CS 577: Introduction to Algorithms — Homework 3

Out: 03/03/2022 — **Due: 03/22/2020**

Ground Rules

- **Reminder: Midterm is scheduled on March 10, Thursday, 11am –12:15pm, In class, losed book, closed notes.**
- Read the lecture notes.
- Read Chapters on Hashing, Binary Search Trees, Red-Black Trees, Greedy Algorithms, Data Structures for Disjoint Sets.
- You should do ***all*** problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1. You are tasked to select a subset of the employees at your company for an evening fun party. The employees of your company are organized into a strict hierarchy, that is, a tree with the company president at the root. The all-knowing oracles in Human Resources have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if the employee's immediate supervisor is also present. Design an efficient algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests.

   Prove that your algorithm produces the optimal solution. What is the running time of your algorithm? Prove your answer on the running time.

2. An element x of an array $A[1:n]$ is called a *majority element* if more than half of A are equal to x. Your task is to find whether a given array has a majority element and if so, what it is. We do not assume the elements of A are integers, or otherwise ordered; the only query you can access the elements of A is of the form whether $A[i] = A[j]$ in constant time, for any i and j.

   - Solve this problem with a Divide-and-Conquer algorithm that runs in time $O(n \log n)$. Prove your algorithm is correct, and prove the time bound. (Hint: By dividing A into two halves $A1$ and $A2$, does the knowledge of whether $A1$ and/or $A2$ have a majority element help? You must be rigorous in your reasoning.)
   - Here is another Divide-and-Conquer approach: We will pair up elements of A arbitrarily. If the pair of elements are different, then discard them both. If the pair of elements are the same, then keep exactly one of them. (If n is odd, think about what you should do with the unique unpaired element.) This produces an array of at most $n/2$ elements.

     Prove that if A has a majority element, then no matter how the pairing is done, the reduced array also has a majority element and it must be the same. Does the converse hold?

     Now devise a Divide-and-Conquer algorithm following this strategy that runs in time $O(n)$.

3. In class we described universal hash functions. Let p be a prime, and let $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$. In class we described a family a universal hash functions of the form $h_{a,b}(x) = ax + b \bmod p$. You should also review the Eulidean Algorithm that finds the gcd of any positive integers a and b, including two integers (could be negative) x and y such that $ax + by = \gcd(a, b)$.

- Define the following family of hash functions

$$\mathcal{H} = \{h_{a,b,c} \mid h_{a,b,c}(x) = ax^2 + bx + c \bmod p, \text{ where } a, b, c, x \in \mathbb{Z}_p\}$$

Prove the following "three-way independence property": For any $x, y, z \in \mathbb{Z}_p$ that are pairwise distinct, and for any target values $i, j, k \in \mathbb{Z}_p$,

$$\Pr_{(a,b,c)}[(h_{a,b,c}(x) = i) \wedge (h_{a,b,c}(y) = j) \wedge (h_{a,b,c}(z) = k] = \frac{1}{p^3}.$$

Conclude that

$$\Pr_{(a,b,c)}[(h_{a,b,c}(x) = i) \wedge (h_{a,b,c}(y) = j)] = \frac{1}{p^2}.$$

and

$$\Pr_{(a,b,c)}[h_{a,b,c}(x) = i] = \frac{1}{p}.$$

- Let $N \geq n$, and let A and b denote matrices in $\mathbb{Z}_p^{N \times n}$ and vectors in $\mathbb{Z}_p^n$. Prove that the following family is a universal family of hash functions

$$\mathcal{M} = \{f_{A,b} \mid f_{A,b}(x) \equiv Ax + b \bmod p, \text{ where } x \in \mathbb{Z}_p^N\}$$

where Ax is a matrix-vector product and all arithmetic operations are done mod p (component by component).

The random choice of $f_{A,b}$ is to pick a matrix $A \in \mathbb{Z}_p^{N \times n}$ among all p^{Nn} such matrices with equal probability, and to pick $b \in \mathbb{Z}_p^n$ among all p^n such vectors with equal probability independently from A.

4. **Graded Problem (Page limit: 1 sheet; 2 sides)** Suppose two parties A and B agree to communicate through the internet where messages are publicly readable, and anyone can try to pretend to be anyone else. A and B agree (beforehand) on a large prime p (which is made public, and so not a secret), and two secret numbers a and $b \in \mathbb{Z}_p$ (both a, b are known only to A and B). This defines a hash function $x \mapsto h(x) = ax + b \bmod p$.

When A wishes to send a message to B (and we assume it can be coded via ASCII code as an integer $x < p$), A sends the message x and an authentication $v = ax + b \bmod p$. B convinces herself that this message x was sent by A by verifying that $h(x) = v$.

- Now suppose some nefarious entity C wants to pretend to be A and sends B a message (x', v') (where $x' \neq x$). Suppose C has seen a legitimate message (x, v) from A to B. If C does not know the secret a and b that define the hash function (even knowing the form of the hash function and the prime p), prove that C cannot fool B into believing that this message (x', v') was sent from A. (You prove that the probability that C can send the correct v' is $1/p$.)
- Now suppose C has intercepted two messages from A to B, in the form of (x, v) and (y, w). Show that C now can pretend to be A and convinces B of any message x' by appending the proper v' pretending that it was sent from A.
- Suggest a way to change the communication scheme that A and B agree beforehand that can tolerate some adversary C from intercepting up to two messages.

  (Extra Credit:) How about up to k intercepted messages?

5. **Graded Problem (Page limit: 1 sheet; 2 sides)** We now revisit the randomized MAXCUT problem presented in class.

In our presentation, we assigned a "monkey" on every vertex of the input graph G and asked them to each flip a fair coin and determine the vertex is to be put on the left or the right side. We showed that this randomized algorithm achieves a cut size in expectation at least 50% of the maximum possible.

Note that the total number of random choices the "monkeys" could make collectively is 2^n for a graph of n vertices.

Use universal hash function to design an alternative randomized algorithm that also achieves this performance in expectation, furthermore your alternative randomized algorithm has the following advantage: If we tried all possible choices made by your randomized algorithm there are only a polynomial number $O(|G|^k)$ (where $|G|$ is the size of G, and k is some constant) of possible choices, and therefore one could try all these choices deterministically and pick the largest cut produced. Show that this gives a deterministic algorithm that runs in time $O(|G|^k)$, for some constant k, and achieves a cut size at least 50% of the maximum possible.