COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Problem Set 2

Module: Classic Algorithms

Below is a reminder of key aspects of the PSet:

- The only goal of this PSet is to help you develop your problem-solving skills in preparation for the exams. Your performance on this PSet will not directly contribute to your grade, but will indirectly improve your ability to do well on the exams.
- Because your performance does not directly impact your grade, you may use any resources you like (collaboration, AI, etc.) to help you complete the PSet.
- We suggest taking a serious stab at the PSet alone, to help self-evaluate where you're at. But, we also suggest collaborating with friends, visiting office hours, asking on Ed, and/or using AI tools to help when stuck. Even when able to complete the entire PSet on your own, you may still find any of these methods useful to discuss the PSet afterwards.
- Throughout the PSet, we've included some general tips to help put these into broader context. Exams will not have these, and future PSets may have fewer.

Problem 1 : Flow State

Consider an edge-weighted directed graph $G = (V, E)$ with two special vertices $s, t \in V$. The weight of an edge $(u, v) \in E$ is denoted $w(u, v)$, and represents the capacity of that edge. Assume that $|V| = n$ and $|E| = m$, weights are integers, and the graph is connected.

(a) The bottleneck capacity of a path P from s to t is the minimum weight of any edge in P, i.e., $\min_{(u,v)\in P} w(u, v)$.

Design an $O(m \log m)$ time algorithm that finds a path from s to t with maximum bottleneck capacity.

(b) Suppose that F is the value of the maximum flow in the network. Prove there is always a path with bottleneck capacity at least F/m.

(c) Consider the following modification of the Dinitz-Edmonds-Karp algorithm for computing maximum flows. Find an augmenting path from s to t with maximum bottleneck capacity (as in part (a)), augment flow along this path, and repeat until there are no more augmenting paths. Prove that this algorithm makes at most $O(m \log F)$ augmentations, where F is the capacity of the maximum flow in the network.

(Hint. Repeatedly apply part (b) to the residual graph.)

Problem Solving Tips

While solving this problem, you may derive an inequality that is related but not exactly what you want. This is common in problem-solving. Aim first to secure a valid bound, then simplify it using a standard inequality. A particularly common one (which may or may not be useful here) is $1 - x \leq e^{-x}$ for all x, which lets you convert multiplicative expressions into exponential ones.

In general, when you have an inequality that's close to your target, pause and ask whether a familiar bound can bridge the gap. If you are not very familiar with common inequalities (which the typical COS 330 student is not expected to be), you might want to look up a catalog of inequalities (or ask AI to help you) and see if it applies. Don't worry about memorizing a catalog: on exams we won't expect it, we'll provide any non-basic inequalities you might need (beyond very obvious ones like $x^2 \geq 0$).

Problem 2 : It Takes Three to Tango

One nice thing about the theory of fine-grained reductions is that there are many problems that can be used as a canonical hard problem to reduce from. We saw a couple in lecture, and here we will see another one and play with it a bit.

Definition 2.1 (3-SUM)**.** You are given an unsorted list of n integers, determine if there are 3 integers in the list that sum to zero.

(a) Consider the following variant of 3-SUM, which we call Colorful-3-SUM.

Definition 2.2 (Colorful-3-SUM)**.** You are given three unsorted lists of n integers A, B, and C. Determine if there are 3 indices $i, j, k \in [n]$ such that $A_i + B_j + C_k = 0$.

Find an $O(n)$ reduction from Colorful-3-SUM to 3-SUM, or in other words, prove that if 3-SUM can be solved in $T(n)$ time, then Colorful-3-SUM can be solved in $O(T(n))$ time.

(b) Consider the following graph problem, which we call Triangle-Detection.

Definition 2.3 (Triangle-Detection)**.** You are given a connected undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Determine if there are three vertices $u, v, w \in V$ such that $(u, v), (v, w), (w, u) \in E$.

Find an $O(n)$ reduction from Triangle-Detection to 3-SUM, or in other words, prove that if 3-SUM can be solved in $T(n)$ time, then Triangle-Detection can be solved in $O(T(m))$ time.

(Hint. You can use the result from part (a) to reduce to Colorful-3-SUM.)

Problem 3 : High Maintenance

A telecommunications company is deploying a network across a metropolitan area with N cell towers. Each tower must operate on one of two frequency bands: Low-band or High-band. The network topology forms a graph where edges connect towers that have overlapping coverage areas.

To maintain this network, the company needs to pay a cost to audit each pair of towers connected by an edge. The cost is B dollars for adjacent towers that use the same frequency band, and C dollars for adjacent towers use different bands, where $C \geq B$. Additionally, the company can pay A dollars to reconfigure any tower to switch frequency bands (this is A dollars per tower reconfigured).

Design an $O(VE^2)$ algorithm that determines the minimum cost necessary to audit this network. (Hint. You may find it helpful to first consider the case where $B = 0$.)

Extra Credit

Recall that extra credits are quite challenging. We do not suggest attempting the extra credit problems to practice for the exam, but only to engage deeper with the course material. If you are interested in pursuing an IW/thesis in CS theory, the extra credits will give you a taste of what that might be like. You are welcome to discuss the extra credit problems with your TA/UCA coach.

Problem : Reuse Reduce Recycle

In this problem you will be designing time and space efficient algorithms, so let's first look at a definition of a computational model to use.

Definition 3.1 (A Computational Model)**.** Suppose input is given as a read-only array of n integers, each of which can be stored in $O(1)$ space, and arithmetic operations as well as comparison on these integers take $O(1)$ time. Additionally assume that the sum of the absolute values of the n integers is an integer that can be stored in $O(1)$ space like the above. (This is a mild simplification of a well-known model called the word-RAM model.)

We can now give a definition of time and space complexity in this model.

Definition 3.2 (Time and Space Complexity)**.** We say that a problem on an input of size n is solvable in $\mathrm{TISP}(t(n), s(n))$ if there is a single algorithm that solves the problem in $O(t(n))$ time and $O(s(n))$ space in the above computational model.

Note that in the above model, sorting in-place (e.g., using quicksort) takes $\mathrm{TISP}(n \log n, n)$ time and space, since the input is read-only, so we still have to produce a new array to store the sorted output. Let's now consider a specific problem.

Definition 3.3 (k-SUM)**.** You are given an unsorted list of n integers, determine if there are k integers in the set that sum to zero.

Note that there is a trivial $\mathrm{TISP}(n^k, k \log n)$ algorithm for k-SUM based on brute force search over all k-tuples of integers, but we can do better.

(a) Design a $\mathrm{TISP}(n^{\lceil k/2 \rceil} \log n, n^{\lfloor k/2 \rfloor})$ algorithm for k-SUM (assume k is a constant).

The above algorithm is very efficient in time, but uses a lot of space when compared to the trivial brute force algorithm. So let's try to reduce the space complexity using a reduction approach.

Definition 3.4 (Grouping)**.** Given a list L of n integers and a parameter g that divides n, we say that a grouping of the list is a partition into g groups $L_1, \ldots, L_g$ such that each group has size n/g and for all $a \in L_i$ and $b \in L_{i+1}$, $a \leq b$.

Definition 3.5 (Subproblem)**.** Given a grouping $L_1, \ldots, L_g$ of a list L, a subproblem is k-tuple of indices $(i_1, \ldots, i_k)$ where $1 \leq i_1, \ldots, i_k \leq g$. A subproblem is trivial if the smallest sum of k elements in the groups $L_{i_1}, \ldots, L_{i_k}$ is greater than 0, or if the largest sum of k elements in the groups $L_{i_1}, \ldots, L_{i_k}$ is less than 0.

(b) Show that the number of non-trivial subproblems for a grouping into g groups is $O(kg^{k-1})$. (Hint. Dilworth's theorem might be useful here.)

Use the above to obtain the following "self-reduction" for 3-SUM.

(c) Suppose there is an algorithm that solves 3-SUM in $\mathrm{TISP}(t(n), s(n))$, then for any g, 3-SUM can be solved in $\mathrm{TISP}(g^2(n + t(n/g)), n/g + s(n/g))$.

(d) Conclude that there is a $\mathrm{TISP}(n^2 \log n, \sqrt{n})$ algorithm for 3-SUM.