More on Hashing

CSCI2100 Tutorial 7
Hou Pong CHAN, Ken
Review on Hash Table

• Given a set of integers $S$ in $[1, U]$
• Main idea: divide $S$ into a number $m$ of disjoint subsets
• Guaranteed
  • Space consumption: $O(n)$
  • Query cost: $O(1)$ in expectation
  • Preprocessing cost: $O(n)$
Review on Hash Table

• No single hash function works for all sets
• Construct a hash function from a universal family
  • Pick a prime number $p$ such that $p \geq m$ and $p \geq U$
  • Choose an integer $\alpha$ from $[1, p - 1]$ uniformly at random
  • Choose an integer $\beta$ from $[0, p - 1]$ uniformly at random
  • Define a hash function:
    $$h(k) = 1 + ((\alpha k + \beta) \mod p) \mod m$$
Example

• Let $S = \{33, 42, 70, 38, 6, 22, 17, 51, 8, 14, 63, 27\}$

• We choose $m = 10, p = 71$, suppose that $\alpha$ and $\beta$ are randomly chosen to be 3 and 7, respectively

• $h(k) = 1 + (((3k + 7) \mod 71) \mod 10)$

Adapted from Dan He’s slides
Regular Exercise 7 Problem 3

• Let $S$ be a multi-set of $n$ integers
• Frequency of an integer $x$:
  • No. of occurrences of $x$ in $S$
• Design an algorithm to produce an array that sorts the distinct integers by frequency in $O(n)$ expected time
• E.g.,
  • $S = \{23, 75, 17, 17, 23\}$
  • You should output $(75, 17, 23)$ or $(75, 23, 17)$
  • If two integers have the same frequency, their relative order is not important
Solution

• First, choose a hash function $h$ and create a hash table $H$

• For each integer $x \in S$
  • If the $H$ already contained a copy of $x$
    • Ignore $x$
  • Else
    • Compute $h(x)$
    • Insert $(x, 0)$ into the $H[h(x)]$

• The checking in each iteration takes $O(1)$ in expectation

• Overall: $O(n)$ in expectation
Solution

• Second, obtain the frequency of every distinct integers

• For each integer \( x \in S \)
  • Find its copy in \( H \)
  • Increase the counter of the copy by one
  • Takes \( O(1) \) expected time

• This part takes \( O(n) \) in expectation
Solution

• Finally, sort all the distinct integers by frequency
• Since the frequency of every integer in $S$ is in the domain $[1, n]$
• Use counting sort to sort the integers by frequency (see tutorial 5), takes $O(n)$ time
• E.g., we get $[(75,1), (23,2), (17,2)]$
• Generate output from these sorted tuples, takes $O(n)$ time
• E.g., $[75, 23, 17]$
Time Complexity

- Overall complexity: $O(n)$ in expectation

\[
\{23, 75, 17, 17, 23\} \rightarrow H \rightarrow \{75, 23, 17\} \rightarrow H \\
\{75, 23, 17\} \rightarrow H \rightarrow [(75, 1), (23, 2), (17, 2)]
\]
Hash Table

- Expected query cost: $O(1)$
  - Pick a hash function from a universal family
- Worst-case query cost: $O(n)$
  - All elements are hashed into the same value

- Can we improve the worst-case query cost?
Hash Table

- Replace linked lists with arrays
- Sort the arrays, cost $O(n \log n)$ for preprocessing
Hash Table

- Query: whether 29 exists
- Step 1:
  - Access the hash table to obtain the address of corresponding array
  - Takes $O(1)$ time in expectation
Hash Table

• Query: whether 29 exists
• Step 2:
  • Perform binary search on the array to find the target
  • Takes $O(\log n)$ time
• Overall worst-case complexity: $O(\log n)$
Hash Table

• This method retains the $O(1)$ expected query time

• Proof:
  • Suppose we look up an integer $q$
  • Define random variable $L_{h(q)}$ to be the length of array that corresponds to the hash value $h(q)$
  • Expected query time:
    • $E[\log_2 L_{h(q)}] = \sum_{l=1}^{n} \log_2 l \Pr(L_{h(q)} = l)$
    • $\leq \sum_{l=1}^{n} l \Pr(L_{h(q)} = l)$
    • $= E[L_{h(q)}]$
    • $= O(1)$
Hash Table

- Can we keep the original hash table, while still achieve the $O(1)$ expected query cost and $O(\log n)$ worst-case query cost?
- Yes, but the space consumption will be doubled
- Construct a new array to store the integers in $S$
- Sort the array using counting sort
Hash Table

• Perform linear search on the linked list, stop when it finds the target or reaches $\log_2 n$
Hash Table

- Expected cost: $O(1)$
- Worst-case cost: $O(\log n)$
Hash Table

• If we cannot find the target in linear search
• Then we perform binary search on the array
• Worst-case cost: $O(\log n)$

| 2 | 6 | 9 | 10 | 14 | 18 | 24 | 26 | 28 | 29 |

• Overall worst case complexity:
  • $O(\log n) + O(\log n) = O(\log n)$