

CS 209

Data Structures and Mathematical
Foundations

04 / 15 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions?/Comments?
- Hash Tables

Hashing

- Hashing is used for what?

Hashing

- Hashing is used to allow
 - inserting an item
 - removing an item
 - searching for an item

all in constant time (in the average case).
- Hashing does not provide efficient sorting nor efficient finding of the minimum or maximum item etc.

Hashing

- Let's continue our implementation of the hash table using separate chaining
- Let's count the number of collisions that happen
- Let's consider a different hash function to hopefully reduce the collisions
- Let's implement a probing style (quadratic probing) collision strategy for a hash table.

Hashing

- Strategies for best performance
 - want items to be distributed evenly (uniformly) throughout the hash table and we want few (or no) collisions
 - so that depends on our data items, our choice of hash function and our size of the hash table
 - also need to decide
 - whether to use a probing (linear or quadratic) hash table
 - or separate chaining - adding the item to a linked list for the index (hash value)
 - another method is called double hashing.
 - if choices are done well we get the retrieval time to be a constant, but the worst case is $O(n)$
 - we also need to consider the computations needed to compute the hash value (note: this will be a constant amount of work, but we should avoid high constants if possible)

Hashing

- Clustering
 - Why is it bad?
 - How to avoid it?
 - Quadratic probing (avoids it to some degree)
 - Double Hashing can avoid it

Hashing

- It is possible, with a poor choice of table size, and quadratic probing, we could end up never finding a spot in the table to place an item.
- For quadratic probing, if the table size is prime, then a new element can always be inserted if the table is at least half empty.

Hashing

- Double hashing is an open address hashing method
 - - Create a hash table as a list of items
 - - Create a hash function `hashf1` that will return a hash value that we will call `hv1` to place the item in the table.
 - - If a collision occurs, call a second hash function `hashf2` which returns an int that we will call `hv2`. Use this `hv2` as the amount of indices to hop to find another place to put the item.

Hashing

- Example: if an item initially hashes to value $h_1=5$ and this causes a collision, then using the same item we compute h_2 to be 7 using the second hash function.
- We would then check if there's an open slot in $5+7=12$. If it's open, place the item there. If that's not open, look in $12+7=19$, and if that's not open continue checking 7 slots away until we find an open slot. Wrap around when necessary (using mod by the size of table).
- Remember the goals of hashing:
 - - we want the data to be distributed well throughout the hash table
 - - we want few collisions in the average case and the worst case

Hashing

- Another thing that must be taken care of is a situation such as this.
- Suppose we have a hash table of size 20 and all of the even numbered slots are filled (that is, index 0, 2, ... 18) and we are trying to insert an item with h_{v1} of 4. This is a collision. So if we compute h_{v2} to be say 6,

Hashing

- $(4+6)\%20=10$ is filled,
- $(10+6)\%20=16$ is filled,
- $(16+6)\%20= 2$ is filled,
- $(2+6)\%20= 8$ is filled,
- $(8+6)\%20=14$ is filled,
- $(14+6)\%20= 0$ is filled,
- $(0+6)\%20= 6$ is filled,
- $(6+6)\%20=12$ is filled,
- $(12+6)\%20=18$ is filled,
- $(18+6)\%20= 4$ is filled,
- $(4+6)\%20=10$ (we're back where we started ...)

Hashing

- and this would go on forever, because we came back to the starting index without examining all the slots. Actually in this case we examined all the filled slots, and ignored all the empty slots!

Hashing

- To avoid this problem, double hashing requires that the table size must be relatively prime with respect to the value (h_2) returned by $hash_2$. This is important because this will guarantee that if there's an empty slot, we will eventually find it.
- Numbers that are relatively prime are those which have no common divisors besides 1.
- You could have a `table_size` which is an integer p , such that p and $p-2$ are prime (twin primes). Then the procedure is to compute h_1 to be some int within the range 0 to `table_size-1`, inclusive. Then compute h_2 to be some int within the range 1 to `table_size-3`.
- This will guarantee that if there's an empty slot, we will find it. This method was devised by Donald Knuth.

Hashing

- Strategies for best performance
 - For double hashing & quadratic probing we should have a prime as our table size
 - Additionally for double hashing we need the second hash value to be relatively prime to the table_size.
 - This will prevent the situation described earlier with the table being half full but we never looked at any of the open slots.
 - The hv2 being relatively prime with table_size will guarantee that if there's an open slot, we will find it.