**9. Table Doubling, Karp Rabin**

Link: <https://www.youtube.com/watch?v=BRO7mVIFt08>

How to choose m?

* We want m = Θ(n)
  + α = Θ(1)
* IdeaL start small: m = 8
* grow/shrink as necessary

Grow table:

* m -> m’
* Allocate memory and rehash
* Make table of size m’
  + Build new hash f’
  + Rehash:
    - For each item in T:
      * T’.insert(item)
* Will take Θ(n+m+m’) which is Θ(n)
  + n is the amount of k,v in linked list, m is spots in original hash table, and m’ is writing them to the new hash table
* m’ = 2m aka table doubling
  + Grows in Θ(1 + 2 + 4 + 8 + … + n) <- Grows in a geometric fashion
  + Θ(n)
  + Some of them are restructred at logn speed while others grow at n due to a quick insertion in table with no collision

Amortization:

* Operation takes “T(n) amortized”
  + If k operations
  + Take <= k \* T(n) time
  + Think of meaning
  + T(n) on average where average over all operations
* Table doubling:
  + K inserts
  + Take Θ(k) time
  + Θ(1) amortized/insert
  + Also, k inserts and deletes take O(k)
* Table shrink
  + Slow method: If m = n/2 then shrink by m/2
    - The issue is that if you have 8 keys in a table size of 8 and add a 9th. You double your table size. Yet, if you minus 1 out, you shrink again. So you’re running an expensive operation of shrinking and growing the table when alternating between 8 and 9.Θ(n) per operation
  + If m = n/4 then shrink -> m/2
    - Amortized time -> constant Θ(1)

String Searching

Simple (Naive) Algorithm:

* any(s==t[i:i + len(s)] for i in range(len(t) - len(s)))
* Runs in O(|s| \* |t|) , can be quadratic

Rolling Hash ADT:

* r.append(c):
  + Add chat c to end of x
* r.skip(c): delete first char of x (assuming it is c)
* r maintains a string x
  + r(): hash value of x x = h(x)

Karp Rabin algorithm:

* Uses rolling hashes
* For c in s: rs.append(c)
* For c in t[:len(s)]:
  + rt.append(c)
* If hs() == ht():...
* For i in range(len(s), len(t)):
  + rt.skip(t[i - len(s)]
  + rt.append(t[i])
  + If rs() == rt()
    - # This does not mean the strings are equal, there could be a collision hash
    - Check whether s == t[i-len(s) + 1: i+1]
    - If equal
      * Found match
    - Else:
      * Happens with probability <= 1/|s|
      * O(1) expected time
* It all amounts to linear time O(|s| + |t| + #match |s|) in expectation
* Use a random prime number >= |s|
* Treat x as multidigit number u in base a (alphabet size)
* The picture below shows the operation and code for recalculating the hash value when appending and skipping
* 