// 最长上升子序列数量，cdq

using namespace std;

const int INF = 0x7f7f7f7f;

const int MAXN = 5e4 + 111;

const int mod = 1000000007;

int a[MAXN];

int id[MAXN];

pii dp[MAXN];

void getMax(pii &a, pii b) {

if (a.first < b.first) a = b;

else if (a.first == b.first) {

if ((a.second += b.second) >= mod) {

a.second -= mod;

}

}

}

bool cmp(int x, int y) {

return a[x] == a[y] ? x > y : a[x] < a[y];

}

void cdq(int l, int r) {

if (l == r) return;

int mid = l + r >> 1;

cdq(l, mid);

for (int i = l; i <= r; ++i) id[i] = i;

sort(id + l, id + r + 1, cmp);

pii mx = pii(0, 0);

for (int i = l; i <= r; ++i) {

if (id[i] <= mid) getMax(mx, dp[id[i]]);

else {

pii temp = mx;

++temp.first;

getMax(dp[id[i]], temp);

}

}

cdq(mid + 1, r);

}

int main()

{

int n; scanf("%d", &n);

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

scanf("%d", &a[i]);

dp[i] = pii(1, 1);

}

cdq(0, n - 1);

pii ans = pii(0, 0);

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

getMax(ans, dp[i]);

}

printf("%d\n", ans.second);

return 0;

}

// 字符串匹配

using namespace std;

const int INF = 0x7f7f7f7f;

const int MAXN = 2e6 + 111;

char s[MAXN];

char st[70];

// 匹配串长度500，字符数256个

bitset<505> p[256], match;

int main()

{

int n, m;

while (gets(s)) {

scanf("%d", &n);

for (int i = 0; i < 256; ++i) p[i].reset();

match.reset();

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

scanf("%d%s", &m, st);

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

p[st[j]].set(i);

}

}

int len = strlen(s);

bool flag = false;

match.set(0);

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

match &= p[s[i]];

match <<= 1;

match.set(0);

// 只有一路连续过来才可能在n位置表现为1

if (match.test(n)) {

flag = true;

printf("%d\n", i - n + 2);

}

}

if (!flag) puts("NULL");

getchar();

}

return 0;

}

// 找不同构，点度数不大于3，直径为k的构造数

using namespace std;

const int INF = 0x7f7f7f7f;

const int MAXN = 1e5 + 111;

const int MOD = 1e9 + 7;

// dp:深度i不同构的二叉树个数

// sum:深度不超过i不同够的二叉树总数

ll dp[MAXN], sum[MAXN];

ll quick\_pow(ll base, ll p) {

ll ret = 1;

for (; p; base = base \* base % MOD, p >>= 1) {

if (p & 1) {

ret = ret \* base % MOD;

}

}

return ret;

}

int main()

{

int k;

dp[0] = 1;

sum[0] = 1;

dp[1] = 1;

sum[1] = 2;

ll inv2 = quick\_pow(2, MOD - 2);

ll inv6 = quick\_pow(6, MOD - 2);

for (int i = 2; i <= 100000; ++i) {

// dp[i]分两种情况：

// 1.一个分支为i - 1，另一个小于i - 1

// 2.两个分支都为i - 1

dp[i] = ((dp[i - 1] + (dp[i - 1] \* (dp[i - 1] - 1) % MOD \* inv2) % MOD)

+ dp[i - 1] \* sum[i - 2]) % MOD;

sum[i] = (sum[i - 1] + dp[i]) % MOD;

}

while (~scanf("%d", &k) && k) {

int cur = k / 2;

ll ans = 0;

if (k % 2 == 0) {

// 两边一样，分相同和不同两种情况

ans = (dp[cur] + (dp[cur] \* (dp[cur] - 1) % MOD \* inv2)) % MOD;

}

else {

// 第三边小于cur

ans = (dp[cur] + (dp[cur] \* (dp[cur] - 1) % MOD \* inv2 % MOD)) \* sum[cur - 1] % MOD;

// 三边长度相同时，三种情况：

// 两边相同

ans = (ans + dp[cur] \* (dp[cur] - 1) % MOD) % MOD;

// 三边相同

ans = (ans + dp[cur]) % MOD;

// 三边不同

ans = (ans + dp[cur] \* (dp[cur] - 1) % MOD \* (dp[cur] - 2) % MOD \* inv6 % MOD) % MOD;

}

printf("%lld\n", (ans + MOD) % MOD);

}

return 0;

}

// 生成的文本串含特定串的种数

#define ll long long

#define pr(x) cout << #x << " = " << (x) << '\n';

using namespace std;

const int INF = 0x7f7f7f7f;

const int MAXN = 111;

const int mod = 10007;

int dp[MAXN][MAXN \* 60];

struct Trie

{

int next[8000][30],fail[8000],end[8000];

bool isend[8000];

int root,L;

int newnode()

{

for(int i = 0;i < 26;i++)

next[L][i] = -1;

isend[L] = 0;

end[L++] = 0;

return L-1;

}

void init()

{

L = 0;

root = newnode();

}

void insert(char buf[])

{

int len = strlen(buf);

int now = root;

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

{

if(next[now][buf[i] - 'A'] == -1)

next[now][buf[i] - 'A'] = newnode();

now = next[now][buf[i] - 'A'];

}

isend[now] = 1;

end[now]++;

}

void build()

{

queue<int>Q;

fail[root] = root;

for(int i = 0;i < 26;i++)

if(next[root][i] == -1)

next[root][i] = root;

else

{

fail[next[root][i]] = root;

Q.push(next[root][i]);

}

while( !Q.empty() )

{

int now = Q.front();

if (isend[fail[now]]) isend[now] = 1;

Q.pop();

for(int i = 0;i < 26;i++) {

if(next[now][i] == -1) {

next[now][i] = next[fail[now]][i];

}

else

{

fail[next[now][i]]=next[fail[now]][i];

Q.push(next[now][i]);

}

}

}

}

int query(char buf[])

{

int len = strlen(buf);

int now = root;

int res = 0;

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

{

now = next[now][buf[i] - 'A'];

int temp = now;

while( temp != root )

{

res += end[temp];

temp = fail[temp];

}

}

return res;

}

void debug()

{

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

{

printf("id = %3d,fail = %3d,end = %3d,chi = [",i,fail[i],end[i]);

for(int j = 0;j < 26;j++)

printf("%2d",next[i][j]);

printf("]\n");

}

}

};

char buf[200];

Trie ac;

ll quickpow(ll x, ll p) {

ll ret = 1;

while (p) {

if (p & 1) ret \*= x, ret %= mod;

x \*= x;

x %= mod;

p >>= 1;

}

return ret;

}

int main()

{

int n, m, t; scanf("%d", &t);

while (t --) {

// 单词数 生成文本串长度

// 本题求，生成文本串中，至少含有一个单词的个数

scanf("%d%d", &n, &m);

ac.init();

while (n --) {

scanf("%s", buf);

ac.insert(buf);

}

ac.build();

memset(dp, 0, sizeof dp);

dp[0][0] = 1;

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

for (int j = 0; j < ac.L; ++j) {

for (int k = 0; k < 26; ++k) {

if (ac.isend[j]) continue;

int nx = ac.next[j][k];

if (ac.isend[nx]) continue;

dp[i + 1][nx] += dp[i][j];

dp[i + 1][nx] %= mod;

}

}

}

int ans = 0;

for (int i = 0; i < ac.L; ++i) {

if (!ac.isend[i]) {

ans += dp[m][i];

ans %= mod;

}

}

ans = ((quickpow(26, m) - ans) % mod + mod) % mod;

printf("%d\n", ans);

}

return 0;

}

import java.io.\*;

import java.util.\*;

import java.math.BigInteger;

public class Main {

public static void main(String[] args) {

//Scanner in = new Scanner(System.in);

// 读入string进行进制转换 36->10

// BigInteger ans = BigInteger.ZERO;

// String s = in.nextLine();

// for (int i = 0; i < s.length(); ++i) {

// char cur = s.charAt(i);

// ans = ans.multiply(BigInteger.valueOf(36));

// if ('0' <= cur && cur <= '9') {

// ans = ans.add(BigInteger.valueOf(cur - '0'));

// }

// else ans = ans.add(BigInteger.valueOf(cur - 'A' + 10));

// }

// System.out.println(ans);

// 求最接近平方根的整数

// BigInteger n = in.nextBigInteger();

// BigInteger l = BigInteger.ZERO, r = n;

// BigInteger ans = BigInteger.valueOf(-1);

// while (l.compareTo(r) <= 0) {

// BigInteger mid = (l.add(r)).divide(BigInteger.valueOf(2));

// BigInteger nxt = mid.multiply(mid);

// if (nxt.compareTo(n) <= 0) {

// ans = mid;

// l = mid.add(BigInteger.ONE);

// }

// else r = mid.subtract(BigInteger.ONE);

// }

// System.out.println(ans);

// 除法和余数

// 也可以用MOD得到余数，用divide得到结果

// BigInteger a = in.nextBigInteger(), b = in.nextBigInteger();

// BigInteger []ans = a.divideAndRemainder(b);

// System.out.println(ans[0]);

// System.out.println(ans[1]);

}

}