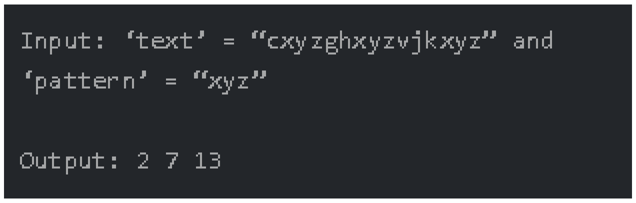
Rabin Karp Algorithm

It is a string pattern matching algorithm one string and a pattern is given in question and task is to find the position at which the pattern is found in the string.



```
Input: ‘text’ = “cxyzghxyzvjkxyz” and
‘pattern’ = “xyz”

Output: 2 7 13
```

Brute force approach is to check the pattern for every window of size of pattern for which the time complexity if O(m*n). Rabin Karp reduces the time complexity to O(m+n) in best case and O(m*n) in worst case.

It makes use of hashing we find the hash value for pattern as well as initial window of size of pattern of the text. For finding the next hash value of text we simply remove the character that us out of the window and include the new character in the window in the hash function. When both hash value matches we perform a check again to be sure that the string is the same so we run a loop matching all the characters in the text window and pattern. If it matches we add it to our answer.

Rabin Karp hash function is also called rolling hash because it computes the hash function for next window using the hash value from the previous window. The number of collisions known as spurious hits here needs to be minimized. We thus make use of this hash function :

p=(p*d+pat[i])%q;

t=(t*d+text[i])%q;

where p and t are initially 0 q is a large prime number to avoid collision and d is the maximum number of characters here taken to be 256

For rolling hash :

t=(d*(t-text[i])+text[i+m]*h)%q;

where h is d^m-1 because the hash we are considering is for example number is 123 ⇒ 1*x^0+ 2*x^1+ 3*x^2 so for removing the least significant bit component is multiplied by x^0 in our case d^0 which is 1 so it is simply subtracted but while adding we need to add it multiplying by d^m-1 which is stored in h. h also needs to be in range q so it is modulo q. if t is negative we make it positive by adding q because our hash table contains only positive values.

Code :

```
vector<int> stringMatch(string text, string pattern) {
  // Write your code here.
  int n=text.size(),m=pattern.size(),h=1,p=0,t=0,q=11,d=256;
  vector<int> ans;
  for(int i=0;i<m-1;i++)
  {
    h=(h*d)%q;
  }
  for(int i=0;i<m;i++)
  {
    p=(d*p+pattern[i])%q;
    t=(d*t+text[i])%q;
  }
  for(int i=0;i<=n-m;i++)
  {
    if(p==t)
    {
      int j;
      for(j=0;j<m;j++)
      {
        if(text[i+j]!=pattern[j])
        {
          break;
        }

      }
      if(j==m)
      ans.push_back(i+1);

    }
    if(i<n-m)
      t=(d*(t-text[i]*h)+text[i+m])%q;
    if(t<0)
    {
      t=t+q;
    }
  }
  return ans;

}
```

- Time Complexity : O(m+n) Best case , O(m*n) worst case
- Space Complexity : O(1)