. Tree edges are shaded. (b) The graph $G^T$, the transpose of $G$. The depth-first forest computed in line 3 of STRONGLY-CONNECTED-COMPONENTS is shown, with tree edges shaded. Each strongly connected component corresponds to one depth-first tree. Vertices $b$, $c$, $g$, and $h$, which are heavily shaded, are the roots of the depth-first trees produced by the depth-first search of $G^T$. (c) The acyclic component graph $G^{SCC}$ obtained by contracting all edges within each strongly connected component of $G$ so that only a single vertex remains in each component.
Strongly-Connected-Components($G$)

1. call DFS($G$) to compute finishing times $f[u]$ for each vertex $u$
2. Compute $G^T$
3. Call DFS($G^T$), but in main loop of DFS, consider vertices in order of decreasing $f[u]$ (as computed in line 1)
4. Output vertices of each tree in depth-first forest formed in line 3 as separate strongly connected component
Details:

- 22.5, pp. 552-557
- Read!!
- Contact me with questions
- Might appear in HW / exam!!!