(1) Create a Token Object. with line no where it originated.
(2) Internally each token is represented by a single character for the comparisons.

---

(1) Comparisson of token strings
(1) Algorithm - Greedy String Tiling.
(2) Aim: Find Set of substrings that are the same and satisfy following rules :-
    (1) Match only 1 token of A. string with 1 token of B string.
    (2) Substrings to be found independent of their position in their string.
    (3) Long substring matches are preferred over short ones

Robin Karp Algorithm for String Matching.

Text
Pattern

① Preprocessing text pattern ② Matching

Let us assume $\Sigma = \{0, 1, 2, 3, 4 \ldots 9\}$

→ each character is a decimal digit.

We can view a string of k digits as length k decimal number

string = 31456 = decimal number. 31456

string number

1 2 3 4 5 7 2

substring / Number.

a - - - z

A - - - z

Radix d = 52 number.

radix d no system.

[7898].

2246 7898

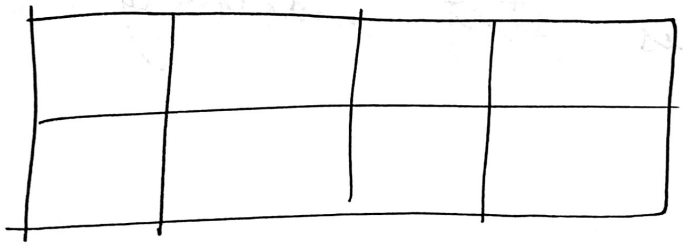
Pattern Matching.

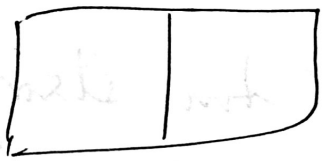
p = decimal value.

t[1..n] . ts = .

=

Tiling Problem:





Horner's Rule.

$3\times10^4 + 2\times10^3 + 7\times10^2 + 3\times10^1 + 2\times10^0$.

$= [3^5\times10^4 + 2\times10^3 + 7\times10^2 + 3\times10^1 + 1\times10^0]$.

$= 1 \quad 0$.

count(n) = count(n-1) + count(n-2)

O(logn) .- Fibonacci sequence.

Text: "GEEKS FOR GEEKS" length = N

Pattern : "GEEK" length = M

index = 0, 8

Slide pattern - match hash value of pattern with hash value of current substring of text.

If hash value matches start matching individual characters

hashing converts text to numeric value.

Algorithm uses the fact that if two strings are equal their hash values are also equal.

Assume length of text > length of pattern M.

hash ("GEEK") = X.

window size = M.

Hash Function:

Requirement: Hash at the next shift must be efficiently computable (O(1)). from current hash value and next character in text.

Rehashing: hash(txt[s+1...s+m]) =
d(hash(txt[s...s+m-1]) - txt[s]*h) + (txt[s+m]) mod q.

hash next = hash current # - trailing current char + Leading current char

d: No of characters in alphabet - 256

h = d^(m-1).

q = prime number.

```
#define d 256

// pat → pattern
// txt → text
// q → prime no

void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0; // hash value for pattern
    int t = 0; // hash value for txt.

    int h = 1;

for (i=0; i<M-1; i++)
    h = (h*d) % q;
for (i=0; i<M; i++)
{   p = (d*p + pat[i]) % q;
    t = (d*t + txt[i]) % q; }
```

txt = "ABCDB"

pat = "DC"

q = 11

M = 2 N = 5

h = (1 * 256) % 11 = 3

Iter 1 p = (256 * 0 + 68) % 11 = 2

t = (256 * 0 + 65) % 11 = 10

Iter 2 p = (256 * 2 + 67) % 11 = 7.

t = . - - - - - -

Slide over - patterns on text.

Repeat iterations with +ve.