

Solution to Exercise 10.2-7

A $\Theta(n)$ -time non-recursive procedure to reverse a singly linked list of n elements with constant storage is the following:

```
LIST-REVERSE (L)
   $p \leftarrow \text{head}[L]$ 
  if  $p = \text{NIL}$ 
    return error “empty list”
   $q \leftarrow \text{next}[p]$ 
   $r \leftarrow \text{NIL}$ 
  while  $q \neq \text{NIL}$ 
    do  $\text{next}[p] \leftarrow r$ 
        $r \leftarrow p$ 
        $p \leftarrow q$ 
        $q \leftarrow \text{next}[p]$ 
   $\text{next}[p] \leftarrow r$ 
   $\text{head}[L] \leftarrow p$ 
```

Solution to Exercise 10.3-4

Suppose we have a multi-array representation of length n such that the nodes of the doubly linked list with m nodes are compacted in the first m index entries. We organize the remaining $n-m$ entries into a stack representing the free space. The top of the stack is $\text{top}[\text{free}]$, which is initially equal to 1 before the creation of the doubly linked list. Allocating a node is straightforward. However when a node is to be returned to free space, we need to recompact the linked list so that the nodes reside in the first indexed entries, and all the pointers are adjusted appropriately. We do this by moving the node that appears right before the top of the stack free into the location freed and adjust the pointers to that node.

Allocate-Object()

```
If  $\text{top}[\text{free}] > n$  then error “out of space”
  else {  $x = \text{top}[\text{free}]$ ;
         $\text{top}[\text{free}] = \text{top}[\text{free}] + 1$ ;
        return  $x$ ;
        }
```

Free-Object(x)

```
y=top[free]-1;
if (y=x) then {top[free]=y; return }
else
  {if prev[y] is not NIL then next[prv[y]]=x;
   else head[L]=x;
  if next[y] is not NIL then prev[next[y]]=x;
  next[x]=next[y];
  prev[x]=prev[y];
  key[x]=key[y];
  top[free]=y;
}
```

Solution to Exercise 10.4-6

If only two pointers and a Boolean value can be used in each node, we will use one pointer as before to point to the leftmost child $left[x]$, and the other pointer either to point to a sibling unless it is the rightmost child in which case it points to the parent. To distinguish between these two cases, we use the Boolean value (say 0 when the pointer is used for a sibling and 1 when it points to a parent). It is clear that for any node we can explore all its children in time linear in the number of children and we can reach the parent by following the sibling pointer until we reach a node with the Boolean value set to 1 in which case we can reach the parent immediately.

Solution to Exercise 11.2-1

For each pair of keys k, l , where $k \neq l$, define the indicator random variable

$X_{kl} = I\{h(k) = h(l)\}$. Since we assume simple uniform hashing, $\Pr\{X_{kl} = 1\} = \Pr\{h(k) = h(l)\} = 1/m$, and so $E[X_{kl}] = 1/m$.

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

$$\begin{aligned} E[Y] &= E\left[\sum_{k \neq l} X_{kl}\right] \\ &= \sum_{k \neq l} E[X_{kl}] \quad (\text{linearity of expectation}) \\ &= \binom{n}{2} \frac{1}{m} \\ &= \frac{n(n-1)}{2} \cdot \frac{1}{m} \\ &= \frac{n(n-1)}{2m} \end{aligned}$$

Solution to Exercise 11.2-4

The flag in each slot will indicate whether the slot is free.

- A free slot is in the free list, a doubly linked list of all free slots in the table. The slot thus contains two pointers.
- A used slot contains an element and a pointer (possibly NIL) to the next element that hashes to this slot. (Of course, that pointer points to another slot in the table.)

Operations

Insertion:

- If the element hashes to a free slot, just remove the slot from the free list and store the element there (with a NIL pointer). The free list must be doubly linked in order for this deletion to run in $O(1)$ time.
- If the element hashes to a used slot j , check whether the element x already there “belongs” there (its key also hashes to slot j).
 - If so, add the new element to the chain of elements in this slot. To do so, allocate a free slot (e.g., take the head of the free list) for the new element and put this new slot at the head of the list pointed to by the hashed-to slot (j).
 - If not, E is part of another slot’s chain. Move it to a new slot by allocating one from the free list, copying the old slot’s (j ’s) contents (element x and pointer) to the new slot, and updating the pointer in the slot that pointed to j to point to the new slot. Then insert the new element in the now-empty slot as usual. To update the pointer to j , it is necessary to find it by searching the chain of elements starting in the slot x hashes to.

Deletion: Let j be the slot the element x to be deleted hashes to.

- If x is the only element in j (j doesn’t point to any other entries), just free the slot, returning it to the head of the free list.
- If x is in j but there’s a pointer to a chain of other elements, move the first pointed-to entry to slot j and free the slot it was in.
- If x is found by following a pointer from j , just free x ’s slot and splice it out of the chain (i.e., update the slot that pointed to x to point to x ’s successor).

Searching: Check the slot the key hashes to, and if that is not the desired element, follow the chain of pointers from the slot.

All the operations take expected $O(1)$ times for the same reason they do with the version in the book: The expected time to search the chains is $O(1 + \alpha)$ regardless of where the chains are stored, and the fact that all the elements are stored in the table means that $\alpha \leq 1$. If the free list were singly linked, then operations that involved removing an arbitrary slot from the free list would not run in $O(1)$ time.

Solution to Problem 11-1

a. Since we assume uniform hashing, we can use the same observation as is used in Corollary 11.7: that inserting a key entails an unsuccessful search followed by placing the key into the first empty slot found. As in the proof of Theorem 11.6, if we let X be the random variable denoting the number of probes in an unsuccessful search, then $\Pr\{X \geq i\} \leq \alpha^{i-1}$. Since $n \leq m/2$, we have $\alpha \leq 1/2$. Letting $i = k + 1$, we have $\Pr\{X > k\} = \Pr\{X \geq k + 1\} \leq (1/2)^{(k+1)-1} = 2^{-k}$.

b. Substituting $k = 2 \lg n$ into the statement of part (a) yields that the probability that the i th insertion requires more than $k = 2 \lg n$ probes is at most $2^{-2 \lg n} = (2^{\lg n})^{-2} = n^{-2} = 1/n^2$.

c. Let the event A be $X > 2 \lg n$, and for $i = 1, 2, \dots, n$, let the event A_i be $X_i > 2 \lg n$. In part (b), we showed that $\Pr\{A_i\} \leq 1/n^2$ for $i = 1, 2, \dots, n$. From how we defined these events, $A = A_1 \cup A_2 \cup \dots \cup A_n$. Using Boole's inequality, (C.18), we have

$$\begin{aligned} \Pr\{A\} &\leq \Pr\{A_1\} + \Pr\{A_2\} + \dots + \Pr\{A_n\} \\ &\leq n \cdot \frac{1}{n^2} \\ &\leq 1/n \end{aligned}$$

d. We use the definition of expectation and break the sum into two parts:

$$\begin{aligned} E[X] &= \sum_{k=1}^n k \cdot \Pr\{X = k\} \\ &= \sum_{k=1}^{\lceil 2 \lg n \rceil} k \cdot \Pr\{X = k\} + \sum_{k=\lceil 2 \lg n \rceil+1}^n k \cdot \Pr\{X = k\} \\ &\leq \sum_{k=1}^{\lceil 2 \lg n \rceil} \lceil 2 \lg n \rceil \cdot \Pr\{X = k\} + \sum_{k=\lceil 2 \lg n \rceil+1}^n n \cdot \Pr\{X = k\} \\ &= \lceil 2 \lg n \rceil \cdot \sum_{k=1}^{\lceil 2 \lg n \rceil} \Pr\{X = k\} + n \cdot \sum_{k=\lceil 2 \lg n \rceil+1}^n \Pr\{X = k\} \end{aligned}$$

Since X takes on exactly one value, we have that

$$\sum_{k=1}^{\lceil 2 \lg n \rceil} \Pr\{X = k\} = \Pr\{X \leq \lceil 2 \lg n \rceil\} \leq 1 \text{ and } \sum_{k=\lceil 2 \lg n \rceil+1}^n \Pr\{X = k\} \leq \Pr\{X > 2 \lg n\} \leq 1/n, \text{ by}$$

part (c). Therefore,

$$\begin{aligned} E[X] &\leq \lceil 2 \lg n \rceil \cdot 1 + n \cdot (1/n) \\ &= \lceil 2 \lg n \rceil + 1 \\ &= O(\lg n) \end{aligned}$$

Solution to Problem 11-2

a. A particular key is hashed to a particular slot with probability $1/n$. Suppose we select a specific set of k keys. The probability that these k keys are inserted into the slot in question and that all other keys are inserted elsewhere is

$$\left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k}.$$

Since there are $\binom{n}{k}$ ways to choose our k keys, we get

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}.$$

b. For $i = 1, 2, \dots, n$, let X_i be a random variable denoting the number of keys that hash to slot i , and let A_i be the event that $X_i = k$, i.e., that exactly k keys hash to slot i . From part (a), we have $\Pr\{A_i\} = Q_k$. Then,

$$\begin{aligned} P_k &= \Pr\{M = k\} \\ &= \Pr\left\{\left(\max_{1 \leq i \leq n} X_i\right) = k\right\} \\ &= \Pr\{\text{there exists } i \text{ such that } X_i = k \text{ and that } X_i \leq k \text{ for } i = 1, 2, \dots, n\} \\ &\leq \Pr\{\text{there exists } i \text{ such that } X_i = k\} \\ &= \Pr\{A_1 \cup A_2 \cup \dots \cup A_n\} \\ &\leq \Pr\{A_1\} + \Pr\{A_2\} + \dots + \Pr\{A_n\} \text{ (by inequality (C.18))} = nQ_k. \end{aligned}$$

c. We start by showing two facts. First, $1 - 1/n < 1$, which implies $(1 - 1/n)^{n-k} < 1$. Second, $n!/(n-k)! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot (n-k+1) < n^k$. Using these facts, along with the simplification $k! > (k/e)^k$ of equation (3.17), we have

$$\begin{aligned} Q_k &= \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \frac{n!}{k!(n-k)!} \\ &< \frac{n!}{n^k k!(n-k)!} \quad ((1 - 1/n)^{n-k} < 1) \\ &< \frac{1}{k!} \quad (n!/(n-k)! < n^k) \\ &< \frac{e^k}{k^k} \quad (k! > (k/e)^k). \end{aligned}$$