HASH TABLES AND CHAINING

CS340

Bad Sorting Algorithm

```
public static void mySort(int[] A, int p ,int q){
    int[] scratch = new int[q-p+1];
    for (int i = 0; i < A.length; i++) {
        scratch[A[i]-p] = A[i];
    }
    int counter = 0;
    for (int i = 0; i < scratch.length; i++) {
        if (scratch[i] > 0){
            A[counter] = scratch[i];
            counter++;
        }
    }
```

Bad Sorting Algorithm

1,2,3,4,5 = good
1000,2000,3000,4000,5000 = bad

- What are the assumptions?
- How valid are they?
- Is speed really that important?
- Do things have to go fast in the real world?
- What if we don’t need to sort, and we’re just looking to find an entry?
- Linear search = O(n). Is there a better way?

```
public static void mySort(int[] A, int p ,int q){
    int[] scratch = new int[q-p+1];
    for (int i = 0; i < A.length; i++) {
        scratch[A[i]-p] = A[i];
    }
    int counter = 0;
    for (int i = 0; i < scratch.length; i++) {
        if (scratch[i] > 0){
            A[counter] = scratch[i];
            counter++;
        }
    }
```

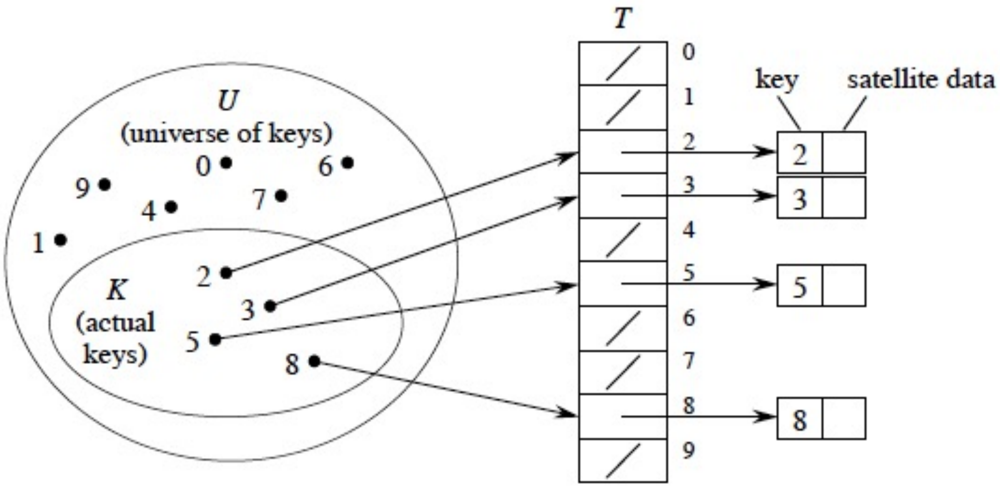
Why A Hash Table?

- Many applications require a dynamic set that supports only the dictionary operations INSERT, SEARCH, and DELETE.
- A hash table is effective for implementing a dictionary.
  - The expected time to search for an element in a hash table is O(1) under some reasonable assumptions. Worst-case O(n).

Hash Tables

- A hash table is a generalization of an ordinary array.
  - With an ordinary array, we store the element whose key is k in position k of the array.
  - Given a key k, we find the element whose key is k by just looking in the kth position of the array. This is called direct addressing.
  - Direct addressing is applicable when we can afford to allocate an array with one position for every possible key.

Direct Address Tables



Dictionary operations are trivial and take $O(1)$ time each:

DIRECT-ADDRESS-SEARCH(T, k)
  return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
  $T[key[x]] = x$

DIRECT-ADDRESS-DELETE(T, x)
  $T[key[x]] = \text{NIL}$

Direct addressing issues

- The problem with direct addressing is if the universe U is large, storing a table of size |U| may be impractical or impossible.
- Often, the set K of keys actually stored is small, compared to U, so that most of the space allocated for T is wasted.
- When K is much smaller than U, a hash table requires much less space than a direct-address table.
- Can reduce storage requirements to $\Theta(|K|)$.
- Can still get $O(1)$ search time, but in the average case, not the worst case.

Hash Tables

- We use a hash table when we do not want to (or cannot) allocate an array with one position per possible key.
  - Use a hash table when the number of keys actually stored is small relative to the number of possible keys.
  - A hash table is an array, but it typically uses a size proportional to the number of keys to be stored (rather than the number of possible keys).
  - Given a key k, don’t just use k as the index into the array.
  - Instead, compute a function of k, and use that value to index into the array. We call this function a hash function.

The idea

- Instead of storing an element with key k in slot k, use a function h and store the element in slot h(k).
- We call *h* a *hash function*.
- h : U → (0, 1, … ,m -1), so that h(k) is a legal slot number in T.
- We say that k *hashes* to slot h(k).

Hash function

- Given a key, k, and m available slots, one hash function might be h(k) = k mod m

- For m=23, The following keys go into the following slots:
  - 14 goes into slot 14%23 = 14
  - 46 goes into slot 0
  - 48 goes into slot 2
  - 60 goes into slot 14

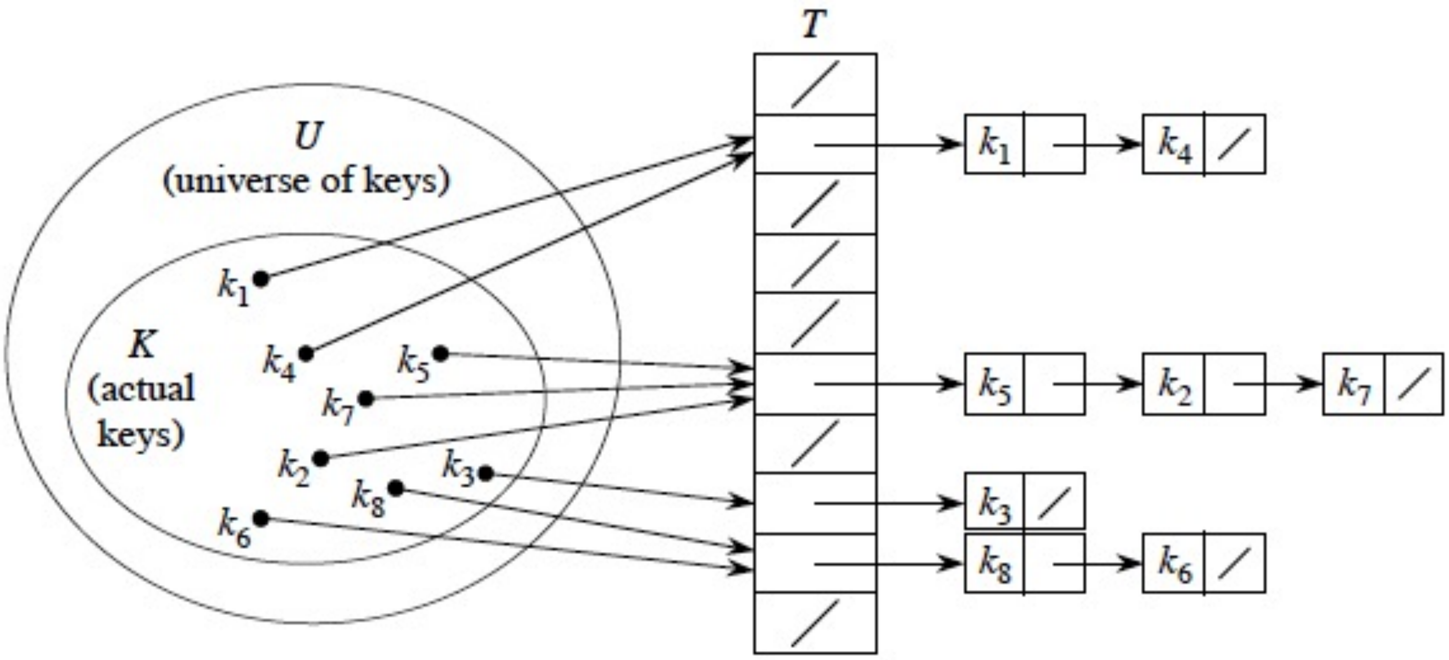
  - Notice that 14 and 60 go into the same slot. This is a problem.

Collisions

- When two or more keys hash to the same slot.
  - Can happen when there are more possible keys than slots (|U| > m).
  - For a given set K of keys with |K| <= m, may or may not happen. Definitely happens if |K| > m.
  - Therefore, must be prepared to handle collisions in all cases.
  - Use two methods: chaining and open addressing.
  - Chaining is usually better than open addressing.

Chaining into a linked list

Put all elements that hash to the same slot into a linked list.



Dictionary Operations with Chaining

- CHAINED-HASH-INSERT(T,x)
  - Insert x at the head of list T[h(key[x])]
    - Worst case running time is O(1)
    - Assumes item is not already in list. Additional search is needed to determine if item is already in list.
- CHAINED-HASH-SEARCH(T,k)
  - Search for an element with key k in list T[h(k)]
  - Time is proportional to the length of the list of elements in the slot
- CHAINED-HASH-DELETE(T,x)
  - Delete x from the list T[h(key[x])]
  - Time is proportional to the length of the list of elements in the slot

Analysis

- What is the worst case with chaining?
- What is the best case?
- What is the average case?