Hashing

- One of the most important data structures, with numerous applications to both algorithms and complexity
- Applications:
  Dictionary data structure

Dictionaries

- Large universe of possible keys – universe size U. Generally U
- Storing a small subset S of U: |S|=n
- Operations supported
  - Insert(k) – add the key k to the set S
  - Find (k) – is the key k in S?
  - Delete (k) – remove the key k from S.
- Sometimes only care about the static case.

Hashing

- One of the most important data structures, with numerous applications to both algorithms and complexity
- Applications:
  Dictionary data structure
  Load balancing
  cryptography

Next few lectures

- What we want from a hash function
- Constructions (universal hashing)
- Applications and analyses:
  - Perfect Hashing
  - Linear probing
  - Bloom Filters
  - Hashing for load balancing (Power of two choices, Cuckoo hashing)
  - Hashing for document similarity (min-hashing, locality sensitive hashing)
  - Applications to streaming

Dictionaries via hashing

- Universe size U, |S|=n
- Define a hash function h: U → [m]
- Store each key x in location h(x).
- What to do about collisions?

What do we want from hash function

- small number of collisions
- m small, specifically O(n).
- hash function easy to describe (small representation)
- hash function easy to compute

The importance of being random

- For any fixed hash function there is a set of bad keys.
  - Example: h(x) = x mod m
- If input comes from such a subset, disaster!

Input data is not random!

So good hash functions must be random!

Suppose hash function h is random

Claim: If h is random, then the expected time to perform any sequence of m operations is O(m).

Assume that all items that hash to the same location are stored in a linked list from that location.

Claim: If h is random, then the expected time to perform any sequence of m operations is O(m).

Claim: If h is random, then the expected time to perform any sequence of m operations is O(m).

- Conclusion: random hash function is great!!
- But useless… except as inspiration…
- [Carter, Wegman]: We didn't use very much about the randomness.

[CW] simple but brilliant idea

- Choose h at random, but from a small space of possible hash function.
- Let *H* be a class of functions mapping U to [m]. We say that H is universal if for any x, y in U (not equal), and h chosen uniformly at random from H,

- Claim: If h is universal, then the expected time to perform any sequence of m operations is O(m).
- Question: how to construct small, efficient, universal family of hash functions?

- Let p be a prime > |U|. The following family is universal:

Your turn: show the following family of hash functions is universal.

- Take a u by k matrix A and fill it with random bits. (2^k=m)
- For x in U, view it as a u-bit vector and define

  h(x) := Ax,

  where calculations are done mod 2.

Can we create a collision-free hash table?

Perfect Hashing [FKS]

- How can we use these ideas to create a hash-table based data structure, where the worst-case time to perform an operation is constant.
- Consider static case

Linear probing

Linear probing and k-wise universal hash functions

- Analysis we just did was for random hash functions..
- Universal hash functions bad
- Something in between?
- k- (strongly) universal hash functions

Linear probing

- Analysis we just did was for random hash functions..
- Universal hash functions bad
- Similar results can be shown with 5-independent hash functions, but not 4-independent!