**(CS1.301) Advanced Algorithm Design (Monsoon 2022)**

# Practice Problem Set

## Basic Graph Algorithms

1. A graph $(V, E)$ is bipartite if the vertices $V$ can be partitioned into two subsets $L$ and $R$, such that every edge has one vertex in $L$ and the other in $R$.

   (a) Prove that every tree is a bipartite graph.

   (b) Prove that a graph $G$ is bipartite if and only if every cycle in $G$ has an even number of edges.

   (c) Describe and analyze an efficient algorithm that determines whether a given undirected graph is bipartite.

2. Recall that a directed graph $G$ is strongly connected if, for any two vertices $u$ and $v$, there is a path in $G$ from $u$ to $v$ and a path in $G$ from $v$ to $u$.

   Describe an algorithm to determine, given an undirected graph $G$ as input, whether it is possible to direct each edge of $G$ so that the resulting directed graph is strongly connected.

3. A directed graph $G = (V, E)$ is strongly connected if and only if every pair of vertices is strongly connected. Equivalently, a strong component of $G$ is a maximal strongly connected subgraph of $G$. A directed graph $G$ is strongly connected if and only if $G$ has exactly one strong component; at the other extreme, $G$ is a DAG if and only if every strong component of $G$ consists of a single vertex.

   Give an algorithm that computes all strong components of a graph in time at most $O(V \cdot E)$. Bonus: Can this be improved to $O(V + E)$?

## Greedy Algorithms

1. In a city there are $N$ houses, each of which is in need of a water supply. It costs $w_i$ dollars to build a well at house $i$, and it costs $c_{i,j}$ to build a pipe in between houses $i$ and $j$. A house can receive water if either there is a well built there or there is some path of pipes to a house with a well. Design an algorithm to find the minimum amount of money needed to supply every house with water.

2. A spanning tree $T$ of a undirected (positively) weighted graph $G$ is called a minimum bottleneck spanning tree (MBST) if the edge with the maximum cost is minimum among all possible spanning trees. Show that a MST is always a MBST. What about the converse?

3. Consider an undirected (positively) weighted graph $G = (V, E)$ with a MST $T$ and a shortest path $\pi(s, t)$ between two vertices $s, t \in V$. Will $T$ still be an MST and $\pi(s, t)$ be a shortest path if

   (a) Weight of each edge is multiplied by a fixed constant $c > 0$.

   (b) Weight of each edge is incremented by a fixed constant $c > 0$.

4. A path is monotonic if the weight of every edge on the path is either strictly increasing or strictly decreasing. Given an edge-weighted directed graph, find a monotonic shortest path from $s$ to every other vertex.

5. Suppose we are given a directed graph $G$ with weighted edges and two vertices $s$ and $t$. Describe and analyze an algorithm to find the shortest path from $s$ to $t$ when exactly one edge in $G$ has negative weight. [Hint: Modify Djikstra's algorithm?]

6. Suppose you are given a directed graph $G$ in which every edge has negative weight, and a source vertex $s$. Describe and analyze an efficient algorithm that computes the shortest-path distances from $s$ to every other vertex in $G$. Specifically, for every vertex $t$:

   - If $t$ is not reachable from $s$, your algorithm should report $\text{dist}(t) = \infty$.
   - If $G$ has a cycle that is reachable from $s$,and $t$ is reachable from that cycle, then the shortest-path distance from $s$ to $t$ is not well-defined, because there are paths (formally, walks) from $s$ to $t$ of arbitrarily large negative length. In this case, your algorithm should report $\text{dist}(t) = -\infty$.
   - If neither of the two previous conditions applies, your algorithm should report the correct shortest-path distance from $s$ to $t$.

7. Let $F$ be a spanning tree of $G$, then $F$ is a minimum spanning tree of $G$ if and only if every edge $e = (u, v)$ in $E \setminus F$ is $F$-heavy, that is, weight of the edge $e = (u, v)$ is greater than the weight of the $u \rightsquigarrow v$ path in $F$.

## Divide and Conquer

1. Describe an algorithm to compute the median of an array $A[1..5]$ of distinct numbers using at most 6 comparisons.

2. Consider the following algorithm given an arrary as input. Divide the input array into $\lceil n/5 \rceil$ blocks, each containing exactly 5 elements, except possibly the last. (If the last block isn't full, just throw in a few 1s.) Then compute the median of each block by brute force, collect those medians into a new array $M[1..\lceil n/5 \rceil]$, and then recursively compute the median of this new array. Show that this algorithm computes an element that is at least 3/10th largest and at most 7/10th largest.

3. Describe and analyze a variant of Karatsuba's algorithm that multiplies any $m$-digit number and any $n$-digit number, for any $n \geqslant m$, in $O(nm^{\log_2 3-1})$ time.

## Recursion and Dynamic Programming

1. The greatest common divisor of two positive integer $x$ and $y$, denoted $\text{gcd}(x, y)$, is the largest integer $d$ such that both $x/d$ and $y/d$ are integers. Euclid describes the following recursive approach.

   - If $x = y$ then return $x$.
   - Else if $x > y$, compute $\text{gcd}(x - y, y)$.
   - Else, compute $\text{gcd}(x, y - x)$.

Prove that the above procedure correctly computes $\gcd(x, y)$. In particular, prove that $\gcd(x, y)$ divides both $x$ and $y$. Prove that every divisor of $x$ and $y$ is a divisor of $\gcd(x, y)$.

2. Suppose you are given an array $A[1..n]$ of numbers, which may be positive, negative, or zero, and which are not necessarily integers.

   (a) Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray $A[i..j]$.

   (b) Describe and analyze an algorithm that finds the largest product of elements in a contiguous subarray $A[i..j]$.

3. Edit distance between two strings is the minimum number of letter insertions, letter deletions, and letter substitutions required to transform one string into the other. For example, the edit distance between FOOD and MONEY is at most four, and distance between the strings ALGORITHM and ALTRUISTIC is at most 6.

   For any two input strings $A[1..m]$ and $B[1..n]$, we can formulate the edit distance problem recursively as follows: For any indices $i$ and $j$, let $\mathtt{Edit}(i, j)$ denote the edit distance between the prefixes $A[1..i]$ and $B[1..j]$. We need to compute $\mathtt{Edit}(m, n)$. Please complete the arguments here on how $\mathtt{Edit}(i, j)$ can be computed using memoization.

4. An independent set in a graph is a subset of the vertices with no edges between them. Finding the largest independent set in an arbitrary graph is extremely hard; in fact, this is one of the canonical NP-hard problems. But in some special classes of graphs, we can find largest independent sets quickly. In particular, when the input graph is a tree with $n$ vertices, we can actually compute the largest independent set in $O(n)$ time. Please describe the algorithm for the same.

5. A string $w$ of parentheses ( and ) and brackets [ and ] is *balanced* if it satisfies one of the following conditions:

   - $w$ is the empty string.
   - $w = (x)$ for some balanced string $x$.
   - $w = [x]$ for some balanced string $x$
   - $w = xy$ for some balanced strings $x$ and $y$.

   Describe and analyze an algorithm to determine whether a given string of parentheses and brackets is balanced.

## Network Flows[1]

1. A given flow network $G$ may have more than one minimum $(s, t)$-cut. Let us define the best minimum $(s, t)$-cut to be any minimum cut $(S, T)$ with the smallest number of edges crossing from $S$ to $T$. Describe an algorithm to find the best minimum $(s, t)$-cut when the capacities are integers.

2. Suppose you are given a flow network $G$ with integer edge capacities and an integer maximum flow $f^*$ in $G$. Describe algorithms for the following operations:

[1]Please treat Assignment 4 questions also as practice questions.

- Increment($e$): Increase the capacity of edge $e$ by 1 and update the maximum flow.
- Decrement($e$): Decrease the capacity of edge $e$ by 1 and update the maximum flow.

Both algorithms should modify $f^*$ so that it is still a maximum flow, more quickly than recomputing a maximum flow from scratch.

## P **vs** NP

1. Show that Sudoku is in NP.
2. Show that checking if a given number is prime is in NP.