1 Hashing (34 points)

Let $H = \{h_1, h_2, \ldots\}$ be a collection of hash functions, where $h_i : U \to \{0, 1, \ldots, M - 1\}$ for every $i$ and we assume that $|U| = 2^u$ and that $M = 2^b$ (the same setup as in class when we designed a universal hash family). Recall that $H$ is a universal hash family if $\Pr_{h \sim H}[h(x) = h(y)] \leq 1/M$ for all $x, y \in U$.

Consider the following, slightly different definition. We say that $H$ is a 2-universal hash family if $\Pr_{h \sim H}[h(x) = a \land h(y) = b] \leq 1/M^2$ for all $x, y \in U$ with $x \neq y$ and $a, b \in \{0, 1, \ldots, M - 1\}$.

(a) (17 points) Prove that any 2-universal hash family is also a universal hash family.

Solution: Let $H$ be a 2-universal hash family. Let $x, y \in U$ be two arbitrary elements. For each outcome $a \in \{0, 1, \ldots, M - 1\}$, let $Z_a$ be the event that $h(x) = h(y) = a$, and let $Z$ be the event that $h(x) = h(y)$. Note that by the definition of a universal hash family, $\Pr_{h \sim H}[Z_a] \leq 1/M^2$ for all $a$. Clearly the events $\{Z_a\}$ are disjoint, and their union is exactly $Z$. Hence

$$\Pr_{h \sim H}[h(x) = h(y)] = \Pr_{h \sim H}[Z] = \sum_{a=0}^{M-1} \Pr_{h \sim H}[Z_a] \leq M \cdot \frac{1}{M^2} = \frac{1}{M}.$$

Since $x$ and $y$ were arbitrary elements, we get that $H$ is a universal hash family.

(b) (17 points) Prove that there is some universal hash family which is not a 2-universal hash family.

Solution: Let $H$ be the universal hash family we constructed in class, i.e., the hash functions in $H$ are random 0/1 matrices with the appropriate dimensions. Let $x$ be the all 0’s vector, and let $y$ be any other element. Note that $h(x) = \vec{0}$ for all $h \in H$. Moreover, it is easy to see that $\Pr_{h \sim H}[h(y) = \vec{0}]$ is exactly $1/M$ (as we discussed in class, for this hash family we know that $\Pr_{h \sim H}[h(y) = a] = 1/M$ for all $a \in U$, as long as $y \neq \vec{0}$). Thus by setting $a = b = \vec{0}$ we get that

$$\Pr_{h \sim H}[h(x) = h(y)] = \Pr_{h \sim H}[h(y) = \vec{0}] = 1/M.$$

Since $1/M > 1/M^2$, this implies that $H$ is not a 2-universal hash family.
2 Union-Find (33 points)

In this problem we’ll consider what happens if we change our Union-Find data structure to not use path compression. We will still use union-by-rank, but Find operations will no longer compress the tree. More formally, consider the following tree-based data structure. Every element has a parent pointer and a rank value.

**Make-Set**(x): Set \( x \rightarrow \text{parent} := x \) and set \( x \rightarrow \text{rank} := 0 \).

**Find**(x): If \( x \rightarrow \text{parent} == x \) then return \( x \). Else return \( \text{Find}(x \rightarrow \text{parent}) \).

**Union**(x, y):

Let \( w := \text{Find}(x) \) and let \( z := \text{Find}(y) \).

If \((w \rightarrow \text{rank}) \geq (z \rightarrow \text{rank})\) then set \( z \rightarrow \text{parent} := w \), else set \( w \rightarrow \text{parent} := z \).

If \((w \rightarrow \text{rank}) == (z \rightarrow \text{rank})\), set \((w \rightarrow \text{rank}) := (w \rightarrow \text{rank}) + 1\).

In this problem we will analyze the running time of this variation.

(a) (11 points) Recall that the height of any node \( x \) is the maximum over all of the descendants of \( x \) of the length of the path from \( x \) to that descendant. Prove that for every node \( x \), the rank of \( x \) is always equal to the height of \( x \).

**Solution:** We prove this by induction. For the base case, it is easy to see that it is initially true: after the first call to Make-Set we have a single node with rank 0 and height 0. For the inductive step, assume that this is true at some point in time and then we execute another operation. If this operation is a Find it has no effect on the ranks or the trees (since we are not using path compression), so all nodes still have rank = height. Similarly, if the operation is a Make-Set then it is true of the new tree (for the same reason as in the base case), and nothing has changed about the other nodes so it is still true for all nodes.

Now suppose that the operation is a Union. In particular, suppose that it is Union(x, y), and let \( w \) be the root of the tree containing \( x \) and let \( z \) be the root of the tree containing \( y \). Note that the only node whose height might have changed is the whichever of \( z \) and \( w \) becomes the root of the new tree, so we just need to prove that the height of this node is equal to its rank. Let \( r_w = w \rightarrow \text{rank} \) and let \( r_z = z \rightarrow \text{rank} \). If \( r_w > r_z \) then \( w \) becomes the new root with rank \( r_w \). Its heigh will be \( \max\{r_w, r_z + 1\} \), since it will have a path of length \( r_w \) to one of its descendants (by the inductive hypothesis) and a path of length \( r_z + 1 \) to one of \( z \)'s descendants (by the inductive hypothesis). Since \( r_w > r_z \), this implies that the height is equal to the rank, as they are both \( r_w \). Similarly, if \( r_z > r_w \) then \( z \) becomes the new root with rank \( r_z \) and height \( \max\{r_z, r_w + 1\} = r_z \).

So all that remains is to analyze the case of \( r_w = r_z \). In this case, \( w \) becomes the new root with rank \( r_w + 1 \). By the inductive hypothesis, its new height is equal to \( \max\{r_w, r_z + 1\} \). Since \( r_w = r_z \), this is equal to \( r_w + 1 \) and thus equal to the rank, as claimed.

(b) (11 points) Prove that if \( x \) has rank \( r \), then there are at least \( 2^r \) elements in the subtree rooted at \( x \) (we did this in class for the more complicated data structure which uses path compression, but now you should do it for this version without path compression).

**Solution:** This is basically the same analysis as we did in class. We prove this by induction. Initially, after the first Make-Set, there is one element with rank 0 and its subtree has exactly
1 = 2^9 elements in it. Now suppose it is true at some point in time, and consider doing another operation. A Find does not change the trees or the ranks, so it would still be true. A Make-Set would maintain the property for the same reason as in the base case.

Now suppose we do Union(x, y). The only node whose rank or subtree changes is the new root (either z or w depending on the ranks). Let \( r_z = z \rightarrow \text{rank} \) and let \( r_w = w \rightarrow \text{rank} \). If \( r_w > r_z \) or \( r_z > r_w \) then no node changes ranks and the size of each subtree does not decrease, so by the inductive hypothesis it is still true. If \( r_w = r_z \), then the new rank of \( w \) is \( r_w + 1 \) and by the inductive hypothesis the size of its subtree is at least \( 2^{r_w} + 2^{r_z} = 2^{r_w} + 2^{r_w} = 2^{r_w+1} \), as claimed.

(c) (11 points) Using the previous two parts, prove that every operation (Make-Set, Union, and Find) takes only \( O(\log n) \) time (where \( n \) is the number of elements, i.e., the number of Make-Set operations).

**Solution:** Make-Set clearly takes \( O(1) \) time. The running time of each Union is \( O(1) \) plus the time to do two Find operations. Hence we need only prove that the running time of a Find is \( O(\log n) \). Part b clearly implies that every node has rank at most \( \log n \) (since there are only \( n \) elements total), so this together with part a implies that every node has height at most \( \log n \). Since the running time of a Find operation is at most the height of a tree (since it is just a walk up to the root), we get that the running time of a Find is \( O(\log n) \).

3 Submatrices (33 points)

Let \( A \in \{0, 1\}^{n \times m} \) be a matrix with \( n \) rows, \( m \) columns, and where every entry is either 0 or 1. We will let \( A_{ij} \) denote the entry in row \( i \) and column \( j \), so for example \( A_{11} \) is the top-left entry, \( A_{n1} \) is the bottom-left entry, \( A_{1m} \) is the top-right entry, and \( A_{nm} \) is the bottom-right entry. Suppose that we want to find the largest integer \( k \) such that \( A \) contains a \( k \times k \) contiguous submatrix consisting of all 0's. In other words, we want to find the largest \( k \) such there exist values \( i, j \) such that \( A_{xy} = 0 \) for all \( i - k < x \leq i \) and \( j - k < y \leq j \).

We will design a dynamic programming algorithm that runs in \( O(nm) \) time for this problem.

(a) (17 points) For every \( i, j \in \mathbb{N} \) with \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \), let \( S(i, j) \) denote the maximum value of \( k \) such that there is a \( k \times k \) contiguous submatrix of \( A \) consisting of all 0's whose bottom-right corner is at \((i, j)\) (row \( i \), column \( j \)). Write a recursive formula for \( S(i, j) \), and prove that your formula is correct.

Note: you will need to use this formula in the next part to get an \( O(nm) \)-time algorithm, so make sure that your formula is not too big/slow.

**Solution:** We will use the following formula:

\[
S(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
0 & \text{if } A_{ij} = 1 \\
\min(S(i - 1, j), S(i, j - 1), S(i - 1, j - 1)) + 1 & \text{otherwise}
\end{cases}
\]
To see that this is correct, suppose that there is a $k \times k$ submatrix of 0's whose bottom-right entry is at $(i, j)$. Then there must be $(k - 1) \times (k - 1)$ submatrices of all 0's whose bottom-right entries are at $(i - 1, j)$, $(i, j - 1)$, and $(i - 1, j - 1)$, since all three of these submatrices would be contained in the $k \times k$ submatrix whose bottom-right entry is at $(i, j)$. Thus $S(i, j) \leq \min(S(i - 1, j), S(i, j - 1), S(i - 1, j - 1)) + 1$. On the other hand, if all three of those submatrices exist and $A_{ij} = 0$, then a $k \times k$ all-0's submatrix exists with bottom-right at $(i, j)$. Thus if $A_{ij} = 0$ we have that $S(i, j) \geq \min(S(i - 1, j), S(i, j - 1), S(i - 1, j - 1)) + 1$.

(b) (16 points) Give a dynamic programming algorithm based on your solution to part (a), and prove that it correctly finds the largest possible value of $k$ and runs in time $O(nm)$.

Solution: Consider the following bottom-up DP algorithm.

```plaintext
def f(i, j):
    if $A_{ij} = 1$:
        $M[i, j] = 0$
    else:
        $M[i, j] = \min(M[i - 1, j], M[i, j - 1], M[i - 1, j - 1]) + 1$
return $\max_{i=1}^{n} \max_{j=1}^{m} M[i, j]$
```

We prove correctness by induction on $(i, j)$ (ordered lexicographically). In particular, we will prove that $M[i, j] = S(i, j)$, which then together with part (a) implies that the algorithm is correct. For the base case, if $i = 0$ or $j = 0$, then $M[i, j] = 0 = S(i, j)$. For the inductive step, if $A_{ij} = 1$ then $M[i, j] = S(i, j)$. Otherwise, we get that

$M[i, j] = \min(M[i - 1, j], M[i, j - 1], M[i - 1, j - 1]) + 1$

$= \min(S(i - 1, j), S(i, j - 1), S(i - 1, j - 1)) + 1$

$= S(i, j)$,

where the first equality is by the definition of the algorithm, the second is by the inductive hypothesis, and the third by is the definition of $S(i, j)$. Hence $M[i, j] = S(i, j)$, so the algorithm is correct.

For the running time, note that the first two loops take time $O(n)$ and $O(m)$ respectively, then then for each of the $nm$ entries in $M$ the computation takes only $O(1)$ time (a single min over three elements). Finally, the ending max computation requires looking over the entire array $M$ and keeping track of the maximum, which can be done in $O(nm)$ time.