Hashing

Yufei Tao

ITEE
University of Queensland
In this lecture, we will revisit the dictionary search problem, where we want to locate an integer $v$ in a set of size $n$ or declare the absence of $v$. Recall that binary search solves the problem in $O(\log n)$ time. We will bring down the cost to $O(1)$ in expectation.

Towards the purpose, we will learn our first randomized data structure in this course. The structure is called the hash table.
The Dictionary Search Problem (Redefined)

$S$ is a set of $n$ integers. We want to preprocess $S$ into a data structure so that queries of the following form can be answered efficiently:

- Given a value $v$, a query asks whether $v \in S$.

We will measure the performance of the data structure by examining its:

- Space consumption: How many memory cells occupied.
- Query cost: Time of answering a query.
- Preprocessing cost: Time of building the data structure.
Dictionary Search—Solution Based on Binary Search

We can solve the problem by sorting $S$ into an array of length $n$, and using binary search to answer a query. This achieves:

- Space consumption: $O(n)$.
- Query cost: $O(\log n)$.
- Preprocessing cost: $O(n \log n)$.
We will improve the previous solution in expectation:

- Space consumption: $O(n)$.
- Query cost: $O(\log n) \Rightarrow O(1)$ in expectation.
- Preprocessing cost: $O(n \log n) \Rightarrow O(n)$. 
The main idea of hashing is to divide the dataset $S$ into a number $m$ of disjoint subsets such that:

- only one subset needs to be searched to answer any query.
Hash Function

Let $\mathbb{Z}$ denote the set of all integers, and $[m]$ the set of integers from 1 to $m$.

A hash function $h$ is a function from $\mathbb{Z}$ to $[m]$. Namely, given any integer $k$, $h(k)$ returns an integer in $[m]$.

The value $h(k)$ is called the hash value of $k$. 
Any hash function produces a hash table that correctly solves the dictionary search problem. However, the quality of the function has a heavy impact on the query efficiency.
Hash Table – Preprocessing

First, choose an integer \( m > 0 \), and a hash function \( h \) from \( \mathbb{Z} \) to \( \left[ m \right] \).

Then, preprocess the input \( S \) as follows:

1. Create an array \( H \) of length \( m \).
2. For each \( i \in [1, m] \), create an empty linked list \( L_i \). Keep the head and tail pointers of \( L_i \) in \( H[i] \).
3. For each integer \( x \in S \):
   - Calculate the hash value \( h(x) \).
   - Insert \( x \) into \( L_{h(x)} \).

Space consumption: \( O(n + m) \).
Preprocessing time: \( O(n + m) \).

We will always choose \( m = O(n) \), so \( O(n + m) = O(n) \).
Hash Table – Querying

We answer a query with value $v$ as follows:

1. Calculate the hash value $h(v)$.
2. Scan the whole $L_{h(v)}$. If $v$ is not found, answer “no”; otherwise, answer “yes”.

Query time: $O(|L_{h(v)}|)$, where $|L_{h(v)}|$ is the number of elements in $L_{h(v)}$. 

COMP3506/7505, Uni of Queensland  Hashing
Example

Let $S = \{34, 19, 67, 2, 81, 75, 92, 56\}$. Suppose that we choose $m = 5$, and $h(k) = 1 + (k \mod m)$.

To answer a query with search value 68, we scan all the elements in $L_3$, and answer “no”. For this hash function, the maximum query time is the cost of scanning a linked list of 3 elements.
Example

Let $S = \{34, 19, 67, 2, 81, 75, 92, 56\}$. Suppose that we choose $m = 5$, and $h(k) = 2$.

For this hash function, the maximum query time is the cost of scanning a linked list of 8 elements (i.e., the worst possible).
It is clear that a good hash function should create linked lists of roughly the same size, i.e., “spreading out” the elements of $S$ as evenly as possible.

In order to achieve $O(1)$ expected query time, we require that the hash function $h$ (from $\mathbb{Z}$ to $[m]$) should be chosen from a large family of functions to ensure the following 2-universal property:

The following holds for any two different integers $k_1, k_2$:

$$\Pr[h(k_1) = h(k_2)] \leq \frac{1}{m}$$
Next, we will first prove that 2-universality gives us the desired $O(1)$ expected query time. Then, we will describe a way to obtain such a good hash function.
We focus on the case where \( q \) does not exist in \( S \) (the case where it does is similar). Recall that our algorithm probes all the elements in the linked list \( L_{h(q)} \). The query cost is therefore \( O(|L_{h(q)}|) \).

Define random variable \( X_i \) \((i \in [1, n])\) to be 1 if the \( i \)-th element \( e \) of \( S \) has the same hash value as \( q \) (i.e., \( h(e) = h(q) \)), and 0 otherwise. Thus:

\[
|L_{h(q)}| = \sum_{i=1}^{n} X_i
\]
By 2-universality, \( \Pr[X_i = 1] \leq 1/m \), meaning that

\[
E[X_i] = 1 \cdot \Pr[X_i = 1] + 0 \cdot \Pr[X_i = 0] \leq 1/m.
\]

Hence:

\[
E[|L_{h(q)}|] = \sum_{i=1}^{n} E[X_i] \leq n/m.
\]

By choosing \( m = \Theta(n) \), we have \( n/m = \Theta(1) \).
Designing a 2-Universal Function

- Pick a prime number $p \geq m$.
- Choose a number $\alpha$ uniformly at random from $1, \ldots, p - 1$.
- Choose a number $\beta$ uniformly at random from $0, \ldots, p - 1$.
- Construct a hash function:

$$h(k) = 1 + (((\alpha k + \beta) \mod p) \mod m)$$

The proof of 2-universality is not required in this course, but will be covered in the training camp.
Now officially we have shown that, for any set $S$ of $n$ integers, it is always possible to construct a hash table with the following guarantees on the dictionary search problem:

- Space $O(n)$.
- Preprocessing time $O(n)$.
- Query time $O(1)$ \textit{in expectation}. 