ZAMAN UNIVERSITY

Data Structures and Algorithms

Chapter 4

Hash Tables

Outline

- Hash Tables
- Quadratic Probing
- Separate Chaining
- When to Use What

Outline

- Hash Tables
- Quadratic Probing
- Separate Chaining
- When to Use What

Hash Tables: Introducing to Hashing

- A hash table is a data structure that offers very fast insertion and searching, takes constant time O(1)
- It's so fast to look up tens of thousands of items in less than a second

- Hash tables do have several disadvantages:
  - Developed based on arrays, difficult to expand after they've been created;
  - Performance might degrade catastrophically when the table becomes too full;
  - Thus, programmer needs to have a fairly accurate idea of how many data items will need to be stored;
  - There's no convenient way to visit the items in a hash table in any kind of order (such as from smallest to largest).

Hash Tables: Introducing to Hashing (cont.)

- One important concept is how a range of key values is transformed into a range of array index values
- In a hash table this is accomplished with a hash function
- However, for certain kinds of keys, no hash function is necessary; the key values can be used directly as array indices

Hash Tables: Employee Numbers as Keys

- Suppose you're writing a program to access employee records for a small company with, say, 1,000 employees
- Each employee record requires 1,000 bytes of storage

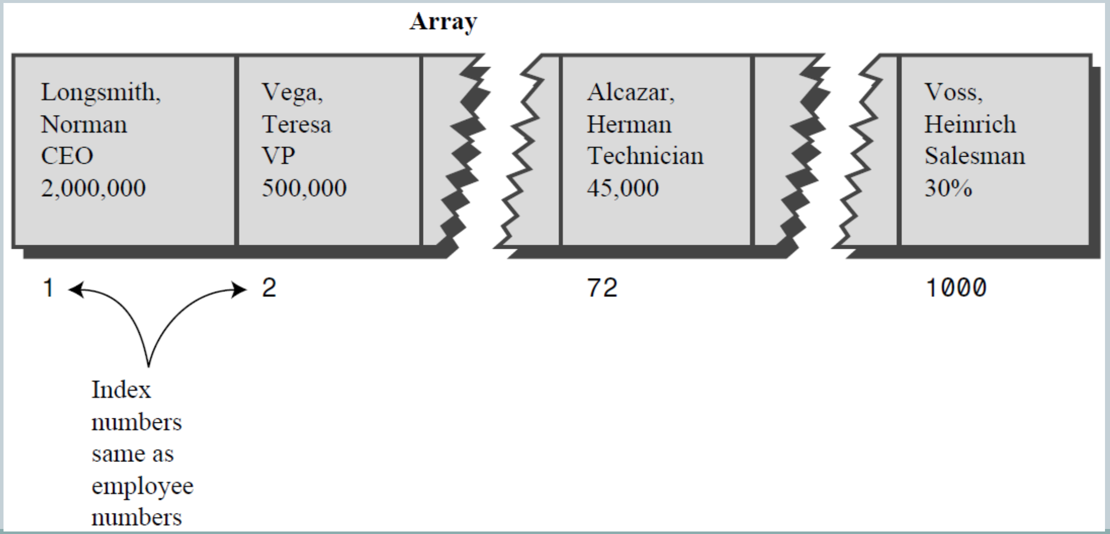
- The company's director has specified that she wants the fastest possible access to any individual record
- Also, every employee has been given a number from 1 (for the founder) to 1,000 (for the most recently hired worker)
- These employee numbers can be used as keys to access the records
- What sort of data structure should you use in this situation?

Hash Tables: Keys Are Index Numbers

- Each employee record occupies one cell of the array, and the index number of the cell is the employee number for that record

- Accessing a specified array element is very fast if you know its index number
- Example: We are looking for Herman Alcazar, his employee number 72.

```
empRecord rec = databaseArray[72];
```


Array
Longsmith, Norman CEO 2,000,000
Vega, Teresa VP 500,000
Alcazar, Herman Technician 45,000
Voss, Heinrich Salesman 30%
1
2
72
1000
Index numbers same as employee numbers

Hash Tables: A Dictionary

- We want to store a 50.000-word English-language dictionary in main memory
- You would like every word to occupy its own cell in a 50,000-cell array, so you can access the word using an index number
- This will make access very fast

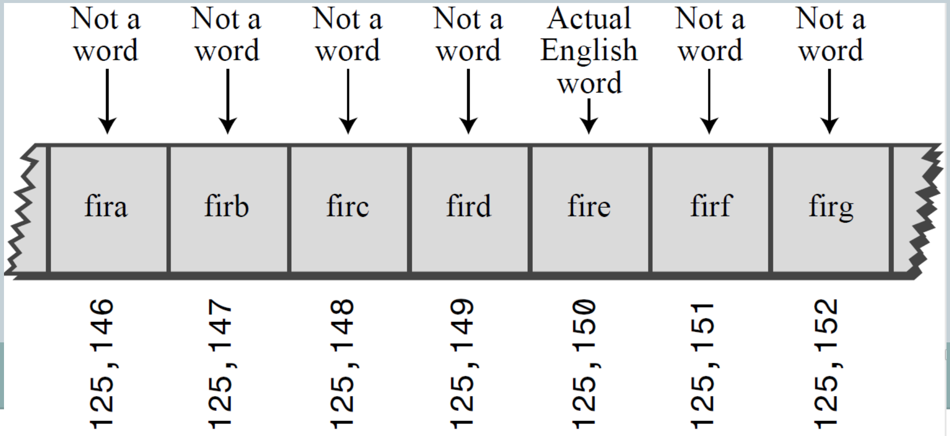
- But what is the relationship of these index numbers to the words? Given the word, how do we define its index number?

Hash Tables: Converting Words to Numbers

- What we need is a system for turning a word into an appropriate index number
- ASCII code runs from 0 to 255, to accommodate capitals, punctuation, and so on
- There are really only 26 letters in English words, so let's devise our own code: Let's say blank is 0, ***a*** is 1, ***b*** is 2, ***c*** is 3, and so on up to 26 for ***z***

Hash Tables: Converting Words to Numbers (cont.)

- **Add the digits**: convert *cats* to a number. First we convert character to number: c=3, a=1, t=20, s=19, thus we add them: 3+1+20+19=43
- Multiply by Powers: different way to map words to number. Examples:
  - Analogous situation, define index of number instead of word thus, 7546 means: $7*10^3 + 5*10^2+4*10^1+6*10^0$
  - Convert the word *cats* to a number. The result: $3*27^3+1*27^2+20*27^1+19*27^0=60,337$



Hash Tables: Hashing

```
arrayIndex = hugeNumber % arraySize;
```

- It hashes (converts) a number in a large range into a number in a smaller range
- This smaller range corresponds to the index numbers in an array
- An array into which data is inserted using a hash function is called a *hash table*.

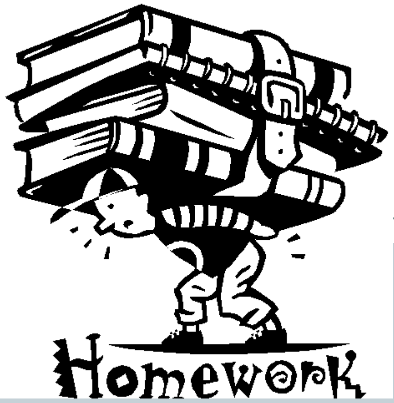
Hash Tables: Collisions

- Perhaps you want to insert a new word into the array
- You hash the new word to obtain its index number, but find that the cell at that number is already occupied by another word, which happens to hash to the exact same number (for a certain size array).

Linear Probing

- In linear probing we search sequentially for vacant cells when a collision occurs

- Example: If 5,421 is occupied when we try to insert *cats* there, we go to 5,422, then 5,423, and so on, incrementing the index until we find an empty cell.
- This is called *linear probing* because it steps sequentially along the line of cells, probing for an empty cell.



1. Create a Hash Table with size 200
2. Create Has Function to convert words to number by Multiply Powers for Insertion and Searching.

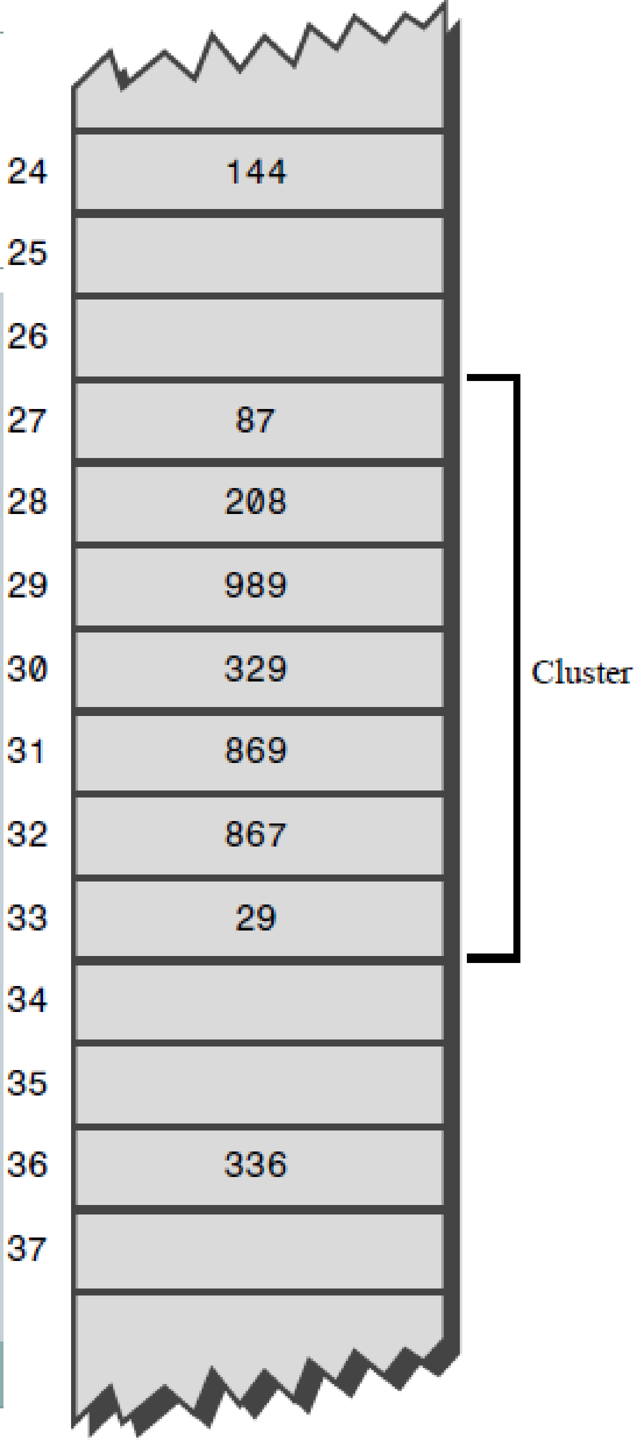
Using Linear Probing to solve collision problem.

Outline

- Hash Tables
- Quadratic Probing
- Separate Chaining
- When to Use What

What Is Clustering?

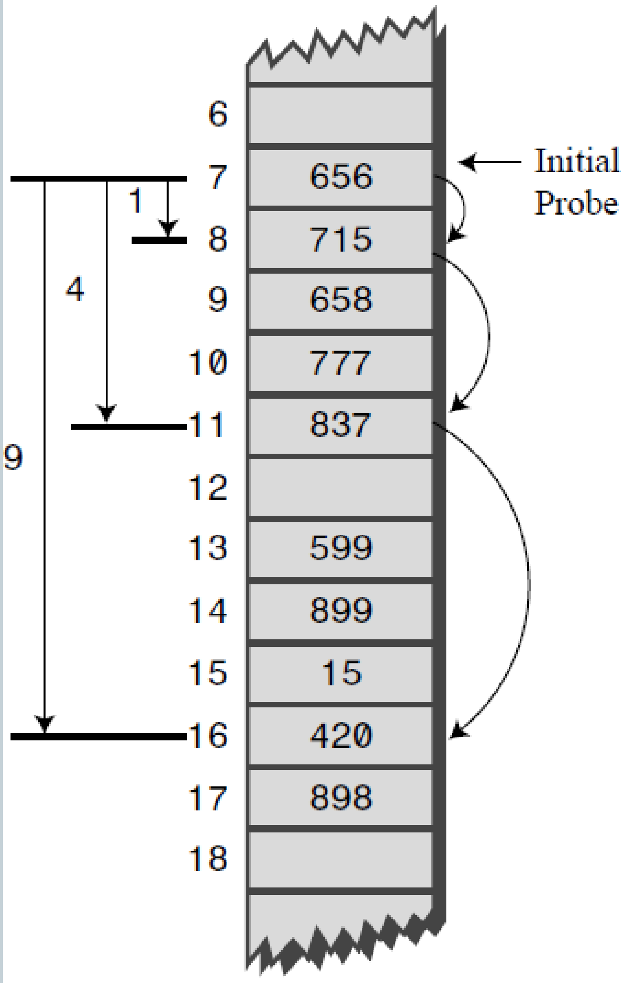
- A sequence of filled cells in a hash table called *filled sequence*
- As you add more and more items, the filled sequences become longer
- This is called *clustering*



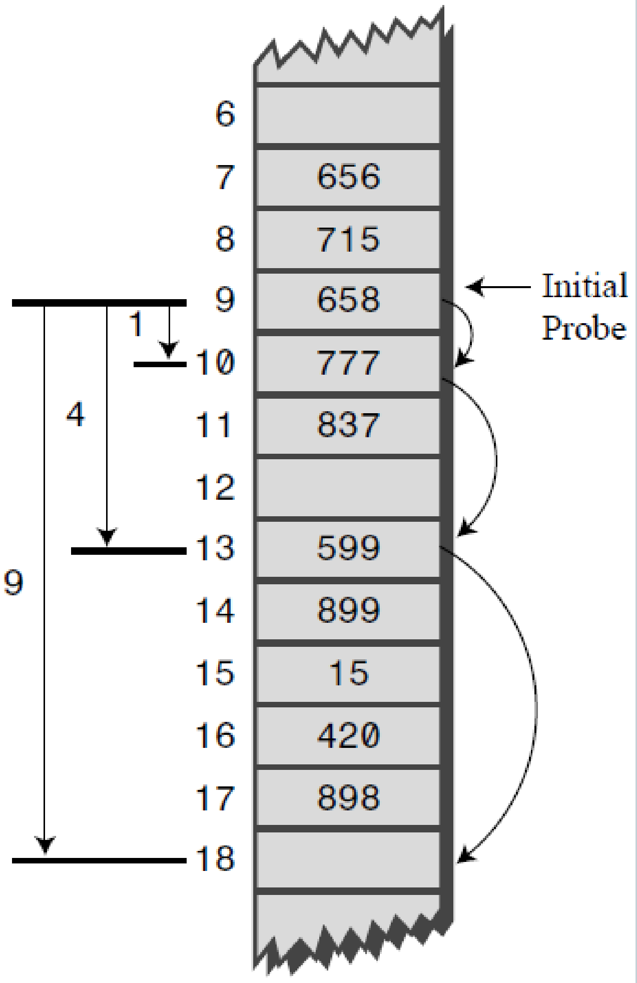
Quadratic Probing

- Quadratic probing is an attempt to keep clusters from forming
- The idea is to probe more widely separated cells, instead of those adjacent to the initial hash site
- The step is the square of the step number
  - In quadratic probing, probes go to x+1, x+4, x+9, x+16, x+25, and so on;
  - It means that the distance from the initial site is the square of the step number, so the probes fall at $x+1^2$, $x+2^2$, $x+3^2$, $x+4^2$, $x+5^2$ and so on.

Example: Searching by Quadratic Probing



a) Successful search for 420



b) Unsuccessful search for 481

Problems with Quadratic Probing

- Quadratic probes eliminate the clustering problem in linear probe, which is called *primary clustering*
- However, the quadratic probe always generates the same steps: 1, 4, 9, 16, and so on, this is called *secondary clustering*.
- To eliminate secondary and primary clustering, another approach can be used: *double hashing* (sometimes called *rehashing*)

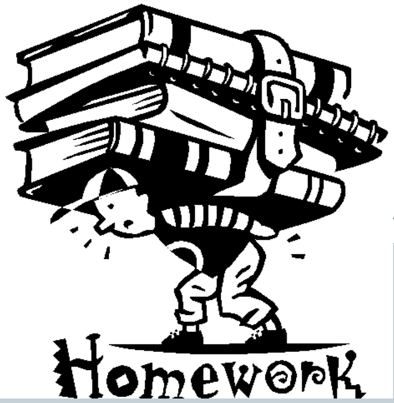
Double Hashing

- What we need is a way to generate probe sequences that depend on the key instead of being the same for every key
- Then numbers with different keys that hash to the same index will use different probe sequences
- Experience has shown that this secondary hash function must have certain characteristics:
  - It must not be the same as the primary hash function;
  - It must never output zero (otherwise there would be no step; every probe would land on the same cell).
- Experts have discovered that functions of the following form work well:

```
stepSize = constant - (key % constant);
```

where constant - prime and smaller than the array size.

For example: `stepSize = 5 - (key % 5);`



1. Create a Hash Table with size 200
2. Words are converted to number by Multiply Powers;
3. Create Hash Function by using Double Hashing

Continue…