Announcement: Homework 1 released! Due: January 24th at 11:59PM (Gradescope)

Recap: Asymptotic Analysis

Five Representative Problems
- Algorithmic Techniques: Greedy, Dynamic Programming, Network Flow...
- Computationally Intractable Problems: Unlikely that polynomial time algorithm exists.

Formal Definition of Big O, Ω, Θ notation
- \( T(n) \in O(f(n)) \) ---- \( f(n) \) upper bounds \( T(n) \)
  - Means we can find constants \( c, N > 0 \) s.t. whenever \( n > N \)
    \( T(n) < \frac{1}{c} f(n) \)
  - Intuition: \( f(n) \) is an upper bound for large enough inputs
- \( T(n) \in \Omega(f(n)) \) ---- \( f(n) \) lower bounds \( T(n) \)
- \( T(n) \in \Theta(f(n)) \) ---- lower bound and upper bound

Polynomial Time function. \( T(n) \in O(n^d) \) for some constant \( d \) (\( d \) is independent of the input size).

2.4 A Survey of Common Running Times

Linear: \( O(n) \)
- Max/Min
- Merge Sorted Lists

Quasilinear: \( O(n \log n) \)
- Sorting
- Many algorithms that use sorting as subroutine

Quadratic: \( O(n^2) \)
- Naive Algorithm to Find Closest Pair of points in Euclidean Space

Linear Time: \( O(n) \)

Merge. Combine two sorted lists \( A = a_1, a_2, \ldots, a_n \) with \( B = b_1, b_2, \ldots, b_n \) into sorted whole.

```
Claim. Merging two lists of size \( n \) takes \( O(n) \) time.
Proof. After each comparison, the length of output list increases by 1.
```

Linear Time: \( O(n \log n) \) Time

\( O(n \log n) \) time. Arises in divide-and-conquer algorithms. Also referred to as linearithmic time.

Sorting. Mergesort and heapsort are sorting algorithms that perform \( O(n \log n) \) comparisons.

Largest empty interval. Given \( n \) time-stamps \( x_1, \ldots, x_n \) on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

\( O(n \log n) \) solution. Sort the \( n \) time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.
Quadratic Time: $O(n^2)$

Quadratic time. Enumerate all pairs of elements.

Closest pair of points. Given a list of $n$ points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest.

$O(n^2)$ solution. Try all pairs of points.

\[
\min_{i=1}^{n} \min_{j=i+1}^{n} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

Remark. $O(n^2)$ seems inevitable, but this is just an illusion.

Cubic Time: $O(n^3)$

Cubic time. Enumerate all triples of elements.

Set disjointness. Given $n$ sets $S_1, \ldots, S_n$ each of which is a subset of $1, 2, \ldots, n$, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pair of sets, determine if they are disjoint.

\[
\text{foreach set } S_i \{
\text{foreach other set } S_j \{
\text{foreach element } p \text{ of } S_i \{
\text{determine whether } p \text{ also belongs to } S_j \\
\text{if no element of } S_i \text{ belongs to } S_j \\
\text{report that } S_i \text{ and } S_j \text{ are disjoint}
\}
\}
\}
\]

Polynomial Time: $O(n^k)$ Time

Independent set of size $k$. Given a graph, are there $k$ nodes such that no two are joined by an edge?

$k$ is a constant

$O(n^k)$ solution. Enumerate all subsets of $k$ nodes.

\[
\text{foreach subset } S \text{ of } k \text{ nodes} \{
\text{check whether } S \text{ is an independent set} \\
\text{if } S \text{ is an independent set} \{
\text{report } S \text{ is an independent set}
\}
\}
\]

\[
\text{Number of } k \text{ element subsets } = \binom{n}{k} = \frac{n!}{k!(n-k)!} \\
\text{is polynomial for } k=17, \text{ but not practicable}
\]

Exponential Time

Independent set. Given a graph, what is the maximum size of an independent set?

$O(2^n)$ solution. Enumerate all subsets.

\[
S^* \leftarrow \phi \\
\text{foreach subset } S \text{ of nodes} \{
\text{check whether } S \text{ is an independent set} \\
\text{if } S \text{ is largest independent set seen so far} \{
\text{update } S^* \leftarrow S
\}
\}
\]

3.1 Basic Definitions and Applications
Undirected Graphs

Undirected graph, $G = (V, E)$
- $V = \text{nodes}$
- $E = \text{edges between pairs of nodes}$
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|, m = |E|$.

Example:

$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$
$E = \{1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6\}$
$n = 8$
$m = 11$

Some Graph Applications

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transportation</td>
<td>Street intersections, highways</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>Computers, fiber optic cables</td>
<td></td>
</tr>
<tr>
<td>World Wide Web</td>
<td>Web pages, hyperlinks</td>
<td></td>
</tr>
<tr>
<td>Social</td>
<td>People, relationships</td>
<td></td>
</tr>
<tr>
<td>Food Web</td>
<td>Species, predator-prey</td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Systems, functions</td>
<td></td>
</tr>
<tr>
<td>Graphs</td>
<td>Functions, function calls</td>
<td></td>
</tr>
<tr>
<td>Circuit</td>
<td>Gates, precedence constraints</td>
<td></td>
</tr>
</tbody>
</table>

World Wide Web

Web graph.
- Node: web page.
- Edge: hyperlink from one page to another.

Ecological Food Web

Food web graph.
- Node: species.
- Edge: from prey to predator.

9-11 Terrorist Network

Social network graph.
- Node: people.
- Edge: relationship between two people.

Graph Representation: Adjacency Matrix

Adjacency matrix: $n \times n$ matrix with $A_{uv} = 1$ if $(u, v)$ is an edge.
- Two representations of each edge.
- Space proportional to $n^2$.
- Checking if $(u, v)$ is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.
Graph Representation: Adjacency List

- **Adjacency list** is a node-indexed array of lists.
- Two representations of each edge.
- Space proportional to $m + n$.
- Checking if $(u, v)$ is an edge takes $O(\text{deg}(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.

Paths and Connectivity

- **Definition**: A path in an undirected graph $G = (V, E)$ is a sequence $P$ of nodes $v_1, v_2, ..., v_k$ with the property that each consecutive pair $v_i, v_{i+1}$ is joined by an edge in $E$.
- **Definition**: A path is simple if all nodes are distinct.
- **Definition**: An undirected graph is connected if for every pair of nodes $u$ and $v$, there is a path between $u$ and $v$.

Cycles

- **Definition**: A cycle is a path $v_1, v_2, ..., v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k-1$ nodes are all distinct.

Trees

- **Definition**: An undirected graph is a tree if it is connected and does not contain a cycle.

- **Theorem**: Let $G$ be an undirected graph on $n$ nodes. Any two of the following statements imply the third:
  - $G$ is connected.
  - $G$ does not contain a cycle.
  - $G$ has $n-1$ edges.

Rooted Trees

- **Definition**: A rooted tree is a tree with a designated root node $r$.

- **Importance**: Models hierarchical structure.

Phylogeny Trees

- **Definition**: Phylogeny trees describe evolutionary history of species.
**Binary Tree**

Def. A rooted tree in which each node has at most 2 children.

Def. Height of a tree is the number of edges in the longest path from root to leaf.

Thm. Number of nodes in binary tree of height $h$ is $n \leq 2^{h+1} - 1 = 2^h + 2^{h-1} + \ldots + 2^0$.

Balanced Binary Tree. Height $h = O(\log n)$.

**GUI Containment Hierarchy**

GUI containment hierarchy. Describe organization of GUI widgets.


**3.2 Graph Traversal**

**Connectivity**

s-t connectivity problem. Given two node s and t, is there a path between s and t?

s-t shortest path problem. Given two node s and t, what is the length of the shortest path between s and t?

Applications.
- Navigation (Google Maps).
- Maze traversal.
- Kevin Bacon number (or Erdős Number).
- Fewest number of hops in a communication network.

**Breadth First Search**

BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.

BFS algorithm.
- $L_0 = \{s\}$.
- $L_i$ = all neighbors of $L_{i-1}$.
- $L_i$ = all nodes that do not belong to $L_0$ or $L_{i-1}$ and that have an edge to a node in $L_{i-1}$.
- $L_i$ = all nodes that do not belong to an earlier layer, and that have an edge to a node in $L_i$.

Theorem. For each i, $L_i$ consists of all nodes at distance exactly i from s. There is a path from s to t iff t appears in some layer.
Breadth First Search

**Property.** Let $T$ be a BFS tree of $G = (V, E)$, and let $(x, y)$ be an edge of $G$. Then the level of $x$ and $y$ differ by at most 1.

---

Breadth First Search: Analysis

**Theorem.** The above implementation of BFS runs in $O(m + n)$ time if the graph is given by its adjacency representation.

**Pf.**
- Easy to prove $O(n^2)$ running time:
  - at most $n$ lists $L[i]$
  - each node occurs on at most one list; for loop runs $\leq n$ times
  - when we consider node $u$, there are $\leq n$ incident edges $(u, v)$, and we spend $O(1)$ processing each edge
- Actually runs in $O(m + n)$ time:
  - when we consider node $u$, there are $\deg(u)$ incident edges $(u, v)$
  - total time processing edges is $\sum_{u \in V} \deg(u) \leq 2m$

---

Connected Component

**Connected component.** Find all nodes reachable from $s$.

Connected component containing node $1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

---

Connected Component

**Theorem.** Upon termination, $R$ is the connected component containing $s$.

- BFS = explore in order of distance from $s$.
- DFS = explore in a different way.

---

Flood Fill

**Flood fill.** Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

recolor lime green blob to blue
3.4 Testing Bipartiteness

Testing Bipartiteness

Testing bipartiteness. Given a graph $G$, is it bipartite?
- Many graph problems become:
  - Easier if the underlying graph is bipartite (matching)
  - Tractable if the underlying graph is bipartite (independent set)
Before attempting to design an algorithm, we need to understand structure of bipartite graphs.

A bipartite graph $G$ another drawing of $G$

An Obstruction to Bipartiteness

Lemma. If a graph $G$ is bipartite, it cannot contain an odd length cycle.

Pf. Not possible to 2-color the odd cycle, let alone $G$.

bipartite (2-colorable) not bipartite (not 2-colorable)

Bipartite Graphs

Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.
(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.
(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

Pf. (i)
- Suppose no edge $(x,y)$ joins two nodes in same layer $L_i$.
- By previous lemma, this implies all edges $(x,y)$ join nodes in adjacent layers (i.e., $x \in L_i$ and $y \in L_{i+1}$).
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.

Case (i)

Pf. (ii)
- Suppose an edge $(x,y)$ joins two nodes in same layer $L_i$.
- By previous lemma, this implies all edges $(x,y)$ join nodes in adjacent layers (i.e., $x \in L_i$ and $y \in L_{i+1}$).
- Bipartition: red = nodes on odd levels, blue = nodes on even levels.

Case (ii)
Bipartite Graphs

Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

Pf. (ii)

\begin{itemize}
  \item Suppose $(x, y)$ is an edge with $x, y$ in same level $L_j$.
  \item Let $z = lca(x, y)$ = lowest common ancestor.
  \item Let $L_i$ be level containing $z$.
  \item Consider cycle that takes edge from $x$ to $y$, then path from $y$ to $z$, then path from $z$ to $x$.
  \item Its length is $1 + (j-i) + (j-i)$, which is odd.
\end{itemize}

Obstruction to Bipartiteness

Corollary. A graph $G$ is bipartite iff it contain no odd length cycle.

Directed Graphs

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

- Edge $(u, v)$ goes from node $u$ to node $v$.

Directed s-t shortest path problem. Given two node $s$ and $t$, what is the length of the shortest path between $s$ and $t$?

Graph search. BFS extends naturally to directed graphs.

Directed reachability. Given a node $s$, find all nodes reachable from $s$.

Web crawler. Start from web page $s$. Find all web pages linked from $s$, either directly or indirectly.

Strong Connectivity

Def. Node $u$ and $v$ are mutually reachable if there is a path from $u$ to $v$ and also a path from $v$ to $u$.

Def. A graph is strongly connected if every pair of nodes is mutually reachable.

Lemma. Let $s$ be any node. $G$ is strongly connected iff every node is reachable from $s$, and $s$ is reachable from every node.

Pf. \( \Rightarrow \) Follows from definition.

Pf. \( \Leftarrow \) Path from $u$ to $v$: concatenate $u$-s path with s-v path.
Path from $v$ to $u$: concatenate v-s path with s-u path.

\[ \text{Path if paths overlap} \]
Strong Connectivity: Algorithm

**Theorem.** Can determine if \( G \) is strongly connected in \( O(m + n) \) time.

**Pf.**

1. Pick any node \( s \).
2. Run BFS from \( s \) in \( G \).
3. Run BFS from \( s \) in \( G^{rev} \).
4. Return true iff all nodes reached in both BFS executions.
5. Correctness follows immediately from previous lemma.

---

**Def.** An **DAG** is a directed graph that contains no directed cycles.

**Ex.** Precedence constraints: edge \((v_i, v_j)\) means \( v_i \) must precede \( v_j \).

**Def.** A **topological order** of a directed graph \( G = (V, E) \) is an ordering of its nodes as \( v_1, v_2, \ldots, v_n \) so that for every edge \((v_i, v_j)\) we have \( i < j \).

---

**Directed Acyclic Graphs**

**Lemma.** If \( G \) has a topological order, then \( G \) is a DAG.

**Pf.** (by contradiction)

1. Suppose that \( G \) has a topological order \( v_1, \ldots, v_n \) and that \( G \) also has a directed cycle \( C \). Let’s see what happens.
2. Let \( v_i \) be the lowest-indexed node in \( C \), and let \( v_j \) be the node just before \( v_i \), thus \((v_j, v_i)\) is an edge.
3. By our choice of \( i \), we have \( i < j \).
4. On the other hand, since \((v_j, v_i)\) is an edge and \( v_1, \ldots, v_i \) is a topological order, we must have \( j < i \), a contradiction.

---

**Directed Acyclic Graphs**

**Lemma.** If \( G \) has a topological order, then \( G \) is a DAG.

**Q.** Does every DAG have a topological ordering?

**Q.** If so, how do we compute one?
Directed Acyclic Graphs

**Lemma.** If \( G \) is a DAG, then \( G \) has a node with no incoming edges.

**Pf.** (by contradiction)
- Suppose that \( G \) is a DAG and every node has at least one incoming edge. Let’s see what happens.
- Pick any node \( v \), and begin following edges backward from \( v \). Since \( v \) has at least one incoming edge \((u, v)\) we can walk backward to \( u \).
- Then, since \( u \) has at least one incoming edge \((x, u)\), we can walk backward to \( x \).
- Repeat until we visit a node, say \( w \), twice.
- Let \( C \) denote the sequence of nodes encountered between successive visits to \( w \). \( C \) is a cycle. □

Directed Acyclic Graphs

**Lemma.** If \( G \) is a DAG, then \( G \) has a topological ordering.

**Pf.** (by induction on \( n \))
- **Base case:** true if \( n = 1 \).
- **Given DAG on \( n > 1 \) nodes, find a node \( v \) with no incoming edges.
- \( G - \{ v \} \) is a DAG, since deleting \( v \) cannot create cycles.
- By inductive hypothesis, \( G - \{ v \} \) has a topological ordering.
- Place \( v \) first in topological ordering; then append nodes of \( G - \{ v \} \) in topological order. This is valid since \( v \) has no incoming edges. □

Topological Sorting Algorithm: Running Time

**Theorem.** Algorithm finds a topological order in \( O(m + n) \) time.

**Pf.** Maintain the following information:
- \( \text{count}[w] \) = remaining number of incoming edges
- \( S \) = set of remaining nodes with no incoming edges
- **Initialization:** \( O(m + n) \) via single scan through graph.
- **Update:** to delete \( v \)
  - remove \( v \) from \( S \)
  - decrement \( \text{count}[w] \) for all edges from \( v \) to \( w \), and
  - add \( w \) to \( S \) if \( \text{count}[w] \) hits 0
- this is \( O(1) \) per edge □