ORF 201

COMPUTER METHODS FOR PROBLEM SOLVING

Lecture 18

Database Data Structures

(C) Princeton University

Database Operations

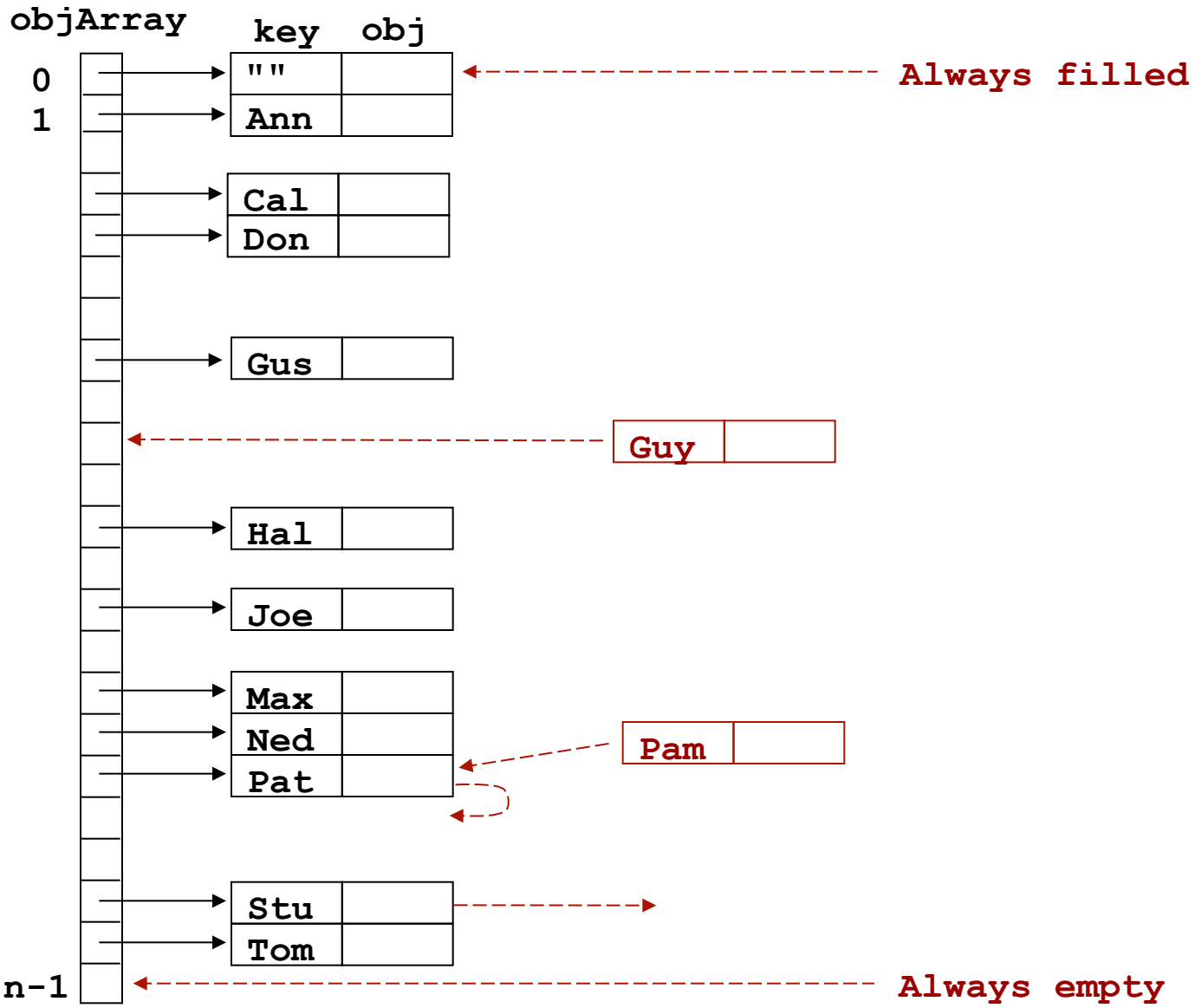
Data Structure	Search	Add	Delete	Next/ Previous
Unsorted Array	n	1	1	n
Sorted Array	$\log n$	n	n	1
Linked List	n	1	1	n
Hash Table	1	1	1	n
Sorted Array w/ Holes	$\log n$	1	1	1
Binary Tree	$\log n$	1	$\log n$	$\log n$

Recall, that for large n:

$$1 < \log n << n$$

Today: Sorted Array with Holes
One Week From Thursday: Binary Trees

Sorted Array w/Holes



Classes KeyObj and Sorted Table

```
class KeyObj
{
    String key;
    Object obj;
}
```

```
class SortedTable
{
    int n;                  // size of table
    int m;                  // number of non-null entries
    KeyObj[] objArray;

    int curr;               // index of current entry
      .
      .
      .
}
```

Sorted Table - Search

Essentially just a binary search.

```
public Object lookUp(String key)
{
    int low, mid=0, high;
    KeyObj ko;

    low = 0;
    high = prev(n-1);
    while ( low <= high ) {
        mid = (low+high)/2;
        ko = objArray[mid];
        if (ko = null) {
            mid = prev(mid);
            ko = objArray[mid];
        }
        if ( key.compareTo(ko.key) < 0) {
            high = prev(mid);
        } else
        if ( key.compareTo(ko.key) > 0) {
            low = next(mid);
        } else {
            curr = mid; return ko.obj;  // found it
        }
    }

    curr = high; return null;  // not found
}
```

`prev()` and **`next()`** return the previous/ next non-null entry in the table.

`s.compareTo(t)` returns positive if string **`s`** is alphabetically after **`t`**, negative if before, and zero if equal.

Always "round down."

Sorted Table - Add

```
public void add(String key, Object obj)
{
    int right, left, j;
    lookUp(key);
    if ( !objArray[curr].key.equals(key) ) {
        KeyObj ko = new KeyObj();
        ko.key = key;
        ko.obj = obj;
        left = curr;
        right = next(left);
        if (right - left > 1) {
            curr = (right + left)/2;
            objArray[curr] = ko;
        } else {
            for (j=right+1; j<n; j++) { // find hole
                if (objArray[j] == null) { break; }
            }
            for (curr=j; curr>right; curr--) { // shift
                objArray[curr] = objArray[curr-1];
            }
            objArray[right] = ko;        // fill hole
        }
        m++;
    }
    if (m/(double)n > 0.7 || m/(double)n < 0.3) {resize();}
    if (objArray[n-1] != null) {resize();}
}
```

Expected number of shifts
= 0(1/2)+1(1/4)+2(1/8)+…
= 1

`resize()` copies to a new array with $n=2m$ and holes evenly spaced.

Sorted Table - Delete

```
public void delete(String key)
{
    Object o;
    o = lookUp(key);
    if (o != null) {
        objArray[curr] = null;
        m--;
    }

    if (m/(double)n > 0.7 || m/(double)n < 0.3) {resize();}
    if (objArray[0] == null) resize();
}
```

Remaining details are in **`PUSortTable.java`**.