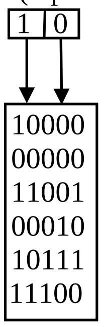
ECS 60 **Written Homework #4: Hash Tables (16 points)** **Fall 2015**

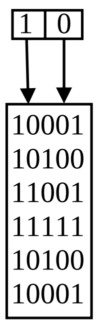
Due Wednesday, October 28th, 4:00 pm in 2131 Kemper

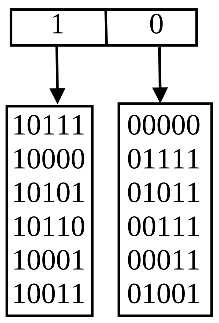
1. (6 points, 2 points each) Show the contents of the final open addressing hash table with an initial size of 5. Each table should rehash when its load factor is about to go over 0.5 because of an insert. Use h_1 = key % table_size as the hash function for all the parts. Please consider each hash table to be separate, and independent of the other two.
   a. Linear probing: Insert 13, insert 2, insert 24, delete 13, insert 4, insert 35
   b. Quadratic probing: Insert 4, insert 12, delete 4, insert 1, insert 10, insert 5, insert 23
   c. Double hashing with h_2 = (key % 4) + 1: Insert 3, insert 58, insert 15, insert 36, insert 25

2. (2 points) Draw a separate chaining hash table of size 7 (with its linked lists) with h_1 = key % 7 for the following number being inserted. Numbers are inserted at the front of their respective lists. 11, 25, 6, 21, 9, 13, 7, 20, 4, 18.

3. (6 points, 2 points each) Insert 10010 into each of the following extendible hashes with M = 6.

a.


b.


c.


4. (2 points) Suppose we want to find the first occurrence of a string $P_1P_2...P_k$ in a long input string $A_1A_2...A_N$. We can solve this problem by hashing the pattern string, obtaining a hash value H_p, and comparing this value with the hash value formed from $A_1A_2...A_k$, $A_2A_3...A_{k+1}$, ..., A_{N-k+1}, $A_{N-k+2}...A_N$. If we have a match of hash values, we compare the strings character by character to verify the match. We return the position (in A) if the strings actually do match, and we continue in the unlikely event that the match is false. Assume we adapt the hash table function implemented for strings that is shown below. Show that if the hash value of $A_0A_1...A_{k-1}$ is known, then the total time for computing ALL of the hash values of $A_iA_{i+2}...A_{i+k}$ for $0 < i < N - k - 1$ can be O(N + k). Note that is not O(N * k). Provide your nextHash() function as well as your proof. Hint: the nextHash() function may have additional parameter(s). Adapted from our text 5.15a.

```
unsigned hash( const char *key, int tableSize )
{
  unsigned hashVal = 0;

  for( int i = 0; key[i]; i++ )
    hashVal = 37 * hashVal + key[ i ];

  hashVal %= tableSize;
  return hashVal;
}

unsigned nextHash(const char *A, unsigned lastHashVal, int k, int position …)
```