Homework Solutions – Unit 3, Chapter 11

CMPSC 465 Spring 2013

Exercise 11.1-1

Suppose that a dynamic set *S* is represented by a direct-address table *T* of length *m*. Describe a procedure that finds the maximum element of *S*. What is the worst-case performance of your procedure?

Solution:

We can do a linear search to find the maximum element in *S* as follows:

Pre-condition: table *T* is not empty; $m \in Z^+$, $m \geq 1$.
Post-condition: FCTVAL == maximum value of dynamic set stored in T.

```
FindMax ($T$, $m$)
{
    $max = -\infty$

    for $i = 1$ to $m$
    {
        if $T[i]$ != NIL && $max < T[i]$
            $max = T[i]$
    }
    return $max$
}
```

In the worst-case searching the entire table is needed. Thus the procedure must take $O(m)$ time.

Exercise 11.2-1

Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$?

Solution:

For each pair of keys k, l, where $k \neq l$, define the indicator random variable $X_{lk} = \mathrm{I}\{h(k) = h(l)\}$. Since we assume simple uniform hashing, $\Pr\{X_{lk} = 1\} = \Pr\{h(k) = h(l)\} = 1/m$, and so $\mathrm{E}[X_{lk}] = 1/m$.

Now define the random variable Y to be the total number of collisions, so that $Y = \sum_{k \neq l} X_{kl}$. The expected number of collisions is

$$\begin{aligned}
E[Y] &= E[\sum_{k\neq l} X_{kl}] && \text{since } Y = \sum_{k\neq l} X_{kl} \\
&= \sum_{k\neq l} E[X_{kl}] && \text{by linearity of expectation} \\
&= \sum_{k\neq l} \frac{1}{m} && \text{since } E[X_{kl}] = \frac{1}{m} \\
&= \binom{n}{2}\frac{1}{m} && \text{by choosing } k \text{ and } l \text{ out of } n \text{ keys} \\
&= \frac{n(n-1)}{2}\frac{1}{m} && \text{by } \binom{n}{r} = \frac{n!}{r!(n-r)!}, \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} \\
&= \frac{n(n-1)}{2m}
\end{aligned}$$

Exercise 11.2-2

Demonstrate what happens when we insert the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \bmod 9$.

Solution: omitted.

Exercise 11.2-3

Professor Marley hypothesizes that he can obtain substantial performance gains by modifying the chaining scheme to keep each list in sorted order. How does the professor's modification affect the running time for successful searches, unsuccessful searches, insertions, and deletions?

Solution:

- Successful searches: $\Theta(1 + \alpha)$, which is identical to the original running time. The element we search for is equally likely to be any of the elements in the hash table, and the proof of the running time for successful searches is similar to what we did in the lecture.
- Unsuccessful searches: 1/2 of the original running time, but still $\Theta(1 + \alpha)$, if we simply assume that the probability that one element's value falls between two consecutive elements in the hash slot is uniformly distributed. This is because the value of the element we search for is equally likely to fall between any consecutive elements in the hash slot, and once we find a larger value, we can stop searching. Thus, the running time for unsuccessful searches is a half of the original running time. Its proof is similar to what we did in the lecture.
- Insertions: $\Theta(1 + \alpha)$, compared to the original running time of $\Theta(1)$. This is because we need to find the right location instead of the head to insert the element so that the list remains sorted. The operation of insertions is similar to the operation of unsuccessful searches in this case.
- Deletions: $\Theta(1 + \alpha)$, same as successful searches.

Exercise 11.3-3

Consider a version of the division method in which $h(k) = k \bmod m$, where $m = 2^p - 1$ and k is a character string interpreted in radix 2^p. Show that if we can derive string x from string y by permuting its characters, then x and y hash to the same value. Give an example of an application in which this property would be undesirable in a hash function.

Solution:
First, we observe that we can generate any permutation by a sequence of interchanges of pairs of characters. One can prove this property formally, but informally, consider that both heapsort and quicksort work by interchanging pairs of elements and that they have to be able to produce any permutation of their input array. Thus, it suffices to show that if string x can be derived from string y by interchanging a single pair of characters, then x and y hash to the same value.

Let x_i be the ith character in x, and similarly for y_i. We can interpret x in radix 2^p as $\sum_{i=0}^{n-1} x_i 2^{ip}$, and interpret y as $\sum_{i=0}^{n-1} y_i 2^{ip}$. So

$$h(x) = \left(\sum_{i=0}^{n-1} x_i 2^{ip}\right) \bmod (2^p - 1). \text{ Similarly, } h(y) = \left(\sum_{i=0}^{n-1} y_i 2^{ip}\right) \bmod (2^p - 1).$$

Suppose that x and y are identical strings of n characters except that the characters in positions a and b are interchanged:

$$x_a = y_b \text{ and } y_a = x_b. \tag{1}$$

Without loss of generality, let $a > b$. We have:

$$h(x) - h(y) = \left(\sum_{i=0}^{n-1} x_i 2^{ip}\right) \bmod (2^p - 1) - \left(\sum_{i=0}^{n-1} y_i 2^{ip}\right) \bmod (2^p - 1) \tag{2}$$

Since $0 \le h(x), h(y) < 2^p - 1$, we have that $-(2^p - 1) < h(x) - h(y) < 2^p - 1$. If we show that $(h(x) - h(y)) \bmod (2^p - 1) = 0$, then $h(x) = h(y)$. To prove $(h(x) - h(y)) \bmod (2^p - 1) = 0$, we have:

$$\begin{aligned}
(h(x) - h(y)) \bmod (2^p - 1) &= \left(\left(\sum_{i=0}^{n-1} x_i 2^{ip}\right) \bmod (2^p - 1) - \left(\sum_{i=0}^{n-1} y_i 2^{ip}\right) \bmod (2^p - 1)\right) \\
&\quad \bmod (2^p - 1) && \text{by (2)} \\
&= \left(\sum_{i=0}^{n-1} x_i 2^{ip} - \sum_{i=0}^{n-1} y_i 2^{ip}\right) \bmod (2^p - 1) && \text{by relation in footnote}^{1} \\
&= ((x_a 2^{ap} + x_b 2^{bp}) - (y_a 2^{ap} + y_b 2^{bp})) \bmod (2^p - 1) && \text{as } x \text{ and } y \text{ are identical strings of } n \text{ characters except that chars. in positions } a \text{ and } b \text{ are interchanged} \\
&= ((x_a 2^{ap} + x_b 2^{bp}) - (x_b 2^{ap} + x_a 2^{bp})) \bmod (2^p - 1) && \text{as } x_a = y_b,\ x_b = y_a \text{ see (1)} \\
&= ((x_a - x_b)2^{ap} + (x_b - x_a)2^{bp}) \bmod (2^p - 1) && \text{by combining like terms} \\
&= ((x_a - x_b)2^{ap} - (x_a - x_b)2^{bp}) \bmod (2^p - 1) && \text{as } (x_b - x_a) = -(x_a - x_b) \\
&= ((x_a - x_b)(2^{ap} - 2^{bp})) \bmod (2^p - 1) && \text{by factoring out } (x_a - x_b) \\
&= ((x_a - x_b)(2^{ap}(2^{bp}/2^{bp}) - 2^{bp})) \bmod (2^p - 1) && \text{by multiplication by } 2^{bp}/2^{bp} = 1 \\
&= ((x_a - x_b)2^{bp}(2^{(a-b)p} - 1)) \bmod (2^p - 1) && \text{by factoring out } 2^{bp} \\
&= \left((x_a - x_b)2^{bp}\left(\sum_{i=0}^{a-b-1} 2^{ip}\right)(2^p - 1)\right) \bmod (2^p - 1) && \text{by substituting } [2^{(a-b)p} - 1]^{2} \\
&= 0 && \text{since one factor is } 2^p - 1
\end{aligned}$$

---

1 Consider the congruence relation: $(m_1 \text{ o } m_2) \bmod n = ((m_1 \bmod n) \text{ o } (m_2 \bmod n)) \bmod n$, where o is $+$, $-$, or $*$

2 Consider the equation $\sum_{i=0}^{a-b-1} 2^{ip} = \dfrac{2^{(a-b)p} - 1}{2^p - 1}$ (geometric series) and multiplying both sides by $2^p - 1$ to get

$$2^{(a-b)p} - 1 = \left(\sum_{i=0}^{a-b-1} 2^{ip}\right)(2^p - 1)]$$

Because we deduced earlier that $(h(x) - h(y)) \bmod (2p - 1) = ((x_a - x_b)2^{bp}(2^{(a-b)p} - 1)) \bmod (2^p - 1)$ and have shown here that $((x_a - x_b)2^{bp}(2^{(a-b)p} - 1)) \bmod (2^p - 1) = 0$, we can conclude $(h(x) - h(y)) \bmod (2^p - 1) = 0$, and thus $h(x) = h(y)$. So we have proven that if we can derive string x from string y by permuting its characters, then x and y hash to the same value.

Examples of applications:
A dictionary which contains words expressed by ASCII code can be one of such example when each character of the dictionary is interpreted in radix $2^8 = 256$ and $m = 255$. The dictionary, for instance, might have words "STOP," "TOPS," "SPOT," "POTS," all of which are hashed into the same slot.

Exercise 11.3-4

Consider a hash table of size $m = 1000$ and a corresponding hash function $h(k) = \lfloor m(kA \bmod 1) \rfloor$ for $A = \frac{\sqrt{5}-1}{2}$. Compute the locations to which the keys 61, 62, 63, 64, and 65 are mapped.

Solution:

$$\begin{aligned} h(61) &= \left\lfloor 1000\left(61\left(\frac{\sqrt{5}-1}{2}\right) \bmod 1\right)\right\rfloor \\ &= \lfloor 1000(37.700 \bmod 1) \rfloor \\ &= \lfloor 1000 \times 0.700 \rfloor \\ &= 700 \end{aligned}$$

$$\begin{aligned} h(62) &= \left\lfloor 1000\left(62\left(\frac{\sqrt{5}-1}{2}\right) \bmod 1\right)\right\rfloor \\ &= \lfloor 1000 \times (38.318 \bmod 1) \rfloor \\ &= \lfloor 1000 \times 0.318 \rfloor \\ &= 318 \end{aligned}$$

$$\begin{aligned} h(63) &= \left\lfloor 1000\left(63\left(\frac{\sqrt{5}-1}{2}\right) \bmod 1\right)\right\rfloor \\ &= \lfloor 1000 \times (38.936 \bmod 1) \rfloor \\ &= \lfloor 1000 \times 0.936 \rfloor \\ &= 936 \end{aligned}$$

$$\begin{aligned} h(64) &= \left\lfloor 1000\left(64\left(\frac{\sqrt{5}-1}{2}\right) \bmod 1\right)\right\rfloor \\ &= \lfloor 1000 \times (39.554 \bmod 1) \rfloor \\ &= \lfloor 1000 \times 0.554 \rfloor \\ &= 554 \end{aligned}$$

$$\begin{aligned} h(65) &= \left\lfloor 1000\left(65\left(\frac{\sqrt{5}-1}{2}\right) \bmod 1\right)\right\rfloor \\ &= \lfloor 1000 \times (40.172 \bmod 1) \rfloor \\ &= \lfloor 1000 \times 0.172 \rfloor \\ &= 172 \end{aligned}$$

Exercise 11.4-1

Consider inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h'(k) = k$ and $h_2'(k) = 1 + (k \bmod (m - 1))$.

Solution:

Linear Probing
With linear probing, we use the hash function $h(k, i) = (h'(k) + i) \bmod m = (k + i) \bmod m$. Consider hashing each of the following keys:

1) Hashing 10:
   $h(10, 0) = (10 + 0) \bmod 11= 10$. Thus we have $T[10] = 10$.
2) Hashing 22:
   $h(22, 0) = (22 + 0) \bmod 11 = 0$. Thus we have $T[0] = 22$.
3) Hashing 31:
   $h(31, 0) = (31 + 0) \bmod 11 = 9$. Thus we have $T[9] = 31$.
4) Hashing 4:
   $h(4, 0) = (4 + 0) \bmod 11 = 4$. Thus we have $T[4] = 4$.
5) Hashing 15:
   $h(15, 0) = (15 + 0) \bmod 11 = 4$, collision!
   $h(15, 1) = (15 + 1) \bmod 11 = 5$. Thus we have $T[5] = 15$.
6) Hashing 28:
   $h(28, 0) = (28 + 0) \bmod 11 = 6$. Thus we have $T[6] = 28$.
7) Hashing 17:
   $h(17, 0) = (17 + 0) \bmod 11 = 6$, collision!
   $h(17, 1) = (17 + 1) \bmod 11 = 7$. Thus we have $T[7] = 17$.
8) Hashing 88:
   $h(88, 0) = (88 + 0) \bmod 11 = 0$, collision!
   $h(88, 1) = (88 + 1) \bmod 11 = 1$. Thus we have $T[1] = 88$.
9) Hashing 59:
   $h(59, 0) = (59 + 0) \bmod 11 = 4$, collision!
   $h(59, 1) = (59 + 1) \bmod 11 = 5$, collision!
   $h(59, 2) = (59 + 2) \bmod 11 = 6$, collision!
   $h(59, 3) = (59 + 3) \bmod 11 = 7$, collision!
   $h(59, 4) = (59 + 4) \bmod 11 = 8$. Thus we have $T[8] = 59$.

The final hash table is shown as:

key	22	88			4	15	28	17	59	31	10
index	1	2	3	4	5	6	7	8	9	10	11

Quadratic Probing
With quadratic probing, and $c_1 = 1$, $c_2 = 3$, we use the hash function $h(k, i) = (h'(k) + i + 3i^2) \bmod m = (k + i + 3i^2) \bmod m$. Consider hashing each of the following keys:

1) Hashing 10:
   $h(10, 0) = (10 + 0 + 0) \bmod 11 = 10$. Thus we have $T[10] = 10$.
2) Hashing 22:
   $h(22, 0) = (22 + 0 + 0) \bmod 11 = 0$. Thus we have $T[0] = 22$.
3) Hashing 31:
   $h(31, 0) = (31 + 0 + 0) \bmod 11 = 9$. Thus we have $T[9] = 31$.
4) Hashing 4:
   $h(4, 0) = (4 + 0 + 0) \bmod 11 = 4$. Thus we have $T[4] = 4$.
5) Hashing 15:
   $h(15, 0) = (15 + 0 + 0) \bmod 11 = 4$, collision!
   $h(15, 1) = (15 + 1 + 3) \bmod 11 = 8$. Thus we have $T[8] = 15$.
6) Hashing 28:
   $h(28, 0) = (28 + 0 + 0) \bmod 11 = 6$. Thus we have $T[6] = 28$.
7) Hashing 17:

$h(17, 0) = (17 + 0 + 0) \bmod 11 = 6$, collision!
$h(17, 1) = (17 + 1 + 3) \bmod 11 = 10$, collision!
$h(17, 2) = (17 + 2 + 12) \bmod 11 = 9$, collision!
$h(17, 3) = (17 + 3 + 27) \bmod 11 = 3$. Thus we have $T[3] = 17$.

8) Hashing 88:
$h(88, 0) = (88 + 0 + 0) \bmod 11 = 0$, collision!
$h(88, 1) = (88 + 1 + 3) \bmod 11 = 4$, collision!
$h(88, 2) = (88 + 2 + 12) \bmod 11 = 3$, collision!
$h(88, 3) = (88 + 3 + 27) \bmod 11 = 8$, collision!
$h(88, 4) = (88 + 4 + 48) \bmod 11 = 8$, collision!
$h(88, 5) = (88 + 5 + 75) \bmod 11 = 3$, collision!
$h(88, 6) = (88 + 6 + 108) \bmod 11 = 4$, collision!
$h(88, 7) = (88 + 7 + 147) \bmod 11 = 0$, collision!
$h(88, 8) = (88 + 8 + 192) \bmod 11 = 2$. Thus we have $T[2] = 88$.

9) Hashing 59:
$h(59, 0) = (59 + 0 + 0) \bmod 11 = 4$, collision!
$h(59, 1) = (59 + 1 + 3) \bmod 11 = 8$, collision!
$h(59, 2) = (59 + 2 + 12) \bmod 11 = 7$. Thus we have $T[7] = 59$.

The final hash table is shown as:

key	22		88	17	4		28	59	15	31	10
index	1	2	3	4	5	6	7	8	9	10	11

Doubling Hashing

With double hashing, we use the hash function:

$h(k, i) = (h'(k) + ih_2'(k)) \bmod m = (k + i(1 + (k \bmod (m - 1)))) \bmod m$.

Consider hashing each of the following keys:

1) Hashing 10:
$h(10, 0) = (10 + 0) \bmod 11 = 10$. Thus we have $T[10] = 10$.

2) Hashing 22:
$h(22, 0) = (22 + 0) \bmod 11 = 0$. Thus we have $T[0] = 22$.

3) Hashing 31:
$h(31, 0) = (31 + 0) \bmod 11 = 9$. Thus we have $T[9] = 31$.

4) Hashing 4:
$h(4, 0) = (4 + 0) \bmod 11 = 4$. Thus we have $T[4] = 4$.

5) Hashing 15:
$h(15, 0) = (15 + 0) \bmod 11 = 4$, collision!
$h(15, 1) = (15 + 1 * h_2'(15)) \bmod 11 = 10$, collision!
$h(15, 2) = (15 + 2 * h_2'(15)) \bmod 11 = 5$. Thus we have $T[5] = 15$.

6) Hashing 28:
$h(28, 0) = (28 + 0) \bmod 11 = 6$. Thus we have $T[6] = 28$.

7) Hashing 17:
$h(17, 0) = (17 + 0) \bmod 11 = 6$, collision!
$h(17, 1) = (17 + 1 * h_2'(17)) \bmod 11 = 3$. Thus we have $T[3] = 17$.

8) Hashing 88:
$h(88, 0) = (88 + 0) \bmod 11 = 0$, collision!
$h(88, 1) = (88 + 1 * h_2'(88)) \bmod 11 = 9$, collision!
$h(88, 2) = (88 + 2 * h_2'(88)) \bmod 11 = 7$. Thus we have $T[7] = 88$.

9) Hashing 59:
$h(59, 0) = (59 + 0) \bmod 11 = 4$, collision!
$h(59, 1) = (59 + 1 * h_2'(59)) \bmod 11 = 3$, collision!
$h(59, 2) = (59 + 2 * h_2'(59)) \bmod 11 = 2$. Thus we have $T[2] = 59$.

The final hash table is shown as:

key	22		59	17	4	15	28	88		31	10
index	1	2	3	4	5	6	7	8	9	10	11