CS109A Notes for Lecture 3/1/95

Representing Sets

1. List. Simple, $O(n)$ dictionary operations.

2. Binary Search Tree. $O(\log n)$ average dictionary operations. Supports other operations like range queries, sorting.

3. Characteristic Vector. $O(1)$ dictionary operations, but limited to sets that are subsets of some small set.

4. Hash Table. $O(1)$ average for dictionary operations is possible.

Sorted List for Union, Etc.

- The operations insert, delete, lookup on sets represented by lists is identical to that for lists themselves.

  - □ Note that a list might allow duplicates, while the set it *represents* is deemed to have only one copy.

- Union, intersection, difference on sets represented by lists profits greatly from having the lists sorted.

- Obvious $O(n^2)$ approach to take union (e.g.) of two sets of size n:

  1. Start with (a copy of) one set as the answer list.

  2. For each member of the second set, check if it is in the first set, and if not, append it to the list being formed for the answer.

- To take union, etc., of sorted lists, we only have to examine the heads, take the smaller to the answer and recurse on the lists with the selected element gone.

Example: Intersection of sorted lists:

```
fun inter(L, nil) = nil
|   inter(nil, L) = nil
|   inter(x::xs, y::ys) =
        if x=y then
            x::inter(xs, ys)
        else if (x:int) < y then
            inter(xs, y::ys)
        else inter(x::xs, ys);
```

- Key trick: whichever head is smaller cannot appear on the other list and so can be excluded from consideration for the intersection.
- $O(n)$ if initial lists are sorted and of length $\leq n$.
    - □ Even if lists must be sorted first, it's only $O(n \log n)$.

Characteristic Vectors

Boolean strings whose positions correspond to the members of some fixed "universal" set.

- 1 in a position means the element is in the set, 0 means it's not.

Example: There are 9 "privileges" that can be associated with a UNIX file: Each of (user, group, others) may have any of (read, write, execute).

- The usual order is rwx for each of user (= owner), group, others.
- Thus, e.g., a "protection mode" 110100000 means that the owner may read and write (but not execute), the group can read only, and others may not even read.
    - □ As a set: $\{ur, uw, gr\}$.

Advantages of Characteristic Vectors

If universal set is small, sets can be represented by bits packed 32 (or more) to a word.

- Insert, delete, lookup are $O(1)$ operations on the proper bit.

- Union, intersection, difference are implemented by machine operations on a word-by-word basis.

  - □ Thus, the running time is $O(m)$, where m is the size of the universal set.

  - □ But constant factor (1/32?) is very low.

Hashing

A "magical" way to get from an element x to the place where x can be found.

- An array `[0..B-1]` of *buckets*.

  - □ Bucket = list of set elements. Array holds *header* (pointer to first cell).

  - □ B = number of buckets.

- A *hash function* that takes potential set elements and produces a "random" integer in the range `[0..B-1]`.

Example: If set of integers, simplest/best hash function is usually $h(x) = x \bmod B$.

- Suppose $B = 6$, and we wish to store the integers 70, 53, 99, 94, 83, 76, 64, 30.

  - □ These were obtained by turning to a random page of the Stanford phone book.

- They belong in buckets 4, 5, 3, 4, 5, 4, ,4, and 0, respectively.

Pitfalls in Hash Function Selection

- Goal is uniform distribution of elements into buckets.

- Beware of a physical phenomenon that causes nonuniform distribution.

Example:

- If integers were all even, then $B = 6$ would cause only buckets 0, 2, and 4 to fill.

- If we hash the words in `/usr/dict/words` into 10 buckets by length mod 10, then 20% go into bucket 7.

Implementing Dictionary Operations

Lookup x by

1. Go to the head of bucket $h(x)$.

2. Search the bucket list. If x is anywhere it is in this bucket.

- Insert similar: Go to bucket $h(x)$ and search for x. If not there, append x.

- Deletion similar: Go to bucket $h(x)$ and delete x from the list if it is there.

Analysis of Dictionary Operations/Hashing

- If we pick B to be approximately n, the number of elements in the set, then the average list is $O(1)$ long.

- Thus, dictionary operations take average $O(1)$ time each.

- However, in the worst case, all elements are in one bucket, and we get $O(n)$ per operation.

But How Do We Keep B Near n?

If n gets as high as $2B$, create a new hash table with $2B$ buckets.

- Rehash every element into the new table.

  - □ Takes $O(n)$ time total.

- But there were at least n inserts since the last time we "rehashed."

  - □ These inserts took time $O(n)$.

- Thus, we may "amortize" the cost of rehashing over the inserts since the last rehash, at most multiplying the time for those inserts by a constant factor.

  - □ i.e., even with rehashing, hashing takes $O(1)$ time average per dictionary operation.