CS 473 ✧ Fall 2024

Homework 1

Due Tuesday, September 10, 2024 at 9pm Central Time

For this and all future homeworks, groups of up to three students can submit joint solutions. All students in the group must collaborate on each problem. Please review the homework policies on the course webpage for more details.

---

1. The social media site FRIENDER represents its social network as an undirected graph G, where each vertex represents a user, and each edge connects two users who are friends. The FRIENDER API ***only*** allows you to access this graph through a subroutine whose input and output behavior is described below:

   > ANYFRIENDSBETWEEN(S, T):
   > **Input**: distinct subsets S and T of vertices of G
   > **Output**: TRUE if G has at least one edge st where $s \in S$ and $t \in T$; FALSE otherwise.

   Design an algorithm that returns *the number of* edges between two given subsets of vertices, using ***only*** calls to ANYFRIENDSBETWEEN. Specifically, given two subsets S and T of vertices, your algorithm should return the number of edges with one endpoint in S and one endpoint in T. Your algorithm does not have direct access to the underlying graph.

   To avoid delving into irrelevant details of how G is represented, how sets of users are represented, or how ANYFRIENDSBETWEEN is implemented, analyze the worst-case number of times your algorithm calls ANYFRIENDSBETWEEN, as a proxy for running time.

   Suppose S and T each have at most n vertices. For full credit, your algorithm should run in subquadratic "time" when the number of edges between S and T is significantly less than n^2.

2. Suppose you are given an array $A[1..n]$ of positive integers. An *interval* in A is a contiguous subarray $A[i..j]$ for some pair of indices $1 \le i \le j \le n$. Suppose we compute the sum of elements in each of the $\Theta(n^2)$ intervals in A; some of these sums might coincide. This question asks you to design and analyze efficient algorithms to compute the number of *distinct* interval sums in A.

   (a) Describe an algorithm that runs in $O(n^2 \log n)$ time.

   (b) Describe an algorithm that runs in $O(M \log M)$ time, where $M = \sum_i A[i]$.
   [Hint: Use FFTs.]

   For example, given the input array $A = [8, 2, 3, 5]$, your algorithms should return 7, because A has seven distinct interval sums: 2, 3, 5, 8, 10, 13, and 18. For example, A contains two intervals $[8, 2]$ and $[2, 3, 5]$ that both sum to 10.

3. The *Hamming distance* between two bit strings is the number of positions where the strings have different bits. For example, the Hamming distance between the strings 01101001 and 11010001 is 4.

   Suppose we are given two bit strings $P[1..m]$ (the "pattern") and $T[1..n]$ (the "text"), where $m \leq n$. Describe and analyze an algorithm to find the minimum Hamming distance between P and a substring of T of length m. For full credit, your algorithm should run in $O(n \log n)$ time.

   For example, given input strings $P =$ 11011011 and $T =$ 111111101010100000, your algorithm should return 2, which is the Hamming distance between P and the substring 11111010 of T:

```
1111111010101000000
  11011011
```

   [Hint: Consider 0*s and* 1*s separately.]*