Hash Tables

Ref: Chapter 11 of text.

Dynamic set: A set which can be modified by INSERT and DELETE operations.

Dictionary:

- A data structure that supports INSERT, DELETE and SEARCH operations.
- Can be used to implement a dynamic set.

Application: Symbol tables in assemblers and compilers.

Hash table:

- One way of implementing a dictionary.
- Each operation takes $O(1)$ expected time.
- SEARCH may take $\Theta(n)$ time in the worst-case.
- Works very well in practice.

Direct Address Table (DAT):

- A hash table that is practical only when the universe of keys is small.
- No two records have the same key.
- Array $T$ of pointers.
  - $T[i]$ points to a record with key $i$.
  - $T[i]$ is NULL if no record with key $i$ exists.


Notes:

- DAT supports each dictionary operation using $O(1)$ worst-case time.
- Required DAT size is very large in practice.

Example: Consider a programming language that allows identifiers of length from 1 to 10. Each identifier starts with a letter and may be followed by additional letters or digits. (Ignore case differences.) How many different identifiers of length 10 are possible?

Answer: $26 \times (36)^9 > 38 \times 10^{13}$. 
**Hash Tables:**

- Choose a suitable integer $m$ and use a table of size $m$. (Table indices vary from 0 to $m - 1$.)
- Map each key $k$ using a **hash function** $h$ whose range is $\{0, 1, 2, \ldots, m - 1\}$.
- $h$ should be computable fast; that is, in $O(1)$ time.

**Example:** Let $m = 23$. Suppose each key is a string of 3 alphabetic characters. Define

$$h(k) = \left( \Sigma_{i=1}^{3} p_{i} \right) \mod 23$$

where $p_{i}$ is the position in the alphabet of the $i^{th}$ letter of the key. Thus,

- $h(\text{art}) = (1 + 18 + 20) \mod 23 = 16$
- $h(\text{box}) = (2 + 15 + 24) \mod 23 = 18$

Note that $h(\text{rat})$ is also 16.

**Collision:**

- The universe of keys is much larger than the size of the hash table.
- So, $h$ cannot be a one-to-one function in general.
- Collision occurs when two keys hash to the same value.

**Hashing with Chaining:**

- One way of resolving collisions.
- The keys that hash to the same location are kept in a linked list.
- Worst-case times:
  - **Insert:** $O(1)$.
  - **Delete:** $O(n)$ with a singly-linked list; $O(1)$ with a doubly-linked list.
  - **Search:** $O(n)$. 
Average-case analysis:

- Uses the quantity load factor (denoted by $\alpha$) defined by
  \[ \alpha = \frac{\text{No. of keys stored} (n)}{\text{No. of slots} (m)}. \]

- **Simple Uniform Hashing assumption**: Any key is equally likely to hash into any of the $m$ slots, independent of other keys.
- So, $\alpha$ is the expected size of a list.
- Analysis is different for unsuccessful and successful searches.

Analysis of unsuccessful search:

**Theorem 1**: Under hashing with chaining, the expected time for an unsuccessful search is $O(1 + \alpha)$.

**Proof**: To be discussed in class.

Analysis of successful search: Needs the following additional assumption.

- The key being searched is equally likely to be any of the $n$ keys inserted into the table.

**Theorem 2**: Under hashing with chaining, the expected time for a successful search is $O(1 + \alpha)$.

**Proof**: To be discussed in class.

Significance of the results:

- If $\alpha = O(1)$, the expected search time using the chaining scheme is $O(1)$.
- When the load factor is chosen to be a constant, hashing with chaining supports each of dictionary operations in $O(1)$ expected time.
Hash functions:

- A “good” hash function should distribute the keys in a “uniform” way.
- Difficult to achieve exactly; heuristic methods used in practice.
- Assumption: Each key is a non-negative integer.

(a) Division Method: Choose the number of slots $m$. The hash function $h$ is given by

$$h(k) = k \mod m$$

Bad choice for $m$: $m = 2^r$.

Reason: The value of $h(k)$ will be the least significant $r$ bits of key $k$ (i.e., hash function won’t depend on the whole key).

Common choice for $m$: A prime number not too close to a power of 2. (This choice is supported by extensive empirical evidence.)

Example: Suppose we want a symbol table with 500 symbols and we can allow the load factor to be 4. At least $500/4 = 125$ slots are needed. Since 125 is too close to $2^7$, a reasonable choice is 139.

Practice: Experiment with several different values of $m$ and choose one that gives the best empirical performance.

(b) Multiplication Method:

- Uses two parameters: $m$ (the number of slots) and a fraction $A$ ($0 < A < 1$).
- Hash function $h(k)$ is given by

$$h(k) = \lfloor m (kA \mod 1) \rfloor$$

where “$kA \mod 1$” is the fractional part of $kA$.

Example: Let $m = 128$, $A = 0.37$, $k = 105$.

Here, $kA = 38.85$ so that $(kA \mod 1) = 0.85$.

Thus, $h(105) = \lfloor 128 \times 0.85 \rfloor = \lfloor 108.8 \rfloor = 108.$
Notes regarding multiplication method:

- The value of \( m \) is not as critical as in the division method. (The value of \( m \) may be a power of 2.)
- Empirical evidence suggests that \( A = (\sqrt{5} - 1)/2 \approx 0.61803 \) works well in practice.

Open Addressing:

- Another way of resolving collisions.
- Each table entry either contains a key or is empty.
- No linked lists are used.
- No. of keys stored \( \leq \) No. of slots in the table (i.e., the load factor \( \alpha \) is at most 1).
- Not a good alternative if \texttt{DELETE} operation must be supported.

Main idea: Compute a “sequence” of slots to be examined (“probes”).

Notation: Hash function \( h(k, i) \) uses two arguments; \( k \) is the key and \( i \) is the probe number.

Pesudocode: Handout 10.2.

Notes regarding deletion:

- When a key is deleted, the table entry cannot be set to NULL. (If this is done, some searches won’t work correctly.)
- Remedy: Use a special marking (Deleted). Let the search procedure distinguish between NULL and Deleted.
- Disadvantage: The search time cannot be estimated in terms of the load factor \( \alpha \); the Deleted entries lead to a false value of \( \alpha \).

Generating probe sequences:

(a) Linear probing:

- The hash function \( h(k, i) \) is given by
  \[
  h(k, i) = (h'(k) + i) \mod m
  \]
  for \( i = 0, 1, \ldots, m - 1; h'(k) \) is a usual hash function.

- Sequence of probes: \( h'(k) \mod m, [h'(k) + 1] \mod m, [h'(k) + 2] \mod m, \) etc.
The value of $h'(k)$ (i.e., the initial slot number) determines the entire probe sequence.

- Easy to implement.
- Causes primary clustering: long runs of occupied slots build up, thus increasing time for unsuccessful search.

(b) Quadratic probing:
- The hash function $h(k, i)$ is given by
  $$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m$$
  for $i = 0, 1, \ldots, m - 1$; $h'(k)$ is a usual hash function; $c_1$ and $c_2 \neq 0$ are constants.
- The value of $h'(k)$ (i.e., the initial slot number) determines the entire probe sequence.
- Generally better than linear probing.
- Causes secondary clustering; less severe than primary clustering.

(c) Double hashing:
- The hash function $h(k, i)$ is given by
  $$h(k, i) = (h_1(k) + i h_2(k)) \mod m$$
  for $i = 0, 1, \ldots, m - 1$; $h_1(k)$ and $h_2(k)$ are usual hash functions.
- The initial slot is $h_1(k)$. Subsequent probes depend on $h_2(k)$.
- Best method in practice.

Important condition: For double hashing to be effective, $m$ and $h_2(k)$ must be relatively prime.

Reason: Otherwise the “$i h_2(k)$” part of the hash function won’t play a role. (Details in class.)

How to satisfy the condition:
1. Choose $m = 2^r$ and let $h_2(k)$ be an odd integer for all $k$.
2. Choose $m$ to be a prime and choose $h_2$ such that $1 \leq h_2(k) < m$ for all $k$. 
Average-case analysis:
Assumptions:
- Uniform Hashing: The probe sequence $\langle h(k, 0), h(k, 1), \ldots, h(k, m - 1) \rangle$ for any key $k$ is equally likely to be any of the $m!$ permutations on $\langle 0, 1, \ldots, m - 1 \rangle$.
- The load factor $\alpha$ is less than 1.
- Each key is equally likely to be searched for.

Theorem 3: Under open addressing, the expected number of probes in an unsuccessful search is at most $1/(1 - \alpha)$.
Proof: To be discussed in class.

Theorem 4: Under open addressing, the expected number of probes to insert a key is at most $1/(1 - \alpha)$.
Proof: Direct consequence of Theorem 3.

Theorem 5: Under open addressing, the expected number of probes in a successful search is at most
\[\frac{1}{\alpha} \ln \frac{1}{1 - \alpha}\]
Proof: To be discussed in class.