October 12 **October 2019**

Tutorial 7: Compression, Search Queries and Hashing

Course Instructor: Himanshu Tyagi *Prepared by: Karthik*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Contents

7.1 Introduction

In this tutorial, we shall study the topic of hashing and hash tables, and their relevance in the context of source compression. Specifically, we consider the following problem of searching in a database. Let $\mathcal{X}$ be a finite alphabet and let $S = \{x_1, \ldots, x_n\}$ be a known subset of $\mathcal{X}$. Suppose that the elements of S have been stored in memory using a total of m bits. Given any $y \in S$ and $\delta > 0$, it is of interest to search the contents of the memory and determine, with accuracy at least $1 - \delta$, whether or not y is present in the set S. In this context, our interest shall be towards understanding how m varies as a function of n and δ, and towards designing schemes that ensure that the complexity of search is $O(1)$ in the variable n (i.e., the complexity grows at most linearly with n).

In the following sections, we shall study various schemes for representing the symbols $x_1, \ldots, x_n$ in memory, and quantify the minimum total number of bits m needed for each of the schemes to represent the aforementioned symbols, so that the search query may be answered with high probability of correctness. While such representations of $x_1, \ldots, x_n$ give us explicit numbers for the minimum number of bits m, towards the end, we shall see a lower bound on m.

7.2 Search with Marginal Error ($\delta > 0$)

In this section, we describe two schemes for storing the information that "$x_1, \ldots, x_n \in S$" in memory, and subsequently using this information to answer the query of whether or not any given $y \in \mathcal{X}$ is in the set S with probability of correctness $1 - \delta$, where $\delta > 0$ is specified beforehand. We begin with the following simple scheme.

Scheme 1: Suppose that we order the elements of $\mathcal{X}$ in some way, and assign labels $1, 2, \ldots, |\mathcal{X}|$ to each of the elements of $\mathcal{X}$ as per this order. Then, one simple way to store the information "$x_1, \ldots, x_n \in S$" in memory is to store $|\mathcal{X}|$ bits in such a way that for each $i \in \{1, \ldots, n\}$, the location of x_i in the above ordering contains bit 1, and all other locations contain bit 0. Then, given any $y \in \mathcal{X}$, we can determine if $y \in S$ by extracting the bit stored at the location corresponding to y in the above ordering. If this bit is 1, then $y \in S$, else $y \notin S$.

We now note the following properties of Scheme 1.

1. The scheme requires us to store a total of $m = |\mathcal{X}|$ bits in memory.
2. The scheme does not make any error, and hence works for all $\delta > 0$.
3. The worst-case complexity of the scheme is $|\mathcal{X}|$ (which is $O(1)$ in the variable n). This is good in the sense that it does not depend on n, but it is a bit of an overkill.

The fact that the above scheme requires us to store $|\mathcal{X}|$ bits in memory is disheartening. We therefore look for schemes that require us to use $m < |\mathcal{X}|$ bits in memory and still answer the search query with high probability of correctness. This can be achieved by using hash functions.

7.2.1 Hash Functions and Hash Tables

We begin this subsection with the following definition of a universal hash family (UHF).

Definition 7.2.1. (Universal Hash Family) Given an integer $m' \geq 1$, a family $\mathcal{F}$ of functions from $\mathcal{X}$ to $\{1, \ldots, m'\}$ is known as an m-universal hash family (or m'-UHF) if for every $x, x' \in \mathcal{X}$, we have

$$P(F(x) = F(x')) \leq \frac{1}{m'},$$

where $F : \mathcal{X} \to \{1, \ldots, m'\}$, $F \sim \text{unif}(\mathcal{F})$, is known as a *random hash function*. The set $\{1, \ldots, m'\}$ is referred to as a *hash table*.

Example 7.2.2. We shall now see an example for a 2-UHF. We may redefine the hash table $\{1, \ldots, m'\} = \{1, 2\}$ as $\{0, 1\}$. Let $\mathcal{X} = \{0, 1\}^k$ and let $\mathcal{F}$ be the family of functions

$$\mathcal{F} = \{f_{\underline{b}} : \underline{b} \in \{0, 1\}^k\},$$

where for each $\underline{b} \in \{0, 1\}^k$, the function $f_{\underline{b}} : \mathcal{X} \to \{0, 1\}$ is defined as

$$f_{\underline{b}}(x) = \left(\sum_{i=1}^{k} b_i x_i \right) \bmod 2, \quad x \in \mathcal{X}.$$

Then, choosing $F \sim \text{unif}(\mathcal{F})$ is equivalent to choosing $\underline{b} = (b_1, \ldots, b_k)$ uniformly at random from $\{0, 1\}^k$. Therefore, for any $x \neq x'$, we have

$$\begin{aligned} P(F(x) = F(x')) &= P\left(\sum_{i=1}^{k} b_i (x_i - x_i') \bmod 2 = 0 \right) \\ &= P\left(\sum_{i=1}^{k} b_i (x_i - x_i') \text{ is an even number} \right), \end{aligned}$$

where $b_1, \ldots, b_k \overset{iid}{\sim} \text{Ber}(0.5)$. Let $a = \sum_{i=1}^{k} 1_{\{x_i \neq x_i'\}}$, i.e., a denotes the number of places in which the k-length binary vectors x and x' differ. Since $x \neq x'$, $a \geq 1$. The probability in the last line above may now be expressed as

$$P\left(\sum_{i=1}^{k} b_i (x_i - x_i') \text{ is an even number} \right) = \begin{cases} \frac{1}{2^a} \left(\binom{a}{0} + \binom{a}{2} + \cdots + \binom{a}{a} \right), & a \text{ even}, \\ \frac{1}{2^a} \left(\binom{a}{0} + \binom{a}{2} + \cdots + \binom{a}{a-1} \right), & a \text{ odd}, \end{cases}$$

where the above follows from the fact that among the random variables $b_1, \ldots, b_k$, it suffices to focus attention on the those random variables that correspond to the places in which x and x' differ, the number of such places is a, in which case there must be an even number of ones in these a places.

From the Binomial theorem, we know that

$$\begin{aligned} \binom{a}{0} + \binom{a}{2} + \cdots + \binom{a}{a} &= 2^{a-1}, \quad a \text{ even}, \\ \binom{a}{0} + \binom{a}{2} + \cdots + \binom{a}{a-1} &= 2^{a-1}, \quad a \text{ odd}, \end{aligned}$$

hence implying that $P(F(x) = F(x')) = \frac{1}{2}$ for all $x \neq x'$. Thus, $\mathcal{F}$ is a 2-UHF.

Remark 1. In the above example, note that the number of elements in $\mathcal{F}$ is 2^k, whereas the total number of functions from $\mathcal{X} = \{0, 1\}^k$ to $\{0, 1\}$ is 2^{2^k}.

7.2.2 Hash Functions and Search

We shall now see how random hash functions may be used for answering search queries efficiently.

Scheme 2: Let $\mathcal{F}$ be an m'-UHF for some integer $m' \geq 1$, and let $F \sim \text{unif}(\mathcal{F})$ be a random hash function sampled from the UHF $\mathcal{F}$. Consider now the following scheme: store m' bits in memory, with 1 at location l if $F(x_i) = l$ for some $i \in \{1, \ldots, n\}$. Here, $l \in \{1, \ldots, m'\}$. Then, given $y \in \mathcal{X}$, we may answer the query of whether or not y is in S by going to the location in memory given by $F(y)$ and reading the bit at this location. If this bit is 1, then we conclude that $y \in S$. Else, we conclude that $y \notin S$.

We now note the following features of Scheme 2.

1. The scheme requires us to store a total of $m = m'$ bits in memory, where m' is the parameter corresponding to the UHF we decide to use.
2. The scheme has $O(1)$ complexity since given any $y \in \mathcal{X}$, we only have to read the bit at the location given by $F(y)$ to answer the query of whether or not y is in S.
3. *Error probability:* We note that if $y \in S$, the scheme does not make any error. However, the scheme makes an error if $y \notin S$ but there exists $x_i \in S$ such that $F(x_i) = F(y)$. The probability of errorP_e of the scheme is thus given by

$$\begin{aligned} P_e &= P\Big(\exists\ i \in \{1, \ldots, n\} \text{ such that } F(x_i) = F(y)\Big) \\ &\overset{(a)}{\leq} \sum_{i=1}^{n} P(F(x_i) = F(y)) \\ &\overset{(b)}{\leq} \sum_{i=1}^{n} \frac{1}{m'} \\ &= \frac{n}{m'}, \end{aligned}$$

   where (a) above is due to union bound, and (b) is due to the fact that F belongs to an m'-UHF family. If the probability of error of our scheme has to be lesser than or equal to δ, we must have $m' \geq \frac{n}{\delta}$.

Thus, by choosing an appropriate $m' \geq \frac{n}{\delta}$, the probability of correctness of search can be guaranteed to be $\geq 1 - \delta$, while ensuring that the complexity of search is $O(1)$.

7.3 Search Without Error ($\delta = 0$)

In this section, we shall describe schemes for answering search queries when no error is allowed. Clearly, Scheme 1 discussed in Section 7.2 may be used in this setting since its probability of error is zero. However, since the complexity of scheme 1 is $|\mathcal{X}|$, we look for schemes that have smaller complexity (which is still $O(1)$ in the variable n).

We may consider a modification of Scheme 2 discussed earlier, where the modification is to account for error. We recall that Scheme 2 makes an error when, given any $y \in \mathcal{X} \setminus S$, there exists $x_i \in S$ such that $F(x_i) = F(y)$. Thus, we are required to find a way to resolve such collisions. Towards this, one option is to create a "linked list" as described in the following scheme.

Scheme 3: Let $\mathcal{F}$ be an m'-UHF for some integer $m' \geq 1$. Let F be sampled uniformly at random from $\mathcal{F}$. Store m' bits in memory with bit 1 at location l if $F(x_i) = l$ for some $i \in \{1, \ldots, n\}$. Here, $l \in \{1, \ldots, m'\}$. Then, for each $i \in \{1, \ldots, n\}$, perform the following operation.

- If there exists $j \neq i$ such that $F(x_i) = F(x_j)$, then store the $\lceil \log |\mathcal{X}| \rceil$-length codewords of each of x_i and x_j in the form of a linked list at location $l = F(x_i) = F(x_j)$.
- If for all $j \neq i$, $F(x_i) \neq F(x_j)$, then store only the $\lceil \log |\mathcal{X}| \rceil$-length codeword of x_i in the form of a linked list at location $l = F(x_i)$.

Given any $y \in \mathcal{X}$, we may answer the query of whether or not y is in S by first going to the location in memory given by $F(y)$ and reading the bit at this location. If this bit is 0, then $y \notin S$. However, if this bit is 1, we then look at all the codewords stored in the linked list at this location (to resolve collision, if any). By reading the individual codewords, we finally know if y corresponds to one of the codewords (in which case $y \in S$) or not (in which case $y \notin S$).

We now note the following features of Scheme 3.

1. The scheme requires us to store a total of $m = m' + n\lceil \log |\mathcal{X}| \rceil$ bits in memory.
2. The worst-case complexity of search is $n\lceil \log |\mathcal{X}| \rceil$, and this corresponds to the case when the uniformly sampled random hash function F is such that $F(x_i) = l$ for all $i \in \{1, \ldots, n\}$. In this case, the codewords of all the symbols $x_1, \ldots, x_n$ will be stored in the linked list at location l. Thus, given any $y \in \mathcal{X} \setminus S$, we may be required to read $n\lceil \log |\mathcal{X}| \rceil$ bits in the worst case to determine if y is present in S or not.
3. The scheme does not make any error since collisions are resolved by means of storing the codewords of all symbols present in the set S.

7.3.1 Towards $O(1)$ Worst-Case Search Complexity

Clearly, the only drawback of scheme 3 is that the worst-case complexity of search grows linearly with n. One possible way to overcome this and ensure that the worst-case complexity of search is $O(1)$ is to choose m' (the parameter corresponding to UHF) large enough so that for each $i \in \{1, \ldots, n\}$, x_i is mapped to a unique index l in the memory. However, since collisions are inherently present in any m'-UHF, we may only hope that the worst-case search complexity is $O(1)$ with high probability.

The following Lemma makes the above exposition concrete.

Lemma 7.3.1. *Let $S = \{x_1, \ldots, x_n\} \subset \mathcal{X}$. Fix $\eta > 0$. Then, for all $y \in \mathcal{X}$, the query of whether or not y is present in S can be answered correctly with probability 1 and with worst-case search complexity being $O(1)$ with probability $\geq 1 - \eta$ if the UHF parameter m' satisfies*

$$m' \geq \frac{1}{\eta}\binom{n}{2}.$$

Proof. As described above, in order to ensure that the worst-case search complexity is $O(1)$, it suffices to choose m' so that for each $i \in \{1, \ldots, n\}$, x_i is mapped to a unique location l in the memory. In other words, if the search complexity is not $O(1)$, then there exist $i, j \in \{1, \ldots, n\}$, $i \neq j$, such that $F(x_i) = F(x_j)$. Recall that here, $F \sim \text{unif}(\mathcal{F})$, where $\mathcal{F}$ denotes an m'-UHF.

Thus,

$$\begin{aligned} P(\text{worst-case search complexity} \neq O(1)) &\leq P(\exists\ i \neq j \text{ such that } F(x_i) = F(x_j)) \\ &\overset{(a)}{\leq} \sum_{i=1}^{n} \sum_{j \neq i} P(F(x_i) = F(x_j)) \\ &\overset{(b)}{\leq} \frac{1}{m'} \binom{n}{2} \\ &\leq \eta \end{aligned}$$

if m' satisfies the bound given in the statement of the Lemma. Here, (a) is due to the union bound, and (b) follows from the definition of m'-UHF. This completes the proof of the Lemma. □

Finally, consider the following scheme.

Scheme 4: Let $\mathcal{F}$ be an m'-UHF, where $m' = \lceil \frac{1}{\eta} \binom{n}{2} \rceil$ for some constant η that is specified beforehand. Let F be sampled uniformly at random from $\mathcal{F}$. Store m' bits in memory with bit 1 at location l if $F(x_i) = l$ for some $i \in \{1, \ldots, n\}$. Here, $l \in \{1, \ldots, m'\}$. Then, for each $i \in \{1, \ldots, n\}$, perform the following operation.

- If there exists $j \neq i$ such that $F(x_i) = F(x_j)$, then store the $\lceil \log |\mathcal{X}| \rceil$-length codewords of each of x_i and x_j in the form of a linked list at location $l = F(x_i) = F(x_j)$.
- If for all $j \neq i$, $F(x_i) \neq F(x_j)$, then store only the $\lceil \log |\mathcal{X}| \rceil$-length codeword of x_i in the form of a linked list at location $l = F(x_i)$.

Given any $y \in \mathcal{X}$, we may answer the query of whether or not y is in S by first going to the location in memory given by $F(y)$ and reading the bit at this location. If this bit is 0, then $y \notin S$. However, if this bit is 1, we then look at all the codewords stored in the linked list at this location (to resolve collision, if any). By reading the individual codewords, we finally know if y corresponds to one of the codewords (in which case $y \in S$) or not (in which case $y \notin S$).

Using Lemma 7.3.1, it follows that Scheme 4

- answers the search query with probability 1,
- has worst-case search complexity $O(1)$ with probability $\geq 1 - \eta$, and
- requires us to store a total of $m = \lceil \frac{1}{\eta} \binom{n}{2} \rceil + n \lceil \log |\mathcal{X}| \rceil = O(n^2)$ bits in memory.

7.3.2 From $O(n^2)$ to $O(n)$ Bits in Memory

As noted above, scheme 4 requires us to store a total of $m = O(n^2)$ bits in memory. Thus, we seek to design (if possible) schemes that require us to store fewer than $O(n^2)$ bits, say only $O(n)$ bits. Towards this, given an m'-UHF $\mathcal{F}$, a random hash function $F \sim \text{unif}(\mathcal{F})$, and $S = \{x_1, \ldots, x_n\}$, let

$$A_l \coloneqq \{i \in \{1, \ldots, n\} : F(x_i) = l\}, \quad N_l \coloneqq |A_l|, \quad l = 1, \ldots, m'.$$

That is, A_l denotes the set of all elements in S which are mapped to l under the random hash function F. Given $\eta > 0$, from Lemma 7.3.1, we may choose numbers $M_1, \ldots, M_{m'}$ such that

$$M_l = \left\lceil \frac{1}{\eta} \binom{N_l}{2} \right\rceil, \quad l = 1, \ldots, m',$$

and the search query over the set A_l may be answered correctly with probability 1 and with worst-case search complexity $O(1)$ with probability $\geq 1-\eta$.

We now use the above idea to construct the following scheme.

Scheme 5: Let $\mathcal{F}$ be an m'-UHF, and let $F \sim \text{unif}(\mathcal{F})$ be a random hash function. Further, for each $l \in \{1, 2, \ldots, m'\}$, let $\mathcal{F}_l$ be an M_l-UHF, and let $F_l \sim \text{unif}(\mathcal{F}_l)$ be a random hash function. Create an array Arr of size m' bits in memory. Furthermore, create m' arrays $\text{Arr}_1, \ldots, \text{Arr}_{m'}$, with Arr_l of size M_l bits. Then, for each $i \in \{1, \ldots, n\}$, perform the following operation.

- If $F(x_i) = l$, then go to location l in the array Arr and store bit 1 at this location.
- Subsequently, go to location $F_l(x_i)$ in the array Arr_l and store the $\lceil \log |\mathcal{X}| \rceil$-length codeword of x_i at this location.

Given any $y \in \mathcal{X}$, we may answer the query of whether or not y is in S by first going to the location given by $F(y)$ in the array Arr and reading the bit at this location. If this bit is 0, then $y \notin S$. However, if this bit is 1, we then go to then location given by $F_l(y)$ in the array Arr_l and read the individual $\lceil \log |\mathcal{X}| \rceil$-length codewords stored in the linked list at this location. By reading the individual codewords, we finally know if y corresponds to one of the codewords (in which case $y \in S$) or not (in which case $y \notin S$).

Clearly, with probability $\geq 1-\eta$, the worst-case search complexity of the above scheme is $O(1)$. Furthermore, the scheme answers the search query correctly with probability 1. The expected total number of bits m that we are required to store in memory is given by

$$\begin{aligned}
E[m] &= m' + n\lceil \log |\mathcal{X}| \rceil + \sum_{l=1}^{m'} E\left[\left\lceil \frac{1}{\eta}\binom{N_l}{2}\right\rceil\right] \\
&\leq m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\sum_{l=1}^{m'} E[N_l^2] \\
&= m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\sum_{l=1}^{m'} E\left[\left(\sum_{i=1}^{n} 1_{\{F(x_i)=l\}}\right)^2\right] \\
&= m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\sum_{l=1}^{m'}\sum_{i,j=1}^{n} P(F(x_i) = l,\ F(x_j) = l) \\
&= m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\sum_{i,j=1}^{n} P(F(x_i) = F(x_j)) \\
&= m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\left(n + \sum_{i\neq j} P(F(x_i) = F(x_j))\right) \\
&\leq m' + n\lceil \log |\mathcal{X}| \rceil + \frac{1}{\eta}\left(n + \frac{n(n-1)}{m'}\right), \qquad (7.1)
\end{aligned}$$

where the last line above follows by noting that F is a member of m'-UHF. Thus, if $m' \geq n-1$, then the right hand side of (7.1) becomes $O(n)$. Thus, scheme 5 requires us to store $O(n)$ bits on the average.

It turns out that $O(n)$ bits in memory is the best we can hope for. The details are present in a proof covered in class. We refer the reader to it for further reference.