Hash Functions for Data Items with Many Bits

Idea:

• Since any data item can be represented as a sequence of bits, it can be viewed as a non-negative integer, and the standard MOD hash function can be used.

• It is convenient to work with integers that fit into a single machine word.

• To convert a data item into an index into the hash table:
  1. Choose a large prime $q$ that can be represented and manipulated with the word size on the machine you are using.
  2. Use arithmetic operations to combine pieces of the data item into a single large number, doing a MOD $q$ as necessary to prevent arithmetic overflow.
  3. MOD the result by $m$ to get a hash table index between 0 and $m−1$. 
Modeling data items as strings:

- For some $k \geq 1$, to partition the bits of each data item into $k$ blocks of $b \geq 1$ bits (if a data item has fewer than $kb$ bits, pad to the left with 0's).
- For example, $b=8$ corresponds to partitioning into bytes.
- Thus, all data items can be viewed as strings of $k$ characters over the alphabet 0 to $b-1$.
- We assume that both $b$ and $k$ are constants $\leq m$. 
Example, the polynomial hash function:

For an integer $p$, compute the polynomial $h(s) = s[k-1]p^{k-1} + \cdots + s[1]p + s[0]$. As $h$ is computed, do operations MOD $q$ to prevent arithmetic overflow:

```
function h(s)
    z := 0
    for $i := k-1$ downto 0 do $z := (pz+s[i]) \text{ MOD } q$
    $z := z \text{ MOD } m$
    return $z$
end
```

The value of $p$ does not have to be prime. Choosing $p = b$ corresponds to shifting bits by blocks. For example, if each block was an ASCII character encoded by 7 bits, choosing $p = 128$ corresponds to shifting by one character.

Note: There is no point in choosing $p$ larger than $b$, but $p$ should be large enough so that for the number of blocks being hashed, values get larger than $m$. In any case, chose $p$ and $q$ so integer overflow cannot occur; that is, values of size on the order of $pq$ must fit into the variables being used.
Example, the weighted hash function:

- Choose a large prime $q$.
- Choose a sequence of weights $W[0]...W[k-1]$, each a prime.
- To hash a string $s = s[0]...s[k-1]$, add up the characters of $s$ times the corresponding weight, and MOD the result by $m$ to get a valid index into the hash table.

\[ function \ h(s) \]
\[ \quad z := 0 \]
\[ \quad for \ i := 0 \ to \ k-1 \ do \ z := (z+s[i]W[i]) \ MOD \ q \]
\[ \quad z := z \ MOD \ m \]
\[ \quad return \ z \]
\[ end \]

Note: Again, one must be careful to choose values for $q$ and the weights that cannot cause an integer overflow.
Note:

For data items represented by long strings, computation of the hash function can dominate the total running time.

For some applications a trie data structure (trees that represent a set of strings by labels on root-to-leaf paths) can be a practical alternative.