Chapter 11

Figure 11.1 Implementing a dynamic set by a direct-address table \( T \). Each key in the universe \( \mathcal{U} = \{0, 1, \ldots, u\} \) corresponds to an index in the table. The set \( k = \{1, 2, 3, 8\} \) of actual keys determines the slots in the table that contain pointers to elements. The other slots, heavily shaded, contain \( \text{null} \).

Figure 11.2 Using a hash function \( \delta \) to map keys to hash table slots. Keys \( k_2 \) and \( k_3 \) map to the same slot, so they collide.
Figure 11.3 Collision resolution by chaining. Each hash-table slot $T[j]$ contains a linked list of all the keys whose hash value is $j$. For example, $h(1) = h(2)$ and $h(3) = h(4) = h(5)$.

Figure 11.4 The multiplication method of hashing. The $w$-bit representation of the key $k$ is multiplied by the $w$-bit value $x = A - 2^n$. The $p$th higher-order bits of the lower $w$-bit half of the product form the desired hash value $h(k)$.

```
HASH-INSERT(T, k)
1   i ← 0
2   repeat j ← h(k, i)
3       if T[j] = NIL
4           then T[j] ← k
5           return j
6       else i ← i + 1
7     until i = m
8   error “hash table overflow”
```
Hash-Search \( T, k \)

1. \( i \leftarrow 0 \)
2. repeat
   1. \( j \leftarrow h(k, i) \)
   2. if \( T[j] = k \)
      then return \( j \)
   3. \( i \leftarrow i + 1 \)
3. until \( T[j] = \text{NIL} \) or \( i = m \)
4. return \( \text{NIL} \)

---

Figure 11.5 Insertion by double hashing. Here we have a hash table of size 13 with \( h_1(k) = k \mod 13 \) and \( h_2(k) = 1 + k \mod 13 \). Since \( 34 \mod 13 \) and \( 76 \mod 13 \), the key 34 is inserted into empty slot 9, after slots 1 and 5 are examined and found to be occupied.

---

Figure 11.6 Using perfect hashing to store the set \( \{10, 22, 37, 48, 70, 75\} \). The main hash function is \( h(k) = \{k \mod 13, k \mod 11\} \) mod 13, where \( a = 3 \), \( b = 11 \), \( p = 101 \), and \( m = 0 \). For example, \( h(75) = 5 \), so key 75 hashes to slot 2 of table \( T \). A secondary hash table \( S_j \) stores all keys hashing to slot \( j \). The size of hash table \( S_j \) is \( m_j \) and the associated hash function is \( h_j(k) = \{k \mod a, k \mod b\} \) mod \( m_j \). Since \( h(75) = 5 \), key 75 is stored in slot 1 of secondary hash table \( S_5 \). There are no collisions in any of the secondary hash tables, and so searching takes constant time in the worst case.