

KMP কি?

kmp স্ট্রিং ম্যাচিং এ্যালগরিদম। Kmp লিনিয়ার টাইমে একটা স্ট্রিং T তে একটা প্যাটার্ন স্ট্রিং P কতবার আছে এবং কোন কোন পজিশনে আছে তা বের করে।

KMP কিভাবে কাজ করে?

Kmp কিভাবে কাজ করে তা বুঝার আগে একটা স্ট্রিং T থেকে প্যাটার্ন স্ট্রিং P কতবার ও কোথায় আছে তার naive solution টা দেখে নেয়া জরুরী।

```
#include<bits/stdc++.h>

char T[10000];

char P[10000];

int main()

{

    scanf("%s %s",T,P);

    for(int i=0;T[i];i++)

    {

        bool flag=true;

        for(int j=0;P[j];j++)

        {

            if(P[j]!=T[i+j])

            {

                flag=false;

                break;

            }

        }

    }
```

```

    }

    if(flag)std::cout<<"match at position "<<i<<std::endl;

    }

    return 0;

}

```

উপরে solution এ T এর প্রতি পজিশন থেকে P স্ট্রিং ম্যাচ করে দেখা হচ্ছে এবং কোথাও ম্যাচ না পেলে আবার P স্ট্রিং এর প্রথম পজিশনে গিয়ে ম্যাচ করে দেখা হচ্ছে।

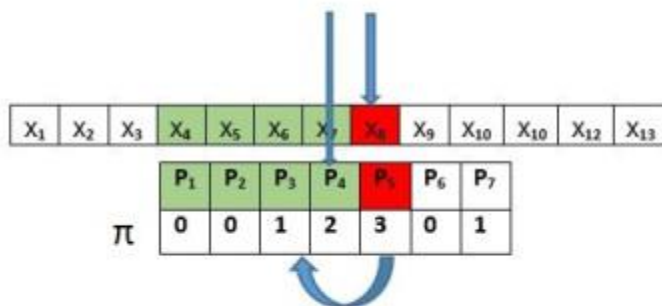
Kmp এ্যালগরিদম ম্যাচ না পেলে P স্ট্রিং এর বার বার প্রথম পজিশন এ গিয়ে ম্যাচ করানোর জিনিস টা স্কিপ করে।

Kmp এ্যালগরিদমে একটা ফাংশন থাকে যাকে prefix/ π /failure ফাংশন বলা হয়ে থাকে। এই ফাংশন একটা π এয়ারে তৈরী করে দেয়, যাতে প্যাটার্ন স্ট্রিং P এর সব পজিশনে কতটুকু প্রিফিক্স ম্যাচ করে তা থাকে।

যখনই T স্ট্রিং এর কোন পজিশন এ P এর স্ট্রিং এর কোন পজিশন মিস ম্যাচ হবে, তখন π এয়ারে দেখে P স্ট্রিং এর কোন পজিশনে যাবে তা নির্ধারণ করা হয়।

ধরা যাক P স্ট্রিং $[P_1, P_2, P_3, P_4, P_5, P_6]$ এর π এয়ারে $[0, 0, 1, 2, 3, 0, 1]$ এমন।

এখন একটা T স্ট্রিং আছে $[X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}]$ এমন।



এখন T স্ট্রিং এর সাথে P স্ট্রিং ম্যাচ করতে করতে X_8, P_5 মিস ম্যাচ হল, তাহলে $\pi[5]$ পজিশন এ জাম্প করবে এবং দেখবে P স্ট্রিং এর পরের পজিশন T এর current পজিশনের সাথে ম্যাচ করে নাকি। যদি ম্যাচ করে তাহলে পরবর্তী সার্চ সেখান থেকে শুরু হবে। কারন আমি T স্ট্রিং এর সাথে P স্ট্রিং এর $\pi[5]$ পর্যন্ত প্রিফিক্স ম্যাচ করে আসেছি। আর যদি ম্যাচ না করে তাহলে সেই পজিশনের π পজিশনে আবার জাম্প করবে ঠিক করবে কোথায় থেকে ম্যাচিং শুরু করবে।

Compute Prefix/Failure function:

Here **Green** character need to match, **Red** character mismatch, **Blue** character match.
Pattern string: "ababaca"

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0						

Step : 1

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0					

Step : 2

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0	1				

Step : 3

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0	1	2			

Step : 4

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0	1	2	3		

Step : 5

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0	1	2	3	0	

Step : 6

i	1	2	3	4	5	6	7
P[i]	a	b	a	b	a	c	a
$\Pi[i]$	0	0	1	2	3	0	1

Step : 7



Π array calculation steps:

```
vector<int> compute_prefix(const char *p)
{
    int m=strlen(p+1);

    vector<int> prefix(m+1);
```

```

prefix[1]=0;

int k=0;

for(int i=2; i<=m; i++)

{

    while(k>0 and p[k+1]!=p[i])k=prefix[k];

    if(p[k+1]==p[i])k=k+1;

    prefix[i]=k;

}

return prefix;

}

```

KMP-Matcher:

Here **Blue** part current matching, **Green** part previously match and **Red** part current mismatch.

Text String T="ababababacaab", Pattern String P="ababaca"

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

this 2 position of T & P are match. So matching will be continue from this position of P.

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

this 2 position are match. So next search will be continue from this position of P.

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T	a	b	a	b	a	b	a	b	a	c	a	a	b
P	a	b	a	b	a	c	a						
	0	0	1	2	3	0	1						

T

a	b	a	b	a	b	a	b	a	c	a	a	b
---	---	---	---	---	---	---	---	---	---	---	---	---

P

a	b	a	b	a	c	a
0	0	1	2	3	0	1

↑

T a b a b a b a b a c a a b

P a b a b a c a
0 0 1 2 3 0 1

```
vector<int> KMP_match(const char *txt,const char *ptrn)
```

{

```
int n=strlen(txt+1);
```

```
int m=strlen(ptrn+1);
```

```
vector<int> Prefix=compute_prefix(ptrn);
```

```
vector<int>Match_position;
```

```
int q=0;
```

```
for(int i=1; i<=n; i++)
```

 $\{$

```
while (q>0 and ptrn[q+1]!=txt[i]) q=Prefix[q];
```

```
if (ptrn[q+1]==txt[i]) q=q+1;
```

```
if (q==m)
```

```

        {

            Match_position.push_back(i-m);

            q=Prefix[q];

        }

    }

    return Match_position;

}

```

Complexity analysis:

Compute prefix function এ লাইন 9 while loop execute হবে $O(m)$ times. এখন $k=0$ এবং 7-12 লাইনে for loop এ k সর্বোচ্চ প্রতিবার বাড়তে পারে আর k এর সর্বোচ্চ মান মনে $m-1$ ।

আবার $k < i$ এবং $\pi[i]$

দেখা যাচ্ছে while loop এর প্রতি iteration এ k time decrease হচ্ছে।

তারমানে পুরো for loop এ while loop total $m-1$ time iterate হচ্ছে। সুতরাং prefix function er runtime হচ্ছে, $\Theta(m)$ ।

একইভাবে kmp matcher function এর runtime analysis করা যায় $\Theta(n)$ ।

সুতরাং kmp algorithm এর total complexity: $\Theta(n+m)$.

All together:

```

#include<bits/stdc++.h>

using namespace std;

char TXT[10000000],ptr[10000000];

vector<int> Match_position;

vector<int> compute_prefix(const char *p)

{

    int m=strlen(p+1);

```

```

vector<int> prefix(m+1);

prefix[1]=0;

int k=0;

for(int i=2; i<=m; i++)

{

    while(k>0 and p[k+1]!=p[i])k=prefix[k];

    if(p[k+1]==p[i])k=k+1;

    prefix[i]=k;

}

return prefix;
}

vector<int> KMP_match(const char *txt,const char *ptrn)

{

    int n=strlen(txt+1);

    int m=strlen(ptrn+1);

    vector<int> Prefix=compute_prefix(ptrn);

    vector<int>Match_position;

    int q=0;

    for(int i=1; i<=n; i++)

```



```

{

    while (q>0 and ptrn[q+1]!=txt[i]) q=Prefix[q];

    if (ptrn[q+1]==txt[i]) q=q+1;

    if (q==m)

    {

        Match_position.push_back(i-m);

        q=Prefix[q];

    }

}

return Match_position;

}

int main()

{

    scanf("%s %s",TXT+1,ptr+1);

    vector<int> Match_position=KMP_match(TXT,ptr);

    for(int i=0; i<Match_position.size(); i++)

    {

        if(!i)printf("%d",Match_position[i]);

        else printf(" %d",Match_position[i]);

    }

}

```

```
    }  
  
    return 0;  
  
}
```

ইনপুট নেয়া লাগবে index 1 থেকে।

Reference:

1. Introduction to Algorithms – Cormen
2. [wiki](#)

Practice problem:

1. [Ministry of Truth](#)
2. [Jack's Last Word](#)
3. [Substring Frequency](#)
4. [Making Huge Palindromes](#)

বিদ্রঃ কোথায় ভুল থাকলে বা বুঝতে সমস্যা হলে কमेंটে জানানোর জন্য অনুরোধ করা হল।

Happy Coding 😊😊