03_String Matching Algos

HOORSPOOLS:

Problem: String / Pattern matching / string search

- Pattern: word or string we’re searching for
- Text: Medium we’re searching through

```
// Size of text -> n
// Size of pattern -> m
//          0 1 2
// pattern: d o g
//       0.1 2 3 4 5 6 7 8 9 10 11 12 13
// text: F V n d o g t x q z 5  1  m  n
// Brute Force Algorithm: check every index,
// or skip the next letters if there's a mismatch.

// come up with a big O for the runtime of the above method:
// T(n,m) = O(M*(N-M + 1) - number of slides): maybe
// aka O(n*m)

// Possible homework:
// Horspool's Algorithm - a varient of the Boyer-More algorithm
// another popular algorithm is known as KMP - Knuth Morris Pratt
// Use the ascii table - 128 charactars
// 97: a
// 97: b...
// 48: 0
// 49: 1...
//
// 1. Make an array that can hold the ascii table
// 2. Initialize array so that everything inside is a 6. //whatever m is.
// 3. Assign letters in the pattern a value in the ascii table. The last gets m, and the ones
//    before that start at the value 1 for the slot before last, and the numbers increment at
//    each decrement of the index.
// 4. Start marker starts at (m-1)
// 5. if a mismatch occurs, slide pattern over x slots, where x is the value in
//     the table of the charactar in the text at your start marker.
//    -Slide over the number of slots that the corresponding letter is assigned in the ascii table.
//
// char pattern[]{"abcdef"};
//        Assign: "543216" //order is important. Large number on right, and count up
//                           while decrementing.
// for each letter in the array, assign the number to the corresponding
// index in the ascii array..
//  So 'a' = 5, 'b' = 4, 'c' = 3, d = '2' etc.
// int numInput = 28
// int m = 6; //size of pattern
//                 V            V           V         V
//        a b c d e f
// Text: "a u v e z x a b c u p q r r a d e f 1 2 3 4 a b c c e f"
// slide pattern to the end match. If there's a mismatch, skip 6. If there's a match, check
// backwards until you've found it, or until there's no match. Slide by the value of the corresponding
// number in the ascii table at table[marker]

// Example: 5 4 3 2 1 6
// Pattern: a b c b c f -- m = 6
//          x r q l a b
// f gets 6, a gets 5 , b gets 4, c gets 3, b gets overwritten with 2,
// and c gets overwritten with 1
//.                V
// text: a b c b x k m n b c f k m a b c b c f e a b c b c y b c a b a b a f c b c f e r
 3  2  1  4
[a][a][b][b]
pattern should have NOT an a in position 2
```

string search:

Brute force: O(n*m)

Horspools worst case O(n*m)

minimum number of comparisons = n/m, there are m average comparisons to jump.

horspools average case is O(n)

HOMEWORK: implement horspools algorithm with 128 char ascii table

output: starting index in the text of the FIRST occurrence of the pattern in the text., else return -1.

secondary output: total number of comparisons.

note: do 0 comparisons once the pattern is past the end of the text.

```
// Rabin Karp algorithm

// Can be any charactars, but we're using numbers instead for example.

//the hash of some strin 'w' will be equal to the sum of the digits % 23.
// the result of this has will be an integer 0-22.
// hash(w) = (epsilon digits) % 23
// (a + b) % m = (a%m + b%m) %m
// if the hash of two strings are different, then the string must be different
// two hashes being the same doesn't mean the patterns are the same

int p[] {3, 8, 7, 6, 2, 6, 3, 8, 4, 9, 7, 6, 3, 2, 8, 4, 5, 5};

int hash(int p[]){
  int sum = 0;
  int len = strlen(p);
  for(int i = 0; i < len; ++i){
    sum += p[i] % 23;
  }
  return sum % 23;
}

rabin karp: worst case: O(mn)
O(n)- probabilistically
```