CS 202 HW 3

Mehmet Akif Şahin
22203673

30 April 2024

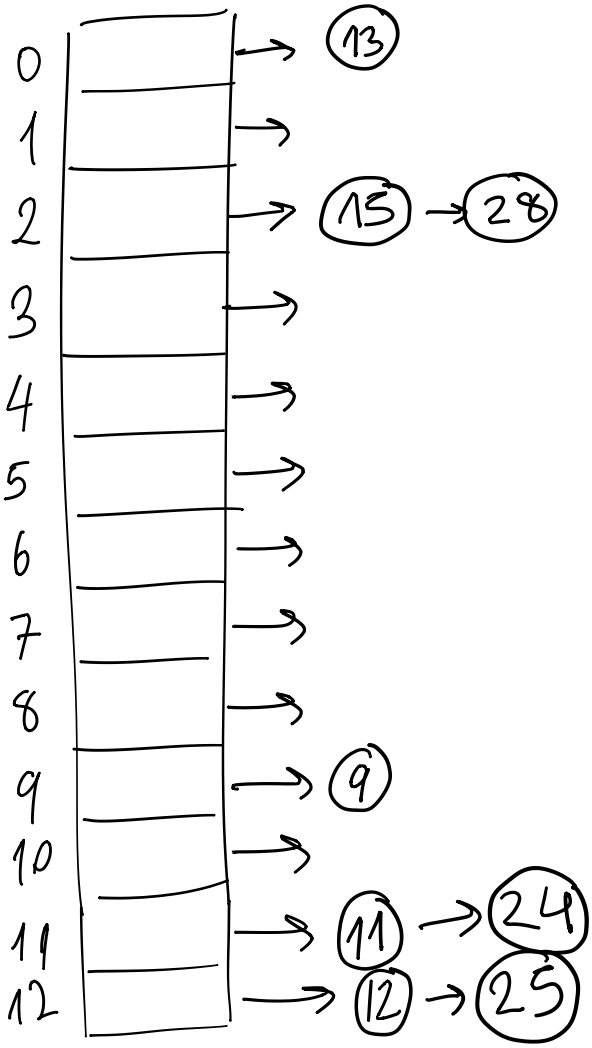
Question 1:

0	13
1	25
2	15
3	28
4	24
5	
6	
7	
8	
9	9
10	
11	11
12	12

Linear probing

0	13
1	
2	15
3	28
4	
5	
6	
7	24
8	25
9	9
10	
11	11
12	12

quadratic probing



Seperate Chaning

Question2:

```
int minTableSize(int *arr, int n) {
    int p = n-1;
    while (true) {
        p++;
        if(!isPrime(p)) continue;
        if(!hasDuplicate(arr, n, p)) return p;
    }
}
bool hasDuplicate(int*arr, int n, int p) {
    for(int i=0; i< n; i++){
        for(int j=i+1; j<n; j++){
            if( myMod(arr[i], p) == myMod(arr[j], p))
                return true;
            }
        }
    }
    return false;
}
int myMod(int x, int mod) {
    while( x <0)  x+=mod;
    while( x >mod) x-=mod;
    return x;
}
```

Question 4

Subtask 1 can be thought as a pattern macthing question. Given strings are patterns, and we search them in prefixes and suffixes of these strings. I used 2 hash tables with 2 prime numbers bigger than the constraints to avoid string comparison. The hash function used is the polynomial hash function with p=31 and rolling hash method is used to calculate the prefix, suffix hashes. Complexity of my solution is

$$\underbrace{O(\text{Total no. of chars})}_{\text{filling the tables}} + \underbrace{(\text{Total no. of chars}) \cdot O(1)}_{\text{calculating prefix, suffix hashes}} = O(\text{Total no. of chars})$$

Subtask-2 is also a pattern matching question. The max. difference bet pattern lengths are given 5 so the question can be divided into similar sub-problems. I used polynomial hash with p=31 with different modulos to calculate (twice) and store the hashes of pattern in a sorted array. Then I used a sliding window to calculate substring hashes in the text with rolling hash and used binary search to find hashes in sorted arrays. The complexity is:

$$\underbrace{O(\text{no. of chars in patterns})}_{\text{pattern hashing}} + \underbrace{O(m \log m)}_{\text{sorting hashes}} + \underbrace{O(n)}_{\text{number of substrings}} \cdot \underbrace{O(\log m)}_{\text{binary search}} + \underbrace{O(\text{no. of chars in text})}_{\text{hashing substrings.}}$$

$$= O(\text{total no. of chars}) + O((n+m)\log m)$$

In subtask3 i used a hash function so that it will give the same output for strings that can be acquired from one another using shift operations. My hash function's only difference with the polynomial hash function is that it hashes all possible strings with rolling hash and selects the minimum. This way it hashes strings that can be generated via shifting to same value. First I generate hashes of strings and sort them. Then with a linear traverse on this sorted hash array, I count distinct elements and check if their reverse's hash exist or not using binary search.

Time complexity of hash function: $\underbrace{O(n)}_{\text{initial hash}} + \underbrace{O(1) \cdot n}_{\text{generating others}} = O(n)$

Time Complexity: $O(\text{no. of chars}) + O(n \log n) + O(\text{no. of chars}) + O(\log n)$

$$= O(\text{no. of chars} \cdot \log n)$$