EECS 114: Engineering Data Structures and Algorithms

Lecture 4

Brian Demsky

bdemsky@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine
Announcements

• Programming Assignment due Monday
Binary Search Trees

- Store data in a tree
- Each node can have up to two child nodes
  - one left child
  - one right child
Binary Search Trees

- For balanced tree, we can search for n nodes for a key in $O(\log n)$ steps
Can we do better?

• Can we find a key/value pair in constant time?
Can we do better?

- For small keys, we could index into an array with the key, and return the corresponding element.
Can we do better?

- Doesn’t work for large key spaces, but what if we compute `inputkey % sizeofarray` as index?
Can we do better?

- Doesn’t work for large key spaces, but what if we compute inputkey % sizeofarray
- Could potentially have a collision – two keys that correspond to the same array element
Use linked list of items

• Can use an array that points to linked lists of key/value pairs
Hash Table

- Components:
  - Hash function $h$ that maps keys of a given type to integers in a fixed interval $[0, N-1]$
    - Example:
      \[ h(x) = x \mod N \]
    - Can have Hash functions for strings, etc...
  - Comparison function
    - Tells us whether two keys are equivalent
  - An array of size $N$
Hash Function

- Can think of hash function as having two components
  - A mapping from keys -> integers
  - A (compression) mapping from integers -> [0, N-1]
- Goal is to get an even distribution of keys throughout the hash table array
- Can also use object pointer values if we want equality to mean the exact same object (not so good for integers)
Hash Functions for Objects with Many Fields

- If any object has many fields, we can compute a hash function for the object that combines these fields in some way
- Typically use
  - Summation
  - Exclusive OR
Polynomial Accumulation

• Partition the bits of the key into a sequence of components of fixed length (8, 16, or 32 bits)
  \[ a_0a_1\ldots a_{n-1} \]

• Evaluate the polynomial
  \[ p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \]
  at fixed value \( x \)

• Use horner’s rule for polynomial time evaluation
  \[ p_0(x) = a_{n-1} \]
  \[ p_i(x) = a_{n-i-1} + x p_{i-1}(x) \]
Idea:

- Data structure known as hash table
- Idea is to grow the array so that the linked lists are always short – gives constant time access to key/value pairs
- Hash function takes key (in this case an integer) and tells us what index in the array to look – in this case we use key % arraysize as the hash function
Search

• Looking for value corresponding to key=8
• First compute key % arraysize(6)=2
• Look in element 2 of array
Search

• Looking for value corresponding to key=8
• First compute key % arraysize(6)=2
• Look in element 2 of array
Search

- Look in element 2 of array
- Follow linked list until we find element with key=8
Search

- Follow linked list until we find element with key=8
- Found key=8, value=12
Search

- If we didn’t find it, then key/value pair isn’t in the hashtable
Adding a key/value pair

- Compute proper array element using the hash function
- Add key/value pair to beginning of the linked list for that element
Alternative Methods for Collision Handling

- Linear probing
  - Store item in next free cell
  - To search, keep looking at consecutive cells until you either find the key, an empty cell, or have wrapped around
  - Removals are complicated - have to use special value to indicate that spot is available but to probe past it
- Collisions can bunch up
Double Hashing

- Have a hash function that produces a series of values and place item in first available slot
- \((i+j \ h'(k)) \mod N\) for \(j=0,1,...N-1\) where \(N\) is prime
- \(i\) is the value of the original hash function
- Secondary hash function \(h'(k)\) can’t have zero values
- Table size \(N\) must be prime to allow probing of all cells
Resizing Hashtables

- Typically, hashtable automatically resize themselves when linked lists begin to get long
  - Have to reallocate array
  - Rebuild linked list of key/value pairs for the new array
  - Free old array
Performance

• Worst case: everything falls into the same bin
• Searches, removals take O(n) time
• Insertions may also take O(n) time
• Load factor alpha = n/N - affects performance
• Can show that expected number of probes is 1/(1-alpha) (for hash tables that store all items in array)
• Expected time is O(1)
Universal Hash Functions

- Idea - hash functions that produce a uniform flat distribution across the array
- Formally, for $0 \leq i, j \leq M-1$, $\Pr(h(i)=h(j)) \leq 1/N$
- Choose $p$ as a prime between $M$ and $2M$.
- Randomly select $0 < a < p$ and $0 \leq b < p$, and define $h(k)=(ak+b \mod p) \mod N$