String Matching
Using
Rabin – Karp Algorithm

-Presented by Adrita Alam

Outline :

- Definition of the Rabin-Karp algorithm
- How Rabin-Karp works
- Example of Rabin-Karp algorithm
- Time Complexity

String Matching Problem

- ❖ To find all occurrences of the pattern p in the given text T
- ❖ Brute Force method can be used to solve this problem by comparing all the letters of the sequence P match over text T.
- ❖ The worst case scenario can reach O(n*m);

  where n= length of given text string,

  m= length of the pattern string

Rabin - karp Algorithm

- A string search algorithm which compares the Hash Value of strings to speed up the search rather than the strings themselves.

- For efficiency, the hash value of next position in the text is easily calculated from the hash value of the current position.

How Rabin-Karp works

- Choose a prime number P and calculate the hash value of pattern string
- Consider an array to store the consecutive hash value for all substrings of the text string
- Check if the hash value matches

  -If the hash values are unequal: Calculate the hash value for next M-character subsequence

  -If the hash values are equal : return the position

Example:

	0	1	2	3	4	5	6	7	8	9	10
Text :	C	C	A	C	C	A	B	E	D	B	A

Text length,n=11

3+3+1=7

	0	1	2
Pattern :	D	B	A

Pattern length,m=3

4+2+1=7

	0	1	2	3	4	5	6	7	8	9	10
Text :	C	C	A	C	C	A	B	E	D	B	A

3+1+3=7

Text length,n=11

	0	1	2
Pattern :	D	B	A

4+2+1=7

Pattern length,m=3

	0	1	2	3	4	5	6	7	8	9	10
Text :	C	C	A	C	C	A	B	E	D	B	A

1+3+3=7

Text length,n=11

	0	1	2
Pattern :	D	B	A

4+2+1=7

Pattern length,m=3

0 1 2 3 4 5 6 7 8 9 10

Text : C C A C C A B E D B A

$3*(11)+3*(11)^2+1*(11)^3 = 1727$

Text length,n=11

Let a prime number, P=11

0 1 2

Pattern : D B A

$4*(11)+2*(11)^2+1*(11)^3 =1617$

Pattern length,m=3

0 1 2 3 4 5 6 7 8 9 10

Text : C $\boxed{\text{C A C}}$ C A B E D B A

$3^*(11)+1^*(11)^2+3*(11)^3 = 4147$

Text length,n=11

Let a prime number, P=11

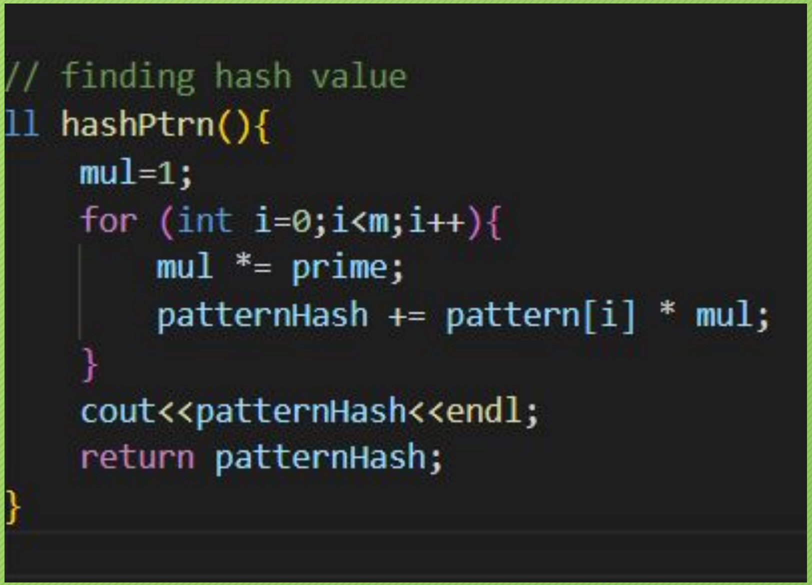
0 1 2

Pattern : D B A

$4^*(11)+2^*(11)^2+1*(11)^3 =1617$

Pattern length,m=3

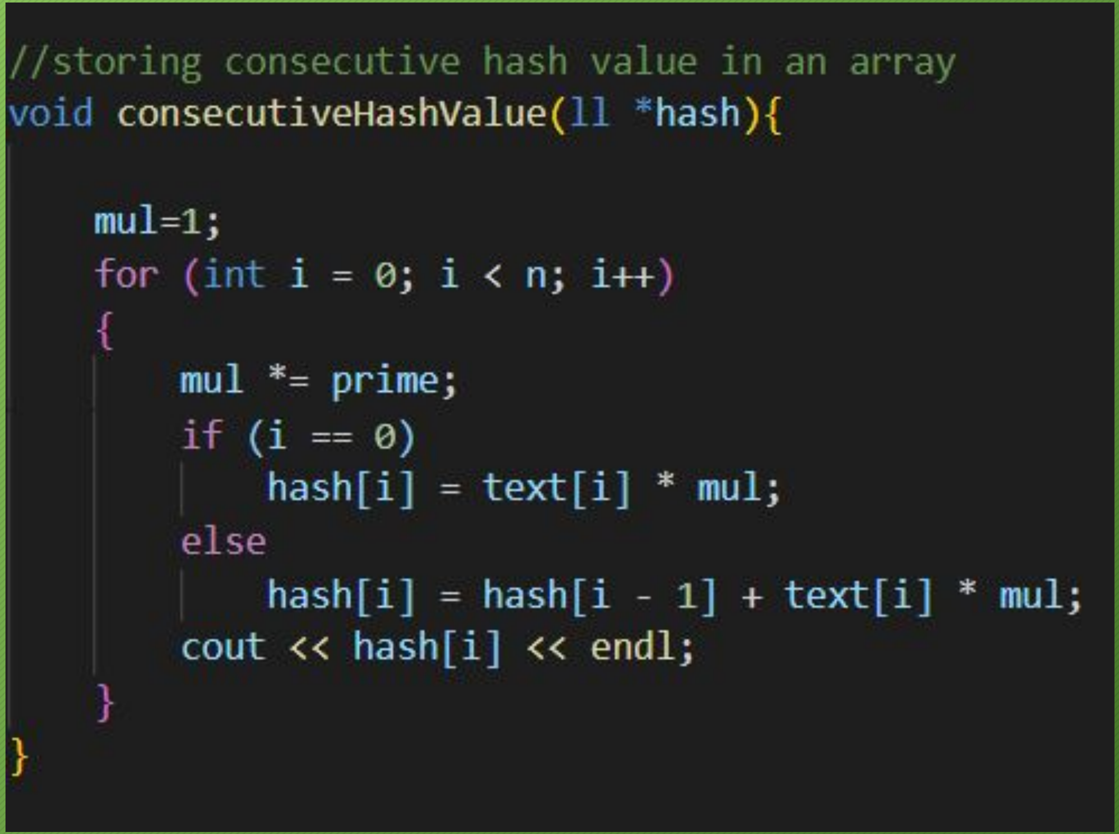
0	1	2	3	4	5	6	7	8	9	10
C	C	A	C	C	A	B	E	D	B	A
33	396	1727	45650	528803	2300364	…	…	…	…	…



```
// finding hash value
ll hashPtrn(){
    mul=1;
    for (int i=0;i<m;i++){
        mul *= prime;
        patternHash += pattern[i] * mul;
    }
    cout<<patternHash<<endl;
    return patternHash;
}
```

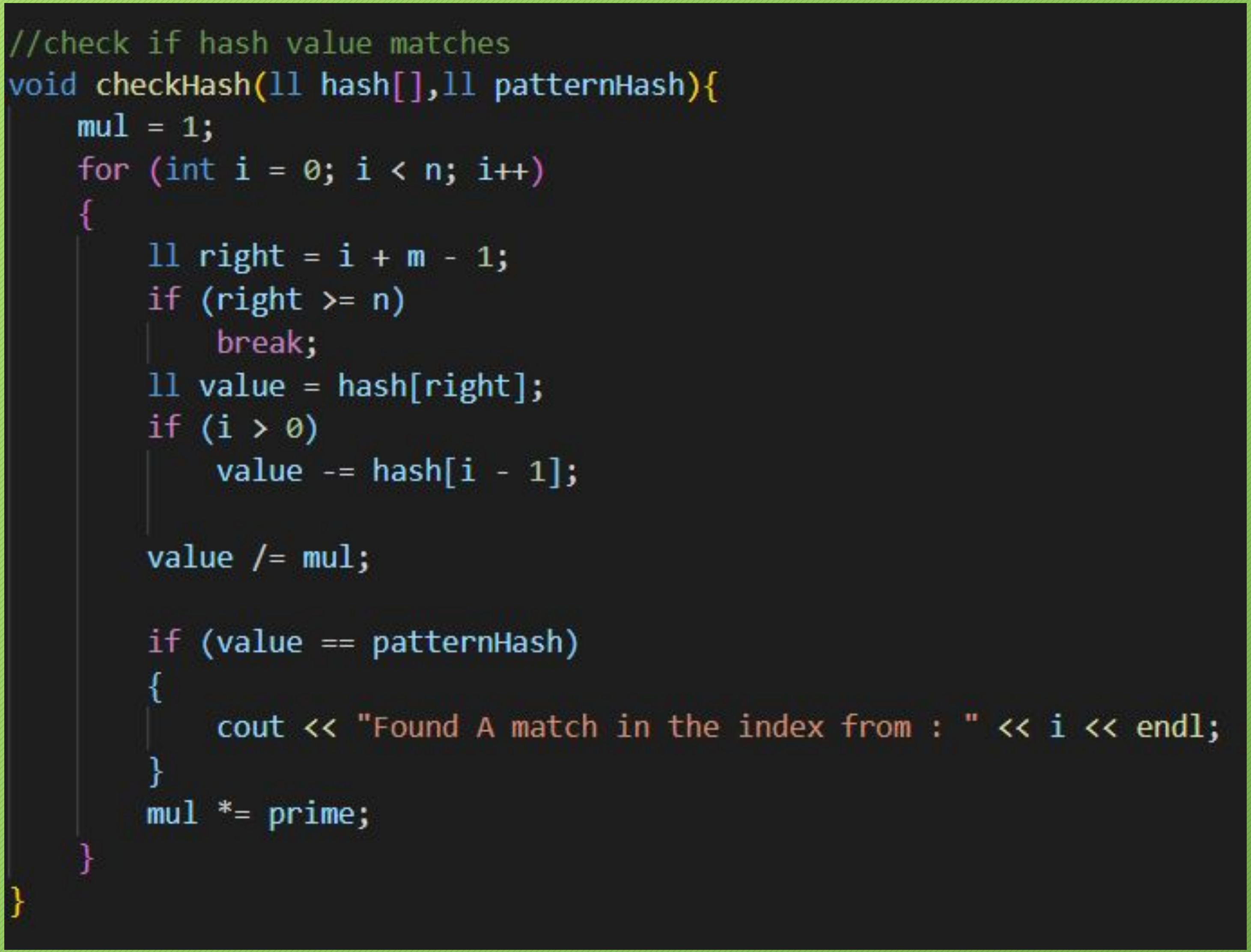
- By considering a prime number we have calculated the hash value of the pattern string

- We have store the consecutive hash values in an 1D array named hash[ ]



```
//storing consecutive hash value in an array
void consecutiveHashValue(ll *hash){

    mul=1;
    for (int i = 0; i < n; i++)
    {
        mul *= prime;
        if (i == 0)
            hash[i] = text[i] * mul;
        else
            hash[i] = hash[i - 1] + text[i] * mul;
        cout << hash[i] << endl;
    }
}
```



```
//check if hash value matches
void checkHash(ll hash[],ll patternHash){
    mul = 1;
    for (int i = 0; i < n; i++)
    {
        ll right = i + m - 1;
        if (right >= n)
            break;
        ll value = hash[right];
        if (i > 0)
            value -= hash[i - 1];

        value /= mul;

        if (value == patternHash)
        {
            cout << "Found A match in the index from : " << i << endl;
        }
        mul *= prime;
    }
}
```

- Set right : i+m-1(i=0,1,2,…,n-1)

- Check whether the required value is equal to the pattern hash value –
  If Yes: Found
  Else : Check the next consecutive hash value

Time Complexity

- For best case: O(n+m)
- For average case: O(n+m)
- For worst case: O(n*m);when all characters of pattern and text are same as the hash values of all substring