Hash Tables II

University of Massachusetts Amherst
ECE 242 – Data Structures and Algorithms
Lecture 32

Hash table

- Index for element computed by hash function
  - E.g., maps string to integer
  - Range matches size of allocated table
  - Does not need to maintain ordering

- Example:

hash(Leonard)
Collisions

• What to do when multiple items have the same hash value?

• Collision resolution techniques
  – Store multiple items in same place (next lecture)
  – Probing (check items following hash index)
    • Linear probing
    • Quadratic probing
    • Etc.

Linear probing

• Insertion
  – Compute hash value
  – Check if table cell is empty at hash value
  – If not, keep checking next cell until empty spot found
  – Insert

• Linear probing sequence
  – hash, hash+1, hash+2, hash+3, etc.
Quadratic probing

• Insertion
  – Compute hash value
  – Check if table cell is empty at hash value
  – If not, keep checking “next” cell until empty spot found
  – Insert

• Quadratic probing sequence
  – hash, hash+1^2, hash+2^2, hash+3^2, etc.

Code

• Quadratic probing:

```java
public void insert(String s) {
    int h = hash(s); // compute hash of item
    System.out.println("inserting "+s+" at "+h);
    int step=1; // step counter for quadratic resolution
    while (table[h]!=null) { // find empty spot in table
        System.out.println("skipping "+h+" due to collision");
        h=(hash(s)+step*step)%size;
        // if occupied, increment by square of step
        step++; // increment step
    }
    table[h]=s; // enter item into table
}
```
Separate chaining

• Collision resolution:
  – Store all elements with same hash in a linked list

• Insert:
  – Add item to linked list at hash location

• Find:
  – Search linear list at hash location

Separate chaining

• Hash table example:
Code

• Separate chaining constructor:

```java
public class HashTable {
    private LinkedList<String>[] table;
    private int size;

    public HashTable(int n) {
        table = (LinkedList<String>[]) new LinkedList[n];
        for (int i=0; i<n; i++) {
            table[i] = new LinkedList<String>();
        }
        size = n;
    }
}
```

Note on Java

• In constructor, we would like to say
  `table = new LinkedList<String>[n];`

• However, Java does not allow it
  – Arrays of generic types need to be created differently

• Instead, we need to use a cast
  `table = (LinkedList<String>[]) new LinkedList[n];`
Code

• Separate chaining insert:

```java
public void insert(String s) {
    int h = hash(s); // compute hash of item
    System.out.println("inserting " + s + " at " + h);
    table[h].add(s);
}
```

Code

• Separate chaining find:

```java
public String find(String s) {
    int h = hash(s); // compute hash of item
    Iterator<String> iter = table[h].iterator();
    while (iter.hasNext()) { // check for next item
        String current = iter.next(); // grab next item
        if (current.equals(s)) { // compare to search item
            return current; // if match return
        }
    }
    return null; // if no match in entire list, return null
}
```
Comments on hash tables

• Other techniques for handling collisions:
  – Double hashing
  – Perfect hash functions (for known set of keys)

• Hash table sizing
  – Hard limit on entries for hash tables with linear/quadratic probing: keep load <30-50%
  – Separate chaining can hold arbitrary number of entries
    • Performance degrades to linked list if too many entries

Next Steps

• Homework 6
  – Deadline tonight 11 p.m. (no extension!)

• Project 5
  – Will be posted Friday

• Lecture on Monday
  – Parallel and distributed algorithms
    • Last topic to be included in final exam