6.1220: DESIGN AND ANALYSIS OF ALGORITHMS LECTURE 7

BRENDON JIANG

Contents

1 Hashing Part II. 1
1.1 Recap of Lecture 6 1
1.2 Plan for this lecture 2
1.3 Open Addressing Analysis. 2
1.4 Static Dictionaries and Perfect Hashing 3

1. Hashing Part II

1.1. Recap of Lecture 6

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)

The solution we went over in the last lecture is combining Hashing and Chaining. We reduce the universe of all keys to a limited amount, and store values to keys in a linked array to account for collisions.

Definition 1.2: Universal hash functions

We say a family of hash functions H is universal if for all$k, k' \in \mathcal{U}$ distinct,

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

The probability of collision for a random hash function is $\leq \frac{1}{m}$.

Date: September 2024

Lemma 1.3

Given a universal hash family, the expected running time of all operations $O(1)$.

Lemma 1.4

We can find efficiently samplable universal hash families exist (from last lecture).

*insert table comparing hash table implementation time complexities

Instead of using linked lists to resolve collisions, we can use a technique called **probing**. When a newly inserted element causes a collision, we can resolve the collision by sending it to another position in the array based on its **probing sequence**.

Definition 1.5

In probing, Every key $k \in \mathcal{U}$ is assigned a probing sequence

$$h(k,0), \ldots, h(k,m-1) \in \{0, \ldots, m-1\}$$

where each succeeding hash value is less preferred.

This works well when the load factor $\alpha = \frac{n}{m}$ is small, as the technique allows cache locality to happen. Cache locality is the effect of retrieving elements near each other in an array to be faster, as they are sent in blocks to the cache.

POP QUIZ!!!!

Question: Open addressing requires $\alpha < 1$. What do we do if α is large (e.g. bigger than $\frac{3}{4}$)?
Answer: Resize the array by doubling the size m. This ensures the expected amortized running time remains $O(1)$. This requires resampling the hash function/basically recreating the entire hash table.

1.2. Plan for this lecture

(1) We perform run time analysis for open addressing assuming uniform permutation hashing (defined below).
(2) Static dictionaries and "perfect" hashing scheme.

1.3. Open Addressing Analysis

Definition 1.6: Uniform Permutation Hashing Assumption

Every key $k \in \mathcal{U}$ receives uniform random permutation $h(k,0), \ldots, h(k,m-1)$ independently.

Lemma 1.7

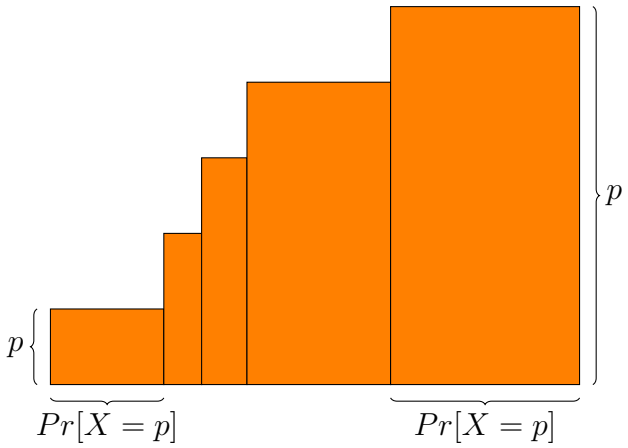
Suppose $\alpha = \frac{n}{m} < 1$. Then for all $k \in \mathcal{U}$, the expected running time of INSERT(k) is $O\left(\frac{1}{1-\alpha}\right)$.

Proof

Let X denote the # of probes INSERT(k) requires. We want $\mathbb{E}[X] \leq \frac{1}{1-\alpha}$.

$$\mathbb{E}[X] = \sum_{p=1}^{m} p \cdot Pr[X = p] = \sum_{p=1}^{m} Pr[X \geq j]$$

The second equality comes from the layered cake representation:



The result comes by looking at the picture horizontally and vertically, summing the areas of the rectangles created.Now we make a subclaim: $Pr[X \geq j] \leq \alpha^{j-1}$.
We prove the subclaim. Define E_p to be the event that the pth probe failed. Then,

$$\begin{aligned} Pr[E \geq j] &= Pr[E_1 \wedge \cdots \wedge E_{j-1}] \\ &= Pr[E_1] \cdot Pr[E_2 \mid E_1] \cdots Pr[E_{j-1} \mid E_1 \wedge \cdots \wedge E_{j-2}] \\ &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-(j-2)}{m-(j-2)} \\ &< \alpha \cdot \alpha \cdots \alpha \\ &= \alpha^{j-1} \end{aligned}$$

Thus, we can finish the proof with

$$\sum_{p-1}^{m} Pr[X \geq j] \leq \sum_{j=1}^{\infty} \alpha^{j-1} = \frac{1}{1-\alpha}$$

Search and Deletions should have similar time complexities, as they are fundamentally not too different from insertion. There is a technicality with deletions (making the element you deleted to be null causes issues in probing), but it is too technical for the lecture.

1.4. Static Dictionaries and Perfect Hashing

In this problem, we are given the keys $k_1, \ldots, k_n \in \mathcal{U}$ up front. Our goal is to build a dictionary supporting SEARCH only. We want a data structure that can achieve these performances:

- $O(1)$ time search in the worst case.
- $O(n)$ space.
- $O(n)$ preprocessing time (The amount of time it takes to create the dictionary).

To start, we first produce a data structure that has $O(n^2)$ space.

Lemma 1.8

Suppose $m \geq n^2$ and H be a universal hash family. For all $k_1, \ldots, k_n \in \mathcal{U}$,

$$Pr_{h\sim H}[\text{no collisions}] \geq \frac{1}{2}$$

Proof

Claim is equivalent to $Pr_{h\sim H}[\exists i \neq j \text{ s.t. } h(k_i) = h(k_j)] < \frac{1}{2}$. We can then use the union bound:

$$Pr_{h\sim H}[\exists i \neq j : h(k_i) = h(k_j)] \leq \sum_{i<j} Pr_{h\sim H}[h(k_i) = h(k_j)] \leq \frac{1}{m}\binom{n}{2} \leq \frac{1}{2}$$

Thus, in expectation, we only need create 2 hash tables with size $m = O(n^2)$ as previously outlined to create a hash table with no collisions. However, we can do better than $O(n^2)$ space, by linking two $O(n)$ space hash tables. This is called the FKS scheme, and the pseudocode looks something like this:

Algorithm 1: FKS

```
Input n keys K ⊆ U
while TRUE do
    Sample h_1 : U → {0, ..., n − 1} from H_1
    Calculate N_i for all buckets i ∈ {0, ..., n − 1}
    if Σ_{i=0}^{n−1} N_i^2 < 4n then
        for i ∈ {0, ..., n − 1} do
            while TRUE do
                Sample h_{2,i} : U → {0, ..., N_i^2 − 1} from H_{2,i}
                If h_{2,i} has collisions: Resample h_{2,i}
                Else: break
        for all k ∈ K do
            insert k at index h_{2,i}(k) of the bucket at index h_1(k)
        return dictionary
return t
```

We expect to generate $h_{2,i}$ two times as per Lemma 1.8. Thus, the total running time is $O(n)$. However, ignoring the if condition, the total space that the algorithm requires is $O(m + \sum_{i=0}^{m-1} N_i^2)$. We now prove that in expectation, we can pick the inner hash tables 2 times to achieve $O(n)$ space.

Lemma 1.9

If $m = n$, then

$$\mathbb{E}[\sum_{i=0}^{m-1} N_i^2] < 2n.$$

We can then turn this with markov inequality to

$$Pr[\sum_{i=0}^{m-1} N_i^2 > 4m] \leq \frac{\mathbb{E}[X]}{4m} \leq \frac{2n}{4m} = \frac{1}{2}$$

This guarantees we expect to only need to pick the inner hash table twice to get $\sum_{i=0}^{m-1} N_i^2 < 4m$, which is $O(n)$ when $m = n$.

Proof

N_i is the number of elements that go into the same bucket with the first hash function. Thus, we can interpret N_i^2 to be the number of pairs of elements that go into the same bucket, and get $N_i^2 = \#\{(k, k') \in k^2 \text{ s.t. } h(k) = h(k') = i\}$. Then,

$$\begin{aligned}
\mathbb{E}[\sum_{i=0}^{m-1} N_i^2] &= \mathbb{E}[\#\{(k, k') \in K^2 : h_1(k) = h_1(k')\} \\
&= n + \sum_{k \neq k'} Pr_{h_1 \sim H_1}[h_1(k) = h_1(k')] \\
&\leq n + n(n-1) \cdot \frac{1}{m} \\
&\leq 2n
\end{aligned}$$