EECS 495: Randomized Algorithms **Lecture 8**
Hashing

Reading: Text:

Hashing

Symbol-table problem

Set S holding n elements:

x pointer to element containing

- $key(x) \in U = \{1, \ldots, N\}$
- satellite data

Operations on S

- $insert(S, x) : S \leftarrow S \cup \{x\}$
- $delete(S, x) : S \leftarrow S - \{x\}$
- $search(S, k)$: returns x if $\exists x \in S, key(x) = k$, otherwise nil

Question: How to store S?

- array: operations $O(1)$, space $O(N)$
- hash table

Def: hash function $h : U \rightarrow \{0, \ldots, b-1\}$, $b = poly(n) << N$

- if well-distributed, get constant operations, near-linear space
- if $h(k_1) = h(k_2)$, collision, increase time for operations

Deterministic function, then adversary can pick keys s.t. all data maps to same bucket, so want to choose function at random from some set; choosing from all functions uniformly at random is bad however, because function must be easy to compute. Solution is to use a small family of functions that are easy to compute and then choose from that family randomly.

Question: how to resolve collisions?

- chaining: store collided data in a linked list
  - worst-case $O(n)$ operations
  - average case $O(1)$ using $b = O(n)$
  - average worst-case $O(\log n / \log\log n)$ using $b = O(n)$ and "random" function (balls and bins)
- perfect hashing, i.e., constant operations worst-case, for static sets: store collided data in secondary hash table

  Claim: $b = O(n)$ size suffices!

  Proof: Let B_i be # elts. in bin i:

$$\begin{aligned} E[\sum_i (B_i)^2] &= n + E[\#\text{colliding pairs}] \\ &= n + n^2/b \end{aligned}$$

$$= \quad O(n)$$

So secondary hash tables in sum use also linear space.

- linear probing: if $h(k)$ occupied, try $h(k)+1$ etc.

  [[*Key advantage, good for cache misses due to sequential access.*]]

- cuckoo hashing: use two hash functions, if $h(k)$ occupied, kick resident elt to bucket in $g(.)$ recursively

  [[*Advantages mostly theoretical.*]]

Linear Probing

[[*See STOC'07 paper of Pagh, Pagh, Ruzic.*]]

Note: Analysis for $b = 3n$ to ease notation.

Consider binary tree spanning array of buckets:

- leaves level 0
- node at level k has 2^k array positions under it
- expect node of level k to have $(1/3)2^k$ items hashed to buckets under it

  [[*In sense of original location $h(x)$, not $h(x)+1$, $h(x)+2$, etc.*]]

Def: A node of level k is *dangerous* if more than $(2/3)2^k$ elts hash under it.

To bound operation time, must bound size of contiguous run of elts. containing $h(q)$:

Claim: If $2^k \leq$ size of run $\leq 2^{k+1}$, either $(k-2)$-ancestor of $h(q)$ or a nearby sibling is dangerous.

Proof: Consider $(k-2)$-level nodes that span run. Given size, there are at least 4 of them. Claim at most 3 can be good:

- 1st good $\rightarrow$ contributes at most $(2/3)2^{k-2}$ to run
- 2nd, 3rd good $\rightarrow$ have each $2^{k-2}/3$ empty slots
- so next two soak up excess from 1st stopping run
- 4th good means get at most $(2/3)2^{k-2}$ extras

Total # elts. in run at most $2^{k-2} + 2^{k-2} + (2/3)2^{k-2} < 4 \times 2^{k-2} = 2^k$.

Let E_k be event a level-k node is dangerous. Expected operation time:

$$\sum_k O(2^k) \Pr[2^k \mathrm{run}(h(q)) \leq 2^{k+1}] \leq \sum_k O(2^k) \Pr[E_{k-2}].$$