1 Hashing (34 points)

Let $H = \{h_1, h_2, \ldots\}$ be a collection of hash functions, where $h_i : U \rightarrow \{0, 1, \ldots, M - 1\}$ 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.

(b) (17 points) Prove that there is some universal hash family which 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 $x \rightarrow parent := x$ and set $x \rightarrow rank := 0$.
- **Find($x$):** If $x \rightarrow parent == x$ then return $x$. Else return $\text{Find}(x \rightarrow parent)$.
- **Union($x, y$):**
  - Let $w := \text{Find}(x)$ and let $z := \text{Find}(y)$.
  - If $(w \rightarrow rank) \geq (z \rightarrow rank)$ then set $z \rightarrow parent := w$, else set $w \rightarrow parent := z$.
  - If $(w \rightarrow rank) == (z \rightarrow rank)$, set $(w \rightarrow rank) := (w \rightarrow rank) + 1$

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 $x$.

(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).
(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).

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.

(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 \(O(nm)\).