Universal Function Bernoulli Approximations: The Hash Map

Table of contents

Oblivious Maps

A set is an unordered collection of distinct elements, typically from some implicitly understood universe. A countable set is a *finite set* or a *countably infinite set*. A *finite set* has a finite number of elements, such as $\{1, 3, 5\}$, and a *countably infinite set* can be put in one-to-one correspondence with the set of natural numbers, $\{1, 2, 3, 4, 5, ...\}$. The cardinality of a set $\mathbb{A}$, denoted by $|\mathbb{A}|$, is a measure on the number of elements in the set, e.g., $|\{a, b\}| = 2$.

A map represents a *many*-to-*one* relationship. A map that associates elements in $\mathbb{X}$ to elements in $\mathbb{Y}$ has a type denoted by $\mathbb{X} \mapsto \mathbb{Y}$. For a map of type $X \mapsto Y$, we denote X the *input* and Y the output. Typically, it is relatively easy to find which output is associated with a given input, but the inverse operation, determining which inputs are associated with a given output is computationally harder. Of course, this is not necessarily the case, and mathematically the map is just a one-to-many relation over $X \times Y$.

For instance, Table **??** depicts a function over a finite domain of n elements, where each input $x \in X$ is associated with a single output $y \in Y$, i.e., $y = f(x)$.

The input does not need to be a simple set like natural numbers, but rather can be any type of set, such as a set of pairs as given in Table **??**.

Table 1: Function negate : $\{0, 1\} \mapsto \{0, 1\}$

Input ($\{0, 1\}$)	Output ($\{0, 1\}$)
0	1
1	0

Table 2: Function f: $X \mapsto Y$
defined over a *finite* domain X

Input ($\{0,1\} \times \{0,1\}$)	Output ($\{0,1\}$)
$(0,0)$	0
$(0,1)$	1
$(1,0)$	1
$(1,1)$	0

Table 3: Function f: $\{0,1\} \mapsto \{0,1\}$ where $b : \{0,1\} \mapsto \{0,1\}^*$ is the encoder for $\mathbb{Y}$

Input ($\mathbb{X}$)	Output ($\mathbb{Y}$)	Encoding ($\mathcal{B}$)
x_1	y_1	b_1
x_2	y_2	b_2
x_3	y_3	b_3
$\vdots$	$\vdots$	$\vdots$
x_n	y_n	b_n

Since we are interested in constructing maps in computer memory, we must have some way to represent them. One technique may be given by the following table.

The oblivious map is given by the following definition. \begin{definition} The *oblivious Bernoulli map* is a specialization of the Bernoulli map. We denote an oblivious map of f by $f^* = (f, \mathcal{C})$ where $\mathcal{C}$ is the subset of the computational basis of f which f^* provides. An oblivious Bernoulli map satisifes the following conditions:

- The function $f^* : X \mapsto Y$ is a Bernoulli map of f.
- If an element of $x \in X$ is not in the domain of definition, $f(x)$ is a random oracle over Y
- A particular mapping $y = f^*(x)$ may only be learned by applying f^* to x.

In an *oblivious map*, a mapping (row in the table) is only learned upon request.

Observe that f^* is an oblivious value. Typically, we are also interested in functions in which the domain and codomain also represent oblivious values, i.e.,

$$f^* : X^* \mapsto Y^*$$

where $X^* = (X, \mathcal{C}_1)$, $Y^* = (Y, \mathcal{C}_2)$, and $f^* = (f, \mathcal{C}_3)$.

It may be the case that X^* is a set of oblivious integers that, say, only supports testing equality and addition. Of course, once we have addition, we may also implement multiplication, powers, and many other operations.

The space complexity of the Bernoulli map with an error rate error_rate$(\hat{f}, x)$ is given by the following theorem. ::: {.theorem} The space complexity of the Bernoulli map with an error… :::

Abstract data type

A *type* is a set and the elements of the set are called the *values* of the type. An *abstract data type* is a type and a set of operations on values of the type. For example, the *integer* abstract data type is the set of all integers and a set of standard operations (computational basis) such as addition, subtraction, and multiplication.

A *data structure* is a particular way of organizing data and may implement one or more abstract data types. An *immutable* data structure has static state; once constructed, its state does not change until it is destroyed. Let $\hat{f} : X \mapsto Y$ model the concept of a Bernoulli approximation of $f : X \mapsto Y$. Then,

- $\hat{f}(x)$

  Returns a value in Y.

- error_rate$(\hat{f}, x)$.

  Returns the *error rate* of $\hat{f}$ applied to x, i.e.,

  $$\Pr\{\hat{f}(x) = f(x)\} = \text{error\_rate}(\hat{f}, x)$$

  for every $x \in \text{dom}(f)$.

The Singular Hash Map

The **Singular Hash Set** is a theoretical data structure that provides an optimal implementation of the *oblivious map* ADT. A more practical implementation is given by the Perfect Map Filter [@pmf].

We consider sets in which the universe of elements is given by the countably infinite set of all bit strings. ::: {.definition} The countably infinite set of all bit strings is denoted by $B^* = \{0, 1\}^*$, where $*$ denotes all non-negative integers. ::: A finite subset of B^* is given by the following definition. ::: {.definition} The finite set of all bit strings of length n is denoted by

$$B^n = \left\{b : b \in \{0, 1\}^n\right\} \tag{1}$$

with a cardinality given by $|B^n| = 2^n$. :::

A hash function is related to countable sets B and B_n and is given by the following definition. ::: {.definition} A hash function hash $: B^* \mapsto B^n$ is a function such that all bit strings of