HASHES

In our study on elementary data structures, we have assumed that:
   a) We have a universe U.
   b) We want to store set S.
   c) Insert, Delete and Member operations should all be supported.

In the previous classes we have studied details of arrays, which have a time requirement of $O(1)$ and a
memory requirement of $O(|U|)$. We have also dealt with linked lists which have a time requirement of $O(|S|)$ and a memory requirement of $O(|S|)$. This brings us to the possibility of a way to achieve a time
requirement of $O(1)$ with a memory requirement of $O(|S|)$. And the answer is yes.
The answer definitely is yes, but $O(1)$ will be expected time and with some overhead on memory.
Consider the situation presented below.

Each element ‘x’ in the universe U will be mapped into a particular slot in the array with a function $h(x)=i$.
However, there is a possibility of a ‘collision’, where an element ‘y’ also wants to be mapped to i. Our
goal is hence, to provide good collision resolution.

There are 2 types of hashing:
   Hashing with chaining.
   Hashing with open addressing.
HASHING WITH CHAINING

In the best case scenario, we have all the values dispersed enough, whereas in the worst case, we have all values mapped to i.

Next, define a function \( h: U \rightarrow \{1, 2, 3, 4 \ldots n\} \) which is a good hashing function if and only if for randomly picked ‘x’ and for every i, we have \( P(h(x)) = \frac{1}{m} \).

We also define that a hash table with ‘m’ buckets and ‘n’ elements has a load factor of \( \frac{n}{m} \). Load factor (\( \alpha \)) is simply the average number of elements per bucket.

**THEOREM 1**

Let ‘h’ be a good hashing function. Also, let \( P(x \text{ is an operand for } k^{th} \text{ operation}) = \frac{1}{|U|} \) [for every k].

Then, the expected cost of a member operation in step \( n \) is bounded by \( 1 + \frac{n-1}{m} \).

In the worst case, the first \( n-1 \) operations are all inserts. In that case, we have \( \alpha = \frac{n-1}{m} \).

**Proof:**
- Computing \( h(x) \) is not expensive. It takes constant time \( \Rightarrow O(1) \).
- Expected cost is proportional to the expected length of the list in the bucket.
- Expected cost = Expected length of the bucket + 1 \{to calculate \( h(x) \)\}. 

...
E[X] = \sum x \cdot P(x)

Cost \leq \sum_{j=0}^{n-1} P(\text{list has length } j) \cdot (1 + 1)

We know that \( P(\text{list has length } j) = \binom{n-1}{j} \left( \frac{1}{m} \right)^j \left( 1 - \frac{1}{m} \right)^{n-1-j} \).

Substituting the above equation in the cost equation, we get

\[
\text{Cost} \leq \sum_{j=0}^{n-1} \binom{n-1}{j} \left( \frac{1}{m} \right)^j \left( 1 - \frac{1}{m} \right)^{n-1-j} (j + 1).
\]

\[
= 1 + \sum_{j=1}^{n-1} j \binom{n-2}{j-1} \left( \frac{1}{m} \right)^j \left( 1 - \frac{1}{m} \right)^{n-2-j}.
\]

We assume \( j-1 = k \). This means when \( j=1 \), \( k=0 \) and when \( j=n-1 \), \( k=n-2 \). Substituting these values in the previous equation, we get,

\[
\text{Cost of operation} = 1 + \frac{n-1}{m} \sum_{k=0}^{n-2} \binom{n-2}{k} \left( \frac{1}{m} \right)^k \left( 1 - \frac{1}{m} \right)^{n-2-k}.
\]

\[
= 1 + \frac{n-1}{m} \sum_{k=0}^{n-2} \binom{n-2}{k} \left( \frac{1}{m} \right)^k \left( 1 - \frac{1}{m} \right)^{n-2-k}.
\]

\[
= 1 + \frac{n-1}{m}.
\]

Hence, cost of operation \( \rightarrow O\left(1+\alpha\right) \). Thus the theorem is proved.

**THEOREM 2**

The expected cost of \( n \) inserts, deletes and/or members is bounded by \( n \left( 1 + \frac{\alpha}{2} \right) \).

**Proof:**

Cost \( \leq \sum_{i=1}^{n} \left( 1 + \frac{i-1}{m} \right) \).

\[
= n + \frac{1}{m} \sum_{i=1}^{n} (i - 1).
\]

Substituting \( i-1 \) as \( k \), when \( i=1 \), \( k=0 \) and when \( i=n \), \( k=n-1 \).
\[
\begin{align*}
&= n + \frac{1}{m} \sum_{k=0}^{m} \frac{1}{k} \\
&= n + \frac{1}{m} \left( \frac{n(n-1)}{2} \right) \\
&= n \left( 1 + \frac{n-1}{2m} \right) \\
&= n \left( 1 + \frac{\alpha}{2} \right)
\end{align*}
\]

Hence the cost is \( \leq n \left( 1 + \frac{\alpha}{2} \right) \). The theorem is proved.