Prior Prefixes

Timothy Salmon

October 26, 2017

Question

Write a program that takes an input String s and prints all palindromic substrings of s.

Explanation and Algorithm

Solution: Forward and Backwards Hashing

To understand this solution, it is helpful to understand simple multiplicative hashing and polynomial evaluation.

Multiplicative Hashing: Hashing a String s

```
LargePrime = 6829
Base = 1
for (char c : s) {
   Base = Base * LargePrime + c
}
```

The above rule is analogous to Horner's Rule for Polynomial Evaluation: Given a polynomial of degree n,

$b_n = a_n$
$b_{n-1} = a_{n-1} + x * b_n$
...
$b_0 = a_0 + x * b_1$

Horner's rule uses the above formula to evaluate polynomials more efficiently than classical polynomial evaluation. We can view the above multiplicative hash value as evaluating a polynomial with $x = LargePrime$ and with the character values as the polynomial coefficients.

Given this Horner's rule view, we can find all palindromic prefixes of a String in linear time. To do this, we use Horner's rule - style hashing, and normal polynomial evaluation - style, to hash the String and its reverse at the same time.

Hints

1. How do you tell if a string is a palindrome?
2. Consider using hashing.
3. Can you hash a string and its reverse in one pass?
4. How many passes will it take to hash all of the substrings?
5. Can you check multiple substrings in a single pass?

Code

```
/*Answer */

public class FindSubstrings{
   public static void main(String [] args){
         String s = args[2];
      for (int i = 0; i <= s.length(); i++){
         int largePrime = 6829;
            int primePower = 1;
         int forwardBase = 0;
            int reverseBase = 0;
         for (int j = i j < s.length(); j++) {
               forwardBase = forwardBase * largePrime + (int) s[j];
                reverseBase = reverseBase + primePower * (int) s[j];
                primePower *= largePrime;
                if (forwardBase == reverseBase) {
                  System.out.println(s.substring(i,j+1))
                }
            }
      }
   }
}
```

Run time analysis

The run time of this solution is O(n^2), where n is the length of the string. The runtime is polynomial because the hashing strategy gives us all prefixes of a

string in linear time, and there are n different starting indices from which to generate different prefixes.