CMPUT 175
Introduction to Foundations of Computing

Hash Tables

Objectives

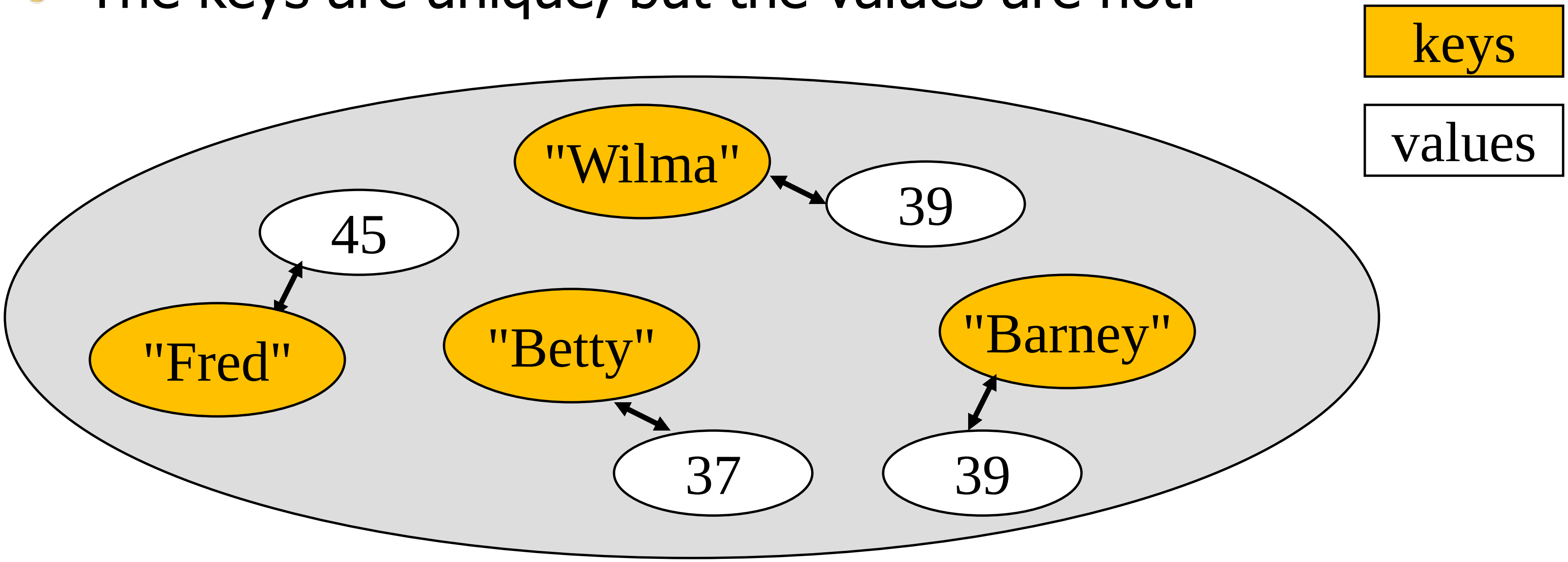
- Understand how dictionaries could be implemented
- Introduce the concept of hash tables and hash functions
- Understand how collusions are resolved

Outline of Lecture

- Dictionaries
- Hashing and Hash Functions
- Collisions
- Hash Tables
- Collision Resolution

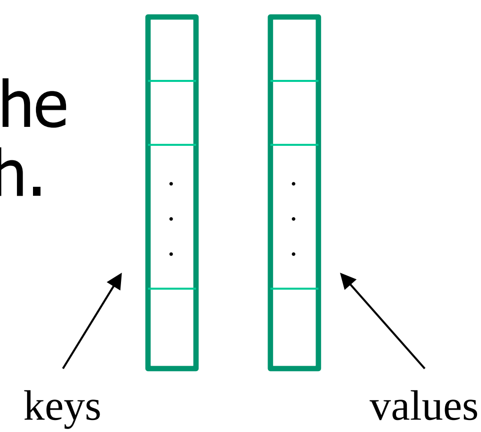
Dictionary

- A Dictionary is an unordered container that contains key-value pairs.
- The keys are unique, but the values are not.



Dictionary - Obvious Implementations

- We could implement a Dictionary using a list that holds associations (key-value).
- We could also implement a Dictionary using two parallel containers (Lists), one for the keys and one for the values.
- If the keys are not comparable (i.e. cannot be sorted using primitive operations), the methods to access the elements would each require O(n) calls to the method comparing the object elements.
- If the keys are comparable we can reduce the comparisons to log (n) using a binary search.
- Can we do better?



Outline of Lecture

- Dictionaries
- Hashing and Hash Functions
- Collisions
- Hash Tables
- Collision Resolution

Dictionary - Parcel Analogy

- Assume that you are about to leave a busy mall and you are one of about a thousand people picking up a parcel at any time during the day.
- This is a Dictionary problem with names as keys and parcels as values.
- Assume the mall has 100 bins that each hold about 10 parcels.
- How should the mall organize these parcels to minimize waiting time?

Parcels - Using Bins

- When you buy your item, you are asked for the last two digits of your phone number and your parcel is sent to that bin.
- When you pick up your parcel the attendant asks for the last two digits of your phone number (00-99), goes to the correct bin (1 - 100) and searches through the parcels (1-10) to get the one with your name.
- This is an example of hashing.
- Each item is assigned a hash number that is used to select a bin which contains a small number of items that can be searched for your item.

Selecting Bin Numbers

- Would the first two digits of a phone number be as good as the last two digits?
- There are only a few combinations of first two digits that most local residents share. Moreover, numbers such as 00, 01,..,09 won't exist as first digits. So a few bins would overflow and others would be empty.
- What about using the first two or last two letters of the name of the person?
- This would take 26*26 = 676 bins but even so, some bins would be fuller than others.
- For maximum efficiency, we want the keys to be uniformly distributed over the bin numbers.

Hash Functions

- A hash function maps keys to integers in the range 0…(#bins-1).
  - It should map keys uniformly across this range.
  - It should be fast to compute.
  - It should be applicable to all objects.
- The hash function should be able to work with any number of bins.
- The most common way to achieve this is to map keys to non-negative integers and then take the integer "mod" the #bins.
- Recall the "mod" function:

X % Y = the remainder when you divide X by Y

e.g. 58 % 5 = 3

Hash Functions

- Given an Alphabetical String, e.g., Names or Words in a Dictionary, we can:
  1. Map Alphabets to Numbers
  2. Possibly Multiply by a Weight depending on the Position of an Alphabet in a String
  3. Add all the numbers multiplied by Weights together
  4. Finally, perform the “mod” operation (% in Python)

- Simple Hash Functions may just Add or Multiply the Numbers in Step 1 above, without including Steps 2 & 3

Outline of Lecture

- Dictionaries
- Hashing and Hash Functions
- Collisions
- Hash Tables
- Collision Resolution

Collisions

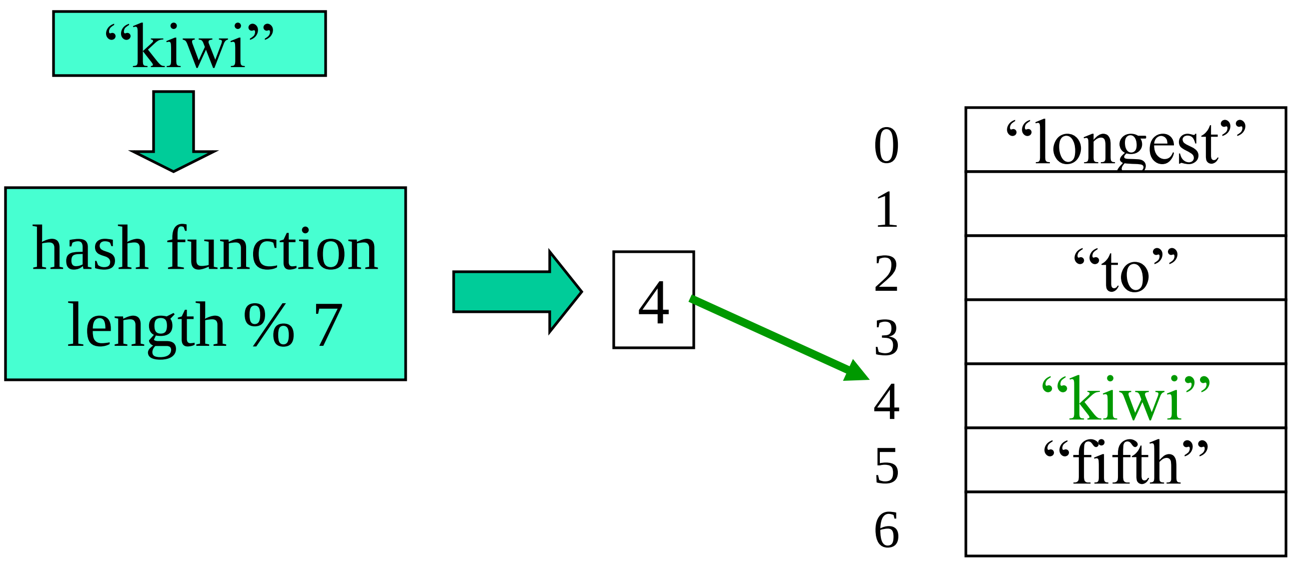
- When two keys map to the same bin, we have a hash collision.
- When a collision occurs, a collision resolution algorithm is used to establish the locations of the colliding keys in the bin.
- In some cases when we know all of the key values in advance we can construct a perfect hash function that maps each key to a different bin (no collisions).

Outline of Lecture

- Dictionaries
- Hashing and Hash Functions
- Collisions
- Hash Tables
- Collision Resolution

Hash Tables

- A hash table is a container (usually an Array or Vector) whose elements are used as bins.
- In the basic implementation, each entry in the hash table is a bin that holds a single element.
- Only the keys are shown in these examples.

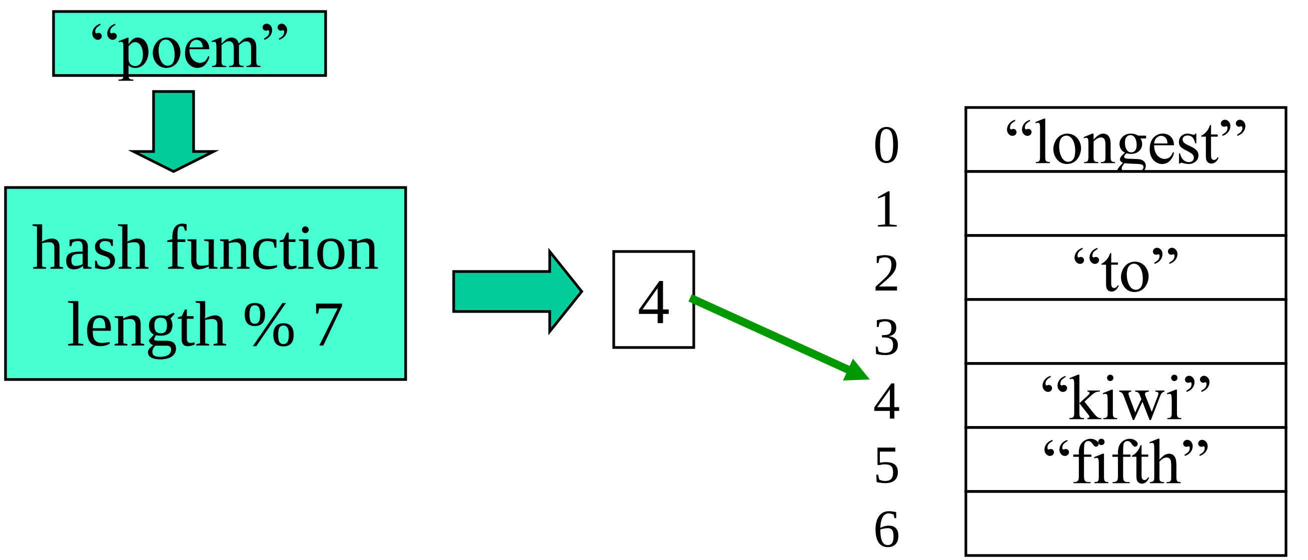


Outline of Lecture

- Dictionaries
- Hashing and Hash Functions
- Collisions
- Hash Tables
- Collision Resolution

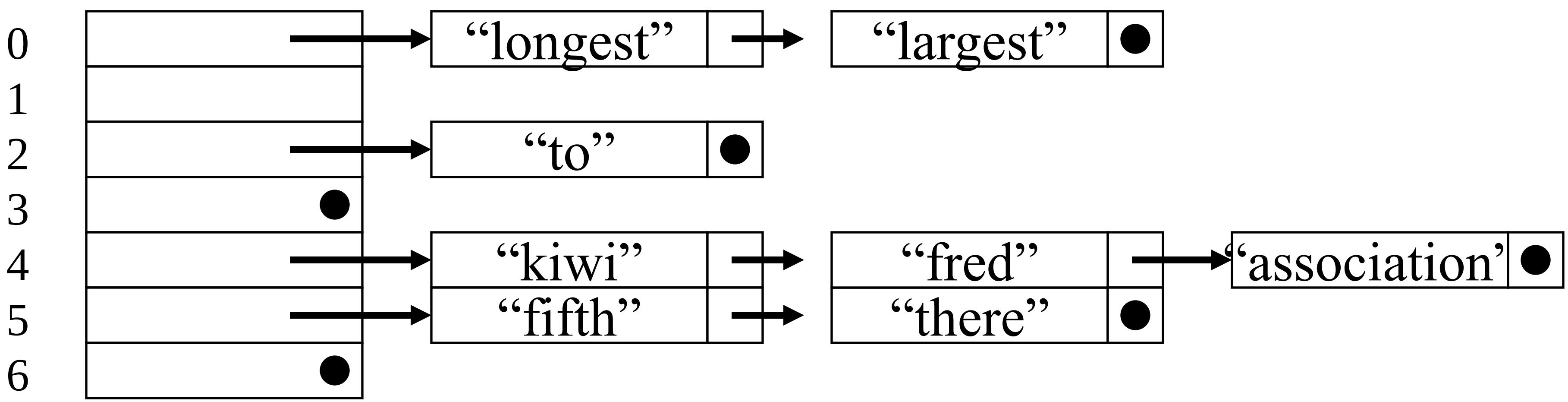
Hash Tables Collisions

- If there is a hash collision, the collision resolution algorithm selects a different bin for the new element to be inserted.
- This is called open addressing and one possible strategy is linear probing.



External Chaining

- Instead of implementing a hash table whose entries are associations, we can have a hash table whose entries are containers for associations.
- Then when there is a hashing collision, we put all elements that collided into a common container.
- This can be done with linked lists



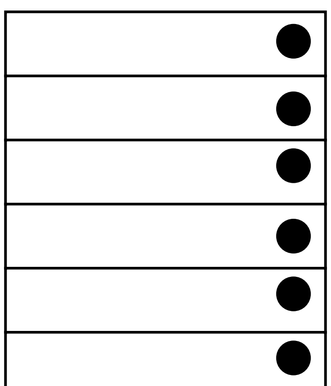
Example Hash Function

1. Map Alphabets to Numbers (A → 1; …; Z → 26)
2. Multiply these numbers together
3. Perform the "mod" operation with 6 to create 6 Bins with External Chaining

Let's look at how the following Strings are stored after Hashing with the above function:

Dog, I, He, Cat, Me, Him, She

0
1
2
3
4
5



(A or a) = 1;
(B or b) = 2;
(C or c) = 3;
(D or d) = 4;
(E or e) = 5;
(F or f) = 6;
(G or g) = 7;
(H or h) = 8;
(I or i) = 9;
(J or j) = 10;
(K or k) = 11;
(L or l) = 12;
(M or m) = 13;
(N or n) = 14;
(O or o) = 15;
(P or p) = 16;
(Q or q) = 17;
(R or r) = 18;
(S or s) = 19;
(T or t) = 20;
(U or u) = 21;
(V or v) = 22;
(W or w) = 23;
(X or x) = 24;
(Y or y) = 25;
(Z or z) = 26

Calculating Hash Function

Dog: 4 x 15 x 7 = 420 % 6 = 0

I: 9 = 9 % 6 = 3

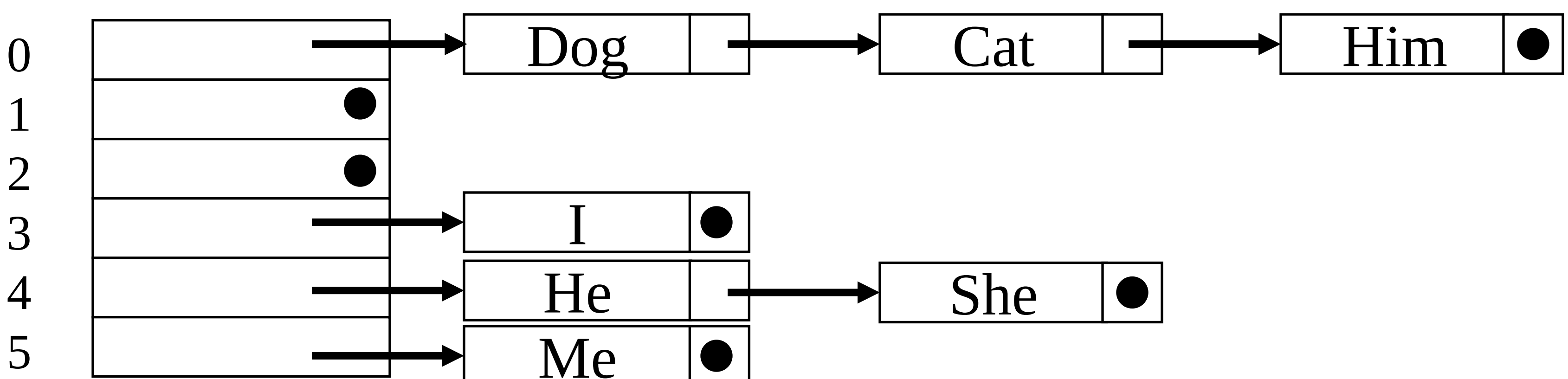
He: 8 x 5 = 40 % 6 = 4

Cat: 3 x 1 x 20 = 60 % 6 = 0

Me: 13 x 5 = 65 % 6 = 5

Him: 8 x 9 x 13 = 936 % 6 = 0

She: 19 x 8 x 5 = 760 % 6 = 4



Efficiency of HashTables

- If the number of collisions is small, searching, adding and removing elements in a hash table requires O(1) time.
- To reduce the number of collisions, in addition to using a good hash function, we must make sure the table does not get too full.
- The load factor of a hash table is the ratio of elements in the table to total space for elements.
- For best results, the load factor should not be above 0.6.
- If it gets higher, we should extend the hash table and re-hash all of its elements.