1. Hashing

Hashing is a mechanism that implements member functions *Insert*, *Remove*, and *Find* in $O(1)$, constant, time complexity. There is no faster implementation.

1.1. Implementation Overview.

To insert, remove, and find information stored, we assume the existence of a key that uniquely identifies information related to a particular key. Information is stored in records that are indexed by a key. For the purpose of this discussion, let us assume that array *I* contains the information we desire. Array *I* is indexed by key *k*. A record is found by coding the expression ***I*[*k*]**.

1.2. Implementation Challenge.

The challenge is that key *k* may be a character string. A character string is not a suitable index for an array. Arrays are usually indexed by an integer or, in some programming languages, a restricted range of integers. The challenge is to map a string to an integer that can be used to index array *I*. The technique used is to add successive characters of the key string *k* to an unsigned integer, *u*, which is shifted left four bits in every iteration. The remainder, *r*, of the unsigned integer *u* divided by an arbitrarily large prime number *p* is the index of array *I*. Array has *p* elements. There are other methods for mapping a string to an integer value.

1.2.1. Example.

Suppose string keys ann, bianca, cosette, and denise map to integer keys 4, 8, 7, 1, respectively. The diagram in Figure 1 illustrates how information for ann, bianca, cosette, and denise is stored and retrieved.

0	
1	**denise**
2	
3	
4	**ann**
5	
6	
7	**cosette**
8	**bianca**
9	
10	
11	
12	

Figure 1. Hashing

1.3. Collisions.

It is possible for two keys, k_1 and k_2, both strings, to map to the same integer key *r*. This event is called a collision.

1.4. Resolving Collisions.

Two methods are used to resolve collisions called *separate chaining* and *open addressing*.

1.4.1. Separate Chaining

In separate chaining, all string keys that collide and map to the same integer index *r* are inserted onto a list. The list is anchored to element *r* in array *I.* Elements of array *I* are pointers to lists.

1.4.1.1. Example.

Suppose strings alice, bethany, clarisse, darla, and edith map to integer keys 3, 7, 1, 3, and 5 respectively. The diagram in Figure 2 illustrates a separate chaining implementation for this example.

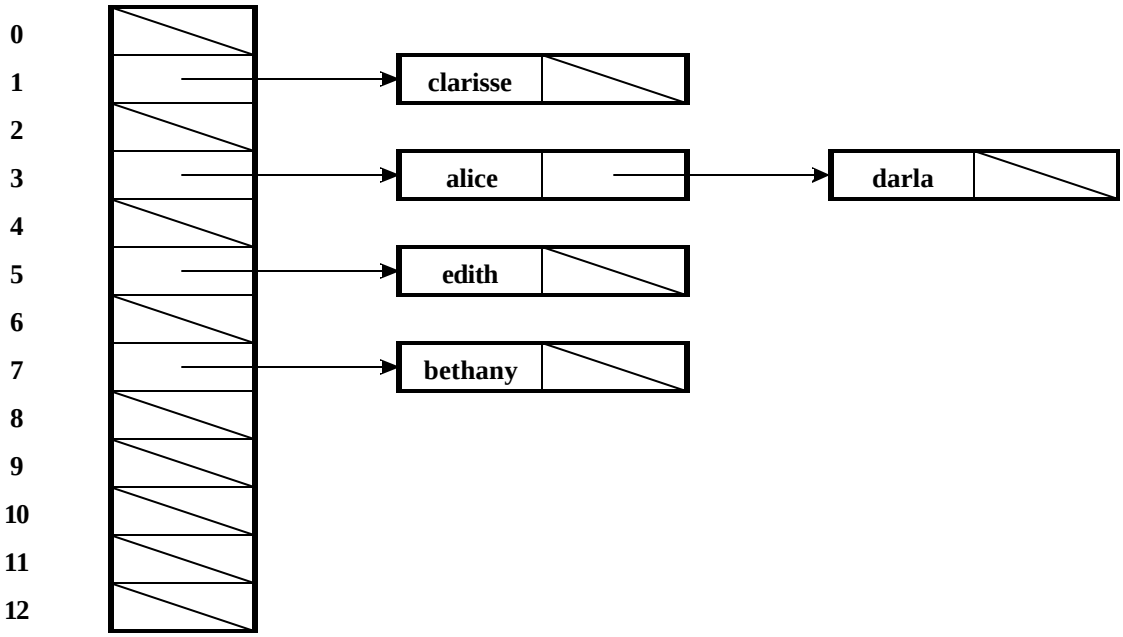

Figure 2. Separate chaining

1.4.2. Open Addressing

In open addressing, all string keys that collide and map to the same integer index *r* are given sequential entries, starting with element *r*. Suppose strings alice, bethany, and clarisse all map to the integer index 5. If alice is entered first, then bethany, and, finally, clarisse, then alice would be given element 5, bethany assigned to element 6, and clarrise would be put in element 7. The algorithm used to resolve collisions is to find the first free element starting with element 5. If element 5 is not available, the subsequent element, element 6, is tested for availability. If the hash table is not full, an element will be found by successively incrementing the index until an available element is found.

1.4.2.1. Example.

Suppose strings alice, bethany, clarisse, darla, and edith map to integer keys 5, 5, 5, 3, and 1 respectively. The diagram in Figure 3 illustrates an open addressing implementation for this example.

	occupied	key
0	**no**	
1	**yes**	**edith**
2	**no**	
3	**yes**	**darla**
4	**no**	
5	**yes**	**alice**
6	**yes**	**bethany**
7	**yes**	**clarisse**
8	**no**	
9	**no**	
10	**no**	
11	**no**	
12	**no**	

Figure 3. Open Addressing