# H問題解説
# Hth Number

haji

# 問題

長さNの文字列Sがある。Sに含まれる文字はすべて0以上9以下の数字である。
Sのすべての部分文字列から作られる数を全列挙してできた項数N×(N+1)/2の数列を作った時、 その数列の中でH番目に小さい値を求めよ。

# sample input 3

例えば

長さ5の文字列”10031”のすべての部分文字列の数の中で13番目に小さい値を求めたい。

“10031”から作ることができる部分文字列は5*(5+1)/2=15通りである。

“1”, “0”, “0”, “3”, “1”,
“10”, “00”, “03”, “01”,
“100”, “003”, “031” ,
“1003”, “0031”,
“10031”


0, 0, 0, 1, 1,
3, 3, 3, 10, 31,
31, 31, **100**, 1003, 10031

よって13番目に小さい値は**100**である。

# 解法

1. 部分文字列の値が0になる通り数を求める。
   **この通り数をAとする。**

1. 答えとなる数の桁数を求める。
   **この桁数をKとする。**

1. 部分文字列の値がK桁になる数を昇順ソートしm番目の値を求める。
   **m = H - A - (0を除くK桁未満の値になる部分文字列の通り数)**

# 1. 部分文字列の値が0となる通り数。

部分文字列の値が0になるような[l, r)の通り数。

これは連続する0の数を数え上げると簡単に求めることができる。

例:
S = “10010001”の部分文字列が0となる通り数は、
1001000001 ⇒ (2 * 3) / 2 + (5 * 6) / 2 = 18通りある。

# 2.　　答えの桁数を求める。

最初にSの文字が1 ~ 9のときにa桁の数の通り数の求めかたを考える。
これは簡単で、先頭の添え字が長さa以上の文字列を作れる必要があるので、(N-i) ≥ aを満たすiの通り数である。

例:
N = 5, S = “13445”　, a = 3

赤色の部分を始点とする長さ3部分文字列をつくることができる。

## 2.　　答えの桁数を求める。

次にSに文字’0’が含まれているときを考える。さっきの方法だと’0’を先頭の添え字としたときに正しく数え上げることができない、そこで、’0を先頭とする添え字を以下のように変換する。

$z_i$ = (i文字目以降で初めて0以外の文字が出現する添え字)とする。

N = 8, S = “10023405”, a = 4のとき

$z_i$ = {0, 3, 3, 3, 4, 5, 7, 7} となる。このとき、長さaの部分文字列は$(N - z_i) \geq 4$となるiの通り数である。5通りになる。

## 3.　　部分文字列の値がK桁になる数を昇順ソートしm番目の値を求める。

**※ m = H - A - (0を除くK桁未満の値になる部分文字列の通り数)**

愚直に長さK部分文字列を作成しようとすると**長さKの文字列を(N-K)個**作る必要がありMLE,TLEしてしまう。

工夫が必要。
よく考えると部分文字列は先頭の添え字だけわかれば表現できる。

S = 12300450 6から3桁となる部分文字列を作りたい場合、

| 123, | 230, | 300, | 450, | 450, | 450, | 506 |
|---|---|---|---|---|---|---|
| | ⇩ | | | | | |
| **0,** | **1,** | **2,** | **5,** | **5,** | **5,** | **6** |

**次にこれを用いてソートすることを考える。**

# 添え字を用いたソート

123, 230, 300, 450, 450, 450, 506

**0, 1, 2, 5, 5, 5, 6**

**これは以下の比較関数が考えられる。**

```
bool compare(int i,int j){
        for(k = 0; k < K; k++) {
                if (S[i + k] < S[j + k]) return true;
                if (S[i + k] > S[j + k]) return false;
        }
        return false;
}
```

**この比較関数はO(K) なので 全体の計算量はO(K * N * logN)になりTLEしてしまう。**

**そこで比較関数の高速化を行う。**

# Rolling Hashと二分探索

比較関数をRolling Hashと二分探索を用いて高速化する。

文字列どうしの比較は左からみていき最初に異なった文字の大小関係をみることで比較可能である。
Rolling Hashはある文字列の任意の部分文字列のhash値を前計算( O(n) )しておくことで**O(1)**で求めることができるアルゴリズムです。
Rolling Hashと二分探索を用いて左から何文字一致してるかを求め、初めて異なる文字の大小関係を比較すればよい。これは**O(logK)**で計算できるので、全体の計算量が**O(N * logN * log K)**となり解ける。

“1234**5**67”
“1234**6**78”

**5** < **6**

“1234**5**67” < “1234**6**78”

※赤い区間の長さを二分探索で求める。

# 最後に

**やってることは結局、Suffix Arrayを求めてるだけなので、もちろんアリ本のSuffix Arrayのアルゴリズを用いてソートすることも可能です。**

# 結果

- First Accepted:
  - Onsite: rupc_latte_chikoku (59m)
  - Online: rupc_latte_chikoku (59m)

- Success Rate: 13.6%

# ジャッジ解

- haji　　C++　79行　1770byte
- beet　　C++　81行　1704byte
- dohatsu　C++　112行 1898byte
- uku　　C++　112行 2256byte