/\*\*

\*

\* Solution to homework assignment 2

\* Introduction to programming course

\* Faculty of Mathematics and Informatics of Sofia University

\* Winter semester 2023/2024

\*

\* @author Petya Petrova

\* @idnumber 1MI0600311

\* @task 1

\*

\*/

#include <iostream>

using namespace std;

constexpr int MAX\_SIZE = 65;

void printSymbols(char firstSymbol, char secondSymbol) {

cout << firstSymbol << secondSymbol;

}

int symbolToIndex(char symbol) {

if (symbol >= '!' || symbol <= 'Z') {

return symbol - '!' + 33;

}

return -1;

}

bool checkBit(unsigned int n, unsigned ind)

{

unsigned int mask = 1;

mask <<= ind;

return (mask & n) == mask;

}

void makeBitOne(int& number, unsigned int ind){

unsigned int mask = 1;

mask <<= ind;

number = number | mask;

}

void fillingNumbers(const int indexOfSymbol, int& index ,int& newNumber) {

int indexOfNumber = 7;

while (indexOfNumber > 0) {

if (checkBit(indexOfSymbol, index)) {

makeBitOne(newNumber, indexOfNumber);

}

else {

makeBitOne(newNumber, indexOfNumber - 1);

}

indexOfNumber -= 2;

index--;

}

}

void codedSymbol(int indexOfSymbol) {

int indexOfFirstSymbol = 0;

int indexOfSecondSymbol = 0;

int index = 7;

fillingNumbers(indexOfSymbol, index, indexOfFirstSymbol);

fillingNumbers(indexOfSymbol, index, indexOfSecondSymbol);

char firstSymbol = indexOfFirstSymbol;

char secondSymbol = indexOfSecondSymbol;

printSymbols(firstSymbol, secondSymbol);

}

void codedStr(const char\* str) {

if (!str) {

return;

}

while (\*str) {

int indexOfSymbol = symbolToIndex(\*str);

codedSymbol(indexOfSymbol);

str++;

}

}

int main()

{

char str[MAX\_SIZE] = "abc";

codedStr(str);

}

/\*\*

\*

\* Solution to homework assignment 2

\* Introduction to programming course

\* Faculty of Mathematics and Informatics of Sofia University

\* Winter semester 2023/2024

\*

\* @author Petya Petrova

\* @idnumber 1MI0600311

\* @task 2

\*

\*/

#include <iostream>

using namespace std;

constexpr int MAX\_SIZE = 1000;

unsigned myStrlen(const char\* str){

if (!str)

return 0;

unsigned result = 0;

while (\*str) {

result++;

str++;

}

return result;

}

bool isNumber(char ch) {

return ch >= '0' && ch <= '9';

}

bool isLower(char ch){

return ch >= 'a' && ch <= 'z';

}

bool isUpper(char ch){

return ch >= 'A' && ch <= 'Z';

}

void myStrcpy(const char\* source, char\* dest){

if (!source || !dest)

return;

while (\*source){

\*dest = \*source;

dest++;

source++;

}

\*dest = '\0';

}

int myStrcmp(const char\* first, const char\* second) {

if (!first || !second)

return 0;

while ((\*first) && (\*second) && ((\*first) == (\*second))) {

first++;

second++;

}

return (\*first - \*second);

}

unsigned int powerOfTwo(unsigned int n){

if (n > 31)

return 0;

return 1 << n;

}

char\*\* createMatrix(int rows, int colls){

char\*\* matrix = new char\* [rows];

for (int i = 0; i < rows; i++){

matrix[i] = new char[colls];

}

return matrix;

}

void freeMatrix(char\*\* matrix, int rows){

for (int i = 0; i < rows; i++)

delete[] matrix[i];

delete[] matrix;

}

void readStr(const char\* pattern, char\*\* strWithAllpossibilities, int allCombinations) {

if (!pattern && !strWithAllpossibilities) {

return;

}

while (allCombinations != 0) {

myStrcpy(pattern, \*strWithAllpossibilities);

strWithAllpossibilities++;

allCombinations--;

}

}

int numberOfPercents(const char\* pattern) {

if (!pattern) {

return -1;

}

unsigned int count = 0;

while (\*pattern) {

if(\*pattern=='%'){

count++;

}

pattern++;

}

return count;

}

void fillSymbolInMask(char\*\* strWithAllpossibilities, int rows, unsigned int\* arrWithCountOfSymbols, unsigned int& step) {

unsigned parts = rows / step;// 8/4 = 2

unsigned indexOfColls;

for (int i = 0; i < rows; i++) {

indexOfColls = arrWithCountOfSymbols[i];

strWithAllpossibilities[i][indexOfColls] = '$';

arrWithCountOfSymbols[i]++;

}

for (int i = 0; i < parts; i++) {

for (int j = 0; j < step; j++) {

if (i % 2 != 0) {

unsigned indexRow = i \* step + j;

unsigned indexOfColls = arrWithCountOfSymbols[indexRow];

strWithAllpossibilities[indexRow][indexOfColls] = '$';

arrWithCountOfSymbols[indexRow]++;

}

}

}

step /= 2;

}

void fillStrWithZiros(char\* strWithAllpossibilities) {

if (!strWithAllpossibilities) {

return;

}

while (\*strWithAllpossibilities) {

\*strWithAllpossibilities = '\0';

strWithAllpossibilities++;

}

}

void revomeDuplicates(char\*\* strWithAllpossibilities, int rows) {

if (!strWithAllpossibilities) {

return;

}

for (int i = 0; i < rows - 1; i++) {

for(int j = i + 1; j < rows; j++){

if (myStrcmp(strWithAllpossibilities[i], strWithAllpossibilities[j]) == 0) {

fillStrWithZiros(strWithAllpossibilities[j]);

}

}

}

}

void isPartOfTheMask(const char\* text, const char\* strWithAllpossibilities, int& count) {

if (!text && !strWithAllpossibilities) {

return;

}

if (!(\*strWithAllpossibilities)) {

count++;

}

else if (\*strWithAllpossibilities == '\*') {

strWithAllpossibilities++;

text++;

isPartOfTheMask(text, strWithAllpossibilities, count);

}

else if (\*strWithAllpossibilities == '@') {

if (isLower(\*text) || isUpper(\*text)) {

strWithAllpossibilities++;

text++;

isPartOfTheMask(text, strWithAllpossibilities, count);

}

}

else if (\*strWithAllpossibilities == \*text) {

strWithAllpossibilities++;

text++;

isPartOfTheMask(text, strWithAllpossibilities, count);

}

else if (\*strWithAllpossibilities == '$') {

if (isNumber(\*text)) {

strWithAllpossibilities++;

text++;

isPartOfTheMask(text, strWithAllpossibilities, count);

}

}

}

void countOfAllTruePatterns(const char\* text, char\*\* strWithAllpossibilities, int rows, int& count) {

if (!strWithAllpossibilities) {

return;

}

for (int i = 0; i < rows; i++) {

if (strWithAllpossibilities[i][0] == '\0')continue;

isPartOfTheMask(text, strWithAllpossibilities[i], count);

}

}

void makeMasks(const char\* text, const char\* pattern, int numberOfPercents, int& count) {

if (!pattern) {

return;

}

int rows = powerOfTwo(numberOfPercents);

char\*\* strWithAllpossibilities = createMatrix(rows, MAX\_SIZE);

unsigned int step = rows / 2;

unsigned int\* arrWithCountOfSymbols = new unsigned int[rows] {0};

int index = 0;

int indexOfColls;

while (pattern[index] != '\0') {

if (pattern[index] != '%') {

for (int i = 0; i < rows; i++) {

indexOfColls = arrWithCountOfSymbols[i];

strWithAllpossibilities[i][indexOfColls] = pattern[index];

arrWithCountOfSymbols[i]++;

}

index++;

}

else {

fillSymbolInMask(strWithAllpossibilities, rows, arrWithCountOfSymbols, step);

numberOfPercents--;

index++;

}

}

revomeDuplicates(strWithAllpossibilities, rows);

countOfAllTruePatterns(text, strWithAllpossibilities, rows, count);

freeMatrix(strWithAllpossibilities, rows);

delete[]arrWithCountOfSymbols;

}

void isPartOfThePattern(const char\* text, const char\* pattern, int& count) {

if (!text && !pattern) {

return;

}

if (!(\*pattern)) {

count++;

}

else if (\*pattern == '\*') {

pattern++;

text++;

isPartOfThePattern(text, pattern, count);

}

else if (\*pattern == '@') {

if (isLower(\*text) || isUpper(\*text)) {

pattern++;

text++;

isPartOfThePattern(text, pattern, count);

}

}

else if (\*pattern == \*text) {

pattern++;

text++;

isPartOfThePattern(text, pattern, count);

}

}

unsigned int timesPatternSeen(const char\* text, const char\* pattern) {

if (!text && !pattern) {

return -1;

}

unsigned int lenOfText = myStrlen(text);

unsigned int lenOfPattern = myStrlen(pattern);

if (lenOfText < lenOfPattern)

return 0;

int count = 0;

int percents = numberOfPercents(pattern);

if (percents) {

makeMasks(text, pattern, percents, count);

}

else {

while (\*text) {

isPartOfThePattern(text, pattern, count);

text++;

}

}

return count;

}

int main()

{

char text[MAX\_SIZE];

char pattern[MAX\_SIZE];

cin.getline(text, MAX\_SIZE, '\n');

cin.getline(pattern, MAX\_SIZE, '\n');

cout << timesPatternSeen(text, pattern);

}

//Напишете функция, която приема като параметри два символни низа - съответно текст и шаблон и връща като резултат колко пъти шаблонът се среща в текста.

//Текстът е съставен само от цифри и латински букви.В шаблона може да има следните специални символи(които не може да се срещнат в текста) :

// \* - съответства на точно един произволен символ;

// % - съответства на една или две цифри(от десетичната бройна система);

// @ - съответства на една буква от латинската азбука

/\*\*

\*

\* Solution to homework assignment 2

\* Introduction to programming course

\* Faculty of Mathematics and Informatics of Sofia University

\* Winter semester 2023/2024

\*

\* @author Petya Petrova

\* @idnumber 1MI0600311

\* @task 3

\*

\*/

#include <iostream>

using namespace std;

constexpr int MAX\_SIZE = 64;

void swap(char\* str, int index1, int index2) {

if (!str) {

return;

}

char temp = str[index1];

str[index1] = str[index2];

str[index2] = temp;

}

unsigned myStrlen(const char\* str) {

if (!str)

return 0;

unsigned result = 0;

while (\*str) {

result++;

str++;

}

return result;

}

void makeItLexicographic(char\* str, size\_t size) {

if (!str) {

return;

}

unsigned int index = 0;

while (\*str) {

char smallestElement = \*str;

unsigned int indexOfSmallerElement = 0;

for (unsigned int i = 0; i < size; i++) {

if (str[i] < smallestElement) {

smallestElement = str[i];

indexOfSmallerElement = i;

}

}

swap(str, index, indexOfSmallerElement);

str++;

size--;

}

}

void myReverse(char\* str, int start, int end) {

while (start < end) {

swap(str, start++, end--);

}

}

bool nextPerm(char\* str, size\_t size) {

int current = size - 1;

while (current > 0 && str[current] <= str[current - 1])

current--;

current--;

if (current == -1)

return false;

int other = size - 1;

while (str[other] <= str[current])

other--;

swap(str, other, current);

myReverse(str, current + 1, size - 1);

return true;

}

int main()

{

char str[MAX\_SIZE];

cin >> str;

int n;

cin >> n;

size\_t size = myStrlen(str);

makeItLexicographic(str, size);

if (n > 0) {

cout << str << endl;

n--;

while (n > 0 && nextPerm(str, size)) {

cout << str << endl;

n--;

}

}

}

/\*\*

\*

\* Solution to homework assignment 2

\* Introduction to programming course

\* Faculty of Mathematics and Informatics of Sofia University

\* Winter semester 2023/2024

\*

\* @author Petya Petrova

\* @idnumber 1MI0600311

\* @task 4

\*

\*/

#include <iostream>

using namespace std;

void readMatrix(int\*\* matrix, int rows, unsigned int colls) {

for (unsigned int j = 1; j < colls; j++) {

int number;

cin >> number;

matrix[rows][j] = number;

}

}

int\*\* createMatrix(int rows){

int\*\* matrix = new int\* [rows];

for (int i = 0; i < rows; i++){

unsigned int colls;

cin >> colls;

matrix[i] = new int[colls + 1];

readMatrix(matrix, i, colls + 1);

matrix[i][0] = colls;

}

return matrix;

}

void freeMatrix(int\*\* matrix, unsigned int rows){

for (unsigned int i = 0; i < rows; i++)

delete[] matrix[i];

delete[] matrix;

}

void print(int\*\* matrix, unsigned int rows){

for (unsigned int i = 0; i < rows; i++){

unsigned int sizeOfRow = matrix[i][0];

for (unsigned int j = 1; j < sizeOfRow + 1; j++)

cout << matrix[i][j] << ' ';

cout << endl;

}

}

void increasingWay(int\*\* matrix, int rows, unsigned int& length, unsigned int& currentLength, int priviousDigit, int x, int y) {

priviousDigit = matrix[x][y];

for (int i = x - 1; i <= x + 1; i++) {

if (i >= 0 && i < rows){

for (int j = y - 1; j <= y + 1; j++) {

if (j >= 1 && j <= matrix[i][0]) {

if (matrix[i][j] > priviousDigit) {

currentLength++;

if (currentLength > length) {

length = currentLength;

}

increasingWay(matrix, rows, length, currentLength, priviousDigit, i, j);

}

}

}

}

}

currentLength--;

}

void longestIncreasingWay(int\*\* matrix, int rows, unsigned int& length) {

int priviousDigit;

for (int x = 0; x < rows ; x++){

for (int y = 1; y <= matrix[x][0]; y++){

priviousDigit = matrix[x][y];

unsigned int currentLength = 1;

increasingWay(matrix, rows, length, currentLength, priviousDigit, x, y);

}

}

}

int main(){

int rows;

cin >> rows;

int\*\* matrix = createMatrix(rows);

unsigned int lengthOfTheWay = 0;

longestIncreasingWay(matrix, rows, lengthOfTheWay);

cout << lengthOfTheWay;

freeMatrix(matrix, rows);

}