Hash tables

Dictionary ADT

• Reminder: A dictionary stores pairs (key, information)

• Operations:
  – find(key k)
  – insert(key k, info i)
  – remove(key k)

• Binary Search Trees implement all these operations in time $O(h)$, where $h$ is the height of the tree, which is $O(\log n)$ if we maintain the tree balanced
  • We can sometimes do better...

Hash tables

• Suppose keys are integers between 0 and K-1
• Then, use an array $A[0...K-1]$ of type "info" to store the dictionary:
  – insert(key k, info i): $A[k] = i$;
  – remove(key k): $A[k] = null$;
  – find(key k): return $A[k]$

• Running time: All operations are $O(1)$
  • It's a miracle! Except that...

Problems with direct array implementation

• If K is large, the array will be very big
  – For McGill student ID, $K = 1\ 000\ 000\ 000$
• The amount of memory needed (K) is independent of the number of items in the dictionary.
  • Idea: compress the array...

Hash functions

Idea: Map the K possible keys to N integers, with N being much smaller than K

Hash function $f$: $[0...K-1] \rightarrow [0...N-1]$

Space of keys: 0 1 2 ..... N-1

Hash function

Hashed key

insert(key k, info i): $A[f(k)] = i$;
remove(key k): $A[f(k)] = null$;
find(key k): return $A[f(k)]$;

Problems with direct array implementation

• Collisions! Many keys map to the same index
• Solution: Each element of the array (bucket) is itself a dictionary, implemented with linked-list, binary search tree, or a hash table...

Hash tables

• Collisions! Many keys map to the same index
• Solution: Each element of the array (bucket) is itself a dictionary, implemented with linked-list, binary search tree, or a hash table...

Hash function $f$: 0 1 2 3 4 5 6 7 8 9 ...

insert(key k, info i): $A[f(k)].insert(k,i)$;
remove(key k, info i): $A[f(k)].remove(k)$;
find(key k): return $A[f(k)].find(k)$;
Importance of good hash functions

- Worst case complexity for hash table containing n elements
  - if all keys end up in the same bucket and we use a linked-list to store buckets??
  - if keys are evenly spread among the N buckets??
- We want a hash function that spreads the keys evenly among the buckets.
- Example: N = 100, key = student ID #
  \[ f(k) = \lfloor k/1000000 \rfloor \mod 100 \]  
  \[ f(k) = k \mod 100 \]  
  \[ f(k) = \text{sum of digits of } k \mod 100 \]
- To avoid patterns in the data, choose N to be a prime number, e.g. 101

Good hash functions

- Choice of hash function depends on application
  - In general, \( f(k) = k \mod N \) is good choice when N is a prime number
- What if the key is not an integer (e.g. a String)?
  - map key to integer first with some function \( g(key) \)
  - use \( f() \) to map the integer to \([0...N-1]\)

Hash functions on Strings

- We need a function \( g: \text{String} \rightarrow \text{Integers} \) that minimizes collisions
  - Linear code:
    \( g(key) = \text{sum of ASCII values of each char.} \)
  - Polynomial code: Choose a small prime number \( a \)
    If key \( k = k_0k_1k_2...k_e \), choose
    \( g(k) = k_0 + k_1a + k_2a^2 + ... + k_ea^e \)