Review on Hash Table

• $S$ = a set of $n$ integers in $[1, U]$

• Main idea: divide $S$ into a number $m$ of disjoint “buckets”

• Guarantees
  • Space consumption: $O(n + m)$
  • Preprocessing cost: $O(n + m)$
  • Query cost: $O(1 + n/m)$ in expectation
Review on Hash Table

- $S = \text{a set of } n \text{ integers in } [1, U]$
- Main idea: divide $S$ into a number $m$ of disjoint “buckets”
  - Set $m = \Theta(n)$
- Guarantees
  - Space consumption: $O(n)$
  - Preprocessing cost: $O(n)$
  - Query cost: $O(1)$ in expectation
Review on Hash Table

• Divide $S$ into a number $m$ of disjoint buckets:
  • Choose a function $h$ from $[1, U]$ to $[1, m]$
  • For each $i \in [1, m]$, create an empty linked list $L_i$
  • For each $x \in S$:
    • Compute $h(x)$
    • Insert $x$ into $L_{h(x)}$

• Important: choose a good hash function $h$
Review on Hash Table

• Construct 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 \( S = \{19, 36, 63, 53, 14, 9, 70, 26\} \)
- We choose \( m = 10, p = 71 \), suppose that \( \alpha \) and \( \beta \) are randomly chosen to be 3 and 7, respectively
- \( h(k) = 1 + ((3k + 7) \mod 71) \mod 10 \)
Relationships between Hash Functions and Queries

• Let $H$ be the universal family defined in the previous slides
• Given a function $h \in H$ and an integer $q \in [1, U]$:
  • Define $\text{cost}(h, q) = |\{x \in S \mid h(x) = h(q)\}|$

<table>
<thead>
<tr>
<th>$h_i$</th>
<th>$1$</th>
<th>$2$</th>
<th>...</th>
<th>$U$</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>$\text{cost}(h_1, 1)$</td>
<td>$\text{cost}(h_1, 2)$</td>
<td>...</td>
<td>$\text{cost}(h_1, U)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$h_2$</td>
<td>$\text{cost}(h_2, 1)$</td>
<td>$\text{cost}(h_2, 2)$</td>
<td>...</td>
<td>$\text{cost}(h_2, U)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$h_{</td>
<td>H</td>
<td>}$</td>
<td>$\text{cost}(h_{</td>
<td>H</td>
<td>}, 1)$</td>
</tr>
<tr>
<td>Average</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Hash Table

• Worst-case expected query cost: $O(1)$
• Worst-case query cost: $O(n)$

• Question:
  • Can we improve the worst-case query cost?
Hash Table: Improving the Worst Cost

- Replace linked lists with sorted arrays
- $O(n \log n)$ preprocessing cost
Hash Table: Improving the Worst Cost

• Query: whether 29 exists

• Step 1:
  • Access the hash table to obtain the address of corresponding array
  • $O(1)$ time
Hash Table: Improving the Worst Cost

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

• This method retains the $O(1)$ worst-case expected query time.

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

• Problem Input:
  • An array $A$ of $n$ distinct integers (not necessarily sorted).

• Goal:
  • Determine whether if there exist two different integers $x$ and $y$ in $A$ satisfying $x + y = v$

• Example: find a pair whose sum is 20
Solution 1: Binary Search the Answer

• Goal: Find a pair \((x, y)\) such that \(x + y = v\)
• Observe that given \(x\), \(y = v - x\), is determined
• Solution:
  • Sort \(A\)
  • For each \(x\) in \(A\):
    • set \(y\) as \(v - x\)
    • Use binary search to see if \(y\) exists in the sequence
• Time complexity: \(O(n \log n)\)
Solution 2: Using the Hash Table

• Step 1 and 2:
  • Choose a hash function $h$ and create an empty hash table $H$
  • Insert each $x$ in $A$ into $L_{h(x)}$

• Step 3:
  • For $i = 1$ to $n$
    • Set $y$ as $v - A[i]$
    • Check if $y$ is in the hash table; if it is, return yes
  • Return no
Time Complexity

• Step 1 and 2: $O(n)$

• Step 3:
  • The step issues $n$ queries (one for each $y$)
  • Let $X_i$ be the time of the $i$-th query
  • We know $E[X_i] = O(1)$
  • The worst-case expected cost of step 3 is $\sum_i E[X_i] = O(n)$

• Overall: $O(n)$ in expectation
Bonus: Sorting by Frequency (a Regular Exercise)

• Problem input:
  • Let $S$ be a multi-set of $n$ integers. The frequency of an integer $x$ as the number of occurrences of $x$ in $S$.

• Goal: Produce an array that sorts the distinct integers in $S$ by frequency.

input: 10 8 8 12 9 9 12 12

output: 12 8 9 10

12 : 3 occurrences
8   : 2 occurrences
9   : 2 occurrences
10  : 1 occurrence
Using a Hash Table to Obtain Frequencies

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]

\[H\]

NIL
NIL
NIL
Using a Hash Table to Obtain Frequencies

\[ H \]

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]
Using a Hash Table to Obtain Frequencies

\[ H \]

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]
Using a Hash Table to Obtain Frequencies

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]

\[
H
\]

\[
\begin{array}{c}
(8, 2) \quad \text{NIL} \\
(10, 1) \quad \text{NIL} \\
\text{NIL} \\
\text{NIL}
\end{array}
\]
Using a Hash Table to Obtain Frequencies

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]

\[
H
\]

(8,2) \arrow{NIL} \\
(10,1) \arrow{NIL} \\
(12,1) \arrow{NIL}

Using a Hash Table to Obtain Frequencies

• The final state:

\[
\begin{array}{cccccccc}
10 & 8 & 8 & 12 & 9 & 9 & 12 & 12 \\
\end{array}
\]

\[
H
\]

- \((8,2)\) -> NIL
- \((10,1)\) -> \((9,2)\) -> NIL
- \((12,3)\) -> NIL
Counting Sort!

- Now we sort the numbers by frequency.
- Key observation: each frequency is in $[1, n]$.
- We can carry out the sorting with counting sort in $O(n)$ time.

Total time complexity: $O(n)$ expected time.