

Report on Offline on HashTable

Prepared By:

Prithu Anan

ID: 2005045

Custom Probe Constants: $c1 = 37$, $c2 = 41$

Seeds Used for Random Functions: seed1 = 2005, seed2 = 045

Results:

Hash Table Size	Collision Resolution Method	Hash Function 1		Hash Function 2	
		Colls.	Probes	Colls.	Probes
N = 5000	Separate Chaining	4171	1.172	4231	1.127
	Double Hashing	6399	1.163	6497	1.127
	Custom Probing	6466	1.159	6468	1.119
N = 10000	Separate Chaining	3724	1.200	3768	1.145
	Double Hashing	6282	1.166	6293	1.128
	Custom Probing	6296	1.158	6279	1.118
N = 20000	Separate Chaining	2082	1.021	2107	1.022
	Double Hashing	3740	1.377	3840	1.357
	Custom Probing	3762	1.389	3762	1.373

Hash Function 1:

```
// Use folding on a string, summed 4 bytes at a time
int sfold(char* key) {
    unsigned int *lkey = (unsigned int *)key;
    int intlength = strlen(key)/4;
    unsigned int sum = 0;
    for(int i=0; i<intlength; i++)
        sum += lkey[i];

    // Now deal with the extra chars at the end
    int extra = strlen(key) - intlength*4;
    char temp[4];
    lkey = (unsigned int *)temp;
    lkey[0] = 0;
    for(int i=0; i<extra; i++)
        temp[i] = key[intlength*4+i];
    sum += lkey[0];

    return sum % M;
}
```

This function takes a string as input. It processes the string four bytes at a time, and interprets each of the four-byte chunks as a single (unsigned) long integer value. The integer values for the four-byte chunks are added together. In the end, the resulting sum is converted to the range 0 to $M - 1$ using the modulus operator.²

For example, if the string “aaaabbbb” is passed to **sfold**, then the first four bytes (“aaaa”) will be interpreted as the integer value 1,633,771,873 and the next four bytes (“bbbb”) will be interpreted as the integer value 1,650,614,882. Their sum is 3,284,386,755 (when viewed as an unsigned integer). If the table size is 101 then the modulus function will cause this key to hash to slot 75 in the table. Note that for any sufficiently long string, the sum for the integer quantities will typically cause a 32-bit integer to overflow (thus losing some of the high-order bits) because the resulting values are so large. But this causes no problems when the goal is to compute a hash function.

Hash Function 2:

■ Approach 3

- Use all N characters of string as an N-digit base-K integer
- Choose K to be prime number larger than number of different digits (characters)
 - I.e., K = 29, 31, 37
- If L = length of string S, then

$$h(S) = \left[\sum_{i=0}^{L-1} S[L-i-1] * 37^i \right] \bmod TableSize$$

- Use Horner's rule to compute h(S)
- Limit L for long strings

```
1  /**
2   * A hash routine for string objects.
3   */
4  int hash( const string & key, int tableSize )
5  {
6      int hashVal = 0;
7
8      for( int i = 0; i < key.length( ); i++ )
9          hashVal = 37 * hashVal + key[ i ];
10
11     hashVal %= tableSize;
12     if( hashVal < 0 )
13         hashVal += tableSize;
14
15     return hashVal;
16 }
```

Auxiliary Hash Function:

```
int h(char* x) {
    int i, sum;
    for (sum=0, i=0; x[i] != '\0'; i++)
        sum += (int) x[i];
    return sum % M;
}
```

This function sums the ASCII values of the letters in a string.