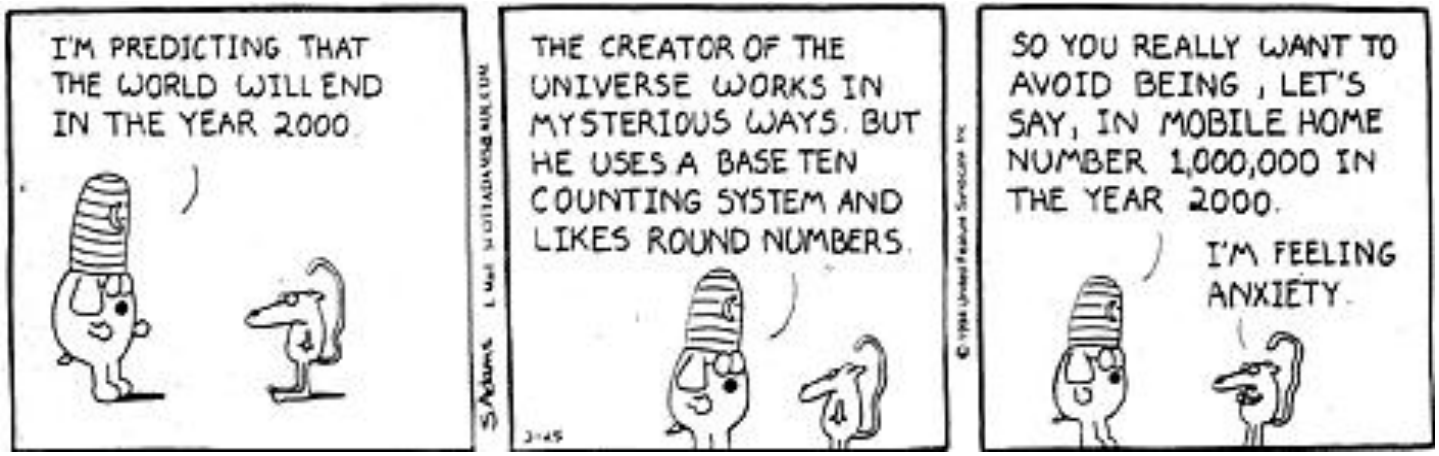
ORF 201

COMPUTER METHODS FOR PROBLEM SOLVING

Lecture 15

Hashing



http://www.princeton.edu/~orf201

Hashing - Ultra Fast Lookup

Database

- Consists of many *records*.
- Each record consists of *fields*.
- One field is designated a *key* field.

Question: How to represent in computer so that key-based access is fast?

Answer: Use *hashing*

```
-----------------------------------
      name:  Adam Stefan Gussow
department:  English
     email:  asgussow@Princeton.EDU
-----------------------------------
      name:  Andrea S Anderson-Ribadeneira
department:  Special Student
     email:  andreaa@Princeton.EDU
-----------------------------------
      name:  Lori A Dauphiny
     phone:  609-258-6373
   address:  5 Dillon Gym
department:  Athletics
     email:  dauphiny@Princeton.EDU
-----------------------------------
      name:  Ann Haver-Allen
     phone:  609-258-3617
   address:  C232 Engrg Quad
department:  School of Engineering and Applied Science
     email:  allen@Princeton.EDU
-----------------------------------
```

- Create an array of **`n`** records: **`recordArray`**.
- Assume **`n`** is larger than the number of records in the database.
- Create a function, hash, mapping the key string to an integer between **`0`** and **`n-1`**.
- Store each record at the place in **`recordArray`** whose index is given by the hash function.

Collisions

Hashing is a great idea **BUT**

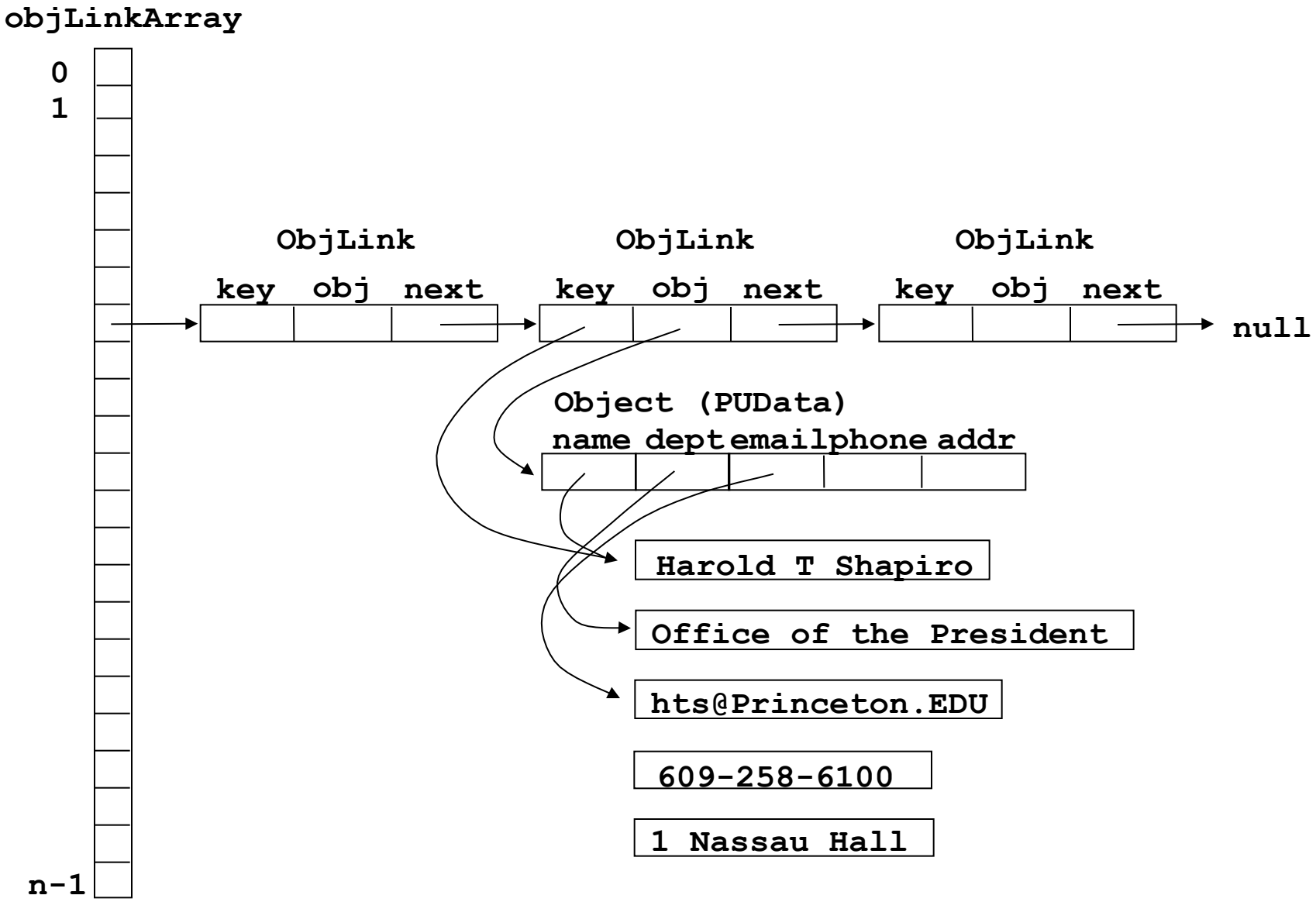
- What if two different keys produce the same hash value?
- What if the number of records exceeds **`n`**?

Resolution
Instead of an array of records, use an array of *linked lists* of records.

Generalization and Abstraction
Hash tables can store anything, not just the Princeton University student/faculty database.

- In Java, every instance of a derived class is also an **`Object`** (because every derived class extends **`Object`**).
- Variables of class **`Object`** are really just generic *pointers*.
- Best to define hashtable using linked lists of key-object pairs.

Memory Schematic



Classes PUData and PUHasher2

```
class PUData
{
    String name;
    String dept;
    String email;
    String phone;
    String addr;
}
```

```
public class PUHasher2
{
    static HashTable ht; // defined later

    static void readData()
    {
        /* details later */
    }

    static void searchNames()
    {
        /* details later */
    }

    public static void main(String[] args)
    {
        ht = new Hashtable(200001);
        readData();
        searchNames();
    }
}
```

readData()

```
static void readData()
{
    int i;
    String line=null;

    Buffered Reader in;
    try { in = new BufferedReader(new FileRead("pu_names.dat")); }
    catch (FileNotFoundException fe)
    { System.err.println("File not found"); return; }

    PUData    p  = new PUData();
    for (i=0;  ; i++) {
        if (i%500 == 0) {System.out.println(i);}

        try { line = in.readLine(); }
        catch (IOException e) {System.err.println("IO Error");}

        if (line == null) break;

        int j = line.indexOf(":")+2;
        int k = line.length();

        if (line.startsWith("             name: ")) {
            p.name = line.substring(j,k);
        } else if (line.startsWith("            phone: ")) {
            p.phone = line.substring(j,k);
        } else if (line.startsWith("          address: ")) {
            p.addr = line.substring(j,k);
        } else if (line.startsWith("      department: ")) {
            p.dept = line.substring(j,k);
        } else if (line.startsWith("            email: ")) {
            p.email = line.substring(j,k);
        } else
        if (line.startsWith("------------------") && p.name!=null) {
            if (ht.lookUp(p.name) == null) {ht.add(p.name, p);}
            p = new PUData();
        }
    }
}
```

searchNames()

```
static void searchNames()
{
     String name;
     while (true) {
          name = Console.in.readLine("Enter a name: ");
          if (name.equals("quit")) {break;}

          PUData p2 = (PUData) ht.lookUp(name);

          if ( p2 != null ) {
               Console.out.println("Found it "+p2.name);
               Console.out.println("          "+p2.dept);
               Console.out.println("          "+p2.addr);
               Console.out.println("          "+p2.phone);
               Console.out.println("          "+p2.email);
          } else {
               Console.out.println("Not found");
          }
     }
}
```

Classes ObjLink and HashTable

```
class ObjLink
{
    String key;
    Object obj;
    ObjLink next;
    public ObjLink(String key, object obj)
    {   this.key = key;
        this.obj = obj;
    }
}
```

```
class HashTable
{
    int n;
    ObjLink[] objLinkArray;
    int theta;

    public HashTable(int n)
    {   this.n = n;
        ObjLinkArray = new ObjLink[n];
        theta = (int)(0.3874576628*n);

        for (int j=0; j<n; j++) {
            objLinkArray[j] = null;
        }
    }

    public void add(String key, Object obj)
    { /* defined later */ }

    public Object lookUp(String name)
    { /* defined later */ }

    public int hash(String s)
    { /* defined later */ }
}
```

add() and lookUp()

```
public void add(String key, Object obj)
{
    Object o;
    ObjLink ol;
    int hashval;

    o = lookUp(key);
    if (o == null) {
        ol = new ObjLink(key, obj);
        hashval = hash(key);
        ol.next = objLinkArray[hashval];
        objLinkArray[hashval] = ol;
    }
}
```

```
public Object lookup(String name)
{
    ObjLink ol;

    for (ol=ObjLinkArray[hash(name)]; ol!=null; ol=ol.next) {
        if (name.equals(ol.key)) {
            return ol.obj;    // found it
        }
    }
    return null;              // not found
}
```

hash()

```
public int hash(String s)
{
    int hashval;

    int len = s.length();
    byte[] b = new byte[len];
    s.getBytes(0, len, b, 0);

    hashval = 0;
    for (int j=0; j<len; j++) {
        hashval = b[j] + theta*hashval;
    }
    if (hashval < 0) {hashval = -hashval;}
    return hashval % n;
}
```