Hash tables:

Some common hash functions:

Table size = m (T[0..(m-1)])

Keys are integers:

$h(k) = k \bmod m$

Keys are strings:

```
int h(char *s) // s is null-terminated string
{
    int sum;
    sum = 0;
    while (*s) {
        sum = 37 * sum + *s;
        sum %= m;
        s++;
    }
}
```

Best if NUM is coprime with m. (no factors in common)

NUM ≠ 1

NUM ≠ m-1

Best if m is prime

Hash table with m entries & storing n items.

Load $\alpha = \frac{n}{m}$

= ave length of a linked list if chaining is used

(aka, utility factor, traffic intensity).

Good to have, say, $\frac{1}{2} \le \alpha \le 2$

Probing (open addressing)

All items stored in the array itself.

Dealing with collisions is fraught.

— one approach: linear search

if new item collides with existing item, search sequentially (cyclically) for free entry to put the new item.

disadvantages:

— deletion is messy

— clustering:

big blocks of contiguous occupied entries grow easily and make search inefficient

General idea with open addr: $\alpha \le 1$

($\alpha \approx 1$ makes poor performance).

Other probing strategy for collision: used in 2nd hash for sequential search.

Binary Search Trees

Collection supporting

Insert(x)

Search(x)

Delete(x)

Advantages over a hash: support other ops: a BST

— (access through items in order)

— selection of the kth smallest item.

Binary tree, each node stores one data item (with key) no duplicates.

Disadvantages over a hash

— items must (easily) be comparable

— worst-case behavior is bad (O(n) for n items)

but there are ways of improving on this

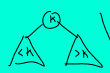
actual worst-case behavior is O(d) where d is the tree depth.

Ways of keeping d small

— AVL trees

— red-black trees (related to 2-3 trees)

Does randomness help? Yes



everything in the tree.

If items inserted in random order (all $n!$ insertion orderings equally likely), then

expected depth is $O(\lg n)$

Analysis is similar to that of randomized quicksort.

n items, laid out in increasing order

$E(n)$ = expected time for n insertions in random order

satisfies

$$E(n) = \Theta(n) + \frac{1}{n}\sum_{j=0}^{n-1}\left(E(j) + E(n-j-1)\right)$$

same recurrence as for randomized quicksort, so $E(n) = n \lg n$

Similar analysis gives expected depth $= O(\lg n)$.

Works if all n items are available beforehand, but this is usually not the case.

Treap data structure

Binary search tree where each item has both a key (for the BST) and a priority (numerical) such that the keys are arranged in BST order and the priorities are in max-heap order (child priority $\leq$ parent priority)

Theorem: Given (key, priority) pairs (no duplicate keys, no duplicate priorities), the shape of the treap is uniquely determined by these pairs, so shape is independent of the insertion order.

Proof: root is max priority element

contents of subtrees uniquely determined by the root.

Applying this same idea recursively key:priority pairs in those their shapes are uniquely determined by the respective subtrees. ∎

Example

key	A	B	C	D	E	F	G	H
priority	1	75	7	37	84	33	12	98