CS 170 Homework 13

Due **Friday 12/1/2023, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. Recall that in the Max-Cut problem, we have an unweighted graph $G = (V, E)$ and we want to find a cut (S, T) that maximizes the number of edges "crossing" the cut (i.e. with one endpoint in each of S, T). Consider the following local search algorithm:

1. Start with any cut (e.g. $(S, T) = (V, \emptyset)$).
2. While there is some vertex $v \in S$ such that more edges cross $(S \setminus \{v\}, T \cup \{v\})$ than (S, T) (or some $v \in T$ such that more edges cross $(S \cup \{v\}, T \setminus \{v\})$ than (S, T)), move v to the other side of the cut.

Now, let us prove a couple of guarantees that this algorithm achieves.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

Hint: when we move v from S to T, v must have more neighbors in S than T. What does this observation suggest about the neighbors of each vertex once the algorithm terminates? Then, what can we say about the number of edges crossing the cut vs. the number of edges within each side of the cut?

3 Randomization for Approximation

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT: given an instance with m clauses each containing exactly 3 distinct literals, find the assignment that satisfies as many of them as possible. Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies c clauses, your algorithm should produce an assignment that satisfies at least $\frac{7c}{8}$ clauses in expectation.

Hint: use linearity of expectation!

(b) Given a Max 3-SAT instance I, let OPT_I denote the maximum fraction of clauses in I satisfied by any variable assignment. What is the smallest value of OPT_I over all instances I? In other words, what is $\min_I \mathsf{OPT}_I$?

Hint: use part (a), and note that a random variable must sometimes be at least its mean.

(c) **(Extra Credit)** Derandomize your algorithm from part (a), i.e give a deterministic algorithm that achieves the same approximation factor. Justify the correctness of your algorithm.

Disclaimer: this subpart is particularly difficult, so please only attempt this after completing the rest of the homework and if you want extra practice.

4 Reservoir Variations

(a) Design an algorithm that takes in a stream $x_1, \ldots, x_M$ of M integers in $[n] := \{1, \ldots, n\}$ and at any time t can output a uniformly random element in $x_1, \ldots, x_t$. Your algorithm may use at most polynomial in $\log n$ and $\log M$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

Hint: for the proof of correctness, note that $\frac{1}{t} = 1 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdots \frac{t-1}{t}$.

(b) For a stream $S = x_1, \ldots, x_{2n}$ of $2n$ integers in $[n]$, we call $j \in [n]$ a *duplicate element* if it occurs more than once.

Prove that S must contain a duplicate element, and design an algorithm that takes in S as input and with probability at least $1 - \frac{1}{n}$ outputs a duplicate element. Your algorithm may use at most polynomial in $\log n$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

Hint: Use $\log n$ *copies of the algorithm from part a to keep track of a random subset of the elements seen so far. For the proof of correctness, try to upper bound the probability that our algorithm fails to output a duplicate; also, note that there are at most* n *indices* t *such that element* x_t *never occurs after index* t.

5 Streaming Integers

In this problem, we assume we are given an infinite stream of integers $x_1, x_2, \ldots$, and have to perform some computation after each new integer is given. Since we may see many integers, we want to limit the amount of memory we have to use in total. **For all of the parts below, give a brief description of your algorithm and a brief justification of its correctness.**

(a) Show that using only a single bit of memory, we can compute whether the sum of all integers seen so far is even or odd.

(b) Show that we can compute whether the sum of all integers seen so far is divisible by some fixed integer N using $O(\log N)$ bits of memory.

(c) Assume N is prime. Give an algorithm to check if N divides the product of all integers seen so far, using as few bits of memory as possible.

Hint: suppose the first 3 integers of the stream are $7, 10, 6$*. How can you check whether* $N = 3$ *divides their product? Why is it not necessary to compute the entire product* $7 \times 10 \times 6 = 420$ *to determine divisibility?*

(d) Now, let N be an arbitrary integer, and suppose we are given its prime factorization: $N = p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$. Give an algorithm to check whether N divides the product of all integers seen so far, using as few bits of memory as possible. Write down the number of bits your algorithm uses in terms of $k_1, \ldots, k_r$.

Hint: keep track of r *values (i.e. one for each prime factor of* N*) throughout your algorithm.*