University of Waterloo
CS240E, Winter 2025
Assignment 4

Due Date: Tuesday, March 18, 2025 at 5pm

Be sure to read the assignment guideliness (`https://student.cs.uwaterloo.ca/~cs240e/w25/assignments.phtml#guidelines`). Submit your solutions electronically to Crowdmark.

Grace period: submissions made before 11:59PM on Mar. 18 will be accepted without penalty. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [2+4=6 marks]

Suppose that we use hashing with a randomly picked hash-function and the uniform hashing assumption holds. As usual, let M be the table-size, and let $n \leq M$ be the number of key-value pairs that we want to insert.

For both part-questions, give an exact answer (no asymptotics), simplify your expression as much as possible, and justify your answer.

a) What is the probability that the first and second item that we inserted are in distinct buckets?

b) What is the probability that after inserting n items there are no collisions, i.e., all buckets contain at most one item?

Hint: The formula that you give should depend on both n and M. It also should be 1 for $n = 1$, and the same as the answer to (a) for $n = 2$. Be sure to check this!

Question 2 [1+2+2+5=10 marks]

Assume that we have a hash function h, and define a probe sequence via $h(k, 0) = h(k)$ and

$$h(k,i) = \Big(h(k,i-1) + i\Big) \bmod M \quad \text{for } 1 \leq i < M$$

a) Write the probe sequence for $h(k) = 0$ and $M = 8$.

b) A probe sequence is called *quadratic probing* if $h(k,i) = \big(h(k) + c_1 i + c_2 i^2\big) \bmod M$ for some hash-function h and some constants $c_1, c_2 \geq 0$. Show that the probe sequence defined above is an instance of quadratic probing.

c) Show that if $h(k,i)=h(k,j)$ for some $0 \leq i < j < M$, then $(j-i)(j+i+1)=0 \bmod 2M$.

d) Assume that M is a power of 2, say $M = 2^m$ for some integer m. Prove that all entries in the probe sequence are different.

Hint: The course notes contains some rules about modular arithmetic that you may use without proof.

Question 3 [2+4+5 = 11 marks]

We have seen one method of obtaining a universal family of hash-functions in class. This assignment discusses another one. Let us assume that all keys come from some universe $\{0, \ldots, U-1\}$, where $U = 2^u$. Therefore any key k can be viewed as bit-string x_k of length u by taking its base-2 representation.

Let us assume further that the hash-table-size M is $M = 2^m$ for some integer m, with $m < u$. To choose a hash-function, we now randomly choose each entry in an $m \times u$-matrix H to be 0 or 1 (equally likely). Then compute $h_k = (Hx_k)\%2$, where x_k is now viewed as a vector and '%2' is applied to each entry. The output is a m-dimensional vector with entries in $\{0, 1\}$; interpreting it as a length-m bit-string gives a number $\{0, ..., M-1\}$ that we use as hash-value $h(k)$. For example, if $k = 18$, $u = 5$, $m = 3$ and H is as shown below, then $h(k) = 1$ since

$$\underbrace{\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}}_{H} \underbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}}_{\text{18 as length-5 bit-string}} \%2 = \underbrace{\begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}}_{Hx_k} \%2 = \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{\text{1 as length-3 bit-string}}$$

a) Let H be the above matrix, $u = 5$ and $m = 3$. Consider the keys 9 and 13. What are their hash-values? Show your work.

b) Consider again $u = 5, m = 3$ and keys $k = 9$ and $k' = 13$. Consider the same matrix H, except that the bits in the middle column are randomly chosen. What is the probability that $h(k) = h(k')$? Justify your answer.

c) Show that (for any u, m) this method of choosing the hash function gives a universal hash function family, or in other words, $P\big(h(k) = h(k')\big) \leq \frac{1}{M}$ for any two keys $k \neq k'$.

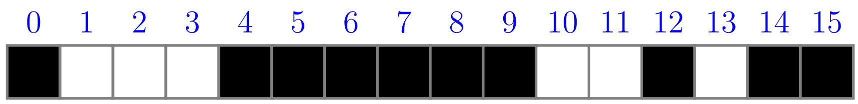
Question 4 [2+3+6=11 marks]

Recall that a quad-tree can be used to store pixelated pictures in a compressed way. This assignment will use the same approach, but for simplicity it will be in dimension 1.

So let $P[0..2^h-1]$ be an array of *pixels*, which are either "black" or "white". Define the *pixel-tree* T as follows: The root is associated with the index-range $[0, 2^h-1]$. If all pixels in this range are the same, then the root becomes a leaf (and stores the colour of the pixels).

Otherwise, the root has two children that are associated each with half of the index-range: if the parent was associated with $[\ell, r]$, then the left child is associated with $[\ell, \frac{r+\ell+1}{2}-1]$ while the right child is associated with $[\frac{r+\ell+1}{2}, r]$. At these children, we recursively build the pixel-trees.

a) Show the pixel-tree T for the following array of pixels:



b) Assume that you are given a binary tree T where the leaves are marked as "black" or "white", and an integer $h \geq 0$. You are told that this is a pixel-tree for the index-range $[0..2^h-1]$, but you do *not* have access to the pixel-array. Give an algorithm that can answer a query "what is the colour of pixel $P[i]$?" for any $0 \leq i < 2^h$ in $O(h)$ time.

c) Assume that you are given T, h as in the previous part. Give an algorithm that finds (for a given $0 \leq i < 2^h$) the maximum range $\ell \leq i \leq r$ such that all pixels in $P[\ell..r]$ have the same colour. (So in the above example, a query for $i = 7$ should return $[4, 9]$.) The run-time should be $O(h)$.

Question 5 [2+3+3+4=12 marks]

Motivation: In class we studied how to search for all points that fall into a query-rectangle. What if we turned this around and asked for all rectangles that are intersected ("pierced") by a query-point?

To keep things simpler, we will only do this in dimension 1, where rectangles becomes segments. So let $S = \{[\ell_i, r_i] : i = 1, \dots, n\}$ be a set of $n \geq 1$ segments that are closed at both endpoints. We want to store these segments such that we can efficiently perform operation *pierce-query*(x), which is given a coordinate x and should return all segments $[\ell_i, r_i]$ in S with $\ell_i \leq x \leq r_i$.

Define a tree (the *S-tree*) as follows:

- The root z stores a *split-coordinate* x_z, which is chosen by taking the upper median of the endpoints of all segments. Thus if the endpoints are $p_0 < p_1 < \cdots < p_{N-1}$ (for some $N \leq 2n$), then x_z is $p_{\lceil N/2 \rceil}$.
- Node z also stores all segments that intersect the split-coordinate x_z, i.e., it stores $M_z = \{[\ell_i, r_i] \in S : \ell_i \leq x_z \leq r_i\}$.
- Let L be all those segments $[\ell_i, r_i]$ that are entirely left of the split-coordinate, i.e., $r_i < x_z$. If L is non-empty, then the left subtree of z is the S-tree for L, otherwise it is empty.

- Let R be all those segments $[\ell_i, r_i]$ that are entirely right of the split-coordinate, i.e., $x_z < \ell_i$. If R is non-empty, then the right subtree of z is the S-tree for R, otherwise it is empty.

a) Consider the following set of segments:

$$[0,1], [0,2], [1,1], [1,2], [1,3], [2,2], [2,3], [2,5], [4,6].$$

Show the S-tree of these segments, and list with each node z the split-coordinate x_z and the set M_z.

b) Show that the height of an S-tree with n segments is in $O(\log n)$.

c) Assume that you have an S-tree with n segments, and $|M_z| \in O(1)$ for all nodes z. Show that *pierce-query*(x) will return $O(\log n)$ segments.

d) Show how to answer *pierce-query*(x) in an S-tree with n segments in $O(\log n + s)$ time, where s is the output-size. Note that we did not tell you how each set M_z is stored; you will need to figure this out yourself. Significant partial credit will be given if the run-time is in $O(\log^2 n + s)$.