Problem Set 4

This problem set is due **in recitation** on **Friday, March 19**.

Reading: Chapters §30.1-30.2, 11.1-11.3, 11.5, 12.1-12.3

There are **three** problems. Each problem is to be done on a **separate sheet** (or sheets) of paper. Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

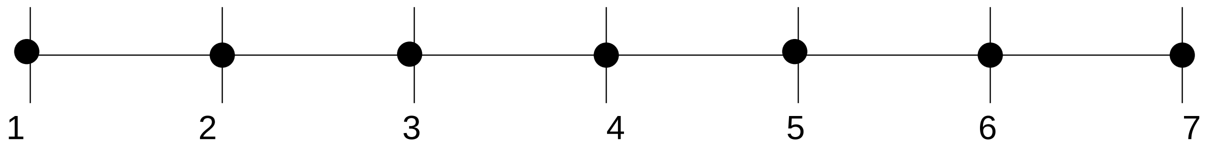
You will often be called upon to "give an algorithm" to solve a certain problem. Giving an algorithm entails:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. A proof (or argument) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

It is also suggested that you include at least one worked example or diagram to show more precisely how your algorithm works. Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions. If you cannot solve a problem, give a brief summary of any partial results.

Problem 4-1. Electric Potential of Equally Spaced Charges

Consider a set of n point charges $p_1, p_2, \ldots, p_n$ located at position $1, 2, \ldots, n$ along the x-axis respectively. Each p_i carries an electric charge q_i. This configuration with $n = 7$ is depicted in the following figure:



In this problem we would like to find the electric potential $v(p_i)$ at each point charge p_i induced by all other point charges, given by the formula where c is a constant:

$$v(p_i) = c \left(\sum_{j \neq i}^{n} \frac{q_j}{|i - j|} \right).$$

(a) The induced potential on all the point charges can be computed by multiplying an $n \times n$ matrix $\mathbf{A}$ by the n-dimensional charge vector $\mathbf{q}$. In other words $\mathbf{v} = c\mathbf{A}\mathbf{q}$, where c is the constant in the above formula. Give the matrix $\mathbf{A}$ for $n = 4$. What is $\mathbf{A}$'s form in general?

Solution: For $n = 4$, $\mathbf{A}$ equals:

$$\begin{pmatrix} 0 & 1 & \frac{1}{2} & \frac{1}{3} \\ 1 & 0 & 1 & \frac{1}{2} \\ \frac{1}{2} & 1 & 0 & 1 \\ \frac{1}{3} & \frac{1}{2} & 1 & 0 \end{pmatrix}$$

In general, $\mathbf{A}$ is a Toeplitz matrix and is uniquely determined by the $2n - 1$ values in the the first column and the first row. Each diagonal is identical, so $a_{ij} = a_{i-1,j-1}$. In this problem, $\mathbf{A}$ happens to be a symmetric Toeplitz matrix and takes the form:

$$\begin{pmatrix} 0 & 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n-1} \\ 1 & 0 & 1 & \frac{1}{2} & \cdots & \frac{1}{n-2} \\ \frac{1}{2} & 1 & 0 & 1 & \cdots & \frac{1}{n-3} \\ \frac{1}{3} & \frac{1}{2} & 1 & 0 & \cdots & \frac{1}{n-4} \\ \vdots & & & & \ddots & \\ \frac{1}{n-1} & \frac{1}{n-2} & \frac{1}{n-3} & \frac{1}{n-4} & \cdots & 0 \end{pmatrix}$$

(b) Give a representation of $\mathbf{A}$ by an $O(n)$-size vector.

Solution:
We can represent matrix $\mathbf{A}$ as a $(2n-1)$-dimensional vector $(a_1, a_2, \ldots, a_{2n-1})^T = (\frac{1}{n-1}, \frac{1}{n-2} \ldots, \frac{1}{2}, 1, 0, 1, \frac{1}{2}, \ldots, \frac{1}{n-2}, \frac{1}{n-1})^T$, where each entry a_i represents a diagonal in $\mathbf{A}$, (starting from the upper right diagonal to the lower left diagonal).

(c) Give an algorithm that computes the potential vector $\mathbf{v} = c\mathbf{A}\mathbf{q}$, in $O(n \log n)$ time.

Solution: Using the $O(n)$ representation of $\mathbf{A}$ from part (b), we have the following, for all $i = 1 \ldots n$:

$$v_i = \sum_{j=1}^{n} a_{n+i-j} q_j$$

Now we define polynomials $p_a(x)$ and $p_q(x)$ as follows:

$$\begin{aligned} p_a(x) &= \sum_{i=1}^{2n-1} a_i x^i, \\ p_q(x) &= \sum_{i=1}^{n} q_i x^i. \end{aligned}$$

Then the term v_i is exactly the coefficient of x^{n+i} in the polynomial $p_a(x)p_b(x)$. Therefore, the potential vector **v** can be obtained by multiplying two polynomials $p_a(x)$ and $p_q(x)$ with degrees $O(n)$, which can be achieved in $O(n \log n)$ time by Fast Fourier Transform.

Problem 4-2. Universal Hashing

Recall that a collection $\mathcal{H}$ of hash function from a universe $\mathcal{U}$ to a range R is called **universal** if for all $x \neq y$ in $\mathcal{U}$ we have

$$\Pr_{h \in \mathcal{H}} [\, h(x) = h(y) \,] = \frac{1}{|R|}$$

We want to implement universal hashing from $\mathcal{U} = \{0,1\}^p$ to $R = \{0,1\}^q$ (where $p > q$).

For any $q \times p$ boolean matrix A and any q-bit vector b we define the function $h_{A,b} : \{0,1\}^p \rightarrow \{0,1\}^q$ as $h_{A,b}(x) = Ax + b$, where by this we mean the usual matrix-vector multiplication and the usual vector addition, except that all the operations are done modulo 2. For example, if $q = 2, p = 3$ and

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

we have

$$h_{A,b}(x) = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x_2 + x_3 + 1 & mod\ 2 \\ x_1 + x_3 & mod\ 2 \end{pmatrix}$$

(a) Prove that the collection $\mathcal{H} = \{\ h_{A,b}\ : A \in \{0,1\}^{q \times p}, b \in \{0,1\}^q\ \}$ is universal.

Solution: Notice that for any function $h_{A,b}$ we have $h(x) = h(y)$ if and only if $A(x - y) = \bar{0}$. We want to show that for any non-zero vector $z (= x - y) \in \{0,1\}^p$, if we choose A at random from $\{0,1\}^{q \times p}$ then the probability of getting $Az = \bar{0}$ is exactly $1/2^q$.

So let z be any non-zero vector in $\{0,1\}^p$, and assume w.l.o.g. that the first coordinate of z is non-zero (the same argument holds if we assume that any other coordinate of z is non-zero). That is, we assume that $z_1 = 1$.

Denote $A = (a_{ij})_{ij}$ and consider any choice of all the elements in A except for the the first column. That is, we assume that we have already chosen all the elements

$$\begin{matrix} a_{12} & a_{13} & \cdots & a_{1p} \\ & & \vdots & \\ a_{q2} & a_{q3} & \cdots & a_{qp} \end{matrix}$$

and the only free variables left are $a_{11}, a_{21}, \cdots a_{q1}$. However, in order to satisfy $Az = \bar{0}$, these a_{i1}'s need to satisfy $a_{i1}z_1 + a_{i2}z_2 + \cdots + a_{ip}z_p = 0 \pmod 2$ for all i. Since $z_1 = 1$, the only choice which will satisfy $Az = \bar{0}$ is

$$\begin{aligned} a_{11} = &\ -(a_{12}z_2 + \cdots + a_{1n}z_p) \pmod 2 \\ &\vdots \\ a_{q1} = &\ -(a_{q2}z_2 + \cdots + a_{qn}z_p) \pmod 2 \end{aligned}$$

Since there are 2^q possible ways to choose all the a_{i1}'s from $\{0, 1\}$ and only one of these choices satisfy $Az = \bar{0}$ then the probability of hitting that one when picking these elements as random is $1/2^q$.

(b) Let $S \subseteq \mathcal{U}$ be the set we would like to hash. Let $n = |S|$ and $m = 2^q$. Prove that if we choose $h_{A,b}$ from $\mathcal{H}$ uniformly at random, the expected number of pairs $(x, y) \in S \times S$ with $x \neq y$ and $h_{A,b}(x) = h_{A,b}(y)$ is $O(\frac{n^2}{m})$.

Solution: For any 2 dintinct elements $x, y \in S$, let I_{xy} be the indicator variable for the event that $h_{A,b}(x) = h_{A,b}(y)$. Since $\mathcal{H}$ is universal, $E[I_{xy}] = \Pr\{h_{A,b}(x) = h_{A,b}(y)\} = \frac{1}{m}$. The expected number of pairs $(x, y) \in S \times S$ with $x \neq y$ and $h_{A,b}(x) = h_{A,b}(y)$ is therefore:

$$E[\sum_{x \neq y} I_{xy}] = \sum_{x \neq y} E[I_{xy}] = \frac{n(n-1)}{m} = O(\frac{n^2}{m}).$$

Problem 4-3. Range Query in Binary Search Tree

Given a binary search tree T and a pair of numbers a, b with $a \leq b$, give an algorithm that returns the set of elements in T with key values in $[a, b]$. Your algorithm should run in $O(h + m)$ time, where h is the height of T and m is the number of elements within the range.

Solution:

```
RANGED-BST-SEARCH($r, a, b$):
 1  S $\leftarrow \emptyset$
 2  $w \leftarrow r$
 3  while $w \neq$ NIL and not $(a \leq w \leq b)$
 4     if $w < a$
 5           then $w \leftarrow right[w]$
 6           else $w \leftarrow left[w]$
 7  if $w =$NIL return S
 8  S$\leftarrow$ S $\cup \{w\}$
 9  $x \leftarrow left[w] \triangleright$ Collect elements in the left subtree
10  while $x \neq$ NIL
11     if $key[x] < a$
12           then $x \leftarrow right[x]$
13           else S$\leftarrow$ S $\cup \{x\} \cup$GET-ELEMENTS-IN-TREE$(right[x])$
14                $x \leftarrow left[l]$
15  $y \leftarrow right[w] \triangleright$ Collect elements in the right subtree
16  while $y \neq$ NIL
17     if $key[y] > b$
18           then $y \leftarrow left[y]$
19           else S$\leftarrow$ S $\cup \{y\} \cup$GET-ELEMENTS-IN-TREE$(left[y])$
20                $l \leftarrow right[y]$
21  return S
```

Lines 3-6 walk down from the root to find the smallest subtree that contains all the elements within the range. The loop invariant is that the subtree rooted at w contains all the elements within the range. At the end of the loop, w is the root of the smallest subtree containing all the elements within the range.

The two loops, lines 9-14 and lines 15-20, collect all the queried elements in the left subtree and right subtree of w respectively. The loop invariant is that all the queried elements are contained in either the set **S** or substrees rooted at x and y. At the end of the loops, both x and y are NIL, so **S** must contain all the queried elements. The procedure GET-ELEMENTS-IN-TREE(x) returns the set of all elements in the subtree rooted at x. This can be achieved in linear time by slightly modifying the INORDER-TREE-WALK procedure.

The algorithm walks from the root to the leaves corresponding to the boundaries of the query, taking $O(h)$ time. For the nodes within the range, they are either along the two paths or collected by the GET-ELEMENTS-IN-TREE procedure, which costs $O(1)$ running time per node. For m nodes collected the total running time will be $O(m)$. Therefore, the overall time complexity of the algorithm is $O(h + m)$.

An alternative solution is to find the smallest element x in the tree with key at least a, using a procedure similar to TREE-SEARCH. Then we can successively call TREE-SUCCESSOR starting with x until the returned element has key $> b$ (or returns NIL). The algorithm clearly works because it keeps collecting elements within the range $[a.b]$ until there is no more left. The algorithm takes $O(h)$ time to find the element x and makes m successive calls of TREE-SUCCESSOR to find the next m elements. It can be shown that m successive calls of procedure TREE-SUCCESSOR take a total of $O(h + m)$ time. Therefore, the total running time for this algorithm is $O(h + m)$.