#include<bits/stdc++.h>

using namespace std;

#define MAX\_N 100010 // second approach: O(n log n)

char T[MAX\_N]; // the input string, up to 100K characters

int n; // the length of input string

int RA[MAX\_N], tempRA[MAX\_N]; // rank array and temporary rank array

int SA[MAX\_N], tempSA[MAX\_N]; // suffix array and temporary suffix array

int c[MAX\_N]; // for counting/radix sort

int Phi[MAX\_N], PLCP[MAX\_N],LCP[MAX\_N];

void countingSort(int k) { // O(n)

int i, sum, maxi = max(300, n); // up to 255 ASCII chars or length of n

memset(c, 0, sizeof c); // clear frequency table

for (i = 0; i < n; i++) // count the frequency of each integer rank

c[i + k < n ? RA[i + k] : 0]++;

for (i = sum = 0; i < maxi; i++) {

int t = c[i]; c[i] = sum; sum += t; }

for (i = 0; i < n; i++) // shuffle the suffix array if necessary

tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];

for (i = 0; i < n; i++) // update the suffix array SA

SA[i] = tempSA[i];

}

void constructSA() { // this version can go up to 100000 characters

int i, k, r;

for (i = 0; i < n; i++) RA[i] = T[i]; // initial rankings

for (i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}

for (k = 1; k < n; k <<= 1) { // repeat sorting process log n times

countingSort(k); // actually radix sort: sort based on the second item

countingSort(0); // then (stable) sort based on the first item

tempRA[SA[0]] = r = 0; // re-ranking; start from rank r = 0

for (i = 1; i < n; i++) // compare adjacent suffixes

tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r

(RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;

for (i = 0; i < n; i++) // update the rank array RA

RA[i] = tempRA[i];

if (RA[SA[n-1]] == n-1) break; // nice optimization trick

}

}

void buildLCP() {

int i, L;

Phi[SA[0]] = -1;

// default value

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

// compute Phi in O(n)

Phi[SA[i]] = SA[i - 1];

// remember which suffix is behind this suffix

for (i = L = 0; i < n; i++) {

// compute Permuted LCP in O(n)

if (Phi[i] == -1) { PLCP[i] = 0; continue; }

// special case

while (T[i + L] == T[Phi[i] + L]) L++;

// L increased max n times

PLCP[i] = L;

L = max(L - 1, 0);

// L decreased max n times

}

for (i = 0; i < n; i++)

// compute LCP in O(n)

LCP[i] = PLCP[SA[i]];

// put the permuted LCP to the correct position

}

int main() {

n = (int)strlen(gets(T)); // input T as per normal, without the ‘$’

T[n++] = '$'; // add terminating character

constructSA();

buildLCP();

//for(int i=0;i<n;i++) cout << LCP[i] << endl;

//for (int i = 0; i < n; i++) printf("%2d\t%s\n", SA[i], T + SA[i]);

memset(LCP, 0, sizeof(LCP));

}