CS 466/666
Topic 3: Streaming
Lectures: October 9

In internet routing decisions must be made very quickly. Often one cannot go “off chip”, so the main constraint is actually memory. This applies to monitoring processes such as statistics gathering. An important problem is detecting “denial of service attacks”, which come down to one address sending a lot of traffic. One (cute) version of this is to ask whether any one value occurs in more than half the locations in a sequence of n values.

```
$Majority(A)$
  $i \leftarrow 0$
  $while\ \ i < n\ \ do$
  $\{x \leftarrow A[++i]$
  $count \leftarrow 1$
  $While\ count > 0\ \ and\ \ i < n\ \ do$
     $\{if\ \ A[++i] = x\ \ then\ ++count$
                       $else\ \ --count$
     $\}$
  $\}$
```

If count $= 0$ *then there is no majority*

If count $> n/2$ *then x is the majority value*

Otherwise x may be the majority value, but we can be certain only by comparing it with previous "majority candidates", and perhaps other values. This can be checked by another scan if this is permitted.

This is an example of a “one sided error”. We can announce there is only one possible candidate.

Now suppose we have room to keep k values and test versus incoming data. We keep k value/counter pairs. On receiving new value

> If value present increment its counter
> Elseif there is a zero counter, put new value there and set counter to 1
> Else decrement all counters.

If a value occurs more than n/(k+1) times it will have a nonzero counter when we finish.

Proof, by contradiction: Consider a value occurring more than n/(k+1) times that is not there at the end. Each time you read the value you do one of the following:

1) decrement k others (so including the value itself this covers (k+1) inputs) or

2) increment its counter, but later someone else causes a decrement to remove this increment. The decrement is applied to all k counters (again, this covers the effect of (k+1) inputs).

This accounts for more than (k+1)(n/(k+1)) = n inputs.

If one is using this scheme to monitor a high speed line, then even if there is space on the appropriate portion of a chip for k values and their counters, the single process of updating counters can be a problem. Presumably we hash to find the appropriate counter in O(1) time (if the location contains some other value we know our value is not present). Incrementing one counter is easy, but decrementing all in O(1) time requires some thought and leaves us with the following data structure problem:

Given k objects, maintain a structure to perform the operations

```
Increment(i)  [add 1 to the count of object  i]
Decrement     [subtract 1 from all counters]
```

We may assume that the decrement operation is never used unless all k counters are positive. Reporting of count values is not required until we are finished scanning the data, at that point we can take O(k) time to do so.

Structure:
For each of at most k+1 counter <u>values</u> currently in use (0, by convention is "in use") maintain:

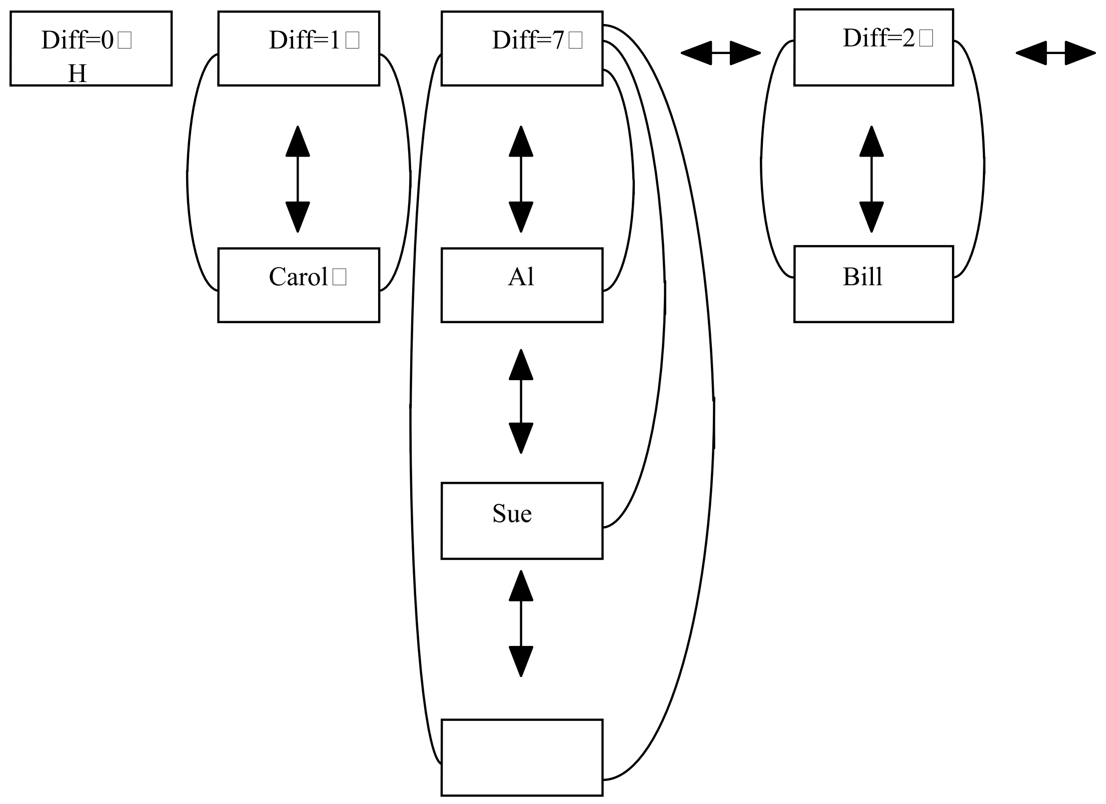
- doubly linked list of elements with this count
- each element has a reference back to the list header and the header has a reference to the end of the list as well.

These list headers are themselves maintained on a doubly linked list. Also included with the list headers is the difference (Diff) between the count represented in that list and the count of the previous list.

H, the "header of the headers" maintains the Diff of 0, indicating a count of 0. We keep it whether or not any counter is actually 0.

```
Increment(i):
    Find element i
    Remove it from its current link and
      Either put it in the next list is the difference value is 1
      Otherwise put it in a new list, after the one from which it was removed
      (give the new list a difference of 1, and decrement the difference of the next one)
    If the list from which we removed the counter is now empty, remove it from the list
    of lists and add its diff value to that of the new list.
```

Decrement: Decrement the Diff value of the list after H. If it goes to 0 move this entire list to the H list.



Here there are initially no unused counters. An attempt to Increment (Joe) causes a Decrement, hence the Diff value of the list containing Carol is reduced to 0, so it is removed and its elements go on the header list.

Increment (Sue) moves Sue to a new list, ahead of her previous list with Diff = 1. The Diff of the following list is reduced by 1 to 1.