Mathematics

Birthday Paradox:

If there are two people in a room,

- Probability that two will have same birthday = 1/365 = 0.00274 = 0.274%
- probability that two will have different birthdays = 1 - (probability that two have same birthday) = 1 - 0.00274 = 0.9973 = 99.73%

What is the minimum number of people that should be present in a room so that there’s 50% chance of two people having same birthday?

In a room of just __ people there’s a 50-50 chance of two people having the same birthday. In a room of __ there’s a 99.9% chance of two people matching.

```
#include<iostream>

using namespace std;

//Here prob defines the probability that two people will have different birthdays
#define days 365.0
#define prob (1.0 - 1/days)

```

```
int minPeople(){
    //res is the prob that x number of pairs will have di
fferent birthdays
    //cnt is the count of minimum pairs such that all of
them have different birthdays with probabbility <= 0.5
    double res = 1;
    int cnt = 0;
    while(res > 0.5){
        res *= prob;
        cnt++;
        //Showing the probability of 'cnt' number of pair
s having different birthdays.
        cout<<cnt<<" "<<res<<endl;
    }

    //Chance that we have a matching pair = (1-res)

    // if we have total pair count cnt then n*(n-1)/2 = c
nt where n is total number of people in the  room
    for(int i = 1;i<cnt;i++){
        //cout<<i<<endl;
        if(i*(i-1)/2 >= cnt)
            return i;
    }
    return 0;
}

int main(){
    cout<<minPeople()<<endl;
    return 0;
}
```

Ques. No of people for 100% probability?

Solving Linear Recurrence

The problem is generally asking you the n-th term of a linear recurrence. It is possible to solve with dynamic programming if n is small, problem arises when n isvery large.

Definition

First, let’s start with a definition. A linear recurrence relation is a function or a sequence such that each term is a linear combination of previous terms. Each term can be described as a function of the previous terms. A famous example is the Fibonacci sequence: f(i) = f(i-1) + f(i-2). Linear means that the previous terms in the definition are only multiplied by a constant (possibly zero) and nothing else. So, this sequence: f(i) = f(i-1) * f(i-2) is not a linear recurrence.

Problem

Given f, a function defined as a linear recurrence relation. Compute f(N). N may be very large.

How to Solve

Break the problem in four steps. Fibonacci sequence will be used as an example.

Step 1. Determine K, the number of terms on which f(i) depends.

More precisely, K is the minimum integer such that f(i) doesn’t depend on f(i-M), for all M > K. For Fibonacci sequence, because the relation is:

$f(i) = f(i-1) + f(i-2)$,

therefore, K = 2. In this way, be careful for missing terms though, for example, this sequence:

$f(i) = 2f(i-2) + f(i-4)$

has K = 4, because it can be rewritten explicitly as:

$f(i) = 0f(i-1) + 2f(i-2) + 0f(i-3) + 1f(i-4)$.

Step 2. Determine vector F1, the initial values.

If each term of a recurrence relation depends on K previous terms, then it must have the first K terms defined, otherwise the whole sequence is undefined. For Fibonacci sequence (K = 2), the well-known initial values are:

f(1) = 1

f(2) = 1

Note: We are indexing fibonacci from 1, f(0) = 0.

We define a column vector Fi as a K x 1 matrix whose first row is f(i), second row is f(i+1), and so on, until K-th row is f(i+K-1). The initial values of f are given in column vector F1 that has values f(1) through f(K):

$$F_1 = \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(K) \end{bmatrix}$$

Step 3. Determine T, the transformation matrix.

Construct a K x K matrix T, called transformation matrix, such that

$$TF_i = F_{i+1}$$

Suppose,

```
f(i) = C1f(i-1) + C2f(i-2) + C3f(i-3) + ..... + Ckf(i-k)
```

$$f(i) = \sum_{j=1}^{K} c_j f(i-j)$$

Putting i = k+1

```
f(k+1) = C1f(k) + C2f(k-1) + C3f(k-2) + ...... + Ckf(1)
```

Hence, the transformation matrix is:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c_K & c_{K-1} & c_{K-2} & c_{K-3} & \cdots & c_1 \end{bmatrix}$$

Example: for fibonacci

```
f(i) = 1f(i-1) + 1f(i-2)
```

C1 = 1, C2 = 1.

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Step 4. Determine FN.

We can obtain Fi for any i, by repeatedly multiplying T with F1. For example, to obtain F2,

$$F_2 = TF_1$$

To obtain F3,

$$F_3 = TF_2 = T^2F_1$$

And so on. In general,

$$F_N = T^{N-1}F_1$$

Therefore, the original problem is now (almost) solved: compute FN as above, and then we can obtain f(N): it is exactly the first row of FN. In case of our Fibonacci sequence, the N-th term in Fibonacci sequence is the first row of:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{N-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

To compute T^N-1 use exponentiation by squaring method that works in O(log N) time, with this recurrence:

- A ^ p = A, if p = 1,
- A ^ p = A * A ^ p-1, if p is odd
- A ^ p = X ^ 2, where X = A ^ p/2, otherwise.

Multiplying two matrices takes O(K ^ 3) time using standard method, so the overall time complexity to solve a linear recurrence is O(K^3 log N).

Variation

The recurrence relation may include a constant, i.e., the function is of the form:

$f(i) = \sum_{j=1}^{K} c_j f(i-j) + d$

In this variant, the vector F is enhanced to remember the value of d. It is of size (K+1) x 1 now:

$$F_i = \begin{bmatrix} f(i) \\ f(i+1) \\ \vdots \\ f(i+K-1) \\ d \end{bmatrix}$$

We now want to construct a matrix T, of size (K+1) x (K+1), such that:

$$[T] \begin{bmatrix} f(i) \\ f(i+1) \\ \vdots \\ f(i+K-1) \\ d \end{bmatrix} = \begin{bmatrix} f(i+1) \\ f(i+2) \\ \vdots \\ f(i+K) \\ d \end{bmatrix}$$

The transformation matrix will become:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_K & c_{K-1} & c_{K-2} & c_{K-3} & \cdots & c_1 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

For example, let f(i) = 2f(i-1) + 3f(i-2) + 5. Then, the matrix equation for this function is:

$$\begin{bmatrix}0 & 1 & 0\\ 3 & 2 & 1\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}f(i)\\ f(i+1)\\ 5\end{bmatrix}=\begin{bmatrix}f(i+1)\\ f(i+2)\\ 5\end{bmatrix}$$

Ques. Recursive Sequence

http://www.spoj.com/problems/SEQ/

```
#include<iostream>
#include<vector>

using namespace std;
typedef long long ll;
#define MOD 1000000000

ll k;
vector<ll> b;
vector<ll> a;
vector<ll> c;

//Multiplication of two matrix
//A --> k*K
//b --> k*k
//return C = A*B --> k*k
vector<vector<ll> > mul(vector<vector<ll> > A, vector<vector<ll> > B){
    vector<vector<ll> > C(k+1, vector<ll>(k+1));
    for(int i = 1;i<=k;i++){
        for(int j = 1;j<=k;j++){
            for(int x = 1;x<=k;x++)
                C[i][j] = (C[i][j] + A[i][x]*B[x][j]) % M
OD;
        }
    }
    return C;
}

//Finding T^n-1 using matrix exponentiation
vector<vector<ll> > pow(vector<vector<ll> > A, ll p){
    if(p == 1)
        return A;
    if(p & 1)
        return mul(A,pow(A,p-1));
    vector<vector<ll> > X = pow(A,p/2);
    return mul(X,X);
}

ll fib(ll n){

    //Base Case
    if(n == 0)
        return 0;
    //If n is less than k,
    //our answer lies in already created F1 vector
    if(n <= k)
```

```
        return b[n-1];

    //Step 2. Determine the F1 vector
    vector<ll> F1(k+1);
    for(int i = 1;i<k+1;i++)
        F1[i] = b[i-1];

    //create transformation matrix T
    vector<vector<ll> > T(k+1,vector<ll>(k+1));
    for(int i = 1;i<=k;i++){
        for(int j = 1;j<=k;j++){
            if(i<k){
                //Put 1 in all (i,j+1) positions
                if(j == i+1)
                    T[i][j] = 1;
                else T[i][j] = 0;
                continue;
            }
            //if i == k i.e. last row
            //For last row, put the coefficient vector
            T[i][j] = c[k - j];
        }
    }

    //Use matrix exponentiation to find T^n-1
    T = pow(T,n-1);

    //answer is the first row of T^n-1 * F1
    ll res = 0;
    for(int i = 1;i<=k;i++){
        res = (res + T[1][i]*F1[i]) % MOD;
    }

    return res % MOD;

}

int main(){
    ll t,n,a;
    scanf("%lld",&t);
    while(t--){
        scanf("%lld",&k);
        for(int i = 0;i<k;i++){
            scanf("%lld",&a);
            b.push_back(a);
        }
        for(int i = 0;i<k;i++){
            scanf("%lld",&a);
            c.push_back(a);
        }
        scanf("%lld",&n);
        printf("%lld\n",fib(n));
        b.clear();
        c.clear();
    }
    return 0;
```

```
}
```

Ques. Recursive Sequence (Version II)

A slightly harder version of the previous problem. This time, instead of determining an, we have to determine am + am+1 +...+an.

Define a series S, where

```
Si = a1 + a2 + a3 + ......+ ai
```

with **S0 = 0**

Now the problem reduces to determining **Sn - Sm-1**.
S itself is a linear recurrence

```
Si = Si-1 + ai
```

The final matrix equation of would look like this:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_k & c_{k-1} & c_{k-2} & \cdots & c_1 \end{bmatrix} \begin{bmatrix} S_{i-1} \\ a_i \\ a_{i+1} \\ a_{i+2} \\ \vdots \\ a_{i+k-1} \end{bmatrix} = \begin{bmatrix} S_i \\ a_{i+1} \\ a_{i+2} \\ a_{i+3} \\ \vdots \\ a_{i+k} \end{bmatrix}$$

with initial values:

$$\begin{bmatrix} S_0 = 0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \end{bmatrix}$$

To determine Sn and Sm-1,

```
Sn = first row of T^n * F1
Sm-1 = first row of T^m-1 * F1
```

Code

```
#include <iostream>
#include <vector>
using namespace std;
typedef long long ll;
#define matrix vector<vector<ll> >
ll m,n,MOD;
int k;
vector<ll> c;
vector<ll> b;


//Multiplication of two matrix
//A --> k+1 * k+1
//b --> k+1 * k+1
//return C = A*B --> k+1 * k+1
```

```
matrix mul(matrix A, matrix B){
    matrix C(k+2, vector<ll>(k+2));
    for(int i = 1;i<=k+1;i++){
        for(int j = 1;j<=k+1;j++){
            for(int x = 1;x<=k+1;x++)
                C[i][j] = (C[i][j] + A[i][x]*B[x][j]) % M
OD;
        }
    }
    return C;
}

//Finding T^n-1 using matrix exponentiation
matrix pow(matrix A, ll p){
    if(p == 1)
        return A;
    if(p & 1)
        return mul(A, pow(A,p-1));
    matrix X = pow(A,p/2);
    return mul(X,X);
}

ll fib(ll n){
     //Base Case
    if(!n)
        return 0;
    //If n is less than k,
    //our answer lies in already created F1 vector
    if(n <= k)
        return b[n-1];

    //Step 2. Determine the F1 vector
    vector<ll> F1(k+2);
    for(int i = 2;i<=k+1;i++){
        F1[i] = (b[i-2]);
    }
    matrix T(k+2, vector<ll>(k+2));
    for(int i = 1;i<=k+1;i++){
        for(int j = 1;j<=k+1;j++){

             //Put 1 at (1,1)
            if(i == 1 && j == 1){
                T[i][j] = 1;
                continue;
            }
            //if i == k + 1 i.e. last row
            //For last row, put the coefficient vector
            if(i == k + 1 && j > 1){
                T[i][j] = c[k- j + 1];
                continue;
            }

            //Put 1 in all (i,j+1) positions
            if(j == i+1){
                T[i][j] = 1;
                continue;
            }
            //else 0 in (i,j)
            T[i][j] = 0;
        }
    }

    //Use matrix exponentiation to find T^n
    T = pow(T,n);

     //answer is the first row of T^n * F1
    ll res = 0;
    for(int i = 1;i<=k+1;i++){
        res = (res + T[1][i] * F1[i]) % MOD;
    }
    return res % MOD;
}

int main(){
    int t;
    ll a;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&k);
        for(int i = 1;i<=k;i++){
            scanf("%lld",&a);
            b.push_back(a);
        }
        for(int i = 1;i<=k;i++){
            scanf("%lld",&a);
            c.push_back(a);
        }
        scanf("%lld %lld %lld",&m,&n,&MOD);
        ll tot = fib(n) - fib(m-1);
        //To handle negative case i.e. if
        //fib(n) - fib(m) becomes negative
        if(tot < 0) tot += MOD;
        printf("%lld\n",tot);
        c.clear();
        b.clear();
    }
    return 0;
}
```

Pigeonhole Principle

The pigeonhole principle is a fairly simple idea to grasp.Say that you have 7 pigeons and 6 pigeonholes. So, now you decide to start putting the pigeons one by one into each pigeonhole.

| p | | p | | p | | p | | p | | p |

So, now, you have one pigeon left, and you can put it into any of the pigeonholes.

|pp | | p | | p | | p | | p | | p |

The point is that when the ***number of pigeons > no. of pigeonholes, there will be at least one pigeonhole with atleast two pigeons.***

Ex: ***Hair counting problem***
If the amount of hair is expressed in terms of the number of hair strands, the average human head has about 150,000 hair strands. It is safe to assume, then, that no human head has more than 1,000,000 strands of hair. Since the population of Delhi is more than 1,000,000 at least two people in Delhi have the same amount of hair.

Ques. Divisible Subset

To find a non-empty subset of the given multiset with the sum of elements divisible by the size of original multiset.

Hint:

```
a % N = x
b % N = x
Then, (b-a) % N = (b % N - a % N) = (x - x) = 0
```

Let’s denote a1 + a2 + … + ak by **bk**. So, we obtain:

```
b0 = 0
b1 = a1
b2 = a1 + a2
.
```

```
.

.

.

bN = a1 + a2 + a3 + a4 +.......+ aN

so, aL + aL+1 + ....... + aR = bR - bL-1
```

Therefore, if there are two values with equal residues modulo N among b0, b1, …, bN then we take the first one for L-1 and the second one for R and the required subsegment is found.

There are **N+1** values of bi and **N** possible residues for N. So, according to the pigeonhole principle the required subsegment will always exist.

Code

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

using namespace std;

#define N (1e6+10)

vector<int> pos(N, -1);
vector<int> v;
vector<int> pre;

int main(){

    int t,n,x,l = 0,r = 0;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i = 0;i<n;i++){
            scanf("%d",&x);
            v.push_back(x % n);
        }

        //Pre vector denotes the prefix sum bi
        pre.push_back(0);
        for(int i = 0;i<n;i++)
            pre.push_back((pre[i] + v[i]) % n);

        //Position vector denotes the index of
        //each segment sum
        for(int i = 0;i<n+1;i++){

            //If this segement sum is encountered first t
ime
            if(pos[pre[i]] == -1){
                pos[pre[i]] = i;
            }

            //If this segment sum has already been encoun
tered
            //we have our answer.
            //L will be its previous index and R current
index
            else{
                //pos[pre[i]] will give L-1
                l = pos[pre[i]] + 1;
                r = i;
                break;
            }
        }

        //Number of elements in the segment
        //is (R-L+1)
        printf("%d\n",r - l + 1);

        //Print the index between l and r
        for(int i = l;i<=r;i++)
            printf("%d ",i);printf("\n");

        v.clear();
        pre.clear();
        for(int i = 0;i<=n;i++)pos[i] = -1;
    }
    return 0;
}
```

T = O(N)

Ques. The Gray-Similar Code

Given ‘N’ 64 bit integers such that any two succesive numbers differ at exactly ‘1’ bit. We have to find out 4 integers such that their XOR is equal to 0.

Hint: If we take XOR of any two succesive numbers, we will get a number with only 1 set bit and all others will be 0.

How to use Pigeonhole Principle

For N = 130, we have ‘65’ pairs i.e. {X1,X2}, {X3,X4}, {X5,X6} … {X129, X130}. But there exists only 64 possible position for the set bit ‘1’, by pigeonhole principle atleast two bits will be set at same positions say {Xi, Xi+1} and {Xj, Xj+1}. If we take xor of pair of these four numbers, we will get 0.

Thus, by pigeonhole principle for all n >= 130, we will always find 4 integers such that their XOR is 0. For n < 130, we can iterate for 3 values of A[i], A[j], A[k] and do a binary search to find 4th number which is (A[i] ^ A[j] ^ A[k]).

Code

```
#include<iostream>
#include<vector>
#include<stdio.h>

using namespace std;

```

```cpp
int main(){
    unsigned long long n,t,a;
    vector<unsigned long long> v;
    scanf("%llu",&n);
    for(int i = 0;i<n;i++){
        scanf("%llu",&a);
        if(n < 130)
            v.push_back(a);
    }
    if(n >= 130){
        printf("Yes");
        return 0;
    }
    for(int i = 0;i<n;i++){
        for(int j = i+1;j<n;j++){
            for (int k = j+1;k<n;k++){
               for(int l = k+1;l<n;l++){
                   if((v[i] ^ v[j] ^ v[k]) == v[l]){
                    printf("Yes\n");
                    return 0;
                   }
               }
            }
        }
    }
    printf("No");
    return 0;
}
```

Ques. Holi

Given a weighted tree, consider there are N people in N nodes. You have to rearrange these N people such that everyone is in a new node, and no node contains more than one person under the constraint that the distance travelled for each person must be maximized.There are N cities having N-1 highways connecting them.

http://www.spoj.com/problems/HOLI/

In order to maximize cost:

- all edges will be used to travel around.
- We need to maximize the use of every edge used.
  Once we know how many time each edge is used, we can calculate the answer.

How to apply Pigeonhole Principle?

Now for any edge Ei, we can partition the whole tree into two subtrees, if one side has n nodes, the other side will have N - n nodes. Also, note that, min(n, N-n) people will be crossing the edge from each side. Because if more people cross the edge, then by pigeon-hole principle in one side, we will get more people than available node which is not allowed in the problem statement. So, Ei will be used a total of 2 * min(n, N-n) times.

Thus the final result is:

```
cost = ∑ 2 * min(ni, N-ni) * wieght(Ei) | for each edge E
i
```

```
#include<iostream>
#include<vector>
#include<string.h>

using namespace std;

const int NN = 1e5 + 2;

vector<int> adj[NN];
vector<int> w[NN];

//Count the number of nodes rechable from i-th node
//Initially all count is 1
int cnt[100000+10];
//Specifies whether the node has been visited or not in d
fs
bool vis[100000+10];

int n;

//Dfs is used to calculate the count of
//number of nodes rechable from each node.
```

```
void dfs(int v){
    //Mark the node visited
    vis[v] = 1;

    //Iterate for all the adjacent nodes of the
    //current node 'v'
    for(int i = 0;i < adj[v].size();i++){
        //find the i-th adjacent node 'u'
        int u = adj[v][i];
        //If the node is already visited
        //do not visit again
        if(vis[u])
            continue;
        //If the node has not neen visited
        //visit this node
        dfs(u);
        //calculate the count of current node v
        //by adding count of all its adjacent nodes
        cnt[v] += cnt[u];
    }
    return;
}

//this function calculates the final answer
long long sol(int v){
    vis[v] = 1;
    long long res = 0;
    for(int i = 0;i<adj[v].size();i++){
        int u = adj[v][i];
        if(vis[u])
            continue;

        //find the minimum number of person that can be s
hifted
        //through ith edge of 'v' i.e. v--u with weight w
[v][i]
        res += min(cnt[u],n - cnt[u]) * 2ll * w[v][i] + s
ol(u);
    }
    return res;
}

int main(){
    int t,a,x,y,in = 1;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i = 0;i<=n;i++){
            adj[i].clear();
            w[i].clear();
            cnt[i] = 1;
            vis[i] = 0;
        }
        for(int i = 0;i<n-1;i++){
            scanf("%d %d %d",&x,&y,&a);
            --x;
            --y;
            adj[x].push_back(y);
            adj[y].push_back(x);
            w[x].push_back(a);
            w[y].push_back(a);
        }
        dfs(0);
        memset(vis,0,sizeof vis);
        printf("Case #%d: %lld\n",in,sol(0));
        ++in;
    }
    return 0;
}
```

The Inclusion-Exclusion Principle

- Every group of objects(or set) A can be associated with a quantity - denoted |A| - called the number of elements in A/ cardinality of A.
- If X = A ∪ B and A ∩ B = Ø, then |X| = |A| + |B|.

If A and B are not disjoint, we get the simplest form of the Inclusion-Exclusion Principle:

- |A ∪ B| = |A| + |B| - |A∩B|.
- |A ∪ B ∪ C| = |A| + |B| + |C| - |A∩B| - |B∩C| - |A∩C| + |A∩B∩C|

In the more general case where there are n different sets Ai, the formula for the Inclusion-Exclusion Principle becomes:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \sum_{1\le i<j<k\le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1} \left|\bigcap_{i=1}^{n} A_i\right|$$

Example: Prime Looking Numbers

How many number are there < 1000 such that they are Prime Looking i.e. **composite but not divisible by 2,3 or 5** (Ex- 49, 77, 91). Given that there are 168 primes upto 1000.

For any positive number N and m, the number of integers divisible by m which are less than N is **floor((N-1)/m)**.

```
divisible by 2 = floor(999/2) = 449
divisible by 3 = floor(999/3) = 333
divisible by 5 = floor(999/5) = 199

divisible by 2.3 = floor(999/6) = 166
divisible by 2.5 = floor(999/10) = 99
divisible by 3.5 = floor(999/15) = 66

divisible by 2.3.5 = floor(999/30) = 33
```

Numbers divisible by atleast 2,3 or 5 =

```
|2 ∪ 3 ∪ 5| = |2| + |3| + |5| - |2 ∩ 3| - |2 ∩  5|  - | 3
 ∩ 5| +  |2 ∩ 3 ∩ 5|
    = 499 + 333 + 199 - 166 - 99 - 66 + 33
    = 733
```

So, there exists 733 integers upto 1000 which have atleast 2,3 or 5 as divisor. This includes {2,3,5}.

Total number not having 2,3 or 5 as divisor = 999 - 733 = **266**.
Note that this set does not include 2,3 or 5.

Since there are 168 prime numbers upto 1000, but we have already excluded 2,3 and 5, number of prime looking numbers upto 1000 = 266 - 165 - 1(since 1 is neither prime nor composite) = **100**.

Ques. Sereja and LCM

We have to find the possible number of arrays: A[1], A[2], A[3],…,A[N] such that A[i] >=1 and A[i] <= M and ***LCM(A[1], A[2], …, A[N])*** is divisible by D.
We have to find the sum of the answers with D = L, L+1, …,R modulo

10^9 + 7.

A/Q we have to find the number of array whose LCM is a multiple of a given number(say **‘x’**).
Using negation **calculate the number of arrays whose LCM is not a multiple of x (say ‘y’)**.
Hence, ans = (possible array with m numbers) - y.

Note: The maximum value of the array elements can be 1000, the maximum number of distinct prime factors possible is **4** (2 * 3 * 5 * 7 * 11 > 1000).

Let the prime factors of **x** be **p,q,r,s**

```
x = (p^a) * (q^b) * (r^c) * (s^d)
p^a: P
q^b: Q
r^c: R
s^d: S
```

To calculate **y**:

- None of element of array have any prime factor that **x** has OR
- it may have some of it missing

So, calculate the number of arrays such that either **(P or its multiple are not present) OR (Q or its multiple are not present) OR (R or**

its multiple are not present) OR (S or its multiple are not present).

```
y = |not(P) U not(Q) U not(R) U not(S)|
```

Applying Principle of Inclusion-Exclusion Principle

```
A = power(m - m/P,n) + power(m - m/Q,n) + power(m - m/R,n
) + power(m - m/S,n);

B = power(m - m/P - m/Q + m/(P*Q), n) + power(m - m/Q - m
/R + m/(Q*R), n).....

C = power(m- m/P -m/Q - m/R + m/(P*Q) + m/(Q*R) + m/(P*R)
 - m/(P*Q*R),n)......

D = powmod(m - m/P - m/Q - m/R - m/S + m/(P*Q) + m/(Q*R)
+ m/(P*R) + m/(R*S) + m/(P*S) + m/(Q*S) - m/(P*Q*R)- m/(Q
*R*S) - m/(P*Q*S) - m/(P*R*S) + m/(P*Q*R*S),n);
```

Final Answer will be:

```
y = A - B + C - D
ans = m ^ n - y
    = m ^ n - A + B - C + D
```

Code

```cpp
#include<iostream>
#include<stdio.h>
#include<algorithm>

using namespace std;

#define MOD 1000000007LL
typedef long long ll;
int prime[1010],F[1010][5],cnt = 0;

//Fast power method
//Returns a^b mod M in log(b)
ll powmod(ll a,ll b){
    if(!a)
        return 0;
    if(b == 1)
        return a;
    ll res = 1;
    while(b){
        if(b & 1)
            res *= (a % MOD);
        res %= MOD;
        a *= a;
        if(a > MOD)
            a %= MOD;
        b >>= 1;
    }

    return res % MOD;
}

int main(){

    //Prime sieve
    //a[i]-->true indicates that i is prime
    bool a[1010];
    for(int i = 0;i<=1000;i++)
        a[i] = 1;
    for(int i = 2;i*i <= 1000;i++){
        if(a[i]){
            for(int j = 2*i;j<=1000;j+=i)
                a[j] = 0;
        }
    }

    // building the primes array which contains all the p
rime numbers upto 1000
    // in increasing order.
    for(int i = 2;i<=1000;i++){
        if(a[i]){
            prime[cnt++] = i;
        }
    }

    //cnt stores the count of total prime numbers upto 10
00
```

```
    for(int i = 0;i<=1000;i++){
        for(int j = 0;j<5;j++){
            F[i][j] = 1;
        }
    }

    //Finding all the prime factors with their power
    //upto 1000
    //F[i][j] = p indicates that j-th prime has a total
    //value of p in factorisation of i

    //Ex: 18 = 2*3*3
    //F[18][0] = 2
    //F[18][1] = 9
    for(int i = 1;i<=1000;i++){
        int num = i,in = 0;
        for(int f = 0;f<cnt;f++){
            int p = 1;
            while((num % prime[f]) == 0){
                p *= prime[f];
                num /= prime[f];
            }
            if(p > 1)
                F[i][in++] = p;
        }
    }
    int t,L,R,m,n;
    scanf("%d",&t);
    while(t--){
        scanf("%d %d %d %d",&n,&m,&L,&R);
        ll res = 0,A = 0,B = 0,C = 0,D = 0;
        int p,q,r,s;
        for(int i = L;i<=R;i++){
            p = F[i][0];
            q = F[i][1];
            r = F[i][2];
            s = F[i][3];
            //cout<<i<<" "<<p<<" "<<q<<" "<<r<<" "<<s<<en
dl;
            A = powmod(m - m/p,n) + powmod(m - m/q,n) + p
owmod(m - m/r,n) + powmod(m - m/s,n);
            A %= MOD;
            B = powmod(m - m/p - m/q + m/(p*q), n) + powm
od(m - m/q - m/r + m/(q*r), n) + powmod(m - m/r - m/s + m
/(r*s), n) + powmod(m - m/p - m/r + m/(p*r), n) + powmod(
m - m/p - m/s + m/(p*s), n) + powmod(m - m/q - m/s + m/(q
*s), n);
            B %= MOD;
            C = powmod(m- m/p -m/q -m/r +m/(p*q)+ m/(q*r)
+m/(p*r) - m/(p*q*r),n)+ powmod(m-m/p-m/q-m/s+m/(p*q)+m/(
p*s)+m/(q*s)-m/(p*q*s),n) + powmod(m-m/s-m/q-m/r+m/(s*q)+
m/(s*r)+m/(q*r)-m/(s*q*r),n)  +  powmod(m-m/p-m/r-m/s+m/(
p*r)+m/(p*s)+m/(r*s)-m/(p*r*s),n);
            C %= MOD;
            D = powmod(m-m/p-m/q-m/r-m/s+m/(p*q)+m/(q*r)+
m/(p*r)+m/(r*s)+m/(p*s)+m/(q*s) - m/(p*q*r)-m/(q*r*s) - m
/(p*q*s) - m/(p*r*s) + m/(p*q*r*s),n);
            D %= MOD;
            //cout<<i<<" "<<(powmod(m,n) - A + B - C + D
+ 2*MOD) % MOD<<endl;
            res += (powmod(m,n) - A + B - C + D + 2*MOD)
% MOD;
            res %= MOD;
        }
        printf("%lld\n",res);
    }
    return 0;
}
```

Mathematical Expectation

Mathematically, for a discrete variable X with probability function P(X), the expected value E(X) is given by **Σ xiP(xi)** the summation runs over all the distinct values xi that the variable can take.

For example, for a dice-throw experiment, the set of discrete outcomes is { 1,2,3,4,5,6} and each of this outcome has the same probability 1/6. Hence, the expected value of this experiment will be 1/6*(1+2+3+4+5+6) = 21/6 = 3.5.

For a continuous variable X with probability density function P(x) , the expected value E(X) is given by **∫ xP(x)dx**.

- Mathematical expectation is some sort of average value of your random variable.
- Expected value is not same as **“most probable value”** - rather, it need not even be one of the probable values. For example, in a dice-throw experiment, the expected value, viz 3.5 is not one of the possible outcomes at all.
- Rather the most probable value is the value with max probability.
- The rule of “linearity of of the expectation” says that E[ax1 + bx2] = aE[x1] + bE[x2].

Ques. What is the expected number of coin flips for getting a head?

Sol:

Ques. What is the expected number of coin flips for getting two consecutive heads?

Sol:

Ques. What is the expected number of coin flips for getting N consecutive heads?

Sol:

Ques. Candidates are appearing for interview one after other. Probability of each candidate getting selected is 0.16. What is the expected number of candidates that you will need to interview to make sure that you select somebody?

Sol:

Ques. The queen of a honey bee nest produces offsprings one-after-other till she produces a male offspring. The probability of produing a male offspring is p. What is the expected number of offsprings required to be produced to produce a male offspring?

Sol:

Thus, observe that in the problems where there are two events, where one event is desirable and other is undesirable, and the probability of desirable event is p, then the expected number of trials done to get the desirable event is 1/p.

Generalizing on the number of events - If there are K events, where one event is desirable and all others are undesirable, and the probability of desirable event is p, then also the expected number of trials done to get the desirable event is 1/p

Ques. What is the expected number of dice throws required to get a four?

Sol:

Note: It is not always necessary that expectation exists.

Ques. Candidates are appearing for interview one

after other. Probability of k-th candidate getting selected is 1/(k+1). What is the expected number of candidates that you will need to interview to make sure that you select somebody?

Sol:

Ques. Favorite Dice - Spoj

Problem Statement: What is the expected number of throws of N sided dice so that each number is rolled at least once?

http://www.spoj.com/submit/FAVDICE/id=18369020

Coupon Collector Problem

Problem Statement: A certain brand of cereal always distributes a coupon in every cereal box. The coupon chosen for each box is chosen randomly from a set of 'n' distinct coupons. A coupon collector wishes to collect all 'n' distinct coupons. What is the expected number of cereal boxes must the coupon collector buy so that the coupon collector collects all 'n' distinct coupons?

Let random variable Xi be the **number of boxes it takes for the coupon collector to collect the i-th new coupon after the i−1th coupon has already been collected**. (Note: this does NOT mean assign numbers to coupons and then collect the i-th coupon. Instead, this means that after Xi boxes, the coupon collector would have collected i distinct coupons, but with only Xi−1 boxes, the coupon collector would have only collected i−1distinct coupons.)

Clearly E(X1)=1, because the coupon collector starts off with no coupons. Now consider the i-th coupon.
After the i−1-th coupon has been collected, then there are n−(i−1) possible coupons that could be the new i-th coupon. Each trial of buying another cereal box, “success” is getting any of the n−(i−1) uncollected coupons, and “failure” is getting any of the already collected i−1 coupons. From this point of view, we see that

```
p = (n-(i-1)) / n
```

This is a bernaulli trial with prob of sucess p and failure (1-p). In bernaulli trial, the expected number of trials for i-th success is 1/p i.e. 1/(success of the i-th outcome).

```
E(Xi) = 1/p = n/n−(i−1).
```

To compute the number of cereal boxes X, required by the coupon collector to collect all n distinct coupons:

```
E(X) = E(X1 + X2 + X3 + X4 + ......... + Xn)
E(X) = E(X1) + E(X2) + ...... + E(Xn)
E(x) = n(1 + 1/2 + 1/3 + 1/4 + .... + 1/n)
```

Code

```
#include<iostream>
```

```
using namespace std;

int main(){
    int t,n;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        double ans = 0.0;
        for(int i = 1;i<=n;i++){
            ans += n/(i*1.0);
        }
        printf("%.9lf\n",ans);
    }
    return 0;
}
```