Pattern Matching Rabin Karp

Tuesday, 19 January 2021 2:18 PM

Pattern matching :-

→ Brute force approach of this is to compare each character of pattern to text and if we observe the pattern we'll return the index

→ We can use rabin karp algorithm.

Rabin karp algorithm :-

→ We take a section in the text which is of same length of pattern

→ Compare if hash code of the section is equal to hash code of pattern.

If they are equal -

then only we compare the string section and pattern.

If they are equal then return index.

Important thing is —

→ To choose a good hash function i.e.,

- → Fast to compute
- → Less collisions - False positives

Normal hash function :-

→ We can add "ascii" values of all the characters and return the sum as the hash code.

→ In this we'll have so many collisions

Optimized hash function :-

d = no.of characters in the text

m = length of pattern.

s = Substring in the middle of text.

$$\Longrightarrow \boxed{hash = s[0] \times d^{m-1} + s[1] \times d^{m-2} + \cdots\cdots + s[m-1] \times d^{0}}$$

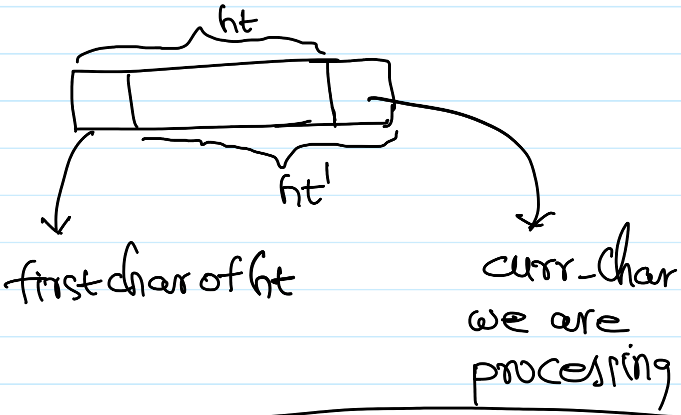
How to compute hash function optimally :

→ If we go on for each string character and compute the hash, it results in same time complexity as that of brute force which is of no use.

→ We can use rolling method to compute hash in O(1) time i.e.,

let, ht be the previous hash

and ht' be the current hash



$$\Longrightarrow \boxed{ht' = (ht - firstcharofht \times d^{m-1}) \times d + curr\_char}$$

Program 1

Tuesday, 19 January 2021 4:30 PM

Problem Statement:

There are pair of words namely W1 and W2 with a limited of word range,
Create a method to return a true value if W2 contains the anagram of W1.
In additional, one of the anagram of first word is the substring of the second word.

Your task is to implement the Solution class, and implement a method in it,
- public boolean checkPalindromeSubstring(String w1, String w2){}.

Input Format:

Two space separated words w1 and w2, consist of lowercase letters only.

Output Format:

Print a boolean value, if W2 contains the anagram of W1 or not.

Sample Input-1:
----
abbcbb abbbabbcb

Sample Output-1:
----
true

Sample Input-2:
----
abbcbbc abbbabbcb

Sample Output-2:
----
false

Code:

```
import java.util.*;
class Solution
{

    public int computeHash(int ht,int currchar, int firstcharofht)
    {
        return ht-firstcharofht+currchar;
    }

    public boolean checkSubstring(String w1,String w2)
    {
        int alphatext[]=new int[26];
        int alphapat[]=new int[26];
        Arrays.fill(alphatext,0);
        Arrays.fill(alphapat,0);
        int lt=w2.length();
        int lp=w1.length();
        if(lt>=lp)
        {
            int hp=0;
            for(int i=0;i<lp;i++)
            {
                hp+=((int)w1.charAt(i))-96;
            }
            int ht=0;
            for(int i=0;i<lp-1;i++)
            {
                ht+=((int)w2.charAt(i))-96;
            }
            // System.out.println(ht);
            int firstcharofht=0;
            int currchar=0;
            int flag=0;
            int i=lp-1;
            while(i<lt)
            {
                flag=0;
                currchar=((int)w2.charAt(i))-96;
                // System.out.println(currchar);
                // System.out.println(firstcharofht);
                ht=computeHash(ht,currchar,firstcharofht);
                // System.out.println(ht);
                if(ht==hp)
                {
                    for(int j=i-lp+1;j<=i;j++)
                    {
                        alphatext[(int)w2.charAt(j)-97]++;
                        alphapat[(int)w1.charAt(j-i+lp-1)-97]++;
                    }
                    for(int j=0;j<26;j++)
                    {
                        if(alphatext[j]!=alphapat[j])
                        {
                            flag=1;
                            break;
                        }
                    }
                    if(flag==0){
                        return true;
                    }
                }
                firstcharofht=((int)w2.charAt(i-lp+1))-96;
                Arrays.fill(alphatext,0);
                Arrays.fill(alphapat,0);
                i++;
            }
            return false;
        }
        else
        {
            return false;
        }
    }
}
```

Test Cases:

case =1
input =dinitrophenylhydrazine acetylphenylhydrazine
output =/false/

case =2
input =abbcbb abbbabbcb
output =/true/

case =3
input =abbcbbc abbbabbcb
output =/false/

case =4
input =listentomotherinlaw wehavesilenthitlerwomantosociety
output =/true/

case =5
input =motherinlawkeepsilentcatdebitcardschoolmasterastronomers
hitlerwomanstarspeeklistenacttheclassroomcreditbadnomore
output =/true/

case =6
input =motherinlawkeepsilentanddebitcardhasbadcreditindirtyroom
peekhitlerwomanlistentheclassroomcreditbadnomore
output =/false/

case =7
input =motherinlawkeepsilentandbadcreditindirtyroom
wehaveanhitlerwomanpeeklistendebitcardindormitory
output =/false/

case =8
input
=motherinlawkeepsilentcatdebitcardschoolmasterastronomershitlerwomanstarspeeklistenacttheclassroomcreditbadnomore
thereisakinglivedinkosalamotherinlawkeepsilentcatdebitcardtenacttheclassroomcreditbadnomoreschoolmasterastronomershitlerwomanstarspeekliswithus
output =/true/

case =9
input =abcdefghijk cbcboooaaah
output =/false/

case =10
input =abcd cbcb
output =/false/

Logic :-

In this we use the normal hash function because we have to find anagrams of the pattern.

→ If the hash code is same i.e., there is possibility that we found an anagram.

So, frequency array of substring is alphatext
frequency array of pattern is alphapat.

→ If these two are equal then, we

→ If these two are equal then, we found an anagram of pattern in the text

Program 2

Tuesday, January 19, 2021 9:02 PM

Problem Statement:

KMIT hosting a Keshav Memorial Badminton League.
They planned to conduct N number of games. Each game begin and ends in perticular time slot.

You are given an array of time slots of the N games, consisting of
begin and end times (b1,e1),(b2,e2),... (b < e).
Your task is to determine minimum number of badminton courts required
to conduct all the games smoothly.

NOTE: If a game begins at time 'a' ends at time 'b',
another game can start at 'b'.

```
Input Format:
-------------
Line-1: An integer N, number of games.
Next N lines: Two space separated integers, begin and end time of each game.

Output Format:
--------------
Print an integer, minimum number of badminton courts required.


Sample Input-1:
---------------
3
0 30
5 10
15 20

Sample Output-1:
----------------
2

Sample Input-2:
---------------
3
0 10
15 25
25 35

Sample Output-2:
----------------
1
```

Code:

```
import java.util.*;
class Badminton
{

   public static int partition(int arr[], int start, int end)
   {
      int pindex=start;
      int pivot=arr[end];
```

```
        int temp=0;
        for(int i=start;i<end;i++)
        {
            if(arr[i]<=pivot)
            {
                temp=arr[i];
                arr[i]=arr[pindex];
                arr[pindex]=temp;
                pindex++;
            }
        }

        temp=arr[end];
        arr[end]=arr[pindex];
        arr[pindex]=temp;

        return pindex;
    }

    public static void quickSort(int arr[],int start, int end)
    {
        if(start<end)
        {
            int pindex=partition(arr,start,end);
            quickSort(arr,start,pindex-1);
            quickSort(arr,pindex+1,end);
        }
        else
        {
            return;
        }
    }

    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int start[]=new int[n];
        int end[]=new int[n];
        for(int i=0;i<n;i++)
        {
            start[i]=sc.nextInt();
            end[i]=sc.nextInt();
        }

        quickSort(start,0,n-1);
        quickSort(end,0,n-1);

        int minCourts=0;
        int maxEndTimeInSpecificCourt=0;
        for(int i=0;i<n;i++)
        {
            if(start[i]<end[maxEndTimeInSpecificCourt])
            {
                minCourts++;
            }
            else
            {
                maxEndTimeInSpecificCourt++;
            }
        }

        System.out.println(minCourts);
    }
}
```

Test Cases:

```
case =1
input =3
0 30
5 10
15 20
output =2

case =2
input =3
0 10
15 25
25 35
output =1

case =3
input =10
1 10
15 25
30 40
45 60
11 15
61 70
41 50
75 90
80 95
91 100
output =2

case =4
input =10
1 15
20 35
30 45
35 50
25 40
10 25
60 75
45 60
40 55
50 65
output =3

case =5
input =15
1 25
10 20
10 35
15 30
25 40
30 50
25 50
40 75
35 60
20 40
40 60
35 50
20 45
25 60
50 75
output =8
```

```
case =6
input =20
1 25
10 20
10 35
15 30
45 60
25 40
35 55
25 50
50 90
55 75
50 80
40 75
35 60
20 40
40 60
70 90
35 50
20 45
25 60
50 75
output =9

case =7
input =10
10 40
40 70
50 80
70 100
100 130
130 150
65 95
55 85
45 75
35 65
output =5

case =8
input =15
1 15
20 35
30 45
35 50
25 40
10 25
60 75
45 60
40 55
50 65
15 35
35 60
30 50
45 70
60 90
output =6
```

Logic :-

→ First we sort both start and end

→ First we sort both start and end arrays independently.

→ Now, we loop i in the start and initialize j=0 ; court=0 ;

j is the maximumpossible end time of current court

→ So, if (start[i] < end[j])

⟹ another match starts before the current court match ends.

So, we need another court to play that match.

⟹ court ++;

→ else i.e., start[i] ≥ end[i].

⟹ another match starts after the current match

⟹ we can play the match in the same court.

So, we increment j because the match is completed and we have to move further

Eg:

5

2 19

4 6

5 31

4	6
5	31
7	29
9	15

↓i

⟹ start : 2 4 5 7 9 counts = 0

end : 6 15 19 29 31

↑
j

⇓ start[i] < end[j]

i

start : 2 4 5 7 9 counts = 1

end : 6 15 19 29 31

↑
j

⇓ start[i] < end[j]

i

start : 2 4 5 7 9 counts = 2

end : 6 15 19 29 31

↑
j

⇓ start[i] < end[j]

i

start : 2 4 5 7 9 counts = 3

end : 6 15 19 29 31

↑
j

⇓ start[i] > end[j]

i

start : 2 4 5 7 9 counts = 3

end : 6 15 19 29 31

end : 6 15 19 29 31
↑ j

Counts = 3

⇓ start[i] < end[j]

↑ i

start : 2 4 5 7 9

end : 6 15 19 29 31
↑ j

Counts = 4

→ Counts = 4

Program 3

Tuesday, January 19, 2021 9:04 PM

Problem Statement:

Mr. James professor of at Illinois state university, as a part of assignment he created a
problem statement related to strings.
He gave a String S, and asked them to design a method to
return the longest substring in S, which is a palindrome.

NOTE: Alphabets are case sensitive
"Aa" is not considered a palindrome here.

Input Format:

A string S, consist of lowercase/uppercase letters or/and digits

Output Format:

Print a string, longest palindrome substring.

Sample Input-1:
----
abbbabbcbbacdb

Sample Output-1:
----
abbcbba

Sample Input-2:
----
thedivideriswide

Sample Output-2:
----
edivide

Code:

```
import java.util.*;
class LongestPalindromicSubString
{
   public static void main(String args[])
   {
        Scanner sc=new Scanner(System.in);
        String s=sc.next();
        int n=s.length();
        boolean dp[][]=new boolean[n][n];
        int maxi=0;
        int maxj=0;

        dp[n-1][n-1]=true;
        for(int i=0;i<n-1;i++)
        {
            dp[i][i]=true;
            if(s.charAt(i)==s.charAt(i+1))
            {
                dp[i][i+1]=true;
                maxi=i;
                maxj=i+1;
            }
            else
                dp[i][i+1]=false;
        }

        int row=0,col=0;
        for(int j=2;j<n;j++)
        {
            row=0;
            col=j;
            while(row<n && col<n)
            {
                if(s.charAt(row)==s.charAt(col))
                {
                    dp[row][col]=dp[row+1][col-1];
                    if(dp[row][col]==true)
                    {
                        maxi=row;
                        maxj=col;
                    }
                }
                else
                {
                    dp[row][col]=false;
                }
                row++;
                col++;
            }
        }

        System.out.println(s.substring(maxi,maxj+1));
    }
}
```

Test Cases:

case =1
input =oneofthemsaidwepanicinapewinabook
output =wepanicinapew

case =2
input =forgeeksskeegfor
output =geeksskeeg

case =3
input =abbbabbcbbacdb
output =abbcbba

case =4
input =thedivideriswide
output =edivide

case =5
input =itisneveroddorevenwhenyougivenazeroasinput
output =neveroddoreven

case =6
input =yourmaidmadeademandthatsiridemandiamamaidnamedirispleaseconsider
output =siridemandiamamaidnamediris

case =7
input =theyhaveprintedaphrasewontloversrevoltnowinapaperheading
output =wontloversrevoltnow

case =8
input =wepanicaspewbutdontnodsoneveroddorevensiridemandiamamaidnameiris
output =emandiamamaidname

Logic :-

→ This can be done using DP.

→ Base cases are -

a) all the substrings whose length is 1 are palindromes. ⟹ dp[i][i] = true;

b) substrings with length 2 —

i) they are palindromes if both characters are equal.

ii) else they are not palindromes.

are equal.

ii) else they are not palindromes.

→ Now, from length 3 lets say in the given string this substring starts from i and ends at j (i, j both inclusive).

$$\Longrightarrow \quad \text{substring} = \begin{array}{cccccc} X & Y & \text{------} & A & B \\ i & i+1 & \text{-----} & j-1 & j \end{array}$$

For this substring to be a palindrome -

a) X and B must be equal.

(If they are not equal if we reverse it doesn't become palindrome)

b) substring - Y.... A must be a palindrome.

i.e.,

a) s.charAt(i) = s.charAt(j)

b) and dp[i+1][j-1] = true.

Eg:

Eg:

abba

	0	1	2	3
0	T	F	F	T
1		T	T	F
2			T	F
3				T

$\Longrightarrow$ maximum length $= 3 - 0 + 1$

$= \boxed{4}$

$\Longrightarrow$ longest palindrome $=$ substring$(0, 4)$

$= \boxed{abba}$