6.1220: DESIGN AND ANALYSIS OF ALGORITHMS LECTURE 6

BRENDON JIANG

Contents

1. Hashing (Part I)

Definition 1.1: Dictionaries/Maps

A dictionary/map is a core data structure that encode associations:

$$\text{Key k} \mapsto \text{Value v}$$

We want this data structure to support three operations

- INSERT(k,v): Add key-value pair(k,v)
- DELETE(k): If there is item x in the dictionary with x.key, delete it from data structure
- SEARCH(k): Given key k, return the value v associated with it (null otherwise)

Here are some data structure specific notations:

- $\mathcal{U}$ = universe of all possible keys (e.g. all ≤ 64 character strings)
- $n = \#$ of objects in dictionary
- m = "size" of the dictionary
- α = n/m "load factor"

We assume $\mathcal{U} >> n$. Here is a naive solution:

Solution 1.2: Doubly Linked List

- INSERT(k,v): Add a node storing (k, v) at the head
- DELETE(k): Traverse list to find and delete x such that x.key = k (and update pointers)
- SEARCH(k): Traverse list to find x such that x.key = k

The time complexities are $O(1)$ insertions and $O(n)$

Solution 1.3: Solution 2: Direct Addressing

Consider each key as simply the address. Then we need $O(|\mathcal{U}|$ space, which is unfeasible.

Date: September 2024

Solution 1.4: Solution 3: Balanced Search Trees

Time complexities will be $O(\log n)$, space will be $m = O(n)$.

Now we build toward the idea of hashing. We reduce the universe size with hash function $h : \mathcal{U} \to \{0, \ldots, m-1\}$. Assume $m = O(n)$. But there are problems:

(1) How do we handle collisions, i.e. $k_1, k_1 \in \mathcal{U}$ s.t. $h(k_1) = h(k_2)$?
(2) How to design the hash function $h : \mathcal{U} \to \{0, \ldots, m-1\}$?
(3) How to represent h?

Solution 1.5: Solution 4: Hashing + Chaining

We use the an array of double linked list to resolve collisions. If two keys conflict, put the newer one at the top of the current linked list.

The time complexity is $O(1)$ insert (assuming hash function is $O(1)$), and $O(m+n)$ space (assuming hash function is $O(1)$ space.

What about complexity of search? (delete is different). At least one of the sets of keys with same hash value has size $\geq \dfrac{\mathcal{U}}{n}$. If we have a deterministic hash function, then an adversary can blow up the hash table by by choosing keys with that same hash value. To confuse our adversary, we use a randomly chosen hash function.

Definition 1.6: Random Oracle

Every key k gets uniformly random $h(k) \in \{0, \ldots, m-1\}$ independently of all others.

It is actually impossible to implement a truly random oracle. However, let us analyze this assuming we have a good implementation, and see if collisions work out in our favor. We look at the expected load.

Claim 1.7: Expected load

For all index $i \in \{0, \ldots, m-1\}$ and for all sequence of keys $k_1, \ldots, k_n \in \mathcal{U}$. Let $N_i = \#\{j : h(k_j) = i\}$. Then, we claim $\mathbb{E}[N_i] = \dfrac{n}{m} = \alpha$.

Proof

Use linearity of expectation. Define $X_{i,j} = \mathbb{I}[h(k_j) = i]$.

$$\mathbb{E}[N_i] = \mathbb{E}[\sum_{j=1}^{n} X_{i,j}] = \sum_{j=1}^{n} Pr[h(k_j) = i] = n \cdot \frac{1}{m} = \frac{n}{m} = \alpha$$

This is good if $m = O(n)$, which means $\alpha = O(1)$. But do we really need a truly random oracle to achieve this?

Definition 1.8: Family of hash functions

We say a family of hash functions $H \subseteq \{\text{set of all hash functions}\}$ is uniform if for all $k \in \mathcal{U}$, for all $i \in \{0, \dots, m-1\}$,

$$Pr_{h\sim H}[h(k) = i] = \frac{1}{m}$$

Example 1.9

Let $H = \{h_i : i \in \{0, \dots, m-1\}\}$ where $h_i(k) = i$ for all $k \in \mathcal{U}$. It is not hard to see that the probability is indeed $\frac{1}{m}$, yet it is still a horrible hash function (only one bucket).

We need some other property on the family of hash function to ensure its "goodness".

Claim 1.10

For all $k_1, \dots, k_n \in \mathcal{U}$, for all j, define $C_j = \#\{j' : h(k_j) = h(k_{j'})\}$ Then $\mathbb{E}[C_j] = \frac{n-1}{m} \leq \alpha$ for the random oracle h.

Proof

For $j' \neq j$, Let $Y_{j,j'} = \mathbb{I}[h(k_j) = h(k_{j'})]$.

$$\mathbb{E}[C_j] = \sum_{j'\neq j} \mathbb{E}[Y_{j,j'}] = \sum_{j'\neq j} Pr[h(k_j) = h(k_{j'})] = \frac{n-1}{m} \leq \alpha$$

Definition 1.11: Universal hash functions

We say a family of hash functions H is universal if $\forall k, k' \in \mathcal{U}$ distinct, $Pr_{h\sim H}[h(k) = h(k')] \leq \frac{1}{m}$. The probability of collision for a random hash function is $\leq \frac{1}{m}$.

Our goal now is to construct a universal family of hash functions. Assume our universe $=\{0, \dots, |\mathcal{U}|-1\}$.

The idea is to use modular arithmetic arithmetic on integers. $h : \{0, \dots, |\mathcal{U}|-1\} \to \{0, \dots, m-1\}$.

Proposition 1.12

Attempt 1: $h(k) = k + s \mod m$, for a uniformly random $s \in \{0, \dots, m-1\}$. But the set of keys that get mapped to the same hash value are the same, meaning for some k, k' we have $Pr[h(k) = h(k')] = 1$ always.

Proposition 1.13

Attempt 2: Suppose we look at $H = \{h_s : s \in \{0, \dots, m-1\}\}$ and $h_s(k) = sk \mod m$. Lets look at $m = 6$. Factors of m have bad hash distributions (ex: $h_2(k) = 2k \mod m$). This is bad if m has many factors.

We can get around the issue with prime m. We now use a more sophisticated construction.

Example 1.14: Definition of a class of universal hash function by example

Assume $m = 7, |\mathcal{U}| = 343 = 7^3$. Mapping $\{0, \dots, 342\} \to \{0, \dots, 6\}$. We use base-7 representation of keys: $k = (k^{(2)}, k^{(1)}, k^{(0)})$. For example, $14 = k = (020)$.

$H = \{h_a : a = (a^{(2)}, a^{(1)}, a^{(0)}) \in \{0, \dots, 6\}\}$ where $h_a(k) = k^{(2)} \cdot a^{(2)} + k^{(1)} \cdot a^{(1)}, k^{(0)} \cdot a^{(0)} \mod m$. We claim this is a universal hash function.

Proof

Proof by example.
$k = (1,1,2), k' = (1,1,3)$. We want to bound $Pr_{h \sim H}[h(k) = h(k')]$.

$$Pr_{h \sim H}[h(k) = h(k')] = Pr[a^{(2)}+a^{(1)}+2a^{(0)} = a^{(2)}+a^{(1)}+3a^{(0)} \mod m] = Pr[a^{(0)} = 0] = \frac{1}{7} = \frac{1}{m}$$

Idea is if one coordinate k and k' is different, then at least one value of $a^{(i)}$ differ in the LHS and RHS, and the probability the equation is correct is $\frac{1}{m}$ as there is only one value of $a^{(i)}$ that makes the equation true.

Example 1.15

Note: Bertrand's postulate ensures there is a prime betwen $n \leq p \leq 2n$, meaning we can always find reasonable p.

Let us explore another approach to address collisions (compared to doubly linked list).

Definition 1.16: Open Addressing

If $h(k_1) = h(k_2)$ and k_1 is already in the dictionary, "search" for a open spot for k_2. Formally, each k gets a permutation of $\{0, \dots, m-1\}$ as $h(k,0), H(k,1), \dots, h(k, m-1)$. Note: need $n < m$.

Definition 1.17: Probing scheme

We call $h(k,0), H(k,1), \dots, h(k, m-1)$ a probing scheme. Probing schemes used in practice includes: Linear probing: $h(k,p) = h(k) + p \mod m$ Double hashing probing: $h(k,p) = h_1(k) + ph_2(k) \mod m$