# **- Suffix Array**

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

//#define \_ ios\_base::sync\_with\_stdio(0);cin.tie(0);

#include <algorithm>

#include <bitset>

#include <cctype>

#include <cmath>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <fstream>

#include <iostream>

#include <list>

#include <map>

#include <queue>

#include <set>

#include <sstream>

#include <stack>

#include <string>

#include <vector>

#include <ctime>

using namespace std;

#define all(a,b,c) for(int I=0;I<b;I++) a[I] = c

#define BE(a) a.begin(),a.end()

#define chng(a,b) a^=b^=a^=b;

#define clr(y,z) memset(y,z,sizeof(y))

#define cntbit(mask) \_\_builtin\_popcount(mask)

#define CROSS(a,b,c,d) ((b.x-a.x)\*(d.y-c.y)-(d.x-c.x)\*(b.y-a.y))

#define EQ(a,b) (fabs(a-b)<ERR)

#define ERR 1e-5

#define FORE(i,a) for(typeof((a).begin())i=(a).begin();i!=(a).end();i++)

#define fr(i,a,b) for(i=a;i<=b;i++)

#define fread freopen("input.txt","r",stdin)

#define fri(a,b) for(int i=a;i<=b;i++)

#define frj(a,b) for(int j=a;j<=b;j++)

#define frk(a,b) for(int k=a;k<=b;k++)

#define frl(a,b) for(int l=a;l<=b;l++)

#define frin(a,b) for(int i=a;i>=b;i--)

#define frjn(a,b) for(int j=a;j>=b;j--)

#define frkn(a,b) for(int k=a;k>=b;k--)

#define frln(a,b) for(int l=a;l>=b;l--)

#define frn(i,a,b) for(i=a;i>=b;i--)

#define fwrite freopen("output.txt","w",stdout)

#define inf (1e9)

#define inpow(a,x,y) int i; a=x;fri(2,y) a\*=x

#define makeint(n,s) istringstream(s)>>n

#define mod 1000000007

#define ISS istringstream

#define ll long long

#define oo (1<<30)

#define OSS ostringstream

#define pb push\_back

#define PI 3.141592653589793

#define pi (2\*acos(0))

#define pp pop\_back

#define PRE 1e-8

#define print1(a) cout<<a<<endl

#define print2(a,b) cout<<a<<" "<<b<<endl

#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl

#define rev(a) reverse(BE(a));

#define round(i,a) i = ( a < 0 ) ? a - 0.5 : a + 0.5;

#define SI set<int>

#define SII set<int>::iterator

#define SIZE(s) ((int)s.size())

#define saja(a) sort(BE(a))

#define sqr(a) ((a)\*(a))

#define SZ 50005

#define SZ1 55

#define typing(j,b) typeof((b).begin()) j=(b).begin();

#define VD vector<double>

#define VI vector<int>

#define VLL vector<long long>

#define VS vector<string>

string str;

int revSA[SZ],SA[SZ];

int cnt[SZ] , nxt[SZ];

bool bh[SZ],b2h[SZ];

int lcp[SZ];

bool cmp(int i,int j)

{

return str[i]<str[j];

}

void sortFirstChar(int n)

{

/// sort for the first char ...

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

SA[i] = i;

sort(SA,SA+n ,cmp);

///indentify the bucket ........

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

{

bh[i] = (i==0 || str[SA[i]]!=str[SA[i-1]]);

b2h[i] = false;

}

return;

}

int CountBucket(int n)

{

int bucket = 0;

for(int i =0 ,j; i<n ; i=j)

{

j = i+1;

while(j<n && bh[j]==false) j++;

nxt[i] = j;

bucket++;

}

return bucket;

}

void SetRank(int n)

{

for(int i = 0 ; i<n ; i=nxt[i])

{

cnt[i] = 0;

for(int j =i ; j<nxt[i] ; j++)

{

revSA[SA[j]] = i;

}

}

return;

}

void findNewRank(int l,int r,int step)

{

for(int j = l ; j<r ; j++)

{

int pre = SA[j] - step;

if(pre>=0)

{

int head = revSA[pre];

revSA[pre] = head+cnt[head]++;

b2h[revSA[pre]] = true;

}

}

return;

}

void findNewBucket(int l,int r,int step)

{

for(int j = l ; j<r ; j++)

{

int pre = SA[j] - step;

if(pre>=0 && b2h[revSA[pre]])

{

for(int k = revSA[pre]+1 ; b2h[k] && !bh[k] ; k++) b2h[k] = false;

}

}

return;

}

void buildSA(int n)

{

///start sorting in logn step ...

sortFirstChar(n);

for(int h =1 ; h<n ; h<<=1)

{

if(CountBucket(n)==n) break;

SetRank(n);

/// cause n-h suffix must be sorted

b2h[revSA[n-h]] = true;

cnt[revSA[n-h]]++;

for(int i = 0 ; i<n ; i=nxt[i])

{

findNewRank(i,nxt[i] , h);

findNewBucket(i , nxt[i] , h);

}

///set the new sorted suffix array ...

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

{

SA[revSA[i]] = i;

bh[i] |= b2h[i]; ///new bucket ....

}

}

return;

}

void buildLCP(int n)

{

int len = 0;

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

revSA[SA[i]] = i;

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

{

int k = revSA[i];

if(k==0)

{

lcp[k] = 0;

continue;

}

int j = SA[k-1];

while(str[i+len]==str[j+len]) len++;

lcp[k] = len;

if(len) len--;

}

return;

}

void printSA()

{

for(int i=0;i<SIZE(str);i++) printf("%d ",SA[i]);

puts("");

for(int i=1;i<SIZE(str);i++) printf("%d ",lcp[i]);

puts("");

return ;

}

int main()

{

int n,p,q;

int tcase,cas=1;

scanf(" %d",&tcase);

while(tcase--)

{

cin>>str;

// cin>>p>>q;

buildSA(SIZE(str));

buildLCP(SIZE(str));

printSA();

// int sol = findSol(p,q,SIZE(str));

// printf("Case %d: %d\n",cas++,sol);

}

return 0;

}

# **- Trie tree using array**

/\*

TRIE tree:

>> The complexity of TRIE is: n.

>> It takes a huge amount of words and then it can search the word with efficient

complexity.

>> Input is: some words which you want to include in your dictionary then give

words to search.

.. Output is: For every searching word, either YES if the word exists or NO if

the word does not exist.

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <string>

#define sz 200005

#define clr(abc,z) memset(abc,z,sizeof(abc))

using namespace std;

// TRIE starts array

#define trie\_sz 26

struct node{

bool ending;

int next[trie\_sz];

node()

{

ending = false;

for (int i = 0; i<trie\_sz; i++) next[i] = 0;

}

}data[sz];

int counter=1;

void insert\_in\_trie(char \*str, int len)

{

int n = 0;

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

{

int now = str[i]-'a';

if(data[n].next[now]==0)

{

data[n].next[now] = counter++;

}

n=data[n].next[now];

}

data[n].ending=true;

}

bool search\_in\_trie(char \*str, int len)

{

int n = 0;

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

{

int now = str[i]-'a';

if(data[n].next[now]==0) return false;

n=data[n].next[now];

}

return data[n].ending;

}

bool delete\_from\_trie() // this is for memset only

{

clr(data,0);

}

// TRIE ends using array

int main()

{

int n;

char s[sz];

puts("How many words in dictionary?");

scanf("%d", &n);

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

{

scanf("%s", s);

insert\_in\_trie(s,strlen(s));

}

puts("How many searches from dictionary?");

scanf("%d", &n);

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

{

scanf("%s", s);

search\_in\_trie(s,strlen(s))==true?printf("YES\n"):printf("NO\n");

}

delete\_from\_trie();

return 0;

}

# **- Trie tree using pointer**

/\*

TRIE tree:

>> The complexity of TRIE is: n.

>> It takes a huge amount of words and then it can search the word with efficient

complexity.

>> Input is: some words which you want to include in your dictionary then give

words to search.

.. Output is: For every searching word, either YES if the word exists or NO if

the word does not exist.

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <string>

#define sz 2000005

using namespace std;

// TRIE starts using pointer

#define trie\_sz 26

struct node{

bool ending;

node \*next[trie\_sz];

node()

{

ending = false;

for (int i = 0; i<trie\_sz; i++) next[i] = NULL;

}

}\*root;

void insert\_in\_trie(char \*str, int len)

{

node \*cur = root;

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

{

int now = str[i]-'a';

if(cur->next[now]==NULL)

cur->next[now]=new node();

cur=cur->next[now];

}

cur->ending=true;

}

bool search\_in\_trie(char \*str, int len)

{

node \*cur=root;

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

{

int now = str[i]-'a';

if(cur->next[now]==NULL) return false;

cur=cur->next[now];

}

return cur->ending;

}

bool delete\_from\_trie(node \*cur) // this is for memset, it should be called by delete\_from\_trie(root) from main

{

for (int i = 0; i<trie\_sz; i++)

if(cur->next[i]!=NULL) delete\_from\_trie(cur->next[i]);

delete(cur);

}

// TRIE ends using pointer

int main()

{

root = new node();

int n;

char s[sz];

puts("How many words in dictionary?");

scanf("%d", &n);

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

{

scanf("%s", s);

insert\_in\_trie(s,strlen(s));

}

puts("How many searches from dictionary?");

scanf("%d", &n);

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

{

scanf("%s", s);

search\_in\_trie(s,strlen(s))==true?printf("YES\n"):printf("NO\n");

}

delete\_from\_trie(root);

return 0;

}

# **- KMP**

/\*

Knuth Morris Pattern (KMP):

>> The complexity of KMP is: n+m. where n is the length of the string and m

varies string to string and it can be at most n-1.

>> It takes a very large line of input and find the highest length of a string

which can be both suffix and prefix of that string.

>> Input is: a very large size (approx. 100000 character) of string.

>> It will return the length of largest possible string which can be both suffix

and prefix.

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <string>

#define sz 2000005

using namespace std;

//KMP starts

char line[sz];

int overlap[sz];

int kmp()

{

int len = strlen(line), v;

overlap[0] = 0;

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

{

v = overlap[i-1];

while(line[v]!=line[i] && v>0)

v = overlap[v-1];

if(line[v]==line[i]) overlap[i] = v+1;

else overlap[i] = 0;

}

return overlap[len-1];

}

//KMP ends

int main()

{

int t, n, m, cas=1;

cout<<"write the string: "<<endl;

cin>>line;

cout<<"Max Length of kmp = "<<kmp()<<endl;

return 0;

}

# **- KMP**

#include <bits/stdc++.h>

#define all(a,b,c) for(int I=0;I<b;I++) a[I] = c

#define BE(a) a.begin(),a.end()

#define chng(a,b) a^=b^=a^=b;

#define clr(y,z) memset(y,z,sizeof(y))

#define cntbit(mask) \_\_builtin\_popcount(mask)

#define CROSS(a,b,c,d) ((b.x-a.x)\*(d.y-c.y)-(d.x-c.x)\*(b.y-a.y))

#define EQ(a,b) (fabs(a-b)<ERR)

#define ERR 1e-5

#define FORE(i,a) for(typeof((a).begin())i=(a).begin();i!=(a).end();i++)

#define fr(i,a,b) for(i=a;i<=b;i++)

#define fread freopen("input.txt","r",stdin)

#define fri(a,b) for(int i=a;i<=b;i++)

#define frj(a,b) for(int j=a;j<=b;j++)

#define frk(a,b) for(int k=a;k<=b;k++)

#define frl(a,b) for(int l=a;l<=b;l++)

#define frin(a,b) for(int i=a;i>=b;i--)

#define frjn(a,b) for(int j=a;j>=b;j--)

#define frkn(a,b) for(int k=a;k>=b;k--)

#define frln(a,b) for(int l=a;l>=b;l--)

#define frn(i,a,b) for(i=a;i>=b;i--)

#define fwrite freopen("output.txt","w",stdout)

#define inf (1e9)

#define print1(a) cout<<a<<endl

#define print2(a,b) cout<<a<<" "<<b<<endl

#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl

#define rev(a) reverse(BE(a));

#define round(i,a) i = ( a < 0 ) ? a - 0.5 : a + 0.5;

#define SI set<int>

#define SII set<int>::iterator

#define SIZE(s) ((int)s.size())

#define saja(a) sort(BE(a))

#define sqr(a) ((a)\*(a))

#define SZ 50005

#define SZ1 55

#define typing(j,b) typeof((b).begin()) j=(b).begin();

#define VD vector<double>

#define VI vector<int>

#define VLL vector<long long>

#define VS vector<string>

VI adj[SZ];//only adj should be cleared

int col[SZ],low[SZ],tim[SZ],timer;

int group\_id[SZ],n,m,components;//components=number of components group\_id = which node belongs to which node

stack<int>S;

void scc(int u)

{

int i,v,tem;

col[u]=1;

low[u]=tim[u]=timer++;

S.push(u);

fr(i,0,SIZE(adj[u])-1)

{

v=adj[u][i];

if(col[v]==1)

low[u]=min(low[u],tim[v]);

else if(col[v]==0)

{

scc(v);

low[u]=min(low[u],low[v]);

}

}

//SCC checking...

if(low[u]==tim[u])

{

do

{

tem=S.top();S.pop();

group\_id[tem]=components;

col[tem]=2; //Completed...

}while(tem!=u);

components++;

}

}

int TarjanSCC() //some change may be required here

{

int i;

timer=components=0;

clr(col,0);

while(!S.empty()) S.pop();

fr(i,0,n-1) if(col[i]==0) scc(i);

return components;

}

VI nadj[SZ];//new adjcency list after SCC(DAG)

void MakeNewDAG\_Graph()

{

int i,j,u,v;

fr(i,0,components-1) nadj[i].clear();

fr(i,0,n-1)

{

fr(j,0,SIZE(adj[i])-1)

{

u=group\_id[i];

v=group\_id[adj[i][j]];

if(u!=v)

nadj[u].pb(v);

}

}

}

int main()

{

int i,j,t,cas=0,u,v,ans;

while(scanf("%d %d",&n,&m)==2)

{

fr(i,0,n-1) adj[i].clear();

fr(i,1,m)

{

scanf("%d %d",&u,&v);

adj[u].pb(v);

}

TarjanSCC();

printf("Total Groups: %d\n",components);

MakeNewDAG\_Graph();

printf("NewGraphLinkUsingSCC: this graph is directed acyclic graph:\n");

//this link between groups no.....

fr(i,0,components-1)

{

fr(j,0,SIZE(nadj[i])-1)

{

u=i;

v=nadj[i][j];

print2(u,v);

}

}

}

return 0;

}

/\*

Input:

8 14

0 1

1 2

1 5

1 4

2 6

2 3

3 2

3 7

4 5

5 6

7 6

7 3

6 5

4 0

Output:

Total Groups: 3

NewGraphLinkUsingSCC: this graph is directed acyclic graph:

1 0

1 0

2 1

2 0

2 0

Another Input:

6 6

0 1

1 2

2 1

3 4

4 5

5 4

Total Groups: 4

NewGraphLinkUsingSCC: this graph is directed acyclic graph:

1 0

3 2

\*/

# **- Mat Expo**

#include <bits/stdc++.h>

using namespace std;

#define print1(a) cout<<a<<endl

#define print2(a,b) cout<<a<<" "<<b<<endl

#define print3(a,b,c) cout<<a<<" "<<b<<" "<<c<<endl

#define oo (1<<30)

#define PI 3.141592653589793

#define pi 2\*acos(0)

#define ERR 1e-5

#define PRE 1e-8

#define SZ(a) (int)a.size()

#define LL long long

#define ISS istringstream

#define OSS ostringstream

#define VS vector<string>

#define VI vector<int>

#define VD vector<double>

#define VLL vector<long long>

#define SII set<int>::iterator

#define SI set<int>

#define mem(a,b) memset(a,b,sizeof(a))

#define fr(i,a,b) for(i=a;i<=b;i++)

#define frn(i,a,b) for(i=a;i>=b;i--)

#define fri(a,b) for(i=a;i<=b;i++)

#define frin(a,b) for(i=a;i>=b;i--)

#define frj(a,b) for(j=a;j<=b;j++)

#define frjn(a,b) for(j=a;j>=b;j--)

#define frk(a,b) for(k=a;k<=b;k++)

#define frkn(a,b) for(k=a;k>=b;k--)

#define frl(a,b) for(l=a;l<=b;l++)

#define frln(a,b) for(l=a;l>=b;l--)

#define EQ(a,b) (fabs(a-b)<ERR)

#define all(a,b,c) for(int I=0;I<b;I++) a[I] = c

#define CROSS(a,b,c,d) ((b.x-a.x)\*(d.y-c.y)-(d.x-c.x)\*(b.y-a.y))

#define sqr(a) ((a)\*(a))

#define FORE(i,a) for(typeof((a).begin())i=(a).begin();i!=(a).end();i++)

#define BE(a) a.begin(),a.end()

#define rev(a) reverse(BE(a));

#define sorta(a) sort(BE(a))

#define pb push\_back

#define popb pop\_back

#define round(i,a) i = ( a < 0 ) ? a - 0.5 : a + 0.5;

#define makeint(n,s) istringstream(s)>>n

#define countbit(mask) \_\_builtin\_popcount(musk)

#define mod 1000000007

struct matrix{

LL x[6][6];

};

matrix base,zero;

matrix matmult(matrix &a,matrix &b,int n)//m\*n and n\*r matrix //1 based

{

matrix ret;

int i,j,k;

fr(i,1,n)

fr(j,1,n)

{

ret.x[i][j]=0;

fr(k,1,n)

ret.x[i][j]=ret.x[i][j]+(a.x[i][k]\*b.x[k][j])%mod; //we can reduce complexity here

ret.x[i][j]%=mod;

}

return ret;

}

matrix bigmod(matrix b,long long p,int n) //have to pass n

{

matrix xx=zero;

int i;

fr(i,1,n) xx.x[i][i]=1;

matrix power=b;

while(p)

{

if((p&1)==1) xx=matmult(xx,power,n);

power=matmult(power,power,n);

p/=2;

}

return xx;

}

int main()

{

int t,cas=0;

cin>>t;

int k;

long long n;

while(t--)

{

cin>>n>>k;

printf("Case %d: ",++cas);

base.x[1][1]=1;

base.x[1][4]=2;

base.x[4][1]=3;

if(n<=1)

{

print1(n);

continue;

}

matrix ans=bigmod(base,n-1,k+2); //n-number of baseconditions+1

print1(ans.x[1][1]);

}

return 0;

}

# **- LCS**

/\*

LCS - Longest Common Subsequence:

>> The complexity of LCS is: n square.

>> It takes two strings and finds a new string which is the longest common subsequence

of the previous two strings. then output the numbering for two string in seperate

lines.

>> Input is: Two strings.

\*/

#include <bits/stdc++.h>

#define \_ ios\_base::sync\_with\_stdio(0);cin.tie(0);

#define sz 100

#define pb(a) push\_back(a)

#define pp pop\_back()

#define ll long long

#define fread freopen("input.txt","r",stdin)

#define fwrite freopen("output.txt","w",stdout)

#define inf (1<<30-1)

#define clr(abc,z) memset(abc,z,sizeof(abc))

#define PI acos(-1)

using namespace std;

int magnitude[sz][sz];

char direction[sz][sz];

void LCS(string X, string Y)

{

int m = X.size(), n =Y.size();

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

{

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

{

if(X[i-1]==Y[j-1])

{

magnitude[i][j] = magnitude[i-1][j-1]+1;

direction[i][j] = 'D'; //'D' denotes its came from diagonal

}

else if (magnitude[i-1][j]>=magnitude[i][j-1])

{

magnitude[i][j] = magnitude[i-1][j];

direction[i][j] = 'U';//'U' denotes its came from up

}

else

{

magnitude[i][j] = magnitude[i][j-1];

direction[i][j] = 'L';//'D' denotes its came from left

}

}

}

return;

}

int main()

{

\_

string a, b;

clr(magnitude,0);

stack<int>p,q;

stack<char>c;

int len;

cin>>a>>b;

int m = a.size(), n =b.size();

LCS(a,b);

len = magnitude[m][n];

while(m&&n)

{

if(direction[m][n]=='D')

{

p.push(m);

q.push(n);

c.push(a[m-1]);

m--,n--;

}

else if(direction[m][n]=='U') m--;

else n--;

}

cout<<"LCS : ";

while(!c.empty())

{

cout<<c.top();

c.pop();

}

cout<<endl<<"Positions in first string : ";

while(!p.empty())

{

cout<<p.top()<<" ";

p.pop();

}

cout<<endl<<"Positions in second string : ";

while(!q.empty())

{

cout<<q.top()<<" ";

q.pop();

}

cout<<endl;

return 0;

}

/\*

ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

GTCGTTCGGAATGCCGTTGCTCTGTAAA

\*/

# **- nCr**

/\*

Finding nCr:

>> The complexity of bubble sort is: unknown.

>> It works with two loops.

>> For a given n and r, we can find the value of nCr recursively using the formula

nCr = (n-1)Cr + (n-1)C(r-1).

>> input is: n and r.

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cmath>

#include <cstdlib>

#include <queue>

#include <stack>

#include <vector>

#include <algorithm>

#include <cctype>

#include <fstream>

#include <map>

#include <list>

#include<set>

#define chng(a,b) a^=b^=a^=b;

#define sz 100

#define pb(a) push\_back(a)

#define pp pop\_back()

#define ll long long

#define fread freopen("input.txt","r",stdin)

#define fwrite freopen("output.txt","w",stdout)

#define inf (1<<30-1)

#define clr(abc,z) memset(abc,z,sizeof(abc))

#define PI acos(-1)

using namespace std;

int dp[sz][sz];

int nCr(int n, int r)

{

if(r==1) return n;

if(n==r) return 1;

int &ret = dp[n][r];

if(ret!=-1) return ret;

ret = nCr(n-1,r)+nCr(n-1,r-1);

return ret;

}

int main()

{

int data[sz], n,r;

clr(dp,-1);

while(cin>>n>>r) cout<<"nCr = "<<nCr(n,r)<<endl;

return 0;

}