Z Algorithm

Z algorithm এবং KMP দুইটাই prefix match store করে রাখে তাই kmp পারি মানেই এইটা আর লাগবে না , শিক্ষার ও দরকার নাই , রান টাইম ও একই (pattern match খুঁজার জন্য কোন string এ) । কিন্তু কিছু প্রবলেম আসলেই kmp থেকে Z algorithm করা সহজ এবং বুঝাও যায় অনেক ভাল । এই প্রবলেমটা দেখার পর আবার নতুন করে অনুপ্রেরণা পাই Z Algorithm শিখার :D তাই শিখতে তো কোন দোষ নাই । এইখানে আমরা AC code এর link । তবে এইটা পুরা লিখাটা পরে তারপর দেখার অনুরোধ করা হল ।

লিখাটা লিখা হচ্ছে Z algorithm শিখে , যেহেতু আমি নিজেই তেমন কিছু পারি না আর খুব একটা প্রবলেম সল্ভ করার হয় নাই স্বাভাবিকভাবে কিছু ভুল থাকতে পারে । যদিও আমি যথাসম্ভব সাবধানতার মাধ্যমে লিখার ট্রাই করতেছি যেন বাংলায় (আমার জানা মনে কোন লিখা এখনও আমার চোখে আসে নাই) একটা টিউটরিয়াল থাকে । একটা ব্যাপার আমাকে অনেক কস্ট দেয় বাংলায় খুব কমই লিখা আছে । এক শাফায়াত ভাইয়া অনেক কস্ট করে অনেক লিখা লিখে ফেলছেন । কিন্তু অন্যান্য দেশ এ চীন বা রাশিয়ার দিকে যদি দেখি ঐখানে ওদের নিজেদের ভাষায় সব লিখাই পাওয়া যায় যাতে school এর বাচ্চাও শিখতে পারে । স্বাভাবিক ভাবেই তারা আমাদের চেয়ে সব সময় আগাইয়া থাকে কারণ তারা আগে শুরু করছে এবং শিক্ষার উপকরণ সহজলভ্য । তাই এই ব্যাপারে আমার সবার প্রতি অনুরোধ আপনি যাই জানেন এইটা কোথাও লিখা রাখা হইলে দেখতে দেখতে সব কিছুর লিখা হয়ে যাবে । আর সব ভার্সিটি/কলেজ/ স্কুল এ ট্রেনিং এর সমান সুযোগ থাকে না । অনেক ভাল ভাল ট্রিক্স থাকে যেইটা অনেক কস্ট করে হয়তো শিখা হইছে এইটা শেয়ার করা হলে এইটা যে কেউ যেখানেই থাকুক না কেন শিখতে পারবে । আর জ্ঞান এমন একটা জিনিস যেটা শেয়ার করা হইলে কখনও কমে না বাড়েই ।

Z algorithm এ কি থাকে বলি । Z algorithm এ একটা array থাকে যেখানে পার পজিশনে i > 0 to size_of_length , longest prefix এর length store থাকে । by default index 0 মানে Z[0] = length_of_string .

Say
String t = ABCABCABC
Z[] = 900600300

এইখানে index 0 স্বাভাবিক ভাবেই পুরা string টাই match আছে তাই Z[0] = lenght_of_string .
Z[1] = 0 কারণ , BCABCABC এর সাথে ABCABCABC এর ০ পজিশন থেকে কোন match নাই ।
Z[2] = 0
Z[3] = 6 কারণ ABCABC এর সাথে ABCABCABC match করছে ।

ঠিক একই কারণে Z[6] = 3

আমরা trivial process দেখি কিভাবে করা যেতে পারে ।

```
const int NX = 1e5 + 10 ; // length of string


char text[NX] ;
int Z[NX] ;


void Z_Algorithm( )
{
    int sz = strlen(text);
    Z[0] = sz ; // always
    for ( int i = 1 ; i < sz ; i++ )
```

```
    {
        while( i + Z[i] < sz && text[Z[i]] == text[ i + Z[i]] ) ++Z[i];
    }
}
```

এইখানে O(n^2) রান টাইম লাগছে , আমরা এইটাকে চাইলে O(n) এ নিয়ে আসতে পারি । কিভাবে এইটা এখন দেখব ।

আমরা i থেকে n - 1 পর্যন্ত Z[i] এর ভ্যালু ক্যালকুলেশন করে যাচ্ছি । আমরা যখন 'i' position এর value এর কাজ করব তখন কোন ভাবে 1 - (i - 1) এ আগে ক্যালকুলেট করা ভ্যালুগুলা ব্যাবহার করার চেস্টা করব । যেন আমরা আমাদের কাজটাকে সহজ করে ফেলতে পারি ।

আমরা কিছু segment এ আমাদের স্ট্রিংটার match বের করতে পারি । প্রত্যেকটা segment কে একটা Z Box ভাবতে পারি , যেমন আগের example এ Z[3] = 6 এইখানে starting point 3 থেকে ending point 9 এ একটা segment আছে মানে S[0 - (ending (9) - starting (3)] == S[starting (3) - ending (9)] মানে আমরা বলতে চাচ্ছি ABCABCABC এর 0 to (9 - 3) == 6 মানে ABCABC == 3 to 9 মানে ABCABC ।
এখন আমরা যখন এই segment টা পেয়ে গেলাম যখন আমরা সামনে অন্যকোন পজিশনের জন্য Z[] এর ভ্যালু ক্যালকুলেট করব তখন আমরা চাব যেন এই ইনফরমেশনটা আমরা use করতে পারি । এতে আমাদের অনেক ক্যালকুলেশন তো কমবেই সেই সাথে আমাদের code এর run time almost O(n) এ চলে আসবে । এইখানে এই অবজারবেশন থেকে দুইটা Case আমরা পাই ।

1. position > ending_point : এই অবস্থায় আমাদের trivial প্রসেস এ ক্যালকুলেশন করে দেখতে হবে আমরা কয়টা ম্যাচ পাচ্ছি । যদি নতুন কোন ম্যাচ আমরা পাই সেই সাথ আমাদের starting_point and ending_point update হবে ।
2. position <= ending_point এর মানে হচ্ছে আমরা আগে এমন একটা segment পেয়ে এসেছি যার মধ্যে আমাদের current_position টা পরে গেছে । এখন আমাদের এই আগের ক্যালকুলেশন এ Z[] এর যে মানগুলা আমাদের কাছে আছে (1 to position - 1) এইগুলাকে আমাদের কাছে লাগাতে হবে । আমরা surely একটা জিনিস বলতে পারি । Z[position] = min (ending_point - position + 1 , Z[position - starting_point]) ;
   এইটা কেন কাজ করে এই জিনিসটা সম্পূর্ণ কোডটা দেখা হলে বুঝা যাবে ।

কোড :

```
const int NX = 1e5 + 10  ; // string size
char text[NX] ;
int Z[NX];


void Z_Algorithm()
{
    int position , starting_point , ending_point ;
    int sz = strlen( text );
    Z[0] = sz ; // always ;
    for ( position = 1 , starting_point = 0 , ending_point = 0 ; position < sz ; position++ )
    {
```

```
        if( position <= ending_point ) Z[position] = min( ending_point - position + 1 , Z[position-starting_point] );
        while( position + Z[position] < sz && text[Z[position]] == text[ position + Z[position] ] ) ++Z[position] ;
        if ( position + Z[position] - 1  > ending_point ) // need to update
        starting_point = position , ending_point = position + Z[position] - 1 ;
    }
}
```

যেহেতু starting_point and ending_point এর কখনও কমবে না তাই এই কোডটার running time হয় O(n) .
একটা ছোট্ট example দেই ।
Say
S = AAAA
Z[] = 4321

এখন position = 1 , position <= ending_point is n't true .
তাই নিচে while_loop এ ভ্যালু update hobe . ফলে next iterration এর সময় starting_point = 1 , ending_point = 3 . position (2) <= ending_point (3) so Z[position] = min (ending_point - position + 1 , Z[position - strating_point])
Z [2] = min (3 - 2 + 1 , Z[2 - 1]) ;
Z[2] = min (2 , Z[1]) ;
Z[2] = min (2 , 3) ;
Z[2] = 2 আমরা বলতে পারি ।
এখন প্রশ্ন হইল কেন Z[1] এ খালি বসল না , কারণ Z[1] এর ক্যালকুলেশন আমরা ending_point (3) পর্যন্ত রেজাল্ট জানি যদি আমাদের কারেক্ট সেভ করা রেজাল্ট আমাদের ending_point cross করে ঐখানে কি আদ্য আছে বা নাই আমরা জানি না ।

প্রায় সব kmp এর প্রবলেম Z algorithm এ করা পসিবল ।