CPS 230 Homework 6

Haoying

October 20, 2002

1.

a. The probability $p = \Pi_{j=1}^{k} P[h(i) = h(j)] = \left(\frac{i-1}{m}\right)^k$
$i \leq n, n \leq m/2 \Rightarrow p < (n/m)^k \leq 2^{-k}$

b. Use the above result, $p \leq 2^{-k} = 2^{-2\lg n} = 1/n^2$
So $p \leq 1/n^2$.

c. $X > 2\lg n \Rightarrow (X_1 > 2\lg n) \bigcup (X_2 > 2\lg n) \bigcup \cdots \bigcup (X_n > 2\lg n)$

$$\begin{aligned}
P[X > 2\lg n] &= P[(X_1 > 2\lg n) \bigcup (X_2 > 2\lg n) \bigcup \cdots \bigcup (X_n > 2\lg n)] \\
&\leq P[X_1 > 2\lg n] + \cdots + P[X_n > 2\lg n] \\
&\leq n\frac{1}{n^2} \\
&= 1/n
\end{aligned}$$

So $Pr\{X > 2\lg n\} \leq 1/n$.

d. Let l be the length of the probe sequence.

$$\begin{aligned}
E[X] &= \sum_{l=0}^{n} l \cdot Pr[X = l] \qquad (l \leq n) \\
&= \sum_{l=0}^{2\lg n} l \cdot Pr[X = l] + \sum_{l=2\lg n+1}^{n} l \cdot Pr[X = l] \\
&= \sum_{l=1}^{2\lg n} Pr[X = l] + \sum_{l=2}^{2\lg n} Pr[X = l] + \cdots + \sum_{l=2\lg n}^{2\lg n} Pr[X = l] + \\
&\quad (2\lg n + 1) \sum_{l=2\lg n+1}^{n} Pr[X = l] + \sum_{l=2\lg n+2}^{n} Pr[X = l] + \sum_{l=2\lg n+3}^{n} Pr[X = l] + \\
&\quad \cdots + \sum_{l=n}^{n} Pr[X = l]
\end{aligned}$$

Because $\sum_{l=0}^{2\lg n} Pr[X = l] < Pr[X < 2\lg n] < 1$, and $\sum_{l=2\lg n+1}^{n} Pr[X = l] < Pr[X > 2\lg n] < 1/n$, we have:

$$\begin{aligned} E[X] &\le 2\lg n + \frac{1}{n}(2\lg n + 1) + \frac{1}{n}(n - 2\lg n - 1) \\ &= 2\lg n + 1 \end{aligned}$$

So $E[X] = O(\lg n)$.

2.

a. $c_1 = c_2 = 1/2$
$h(k,i) = (h'(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod m$
$h(k,i+1) - h(k,i) = (i+1) \bmod m$
In the program, every time we increase position i by $(i+1) \bmod m$. So it is quadratic probing.

b. If key k is not in the hash table, this algorithm will exam every table position.

Let $f(i) = h'(k) + \frac{1}{2}i + \frac{1}{2}i^2$.
$h(k,i) = (h'(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod m$=$f(i) \bmod m$.
If we can prove that $h(k,0)$ to $h(k,m-1)$ are all different, then it is obvious the algorithm will search all the table position if k isn't in the table.

If there are two item $h(k,a)$ and $h(k,b)$ $(0 \le a, b \le m-1)$ which are the same, assume $f(b) > f(a)$, so $f(b) - f(a) \bmod m$ =0.
$f(b) - f(a) = \frac{1}{2}(b-a)(1+b-a)$. One of $(b-a)$ and $(1+b-a)$ must be even and the other is odd, whether a, b is even or odd.
If (b-a) is even, then $m|\frac{1}{2}(b-a)$,but $\frac{1}{2}(b-a) < m$, so $m \nmid \frac{1}{2}(b-a)$.
If (1+b+a) is even, then $m \mid \frac{1}{2}(1+b+a)$, but $\frac{1}{2}(1+b+a) < \frac{2m-2}{2} < m-1$, so $m \nmid \frac{1}{2}(1+b+a)$.
In both the situations, there are conflicts, so the assumption we made is wrong. $h(k,0)$ to $h(k,m-1)$ are all different, and in the worst case, we need to search all the table position.

3.We need a double linked free list. At first, all the slots in the hash table are in the free list.

When adding an element k to the hash table, we first check its hash slot $h(k)$, if it is not occupied, we could use the slot, mark the flag of the slot as 'occupied' and delete it from the free list. If it is occupied by t, compute $h(t)$, if $h(k) = h(t)$, use the first slot in the free list to store k, and delete the slot from the free list. Use a chain to keep track of those conflicting elements, if t points to other element that is conflict with t, search the chain until finding the last element and let it point to k. If $h(k) \ne h(t)$, we must move t to the first slot in the free list. We should try to maitain the chain. Compute $h(t)$ and search the chain, if $p \to link = t$, let $p \to link = t \to link$, find the last element in the chain, and let it point to t. At last let k occupy the slot $h(k)$. In the worst case of add, we must search all the chain, the expected time of unsuccessful search is $O(1+\alpha)$(Theorem 11.1), so the cost of add is $O(1)$ when α is a constant.

When searching, according to Theorem 11.1 and 11.2, the cost is $O(1+\alpha)$ where $\alpha = n/m$, the expected search time is $O(1)$.

When deleting, first do search, because all the conflicting elements are linked, it will cost $O(1)$ time to search, then delete it and add the slot to the end of the free list. The cost of delete is $O(1)$.

4.

(a) Define $\Phi(D) = |3(\#\text{elements in table}) - 2(\text{size of table})|$.
$\Phi(D_0) = 0, \Phi(D_n) \geq 0 (n > 0)$
When the array is not full,

$$\begin{aligned}
\bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\
&= 1 + |3element_{k+1} - 2size_{k+1}| - |3element_k - 2size_k| \\
&= 1 + 3(element_k + 1) - 2size_k - 3element_k + 2size_k \\
&= 4
\end{aligned}$$

When the array is full,

$$\begin{aligned}
\bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\
&= (1+n) + |3element_{k+1} - 2size_{k+1}| - |3element_k - 2size_k| \\
&= (1+n) + 3(element_k + 1) - 2(\frac{3}{2}size_k) - 3element_k + 2size_k \\
&= (1+n) + 3(n+1) - 3n - 3n + 2n \\
&= 4
\end{aligned}$$

So the amortized cost of a push is $O(1)$.

(b) No. If the size of the array is n, and it is 2/3 full, consider the following operations:pop,push,push,pop,pop,push,push,$\cdots$. When we pop from the array that is 2/3 full, it will shrink form size n to size $\frac{2n}{3}$. After one push, the array is full. Then we do another push, the array size will be enlarged from $\frac{2n}{3}$ to n. After one pop, the array is 2/3 full again. Do another pop will cause the array to shrink again.
Assume that decreasing the size of the array from size n to $\frac{2n}{3}$ costs $\frac{2n}{3}$. The actual cost of the above operations is $1 + \frac{2n}{3}, 1, 1 + \frac{2n}{3}, 1, 1 + \frac{2n}{3}, \cdots$. The cost of each pair of push is $\Omega(n)$, and the cost of each pair of pop is $\Omega(n)$. So the amortized cost is $\Omega(n)$, not $O(1)$.

(c) Yes.
Define $\Phi(D) = |\frac{21}{2}(\#\text{element in table}) - 8(\text{size of table})|$
When the array is 2/3 full, and we do a pop

$$\bar{c}_k = c_k + \Phi(D_{k+1}) - \Phi(D_k)$$

$$
\begin{aligned}
&= (1+\frac{2n}{3}) + |\frac{21}{2}element_{k+1} - 8size_{k+1}| - |\frac{21}{2}element_k - 8size_k| \\
&= (1+\frac{2n}{3}) + \frac{21}{2}(\frac{2}{3}n - 1) - 8\frac{5n}{6} + \frac{21}{2}\frac{2n}{3} - 8n \\
&= -\frac{19}{2}
\end{aligned}
$$

When the array is not 2/3 full, and we do a pop

$$
\begin{aligned}
\bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\
&= 1 + |\frac{21}{2}element_{k+1} - 8size_{k+1}| - |\frac{21}{2}element_k - 8size_k| \\
&= 1 + |\frac{21}{2}(element_k - 1) - 8size_k| - |\frac{21}{2}element_k - 8size_k| \\
&= -\frac{9}{2}
\end{aligned}
$$

The amortized cost of a pop is O(1).
When the array is full, and we do a push

$$
\begin{aligned}
\bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\
&= (1+n) + |\frac{21}{2}element_{k+1} - 8size_{k+1}| - |\frac{21}{2}element_k - 8size_k| \\
&= (1+n) + |\frac{21}{2}(n+1) - 8\frac{3n}{2}| - |\frac{21}{2}n - 8n| \\
&= -\frac{19}{2}
\end{aligned}
$$

When the array is not full, and we do a push

$$
\begin{aligned}
\bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\
&= 1 + |\frac{21}{2}element_{k+1} - 8size_{k+1}| - |\frac{21}{2}element_k - 8size_k| \\
&= 1 + |\frac{21}{2}(element_k + 1) - 8size_k| - |\frac{21}{2}element_k - 8size_k| \\
&= \frac{23}{2}
\end{aligned}
$$

The amortized cost of a push is O(1).
This yields an amortized cost O(1) per operation.

5.When we need to do an *Enqueue*, we push the new element in Stack A. When we need to do a *Dequeue*, if Stack B is not empty, we pop from Stack B; if Stack B is empty, we pop all the elements in Stack A into Stack B, and pop from Stack B.
Define $\Phi(D) = |2(\#\text{element in Stack A}) - (\#\text{element in Stack B})|$.
$\Phi(D_0) = 0, \Phi(D_n) \geq 0 (n > 0)$
Assume n is the number of elements in Stack A, m is the number of elements

in Stack B.
When we do *Enqueue*

$$\begin{aligned}\bar{c}_k &= c_k + \Phi(D_{k+1} - \Phi D_k) \\ &= 1 + |2(n+1) - m| - |2n - m| \\ &= 3\end{aligned}$$

When Stack B is not empty, and we do *Dequeue*

$$\begin{aligned}\bar{c}_k &= c_k + \Phi(D_{k+1} - \Phi D_k) \\ &= 1 + |2n - (m-1)| - |2n - m| \\ &= 2\end{aligned}$$

When Stack B is empty, and we do *Dequeue*

$$\begin{aligned}\bar{c}_k &= c_k + \Phi(D_{k+1} - \Phi D_k) \\ &= n + 1 + |0 - (n-1)| - |2n - 0| \\ &= 0\end{aligned}$$

The amortized cost of *Enqueue* and *Dequeue* is O(1).

6.

(a) Use a list to implement *Enqueue* and *Dequeue*, *Enqueue* adds an element at the head of the list, and *Dequeue* returns the end of the list and deletes it from the list, the running time is $O(1)$. *Minimum* should go through the list and return the minimum, the running time is $O(n)$.

(b) Use a list and a minimal heap, *Enqueue* and *Dequeue* is done in the list, each element in the list has a pointer pointing to the corresponding node in the heap. *Enqueue* inserts a element at the head of the list, inserts a node to the heap, and does rotation to maintain the heap a minimum heap. This costs $O(1) + O(\lg n) = O(\lg n)$ time. *Dequeue* returns the end of the list and deletes it from the list and the heap, and does rotation to maintain the property of the heap. This also costs $O(1) + O(\lg n) = O(\lg n)$ time. *Minimum* is to return the top of the heap, it is $O(1)$ time.

(c) Use two lists, list A and list B. List A is to implement *Enqueue* and *Dequeue*. List B stores increasing sequence of the elements, so *Minimum* returns the head of List B, the cost is $O(1)$. *Enqueue* adds the new element p at the head of list A, and compares the element p with the end of list B, if $p < B.end$, compares p with $B.end - 1, B.end - 2, \cdots$, until finding a suitable position for p such that List B is in increasing order, then deletes the element in List B that are greater than p, since these elements q_i won't be the result of *Minimum*, q_i is popped earlier than p, so when p is in List A, the minimum can't be q_i. If $p > B.end$, add p as the end of List B. *Dequeue* returns and deletes the element at the

end of List A, and compare $A.end$ with $B.head$, if $A.end = B.head$, the minimum is popped, and delete $B.head$, if $A.end \neq B.head$, do nothing to List B. The cost of $Dequeue$ is $O(1)$.
Now compute the amortized cost of $Enqueue$. The actual cost is $(1+n)$ where n is the number of elements in List B that are greater than the new element.
Define $\Phi(D)$ =length of List B= L.
$\Phi(D_0) = 0, \Phi(D_n) \geq 0$

$$\begin{aligned} \bar{c}_k &= c_k + \Phi(D_{k+1}) - \Phi(D_k) \\ &= 1 + n + L - n + 1 + L \\ &= 2 \end{aligned}$$

The amortized cost of all the three operations is O(1).