Sam's Numbers

locked

by nikasvanidze

Editorial by nikasvanidze

For 20% of the points

The solution is just brute force, i.e., enumerate all lists satisfying the conditions.

For 40% of the points

There is a **dynamic programming** solution that runs in $O(m^2 s)$.

Define $V_{x,y}$ to be the number of different ways of getting a list where the sum of numbers is x and the last number is y.

We have the following recurrence (for $x > y$):

$$V_{x,y} = \sum_{\substack{1 \le p \le m \\ |y-p| \le d}} V_{x-y,p}$$

The answer is then $V_{s,y}$.

Dynamic programming in more detail

Define $V_{x,y}$ to be the number of different ways of getting a list where the sum of numbers is x and the last number is y. Using $V_{x,y}$, we can now calculate the answer as $\sum_{y=1}^{m} V_{s,y}$. (i.e., we want the sum to be s, but the last number can be anything from 1 to m.) Thus, if we construct the matrix V with dimensions $s \times m$ and with entries $V_{x,y}$ ($1 \le x \le s$, $1 \le y \le m$), then we want to compute its sth row.

We have the following recurrence (for $x > y$):

$$V_{x,y} = \sum_{\substack{1 \le p \le m \\ |y-p| \le d}} V_{x-y,p}$$

The idea for this recurrence is as follows. Suppose we want to construct a list with sum x and whose last number is y. Suppose p is the number before the last number. Then $1 \le p \le m$ and $|y - p| \le d$ must be satisfied. And if we remove the last number, then we are left with a list whose last number is p and whose sum is $x - y$. But there are $V_{x-y,p}$ such lists. Hence, by summing across all valid values of p, we get the formula above.

For the base case, we say $V_{y,y} = 1$, and $V_{x,y} = 0$ if $x < y$.

Thus, to build the matrix V, we can compute each row one by one, and compute each entry using the recurrence above.

For 100% of the points

It can be seen that $V_{x,y}$ is calculated from only a few numbers. Specifically, we only need the last m rows of the matrix. Thus, we only need to remember m^2 values to compute each subsequent row. In this case, we can use *matrix multiplication* to calculate the answer quickly.

To use matrix multiplication, we should find a linear relationship between our DP steps, and then construct a matrix Q out of it. In other words, we should make a matrix Q that will do the following:

Statistics

Difficulty: Medium
Time Complexity: $O(m^6 \log s)$ Required
Knowledge: Dynamic Programming, Fast Matrix Multiplication
Publish Date: May 13 2017

$Q \times DP_i = DP_{i+1}$ where DP_i is the list of numbers that we calculated for step i.

In our case, to calculate the x^{th} row of our DP, we need the last m rows (so m^2 numbers in total). Let's number these values from 0 to $m^2 - 1$ and then set the entry $Q_{x,y}$ to 1 if the x^{th} number is considered in the calculation of the y^{th} number in next step, otherwise we set it to 0.

The answer will then be contained in $Q^s \times DP_0$.

Matrix exponentiation in more detail

Some dynamic programming algorithms can be sped up with a standard technique called *matrix exponentiation*. For an introduction to this technique, you may want to first solve the following problems:

- https://www.hackerrank.com/challenges/fibonacci-finding-easy
- https://www.hackerrank.com/challenges/mutual-recurrences
- https://www.hackerrank.com/challenges/circle-summation

The main takeaway here is that if our dynamic programming recurrence satisfies the following properties:

- each row only depends on the previous row,
- the recurrences are linear, and
- the recurrences are independent of the row number; in other words, each row is dependent on the previous row in exactly the same way, regardless of the row number,

then we can (possibly) speed up the dynamic programming with matrix exponentiation.

Note that our recurrence above satisfies the last two properties, but not the first, since each row actually depends on the *previous* m *rows*, not just the previous row. (If we inspect the recurrence, we find that to compute $V_{x,y}$ we need $V_{x-y,p}$, i.e., to compute the xth row, we need the $(x-y)$th row. But y can be up to m, so we need the previous m rows.)

But we can fix this by making our matrix larger. Suppose instead of V we build a similar matrix W. Whereas V has m columns, W will have m^2 columns, and each row of W will consist of m consecutive rows of V. Now, the matrix W satisfies all the properties above!

To see how matrix exponentiation can be applied, consider the following column vectors of length m^2 (which corresponds to two consecutive rows of matrix W):

$$DP_x = \begin{bmatrix} V_{x-1,1} \\ V_{x-1,2} \\ \vdots \\ V_{x-1,m} \\ V_{x-2,1} \\ V_{x-2,2} \\ \vdots \\ V_{x-2,m} \\ \vdots \\ \vdots \\ V_{x-m,1} \\ V_{x-m,2} \\ \vdots \\ V_{x-m,m} \end{bmatrix} \quad DP_{x+1} = \begin{bmatrix} V_{x,1} \\ V_{x,2} \\ \vdots \\ V_{x,m} \\ V_{x-1,1} \\ V_{x-1,2} \\ \vdots \\ V_{x-1,m} \\ \vdots \\ \vdots \\ V_{x-m+1,1} \\ V_{x-m+1,2} \\ \vdots \\ V_{x-m+1,m} \end{bmatrix}$$

Suppose we want to compute DP_{x+1}, given DP_x. Note that we can do so because of the properties of our recurrence above; DP_{x+1} is completely determined by the values inside DP_x. In fact, using our recurrence above, we can relate DP_{x+1} and DP_x in the following way:

$$Q \cdot DP_x = DP_{x+1}$$

where Q is a particular $m^2 \times m^2$ matrix. For example, suppose $d = 1$ and $m = 4$, then we have the following relationship:

$$\begin{bmatrix}
1 & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & 1 & 1 & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & 1 & 1 & 1 & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & 1 \\
1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & . & .
\end{bmatrix} \cdot \begin{bmatrix}
V_{x-1,1} \\ V_{x-1,2} \\ V_{x-1,3} \\ V_{x-1,4} \\ V_{x-2,1} \\ V_{x-2,2} \\ V_{x-2,3} \\ V_{x-2,4} \\ V_{x-3,1} \\ V_{x-3,2} \\ V_{x-3,3} \\ V_{x-3,4} \\ V_{x-4,1} \\ V_{x-4,2} \\ V_{x-4,3} \\ V_{x-4,4}
\end{bmatrix} = \begin{bmatrix}
V_{x,1} \\ V_{x,2} \\ V_{x,3} \\ V_{x,4} \\ V_{x-1,1} \\ V_{x-1,2} \\ V_{x-1,3} \\ V_{x-1,4} \\ V_{x-2,1} \\ V_{x-2,2} \\ V_{x-2,3} \\ V_{x-2,4} \\ V_{x-3,1} \\ V_{x-3,2} \\ V_{x-3,3} \\ V_{x-3,4}
\end{bmatrix}$$

(In the matrix in the left-hand side, a dot (.) represents a 0.) To verify this, try multiplying the left-hand side. Our matrix "Q", then, is the 16×16 matrix on the left-hand side. Its coefficients are just derived from our dynamic programming recurrence above:

$$V_{x,y} = \sum_{\substack{1 \le p \le m \\ |y-p| \le d}} V_{x-y,p}$$

Now that we have the matrix equation $Q \cdot DP_x = DP_{x+1}$, how does this help us? Note that we want to compute DP_s, so:

$$\begin{aligned}
DP_s &= Q \cdot DP_{s-1} \\
DP_s &= Q \cdot Q \cdot DP_{s-2} \\
DP_s &= Q^2 \cdot DP_{s-2} \\
DP_s &= Q^2 \cdot Q \cdot DP_{s-3} \\
DP_s &= Q^3 \cdot DP_{s-3} \\
DP_s &= Q^3 \cdot Q \cdot DP_{s-4} \\
DP_s &= Q^4 \cdot DP_{s-4} \\
\ldots &= \ldots \\
DP_s &= Q^s \cdot DP_0
\end{aligned}$$

This allows us to compute the sth row, DP_s, by computing the expression $Q^s \cdot DP_0$. Note that Q^s can be computed in $O((m^2)^3 \log s)$ time using *fast matrix exponentiation*! (It takes $O(n^3)$ time to multiply two $n \times n$ matrices, and $O(\log s)$ multiplications to raise something to the sth power.) Finally, DP_0 can be constructed out of our base cases. Hence, the overall algorithm is improved to just $O(m^6 \log s)$ time.

The matrix Q is only dependent on the input values d and m, and its entries can be computed in $O((m^2)^2) = O(m^4)$ time.

 Set by nikasvanidze

Problem Setter's code :

```
#include <bits/stdc++.h>
using namespace std;

const long long P = 1000000009;
long long n;
int m,d;

vector <int> vn,vm;
int M;
vector <vector <long long> > A;

void input() {
    cin >> n >> m >> d;
}

void mul(vector <vector <long long> > &A,
         vector <vector <long long> > &B,
         vector <vector <long long> > &C) {
```

```
     for (int i = 0; i < M; i++) {
        for (int j = 0; j < M; j++) {
            C[i][j] = 0;
            for (int k = 0; k < M; k++) {
                C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % P;
            }
        }
    }
}


void sol(){
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j <= m; j++) {
            vn.push_back(i);
            vm.push_back(j);
        }
    }

    M = vn.size();

    for (int i = 0; i < M; i++) {
        A.push_back({});
        for (int j = 0; j < M; j++) {
            A[i].push_back(0);
            if (vm[j] == 0) {
                if (vm[i]+1 == vn[j] && abs(vn[i]-vn[j]) <= d) {
                    A[i][j] = 1;
                }
            } else {
                if (vm[i] == vm[j] - 1 && vn[i] == vn[j]) {
                    A[i][j] = 1;
                }
            }
        }
    }

    vector <vector <long long> > B(M,vector<long long>(M,0));
    vector <vector <long long> > C = B;

    for (int i = 0; i < M; i++) C[i][i] = 1;

    while (n) {
        if (n & 1ll) {
            mul(C,A,B);
            C = B;
        }
        n >>= 1;
        mul(A,A,B);
        A = B;
    }

    long long ans = 0;
    for (int k = 1; k <= m; k++) {
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < M; j++) {
                if (vn[i] == k && vm[i] == 0 && vm[j] == k) {
                    ans = (ans + C[i][j]) % P;
                }
            }
        }
    }

    cout << ans << endl;
}

int main() {
    input();
    sol();
    return 0;
}
```

Tested by ma5termind

Problem Tester's code :

```
#include <stdio.h>
#include <cassert>
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
```

```
#include <memory.h>
#include <map>
#include <set>
#include <queue>
#include <list>
#include <sstream>
#include <cstring>
using namespace std ;

#define ft first
#define sd second
#define pb push_back
#define all(x) x.begin(),x.end()

#define ll long long int
#define vi vector<int>
#define vii vector<pair<int,int> >
#define pii pair<int,int>
#define plii pair<pair<ll, int>, int>
#define piii pair<pii, int>
#define viii vector<pair<pii, int> >
#define vl vector<ll>
#define vll vector<pair<ll,ll> >
#define pll pair<ll,ll>
#define pli pair<ll,int>
#define mp make_pair
#define ms(x, v) memset(x, v, sizeof x)

#define sc1(x) scanf("%d",&x)
#define sc2(x,y) scanf("%d%d",&x,&y)
#define sc3(x,y,z) scanf("%d%d%d",&x,&y,&z)

#define scll1(x) scanf("%lld",&x)
#define scll2(x,y) scanf("%lld%lld",&x,&y)
#define scll3(x,y,z) scanf("%lld%lld%lld",&x,&y,&z)

#define pr1(x) printf("%d\n",x)
#define pr2(x,y) printf("%d %d\n",x,y)
#define pr3(x,y,z) printf("%d %d %d\n",x,y,z)

#define prll1(x) printf("%lld\n",x)
#define prll2(x,y) printf("%lld %lld\n",x,y)
#define prll3(x,y,z) printf("%lld %lld %lld\n",x,y,z)

#define pr_vec(v) for(int i=0;i<v.size();i++) cout << v[i] << " " ;

#define f_in(st) freopen(st,"r",stdin)
#define f_out(st) freopen(st,"w",stdout)

#define fr(i, a, b) for(i=a; i<=b; i++)
#define fb(i, a, b) for(i=a; i>=b; i--)
#define ASST(x, l, r) assert( x <= r && x >= l )

const int mod = 1e9 + 9;

int ADD(int a, int b, int m = mod) {
    int s = a;
    s += b;
    if( s >= m )
      s -= m;
    return s;
}

int MUL(int a, int b, int m = mod) {
    return (1LL * a * b % m);
}

int power(int a, int b, int m = mod) {
    int res = 1;
    while( b ) {
        if( b & 1 ) {
            res = 1LL * res * a % m;
        }
        a = 1LL * a * a % m;
        b /= 2;
    }
    return res;
}

ll nC2(ll x) {
    return ( x * ( x - 1 ) / 2 );
}

const int MAXN = 110000;
int f[25][25], m, d, t[105][105], res[105][105];
ll n;
void mult(int a[][105], int b[][105], int m) {
```

```
    int tmp[105][105];
    for(int i=1; i<=m * m; i++) {
        for(int j=1; j<=m * m; j++) {
            tmp[i][j] = 0;
            for(int k=1; k<=m * m; k++) {
                tmp[i][j] += 1LL * a[i][k] * b[k][j] % mod;
                if( tmp[i][j] >= mod ) tmp[i][j] -= mod;
            }
        }
    }
    for(int i=1; i<=m*m; i++) {
        for(int j=1; j<=m*m; j++) {
            a[i][j] = tmp[i][j];
        }
    }
}
void solve(int z) {
        cin >> n >> m >> d;
        ASST(n, 1, 1000000000000000000LL);
        ASST(m, 1, 10);
        ASST(d, 0, m-1);
        ms(f, 0);
        ms(t, 0);
        ms(res, 0);
        for(int i=1; i<=m; i++) f[i][i] = 1;
        for(int i=1; i<=m; i++) {
            for(int j=1; j<=m; j++) {
                for(int k=1; k<=m; k++) {
                    if(abs(j - k) <= d) f[i+k][k] += f[i][j];
                    if(f[i+k][k] >= mod) f[i+k][k] -= mod;
                }
            }
        }

        int ans = 0;
        if(n <= m) {
            for(int i=1; i<=m; i++) {
                ans += f[n][i];
                if( ans >= mod ) ans -= mod;
            }
            cout << ans << "\n";
            return;
        }

        n -= m;
        for(int i=1; i<=m * m; i++) {
            res[i][i] = 1;
            int ind1 = ( i - 1 ) % m + 1;
            int val1 = m + 2 - (( i - 1 ) / m + 1);
            for(int j=1; j<=m * m; j++) {
                int ind2 = ( j - 1 ) % m + 1;
                int val2 = m + 1 - (( j - 1 ) / m + 1);
                if( val1 <= m && ind1 == ind2 && val1 == val2 ) {
                    t[i][j] = 1;
                }
                if( val1 > m && abs(ind1 - ind2) <= d && val1 - val2 == ind1) {
                    t[i][j] = 1;
                }
            }
        }

        while( n ) {
            if(n & 1) {
                mult(res, t, m);
            }
            mult(t, t, m);
            n /= 2;
        }

        for(int i=1; i<=m; i++) {
            for(int j=1; j<=m * m; j++) {
                int idx = ( j - 1 ) % m + 1;
                ans += 1LL * res[i][j] * f[m - (j - 1) / m][idx] % mod;
                if( ans >= mod ) ans -= mod;
            }
        }
        cout << ans << "\n";

}

int main() {
    int t = 1; // cin >> t;
    for(int i=1; i<=t; i++) solve(t);
    return 0;
}
```