Problem Set 4 Solutions

MIT students: This problem set is due in recitation on *Day 13*.

Reading: Chapters 10 and 11

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation instructor and time, the date, and the names of any students with whom you collaborated.

MIT students: Each problem should be done on a separate sheet (or sheets) of three-hole punched paper.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudocode.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Graders will be instructed to take off points for convoluted and obtuse descriptions.

Exercise 4-1. Do exercise 10.1-6 on page 204 of CLRS.

Solution:

Use stack A for ENQUEUE operations, and stack B for DEQUEUE operations. Simulate ENQUEUE by pushing new element onto stack A. Simulate DEQUEUE by popping top element from stack B, but if stack B is empty when a DEQUEUE is requested, first empty stack A into stack B by popping elements one at a time from stack A and pushing them onto stack B. (Note that copying the stack reverses its order, so the oldest element is then on top and can be removed with DEQUEUE.)

ENQUEUE takes time $O(1)$. In the average case, DEQUEUE also takes time $O(1)$, but we could get unlucky and have to transfer n elements from one stack to another, so it is worst case $O(n)$. (However, we can *amortize* the cost of the transfer over all of the ENQUEUE and DEQUEUE operations. It is clear that each element must be popped from A and pushed onto B exactly once, so an amortized analysis gives an average worst case time of $O(1)$ for DEQUEUE.)

Exercise 4-2. Do exercise 10.2-4 on page 208 of CLRS.

Solution:

Store the value you're looking for in the sentinel.

Exercise 4-3. Do exercise 10.3-4 on page 213 of CLRS.

Solution:

Use a variable m, that indicates the number of elements currently in the list, and always keep the list in array locations 1 through m. Allocating a new object is accomplished by using array entry $A[m+1]$. Whenever an object that occupies one of the locations 1 through m is freed, take the object at location m and move it there, thus freeing entry m. This way we guarantee that the list is always compact.

Exercise 4-4. Suppose we hash elements of a set U of keys into m slots. Show that if $|U| > (n-1)m$, there is a subset of U of size n consisting of keys that all hash to the same slot, so that the worst-case searching time for hashing with chaining is $\Theta(n)$.

Solution:

Mapping $(n-1)m+1$ keys into a table of size m must result in at least one slot with n keys or more (pigeonhole principle): if each slot held at most $n-1$ keys, there would only be at most $(n-1)m$ keys. The $(n-1)m+1$th key would have to go in some slot which already had $n-1$ keys. Therefore, the worst-case searching time for hashing with chaining is $\Theta(n)$.

Exercise 4-5. Do exercise 11.3-3 on page 236 of CLRS.

Solution:

All permutations can be generated by a sequence of two character interchanges. Thus it suffices to show that if two arbitrary characters i and j are switched, then the values hash to the same place.

Now consider two numbers x and y which have characters i and j interchanged. w.l.o.g., say $i > j$.

$$\begin{aligned}
x - y &= (x_i - y_i)(m+1)^{(i-1)} + (x_j - y_j)(m+1)^{(j-1)} \\
&= (x_i - x_j)(m+1)^{(i-1)} - (x_i - x_j)(m+1)^{(j-1)} \\
&= (x_i - x_j)((m+1)^{(i-1)} - (m+1)^{(j-1)}) \\
&= (x_i - x_j)(m+1)^{(j-1)}((m+1)^{(i-j)} - 1) \\
&= (x_i - x_j)(m+1)^{(j-1)}((m+1) - 1)\sum_{k=0}^{i-j-1}(m+1)^k \\
&\equiv 0 \bmod m
\end{aligned}$$

Problem 4-1. Comparisons among dynamic sets

For each type of dynamic set in the following table, what is the asymptotic running time for each operation listed, in terms of the number of elements n?

For operations that have not been explicitly defined, consider how you would implement the operation given the data structure. You do not need to give the algorithm, just the running time. State any assumptions that you make.

Assume that the hash tables resolve collisions by chaining with doubly linked lists.

	unsorted singly linked list, worst-case	sorted doubly linked list, worst-case	min-heap, worst-case	hash table, worst-case	hash table, average-case
SEARCH(L, k)					
INSERT(L, x)					
DELETE(L, x)					
SUCCESSOR(L, x)					
MINIMUM(L)					
MAXIMUM(L)					

Solution:

	unsorted singly linked list, worst-case	sorted doubly linked list, worst-case	min-heap, worst-case	hash table, worst-case	hash table, average-case
SEARCH(L, k)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
INSERT(L, x)	$O(1)$	$O(n)$	$O(lgn)$	$O(1)$ or $O(n)$	$O(1)$
DELETE(L, x)	$O(n)$	$O(1)$	$O(lgn)$	$O(1)$	$O(1)$
SUCCESSOR(L, x)	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
MINIMUM(L)	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$
MAXIMUM(L)	$O(n)$	$O(n)$ or $O(1)$	$O(n)$	$O(n)$	$O(n)$

Doubly-linked sorted lists can find the maximum in constant-time if they maintain a $tail$ attribute, or are circular; otherwise, they must scan through the entire list to find the end, taking $O(n)$.

Inserting into a hash table takes worst-case $O(n)$ if you want to ensure there are no duplicate entries, because you have to do a search first. Otherwise, it's $O(1)$.

Problem 4-2. k-universal hashing and authentication

Let $\mathcal{H}$ be a class of hash functions in which each hash function $h \in \mathcal{H}$ maps the universe U of keys to $\{0, 1, \ldots, m-1\}$. We say that $\mathcal{H}$ is ***k-universal*** if, for every fixed sequence of k distinct keys $\left\langle x^{(1)}, x^{(2)}, \ldots, x^{(k)} \right\rangle$ and for any h chosen at random from $\mathcal{H}$, the sequence $\left\langle h(x^{(1)}), h(x^{(2)}), \ldots, h(x^{(k)}) \right\rangle$ is equally likely to be any of the m^k sequences of length k with elements drawn from $\{0, 1, \ldots, m-1\}$.

(a) Show that if the family $\mathcal{H}$ of hash functions is 2-universal, then it is universal.

Solution:

If $\mathcal{H}$ is 2-universal then for any two fixed keys $x \neq y$, the sequence $\langle x, y \rangle$ is equally likely to be any sequence in $\{0, 1, \ldots, m-1\}^2$. Therefore, as h varies over $\mathcal{H}$, the number of collisions ($h(x) = h(y)$) is $(1/m)\,|\mathcal{H}|$, and $\mathcal{H}$ is universal.

(b) Suppose that the universe U is the set of n-tuples of values drawn from $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where p is prime. Consider an element $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle \in U$. For any n-tuple $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in U$, define the hash function h_a by

$$h_a(x) = \left(\sum_{j=0}^{n-1} a_j x_j \right) \bmod p \,.$$

Let $\mathcal{H} = \{h_a\}$. This is the family of hash functions shown in lecture to be universal. Show that $\mathcal{H}$ is not 2-universal. (*Hint:* Find a key for which all hash functions in $\mathcal{H}$ produce the same value.)

Solution:

Suppose we take $x = \langle 0, 0, \ldots, 0 \rangle$, and some fixed $y \in U$. Then for any $a \in U$, $\langle h_a(x), h_a(y) \rangle = \langle 0, h_a(y) \rangle$. This shows that the class $\mathcal{H}$ is not 2-universal, since not all sequences $\langle h_a(x), h_a(y) \rangle$ are equally likely to occur.

(c) Suppose that we modify $\mathcal{H}$ slightly from part (b): For any $a \in U$ and for any $b \in \mathbb{Z}_p$, define

$$h'_{a,b}(x) = \left(\sum_{j=0}^{n-1} a_j x_j + b \right) \bmod p \,.$$

and $\mathcal{H}' = \left\{ h'_{a,b} \right\}$. Argue that $\mathcal{H}'$ is 2-universal. (*Hint:* Consider fixed $x \in U$ and $y \in U$, with $x_i \neq y_i$ for some i. What happens to $h'_{a,b}(x)$ and $h'_{a,b}(y)$ as a_i and b range over $\mathbb{Z}_p$?)

Solution:

For each key pair $\langle x, y \rangle$, $x \neq y$, we wish to show that all value pairs $\left\langle h'_{a,b}(x), h'_{a,b}(y) \right\rangle$ are equally likely to occur when h is chosen randomly from $\mathcal{H}$ – that is, when $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ and b are chosen randomly.

If $x \neq y$, we must have $x_i \neq y_i$ for some i. Assume w.l.o.g. that $i = 0$. We define

$$\alpha = h'_{a,b}(x), \quad \beta = h'_{a,b}(y),$$

and

$$X = \sum_{j=1}^{n-1} a_j x_j, \quad Y = \sum_{j=1}^{n-1} a_j y_j.$$

This gives us the equations

$$\begin{aligned} \alpha &= (a_0 x_0 + b + X) \bmod p, \text{ and} \\ \beta &= (a_0 y_0 + b + Y) \bmod p. \end{aligned}$$

Since $x_0 \neq y_0$ and p is prime, there is a unique solution to the above equations for a_0 and b in terms of α and β. To see this more explicitly, consider that if we want to be able to generate all possible pairs $\langle \alpha, \beta \rangle$, then it is sufficient to be able to independently control α and $\alpha - \beta$. We have

$$\begin{aligned} \alpha - \beta &\equiv a_0 (x_0 - y_0) + X - Y \pmod p, \text{ so} \\ a_0 &= (\alpha - \beta - X + Y)(x_0 - y_0)^{-1} \bmod p. \end{aligned}$$

We know that $(x_0 - y_0)^{-1}$ exists and is unique because $x_0 \neq y_0$ and p is prime. So we may make $\alpha - \beta$ whatever we want by choosing a particular a_0. Having done so, we may make α whatever we want by choosing a particular b: this simply applies an identical offset to both α and β, leaving their difference $\bmod p$ the same.

Therefore, for any given $a_1 \ldots a_{n-1}$, we may find a hash function $h'_{a,b}$ which generates any possible $\langle \alpha, \beta \rangle$ by choosing the right a_0 and b. Since there are p^2 possible choices for a_0 and b, and also p^2 possible values for $\langle \alpha, \beta \rangle$, each $\langle \alpha, \beta \rangle$ is generated by exactly one choice of a_0 and b. This is true for all p^{n-1} choices of $a_1 \ldots a_{n-1}$. Therefore, there are exactly p^{n-1} functions $h'_{a,b}$ which generate each value pair $\langle \alpha, \beta \rangle$. All value pairs are then equally likely when $h'_{a,b}$ is chosen randomly, so $\mathcal{H}'$ is 2-universal.

(d) Suppose that Alice and Bob secretly agree on a hash function h from a 2-universal family $\mathcal{H}$ of hash functions. Each $h \in \mathcal{H}$ maps from a universe of keys U to $\mathbb{Z}_p$, where p is prime. Later, Alice sends a message m to Bob over the Internet, where $m \in U$. She authenticates this message to Bob by also sending an authentication tag $t = h(m)$, and Bob checks that the pair (m, t) he receives satisfies $t = h(m)$. Suppose that an adversary intercepts (m, t) en route and tries to fool Bob by replacing the pair with a different pair (m', t'). Argue that the probability that the adversary succeeds in fooling Bob into accepting (m', t') is at most $1/p$, no matter how much computing power the adversary has, even if the adversary knows the family $\mathcal{H}$ of hash functions used.

Solution:

For any key pair $\langle m, m' \rangle$, all hash value pairs $\langle h(m), h(m') \rangle$ are equally likely when h is chosen at random. This is what it means for $\mathcal{H}$ to be 2-universal. In particular, all p pairs $\langle t, h(m') \rangle$ are equally likely. So even if the adversary knows $\mathcal{H}$, seeing (m, t) tells him nothing about $h(m')$. Since the probabilities for the p cases are equal and must sum to 1, all possible $h(m')$s have probability $1/p$, and the adversary can do no better than guessing.