Advanced Algorithms - Notes

Dom Hutchinson

February 3, 2020

Contents

1 Hashing

Definition 1.1 - *Dictionary*
A *Dictionary* is an abstract data structure which stores (*key*, *value*) pairs, with *key* being unique.
A *Dynamic Dictionary* can perform the following operations

Operation	Description
`add(k,v)`	Add the pair `(k,v)`.
`lookup(k)`	Return `v` if `(k,v)` is in dictionary, `NULL` otherwise.
`delete(k)`	Remove pair `(k,v)`, assuming `(k,v)` is in dictionary.

A *Static Dictionary* can only perform lookups

Operation	Description
`lookup(k)`	Return `v` if `(k,v)` is in dictionary, `NULL` otherwise.

Proposition 1.1 - *Implementing a Dictionary*
Many data structures can be used to implement a *Dictionary*. These include, but not limited to:

i) Linked lists.

ii) Binary Search, (2,3,4) & Red-Black Trees.

iii) Skip lists

iv) van Emde Boas Trees.

Remark 1.1 - *Motivation for Hashing*
None of the implementations of a *Dictionary* suggested in **Proposition 1.1** achieves a $O(1)$ run-time complexity in the worst case for all operations. To achieve this we introduce *Hashing*.

Definition 1.2 - *Hash Function*
A *Hash Function* takes in object's key and returns a value which is used to index the object in a *Hash Table*.
Let S be the set of all possible keys a hash function can recieve & m be the number of indexes in its *Associated Hash Table*. Then

$$h : S \to [m]$$

N.B. We want to avoid cases where $h(x) = h(y)$ for $x \neq y (collisions)$.

Remark 1.2 - *Hashing functions assign items to indices with a geometric distribution*

Remark 1.3 - *Avoiding Collisions in Hashing*
When indexing n items to m indicies using a *Hash Function* we only avoid *Collisions* if $m \gg n$.

Definition 1.3 - *Hash Table*
A *Hash Table* is an abstract data structre which extends the *Dictionary* in such a way that time complexity is reduced.
A *Hash Table* is comprise of an array & a *Hash Function*. The *Hash Function* maps an object's key to an index in the array. If multiple objects have the same *Hash Value* then a *Linked List* is used in that index, with new objects added to the end of the *Linked List* (Called *Chaining*).

Proposition 1.2 - *Time Complexity for Dictionary Operations in a Hash Table*
By building a *Hash Table* with *Chaining* we achieve the following time complexities for *Dictionary* operations

Operation	**Worst Case Time Complexity**	Comments
`add(k,v)`	$O(1)$	Add item to the end of *Linked List* if necessary.
`lookup(k)`	O(length of chain containing `k`)	We might have to search through the whole *Linked List* containing `k`.
`delete(k)`	O(length of chain containing `k`)	Only $O(1)$ to perform the actual deletion, but need to find `k` first.

Theorem 1.1 - *True Randomness*
Consider n fixed inputs for a *Hash Table* with m indices. (*i.e.* any sequence of n `add`/`lookup` /`delete` operations).
Pick a *Hash Function*, h, at random from a set of all *Hash Functions*, $H := \{h : S \to [m]\}$. Then

$$\mathbb{E}(\text{Run-Time per Operation}) = O\left(1 + \frac{n}{m}\right)$$

N.B. The expected run-time per operation is $O(1)$ if $m \gg n$.

Proof 1.1 - *Theorem 1.1*
Let x & $y \in S$ be two distincy keys & T be a *Hash Table* with m indexes.
Define $I_{x,y} \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$.
We have $\mathbb{P}(h(x) = h(y)) = \frac{1}{m}$.
Therefore

$$\begin{aligned} \mathbb{E}(I_{x,y}) &= \mathbb{P}(I_{x,y} = 1) \\ &= \mathbb{P}(h(x) = h(y)) \\ &= \tfrac{1}{m} \end{aligned}$$

Let N_x be the number of keys stored in H that are hashed to $h(x)$.
Note that $N_x = \sum_{k \in T} I_{x,k}$.
Now we have that

$$\mathbb{E}(N_x) = \mathbb{E}\left(\sum_{k \in T} I_{x,k}\right) = \sum_{k \in H} \mathbb{E}(I_{x,k}) = n\frac{1}{m} = \frac{n}{m}$$

□

Remark 1.4 - *Why not hash to unique values*
Suppose we want to define a *Hash Function* which maps each key in S to a unique position in the *Hash Table*, T. This requires m unique positions, which in turn require $\log_2 m$ bits for each key. This is an unreasonably large amount of space.

Proposition 1.3 - *Specifying the Hash Function*
Consider a set of *Hash Functions*, $H := \{h_1, h_2, \dots\}$.
When we initialise a *Hash Table* we choose a hash function $h \in H$ at random and then proceed only to use h when dealing with this specific *Hash Table*.

Remark 1.5 - *Randomness in Hashing*
All the randomness in *Hashing* comes from how we choose the *Hash Function* & not from how the *Hash Function* itself runs.

Definition 1.4 - *Weakly Universal Set of Hashing Functions*
Let $H := \{h | h : S \to [m]\}$ be a set of *Hashing Functions*.

H is *Weakly Universal* if for any chosen $x, y \in S$ with $x \neq y$

$$\mathbb{P}(h(x) = h(y)) \leq \frac{1}{m} \text{ when varying } h(\cdot)$$

when h is chosen uniformly at random from H.

Theorem 1.2 - *Expected Run time for Weakly Universal Set*
Consider n fixeds to a *Hash Table*, T, with m indexes.
Pick a *Hash Function*, H, from a *Weakly Universal Set* of *Hash Functions*, H.

$$\mathbb{E}(\text{Run-Time per Operation}) = O(1) \text{ for } m \geq n$$

N.B. Proof is same as for *True Randomness*.

Proposition 1.4 - *Constructing a Weakly Universal Set of Hash Functions*
Let $S := [s]$ be the set of possible keys & p be some prime greater than s[1].
Choose some $a, b \in [0, p-1]$ & define

$$\begin{aligned} h_{a,b}(x) &= \underbrace{[\ (ax+b) \bmod p\ ]}_{\text{spread values over [0,p-1]}} \underbrace{\bmod m}_{\text{causes collisions}} \\ H_{p,m} &= \{h_{a,b}(\cdot) : a \in [1, p-1],\ b \in [0, p-1]\end{aligned}$$

N.B. $H_{p,m}$ is a *Weakly Universal Set* of *Hashing Functions*.
N.B. Different values of a & b perform differently for different data sets.

Remark 1.6 - *True Randomness vs Weakly Universal Hashing*

- For both *True Randomness* & *Weakly Universal Hashing* we have that when $m \geq n$ the expected `lookup` time in the *Hash Table* is $O(1)$.
- Constructing a *Weakly Universal Set* of *Hash Functions* is generally easier.

Theorem 1.3 - *Longest Chain - True Randomness*
If *Hashing Function* h is selected uniformly at random from all *Hashing Functions* to m indicies. Then, over m inputs we have

$$\mathbb{P}(\exists \text{ a chain length} \geq 3\log_2 m) \leq \frac{1}{m}$$

Proof 1.2 - *Theorem 1.3*
This problem is equivalent to showing that if we randomly throw m balls into m bins the probabiltiy of having a bin with at least $3\log_2 m$ balls is at most $\frac{1}{m}$.
Let X_1 be the number of valls in the first bin.
Choose any k of the M balls, the probabiltiy that all of these K balls go into the first bin is $\frac{1}{m^k}$.
By the *Union Bound Theorem* we have

$$\mathbb{P}(X_1 \geq k) \leq \binom{m}{k}\frac{1}{m^k} \leq 1k!$$

Applying the *Union Bound Theorem* again we have

$$\mathbb{P}(\text{at least 1 bin recieves at least } k \text{ balls}) \leq m\mathbb{P}(X_1 \geq k) \leq \frac{m}{k!}$$

[1] There is a theorem that $\forall\ n\ \exists\ p \in [n, 2n]$ st p is prime.

Observe that

$$\begin{aligned} k! &> 2^{k-1} \\ \text{Let } k &= 3\log_2 m \\ \implies k! &> 2^{(3\log_2 m - 1)} \\ &\geq 2^{2\log_2 m} \\ &\geq (2^{\log_2 m})^2 \\ &= m^2 \end{aligned}$$

Thus, setting $k = 3\log_2 m$ means

$$\frac{m}{k!} \leq \frac{1}{m} \text{ for } m \geq 2$$

□

Theorem 1.4 - *Longest Chain - Weakly Universal Hashing*
Let *Hashing Function* h be picked uniformly at random from a *Weakly Universal Set* of *Hashing Functions*.
Then, over m inputs

$$\mathbb{P}(\exists \text{ a chain length} \geq 1 + \sqrt{2m}) \leq \frac{1}{2}$$

N.B. This is a poor bound.

Proof 1.3 - *Theorem 1.4*
Let $x, y \in S$ be two keys and define $I_{x,y} \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$.
Let C be a random variable for the total number of collision (*i.e.* $C = \sum_{x,y \in H, \mathbf{x}<y} I_{x,y}$).
Using *Linearity of Expectation* and that $\mathbb{E}(I_{x,y}) = \frac{1}{m}$ when h is *Weakly Universal*

$$\mathbb{E}(C) = \mathbb{E}\left(\sum_{x,y \in H,\ x<y} I_{x,y}\right) = \sum_{x,y \in H,\ x<y} \mathbb{E}(I_{x,y}) = \binom{m}{2}\frac{1}{m} \leq \frac{m}{2}$$

By *Markov's Inequality*

$$\mathbb{P}(C \geq m) \leq \frac{\mathbb{E}(C)}{m} \leq \frac{1}{2}$$

Let L be a random variable for the length of the longest chain in H.
Then, $C \leq \binom{L}{2}$. Now

$$\mathbb{P}\left(\frac{(L-1)^2}{2} \geq m\right) \leq \mathbb{P}\left(\binom{L}{2} \geq m\right) \leq \mathbb{P}(C \geq m) \leq \frac{1}{2}$$

By rearranging, we have that

$$\mathbb{P}(L \geq 1 + \sqrt{2m}) \leq \frac{1}{2}$$

1.1 Perfect Hashing

Remark 1.7 - *Motivation*
The *Hashing Schemes* discussed in the previous part perform well in the best & average cases but not necessarily in the worst cases (as they can have really long longest chains).

Definition 1.5 - *Static Perfect Hashing*
A *Perfect Static Hashing Scheme* maps values to a *Hash Table* is such a way that `lookup`s can be done in constant time, even in the worst case.

N.B. FKS Hashing Scheme is a *Perfect Static Hashing Scheme.*

Definition 1.6 - *FKS Hashing Scheme*
Below is an algorithm for the *FKS Hashing Scheme*

Algorithm 1: FKS Hashing Scheme

require: $n\{\#\text{ insertions}\}, T\{\text{Table with } n \text{ entries}\}$
Insert all n into T using h
while *Collisions in* $T \geq n$ **do**
    Rebuild T using a new h.
Let $n_i = |T[i]|$.
for $i \in [1, n]$ **do**
    Insert items of $T[i]$ into new table T_i of size n_i^2 using h_i.
    while *Collisions in* $T_i \geq 1$ **do**
        Rebuild T_i using a new h_i.
return T

N.B. $\mathbb{P}(\text{Collisions in } T_i \geq 1) \leq \frac{1}{2}$ and *N.B.* $\mathbb{P}(\text{Collisions in } T \geq n) \leq \frac{1}{2}$ so we expect to have to build each table twice.

Proposition 1.5 - *FKS Hashing Scheme -* `lookup`
Below is an algorithm for `lookup(x)` in the *Hash Tables* produced by the *FKS Hashing Scheme*

Algorithm 2: FKS - `lookup(x)`

require: T {main table}, $\{T_1, \dots, T_m\}$ {sub-tables}, x {key}
Compute $i = h(x)$.
Compute $j = h_i(x)$.
return $T_i[j]$

N.B. This runs in $O(1)$ time.

Proof 1.4 - *FKS Hashing Scheme - Space Requirements*
In the *FKS Hashing Scheme* the main table T requires space $O(n)$ and each sub-table T_i requires space $O(n_i^2)$, where $n_i = |T[i]|$.
Storing each task function, h_i requires space $O(1)$.
Thus the total space used is1

$$O(n) + \sum_i O(n_i^2) = O(n) + O\left(\sum_i n_i^2\right)$$

We know there are $\binom{n_i}{2}$ collisions in $T[i]$ so there are $\sum_i \binom{n_i}{2}$ collisions in T.
We know there are at most n collisions in T so

$$\sum_i \frac{n_i^2}{4} \leq \sum_i \binom{n_i}{2} < n \implies \sum_i n_i^2 < 4n$$

Thus

$$O(n) + O\left(\sum_i n_i^2\right) = O(n)$$

Proof 1.5 - *FKS Hashing Scheme - Expected Construction Time*
The expected construction time for the main table, T, is $O(n)$.
The expected construction time for reach sub-table, T_i, is $O(n_i^2)$ where $n_i := |T[i]|$.

Thus

$$\begin{aligned}
\textit{expect}(\text{construction time}) &= \mathbb{E}\left(\text{construction time of } T + \sum_i \text{construction time of } T_i\right) \\
&= \mathbb{E}\text{construction time of } T) + \mathbb{E}\left(\sum_i \text{construction time of } T_i\right) \\
&= O(n) + \sum_i O(n_i^2) \\
&= O(n) + O\left(\sum_i n_i^2\right) \text{ see Proof 1.4} \\
&= O(n)
\end{aligned}$$

Proposition 1.6 - *FKS Hashing Scheme - Properties*

- Has no collisions.
- `lookup` takes $O(1)$ time in worst-case.
- Uses $O(n)$ space.
- Can be build in $O(n)$ expected time.

0 Reference

0.1 Probability

Definition 0.1 - *Sample Space,* Ω
A *Sample Space* is the set of possible outcomes of a scenario. A *Sample Space* is not necessarily finite.
e.g. Rolling a dice $\Omega := \{1, 2, 3, 4, 5, 6\}$.

Definition 0.2 - *Event*
An *Event* is a subset of the *Sample Space*.
The probability of an *Event*, A, happening is

$$\mathbb{P}(A) = \sum_{x \in A} \mathbb{P}(x)$$

Definition 0.3 - *Disjoint Events*
Let A_1 & A_2 be events.
A_1 & A_2 are said to be *Disjoint* if $A_1 \cap A_2 = \emptyset$.

Definition 0.4 - σ*-Field,* $\mathcal{F}$
A *Sigma Field* is the set of possible events in a given scenario.
A *Sigma Field* must fulfil the following criteria

i) $\emptyset, \Omega \in \mathcal{F}$.

ii) $\forall\ A \in \mathcal{F} \implies A^c \in \mathcal{F}$.

iii) $\forall\ \{A_1, \dots, A_n\} \subseteq \mathcal{F} \implies \bigcup_{i=1}^{n} A_i \in \mathcal{F}$.

Definition 0.5 - *Probability Measure,* $\mathbb{P}$
A *Probability Measure* maps a σ*-Field* to $[0, 1]$ which satisfies

i) $\mathbb{P}(\emptyset) = 0$ & $\mathbb{P}(S) = 1$; and,

ii) If $\{A_1, \dots, A_n\} \subseteq \mathcal{F}$ are pair-wise disjoint then $\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} \mathbb{P}(A_i)$. [$\sigma$-Additivity]

Definition 0.6 - *Random Variable*
A *Random Variable* is a function from the sample space, S, to the real numbers, $\mathbb{R}$.

$$X : S \to \mathbb{R}$$

The probability of a *Random Variable*, X, taking a specific value x is found by

$$\mathbb{P}(X = x) = \sum_{\{a \in \Omega : X(a) = x\}} \mathbb{P}(a)$$

Definition 0.7 - *Indicator Random Variable*
An *Indicator Random Variable* is a *Random Variable* which only ever takes 0 or 1 and is used to indicate whether a particular event has happened (1), or not (0).

$$\mathbb{E}(I) = \mathbb{P}(I = 1)$$

Definition 0.8 - *Expected Value,* $\mathbb{E}$
The *Expected Value* of a *Random Variable* is the mean value of said *Random Variable*

$$\mathbb{E}(X) := \sum_{x} x\mathbb{P}(X = x)$$

Theorem 0.1 - *Linearity of Expected Value*
Let $X_1, \dots, X_n$ be random variables. Then

$$\mathbb{E}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathbb{E}(X_i)$$

Theorem 0.2 - *Markov's Inequality*
Let X be a non-negative random variable. Then

$$\mathbb{P}(X \geq a) \leq \frac{1}{a}\mathbb{E}(X) \quad \forall\ a > 0$$

Theorem 0.3 - *Union Bound*
Let $A_1, \dots, A_n$ be *Events*. Then

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i)\right) \leq \sum_{i=1}^{n} \mathbb{P}(A_i)$$

N.B. This in an equality if the events are disjoint.

Proof 0.1 - *Union Bound*
Define *Indicator RV* I_i st

$$I_i := \begin{cases} 1 & A_i \text{ happened} \\ 0 & \text{otherwise} \end{cases}$$

Define *Random Variable* $X := \sum_{i=1}^{n} I_i$ (the number of events that happened).
Then

$$\begin{aligned} \mathbb{P}\left(\bigcup_{i=1}^{n} A_i)\right) &= \mathbb{P}(X > 0) \\ &\leq \mathbb{E}(X) \text{ by Markov's Inequality} \\ &= \mathbb{E}\left[\sum_{i=1}^{n} I_i\right] \\ &= \sum_{i=1}^{n} \mathbb{E}[I_i] \\ &= \sum_{i=1}^{n} \mathbb{P}(I_i = 1) \\ &= \sum_{i=1}^{n} \mathbb{P}(A_i 1) \end{aligned}$$

□