Data Structures

Elementary Graph Algorithms
BFS, DFS & Topological Sort
A graph, $G = (V, E)$, consists of two sets:

- $V$ is a finite non-empty set of vertices.
- $E$ is a set of pairs of vertices, called edges.

$V = \{1, 2, 3, 4\}$

$E = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$
Graphs

• In an **undirected graph**
  – The pair of vertices are unordered pairs.
  – Thus, the pairs \((v_1, v_2)\) and \((v_2, v_1)\) are the same.
  – No reflective (self) edges.

• In a **directed graph**
  – The edges are represented by a directed pair \((v_1, v_2)\).
  – Therefore \((v_2, v_1)\) and \((v_1, v_2)\) are two different edges.
  – \(v_1\) is the **tail** and \(v_2\) the **head** of the edge.
Graphs

- $G_1$ and $G_2$ are undirected.
- $G_3$ is a directed graph.

- $G_1 = (\{1,2,3,4\}, \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\})$
- $G_2 = (\{1,2,3,4,5,6,7\}, \{(1,2),(1,3),(2,4),(2,5),(3,6),(3,7)\})$
- $G_3 = (\{1,2,3\}, \{<1,2>, <2,1>, <2,3>\})$
Graphs

• The maximum number edges in
  – An undirected graph with n vertices is \( n(n - 1)/2 \)
  – A directed graph with n vertices is \( n^2 \)

• An undirected graph with n vertices and exactly \( n(n-1)/2 \) edges is said to be complete
Graphs

• If \((v_1, v_2)\) is an edge of a graph \(G\), then
  – We shall say that the vertices \(v_1\) and \(v_2\) are **adjacent**
  – And that the edge \((v_1, v_2)\) is **incident** on vertices \(v_1\) and \(v_2\)

• If \((v_1, v_2)\) is a directed edge, then vertex \(v_1\) will be said to be **adjacent to** \(v_2\), while \(v_2\) is **adjacent from** \(v_1\).

• The **degree** of a vertex in undirected graph is the number of edges incident to that vertex.
  – Example: In \(G_1\), degree\((2) = 3\).

• In case \(G\) is a directed graph, we define
  – The **in-degree** of a vertex \(v\) is the number of edges for which \(v\) is the head.
  – The **out-degree** is the number of edges for which \(v\) is the tail.
  – Example: In \(G_3\), in-degree\((3) = 1\); out-degree\((3) = 0\).
Graphs

A **subgraph** of a graph $G$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. 

![Graphs Diagram](image)
Graphs

• A path from vertex \( v_p \) to vertex \( v_q \) (\( v_p \rightarrow v_q \)) in graph \( G \) is a sequence of vertices \( v_p, v_{i_1}, v_{i_2}, ..., v_{i_n}, v_q \) such that \( (v_p,v_{i_1}), (v_{i_1},v_{i_2}), ..., (v_{i_n},v_q) \) are edges

• The length of a path is the number of edges on it

• A simple path is a path in which
  – All vertices except possibly the first and last are distinct

\[(1,2,3,4) – \text{simple path} \]
\[(1,4,2,3,4,5) – \text{not simple} \]
Graphs

• A **cycle** is a path in which
  – The first and last vertices are the same
• A **simple cycle** – same with simple path
• When the graph is **directed**, we add the prefix "directed" to the terms
Graphs

• In an undirected graph, $G$, two vertices $v_1$ and $v_2$ are said to be **connected**
  – If there is a path in $G$ from $v_1$ to $v_2$.
  – Since $G$ is undirected, this means there must also be a path from $v_2$ to $v_1$.

• An **undirected graph** is said to be **connected**
  – If for every pair of distinct vertices $v_i$, $v_j$ in there is a path from $v_i$ to $v_j$ in $G$
Graphs

• A (connected) component of an undirected graph is a maximal connected subgraph

• A tree is a connected acyclic undirected graph

Two connected components
A forest is an acyclic undirected graph (not have to be connected)
Graphs

- Simple cycle
- Nonsimple cycle
- Free Tree
- Forest
- DAG
Graph Representation

- Two common ways to represent a graph (either directed or undirected):
  - Adjacency lists.
  - Adjacency matrix.
Adjacency Lists

- Array Adj of $|V|$ lists,
  - One per vertex.
  - The list Adj[u] contains all vertices v such that $(u, v) \in E$.
  - Works for both directed and undirected graphs.

- **Space**: $\Theta(V + E)$.
  - When expressing space/running time, we’ll drop the cardinality.
- **Time to list** all vertices adjacent to u: $\Theta(\text{degree}(u))$.
- **Time to determine** if $(u, v) \in E$: $O(\text{degree}(u))$. 
Adjacency Matrix

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

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

• Space: $\Theta(V^2)$.
• Time to list all vertices adjacent to $u$: $\Theta(V)$.
• Time to determine if $(u, v) \in E$: $O(1)$. 
Breadth-First Search

• Given a graph $G = (V, E)$ and a vertex $s \in V$
  – Which vertices are reachable from $s$
  – What is the shortest distance to each reachable vertex
  – What are the shortest paths
Breadth-First Search

- Algorithm Idea:
  - Send a wave out from s:
    - First hits all vertices 1 edge from s.
    - From there, hits all vertices 2 edges from s.
    - Etc.
  - Use FIFO queue Q to maintain wave front.
    - $v \in Q$ if and only if wave has hit v but has not come out of v yet.
Breadth-First Search

• **Input:**
  – **Graph** $G = (V, E)$, either directed or undirected.
  – **Source vertex** $s \in V$.

• **Output:**
  – $v.d =$ **distance** to all $v \in V$ from $s$.
    - If $v \in V$ is not reachable from $s$, $v.d = \infty$.
  – $v.\pi$ is the predecessor $u$ of $v$ on **shortest** path $s \rightsquigarrow v$.
    - If $v \in V$ is not reachable from $s$, $v.\pi$ will be **null**.
    - The set of edges $\{(v.\pi, v) : v \neq s\}$ forms a **tree**, such that $u$ is $v$’s **predecessor**.

• **Auxiliary Means:**
  – every vertex has a **color**:
    - **White** - **undiscovered**
    - **Gray** - **discovered, but not finished** (not done exploring from it)
    - **Black** - **finished** (have found everything reachable from it)
BFS($G, s$)

1. for each vertex $u \in G.V - \{s\}$
2. $u.color \leftarrow$ WHITE
3. $u.d \leftarrow \infty$
4. $u.\pi \leftarrow$ NULL
5. $s.color \leftarrow$ GRAY
6. $s.d \leftarrow 0$
7. $s.\pi \leftarrow$ NULL
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)

10. while $Q \neq \emptyset$
11. $u \leftarrow$ DEQUEUE($Q$)
12. for each $v \in G.Adj[u]$
13. if $v.color =$ WHITE
14. $v.color \leftarrow$ GRAY
15. $v.d \leftarrow u.d + 1$
16. $v.\pi \leftarrow u$
17. ENQUEUE($Q, v$)
18. $u.color \leftarrow$ BLACK

Complexity: $O(V + E)$
• The **shortest-path distance** from \( s \) to \( v \), \( \delta(s, v) \), is:
  – The minimum length of a path from \( s \) to \( v \), or \(-\infty\) if there is no path from \( s \) to \( v \).

• A **path** of length \( \delta(s, v) \) from \( s \) to \( v \) is said to be a **shortest path** from \( s \) to \( v \).
Breadth-First Search

**Theorem:**

- Suppose that BFS is run on graph $G = (V, E)$ from a source $s \in V$, then
  
- It discovers every vertex $v \in V$ that is reachable from $s$.

- Upon termination, $v.d = \delta(s, v)$ for all $v \in V$.

- For any vertex $v \neq s$ that is reachable from $s$,
  
  - **One of the shortest paths** from $s$ to $v$ is a shortest path from $s$ to $v.\pi$ followed by the edge $(v.\pi, v)$.
  
  - Hence, the path $(s, (v.\pi. \ldots).\pi, \ldots, (v.\pi).\pi, v.\pi, v)$ is one of the shortest paths from $s$ to $v$. 


Breadth-First Search

• Proof:
  – We will use the following two facts:
    • **Fact 1:** Upon termination, for each \( v \in V \), \( v.d \geq \delta(s, v) \).
    • **Fact 2:** If vertex \( v_i \) is enqueued before \( v_j \) during the execution then \( v_i.d \leq v_j.d \).
  – Assume, for the purpose of contradiction, that there is \( v \in V \) such that upon termination \( v.d \neq \delta(s, v) \).
  – If there are more then one, take \( v \) with minimum \( \delta(s, v) \).
  – Clearly \( v \neq s \).
  – By fact 1, \( v.d \geq \delta(s, v) \), and thus, from the assumption, \( v.d > \delta(s, v) \).
  – Vertex \( v \) must be reachable from \( s \), for if it is not, then \( \delta(s, v) = \infty \geq v.d \).
  – Let \( u \) be the vertex immediately preceding \( v \) on a shortest path from \( s \) to \( v \), so that \( \delta(s, v) = \delta(s, u) + 1 \).
  – Since \( \delta(s, u) < \delta(s, v) \), and because of how we chose \( v \), we have \( u.d = \delta(s, u) \).
  – Putting these together, we have \( v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1 \).
Breadth-First Search

• Consider the time when vertex u is dequeued from Q.
• At this time, vertex v is either white, gray, or black.
  – If v is white, then BFS sets v.d = u.d + 1 = δ(s, v), contradiction.
  – If v is black, then it was already removed from the queue and, by fact 2, we have v.d ≤ u.d < δ(s, v), contradiction.
  – If v is gray, then it was painted gray upon dequeuing some vertex w, which was removed from Q earlier than u and for which v.d = w.d + 1.
  – By fact2, however, w.d ≤ u.d, and so we have v.d ≤ u.d + 1 = δ(s, v), contradiction.
• Thus we conclude that v.d = δ(s, v) for all v ∈ V.
• All vertices reachable from s must be discovered, for if they were not, they would have infinite d values.
• If v.π = u, then v.d = u.d + 1. Thus, we can obtain a shortest path from s to v by taking a shortest path from s to v.π and then traversing the edge (v.π, v).
• By induction, the path (s, (v.π ...).π), ..., ((v.π) .π, v.π), (v.π, v) is one of the shortest paths from s to v.
Printing The Shortest Path

1. PrintPath \((G, s, v)\)
2. \(\text{if } (v = s) \text{ then}\)
3. \(\text{print } s\)
4. \(\text{else if } (v.\pi = \text{null}) \text{ then}\)
5. \(\text{print "no path from" } s \text{ "to" } v \text{ "exists"}\)
6. \(\text{else}\)
7. \(\text{printPath } (G, s, v.\pi)\)
8. \(\text{print } v\)
Depth-First Search

• Given a graph $G = (V, E)$
  – Draw $G$ as a **forest of sub graphs**
  – Determine which vertex is a **descendant** of another in the forest
  – Detect **cycles**
  – Discover **connected components**
  – **Topological Sort**
Depth-First Search

• Algorithm Idea:
  – Search deeper in the graph whenever possible:
    • Start from an arbitrary unvisited vertex.
    • Explore out one of the undiscovered edge of the most recently discovered vertex \( v \).
    • When all of \( v \)'s edges have been explored, backtracks, and continue.
    • Start all over again.
Depth-First Search

• Input:
  – Graph $G = (V, E)$, either directed or undirected.

• Output:
  – $v.d =$ discovery time & $v.f =$ finishing time.
    • A unique integer from 1 to $2|V|$ such that $1 \leq v.d < v.f \leq 2|V|$.
  – $v.\pi = u$ such that $u$ is the predecessor to $v$ in the visit order.
    • The predecessor subgraph $G_\pi = (V, E_\pi)$, where $E_\pi=\{(v.\pi,v) : v.\pi \neq \text{null}\}$ forms a forest composed of several trees.

• Auxiliary Means:
  – every vertex has a color:
    • White - undiscovered
    • Gray - discovered, but not finished (not done exploring from it)
    • Black - finished (have found everything reachable from it)
**DFS(G)**

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

**DFS-VISIT(G,u)**

1. \( u.color \leftarrow \text{GRAY} \)
2. \( time \leftarrow time+1 \)
3. \( u.d \leftarrow time \)
4. for each \( v \in G.\text{Adj}[u] \) // explore edge \((u, v)\)
5. if \( v.color = \text{WHITE} \)
6. \( v.\pi \leftarrow u \)
7. DFS-VISIT(\( G, v \)\
8. \( u.color \leftarrow \text{BLACK} \)
   // blacken \( u \); it is finished.
9. \( time \leftarrow time+1 \)
10. \( u.f \leftarrow time \)
Depth-First Search

• When one vertex is a descendant of another in the forest that was constructed by DFS?
  – Parenthesis Theorem
  – White-path Theorem
Parenthesis Theorem

- **Theorem** (Parenthesis theorem):
  - For all \( u, v \), exactly one of the following holds:
    1. \( u.d < u.f < v.d < v.f \) or \( v.d < v.f < u.d < u.f \) and neither of \( u \) and \( v \) is a descendant of the other.
    2. \( u.d < v.d < v.f < u.f \) and \( v \) is a descendant of \( u \).
    3. \( v.d < u.d < u.f < v.f \) and \( u \) is a descendant of \( v \).
  - So \( u.d < v.d < u.f < v.f \) cannot happen.

- Like parentheses:
  - OK: ( ) [ ] ( [ ] ) [ ( ) ]
  - Not OK: ( [ ] ) [ ( ) ]

- **Corollary** (Nesting of descendants’ intervals):
  - \( v \) is a proper descendant of \( u \) if and only if \( u.d < v.d < v.f < u.f \).
White-path Theorem

• Theorem (White-path theorem):
  – v is a descendant of u if and only if at time u.d, there is a path $u \leadsto v$ consisting of only white vertices (except for u, which was just colored gray)
Classification of Edges

- **Tree edge:**
  - In the constructed forest.
  - Found by exploring \((u, v)\).

- **Back edge:**
  - \((u, v)\), where \(u\) is a descendant of \(v\).

- **Forward edge:**
  - \((u, v)\), where \(v\) is a descendant of \(u\), but not a tree edge.

- **Cross edge:**
  - any other edge.
  - Can go between vertices in same tree or in different trees.

- In an undirected graph, there may be some ambiguity since \((u, v)\) & \((v, u)\) are the same edge.
  - Classify by the first type above that matches

- **Theorem:**
  - In DFS of an undirected graph, we get only tree and back edges. No forward or cross edges.
Classification of Edges

• Edge \((u, v)\) can be classified by the color of \(v\) when the edge is first explored:
  
  – WHITE - indicates a **tree edge**
  
  – GRAY - indicates a **back edge**
  
  – BLACK indicates a **forward or cross edge**.

  • \((u, v)\) is a **forward edge** if \(u.d < v.d\)
  
  • \((u, v)\) is a **cross edge** if \(u.d > v.d\).
Detection of Cycles

• Lemma:
  – A directed graph G is acyclic if and only if a DFS of G yields no back edges.

• Proof:
  – Back edge $\Rightarrow$ Cycle
    • Suppose there is a back edge $(u, v)$
    • Then $v$ is ancestor of $u$ in the constructed forest
    • Therefore, there is a path $v \leadsto u$, so $v \leadsto u \Rightarrow v$ is a cycle
Detection of Cycles

• Cycle \(\leftrightarrow\) back edge.
  
  \(-\) Suppose \(G\) contains cycle \(C\).
  
  \(-\) Let \(v\) be the first vertex discovered in \(C\), and let \((u, v)\) be the preceding edge in \(C\).
  
  \(-\) At time \(v.d\), vertices of \(C\) form a white path \(v \rightarrow u\)
    
    \(\bullet\) since \(v\) is the first vertex discovered in \(C\).
  
  \(-\) By white-path theorem, \(u\) is descendant of \(v\) in depth-first forest.
  
  \(-\) Therefore, \((u, v)\) is a back edge.

• Similar for undirected graph.
Topological Sort

• **Directed acyclic graph (DAG)** is a directed graph with no cycles

• Good for modeling a **partial order**:
  – $a > b$ and $b > c \Rightarrow a > c$.
  – May have $a$ and $b$ such that neither $a > b$ nor $b > c$.

• **Topological sort of a DAG**: a linear ordering of vertices such that if $(u, v) \in E$, then $u$ appears somewhere before $v$. 
Topological Sort

topologicalSort \((G)\)  \hspace{2em} // Assume G is a DAG

Call DFS\((G)\) to compute finishing times \(v.f\) for all \(v \in V\) as each vertex is finished, insert it onto the front of a linked list

return the linked list of vertices

Complexity: \(\Theta(V + E)\)
Topological Sort

• **Correctness:**
  – Just need to show if \((u, v) \in E\), then \(v.f < u.f\).

• When we explore \((u, v)\), what are the colors of \(u\) and \(v\)?
  – \(u\) is gray.
  – \(v\) can’t be gray.
    • Because then \((u, v)\) is a back edge, but \(G\) is a dag.
  – If \(v\) is white.
    • By parenthesis theorem, \(u.d < v.d < v.f < u.f\).
  – If \(v\) is black.
    • Then \(v\) is already finished, but \(u\) doesn't, therefore, \(v.f < u.f\).