, 13, 14, 15, 16,

CS 240 - Data Structures and Data Management **Spring 2017**

Lecture 12, 13, 14, 15, 16,: June 13 - June 29, 2017

Lecturer: Taylor Smith *Notes By: Harsh Mistry*

12.1 Tries

Definition 12.1 *A Trie (Radix Tree) is a dictionary for binary strings*

- *Insert : search for x and suppose we finish at a node v*
- *Delete : search for x*
  - *If x found an internal flagged node, then unflag node*
  - *if x found at a leaf, delete leaf and all ancestors until we reach an ancestor that has two children or is flagged.*

Time complexity of all operations is $\theta(\mid x \mid)$

Compressed Tries (Patricia Tries)

- Reduces storage requirement : eliminate unflagged nodes with only one child
- Every path of one-child unflagged nodes is compressed to a single edge
- Each node stores an index indicating the next bit to be tested during a search
- A compressed trie storing n keys always has at most n-1 internal nodes

- Search : follow the proper path from the root down in the tree to a leaf. If search ends in an unflagged node, its unsuccessful. Otherwise check if the key stored is indeed x
- Delete :
  - Perform search for x
  - if search ends in a internal node, then unflag with there is 2 children or delete the node and promote the only child
  - if search ends in a leaf then delete the leaf
  - if its parent is unflagged, then delete the parent,
- Insert :
  - Perform Search(x)
  - If the search ends at a leaf L with key y, compare x against y.

- If y is a pre
  x of x, add a child to y containing x.
- Else, determine the first index i where they disagree and create a new node N with index i .
- Insert N along the path from the root to L so that the parent of N has index $<$ i and one child of N is either L or an existing node on the path from the root to L that has index $>$ i . The other child of N will be a new leaf node containing x.
- If the search ends at an internal node, we and the key corresponding to that internal node and proceed in a similar way to the previous case.

Multiway Tries

- To represent Strings over any fixed alphabet $\sum$
- Any node will have at most $|\sum|$ children
- Special characters are used to represent the end of a path

12.2 Hashing

Direct addressing isn't possible if keys are not integers and storage is very wasteful.

Definition 12.2 *Array T of size M (the hash table). An item with key k is stored in T(h(k)]. search, insert, and delete should all cost 0(1).*

Collision Resolution

Even the best hash function may have collisions: When we want to insert (k,v) into the table, but T[h(k)] is already occupied.
Two basic strategies to resolve this are :

- Allow multiple items at each table location (buckets)
- Allow each item to go into multiple locations.

Chaining

Each table entry is a bucket containing 0 or more KVPs. This could be implemented by any dictionary (even another has table).
The simplest approach is to use an unsorted linked list in each bucket. This is called collision resolution by chaining.

- Search : Look for for key k in the list T[h(k)] : $\theta(1 + \alpha)$
- Insert (k,v) : Add (k,v) to the front of the list at T[h(k)] $O(1)$
- Delete (k)) : Perform a search, then delete from the linked list $\theta(1 + \alpha)$

Double Hashing

Say we have two hash functions h_1 that are independent. So, under uniform hashing, we assume the probability that a key k has $h_1(k) = a$ and $h_2(k) = b$ for any particular a and b is

$$\frac{1}{M^2}$$

For double hashing, define $h(k, i) = h_1(k) + i \cdot h_2(k) mod M$

Basic Hash Functions

If all keys are integer (or can be mapped to integers) the following two approaches tend to work well

- Division Method : h(k) = k mod M, Given M is prime
- Multiplication method : h(k) = $floor(M(kA - floor(kA)))$

Advantages

- O(1) cost, but only on average
- Flexible load factor parameters
- Cuckoo hashing achieves O(1) worst case for Search and Delete