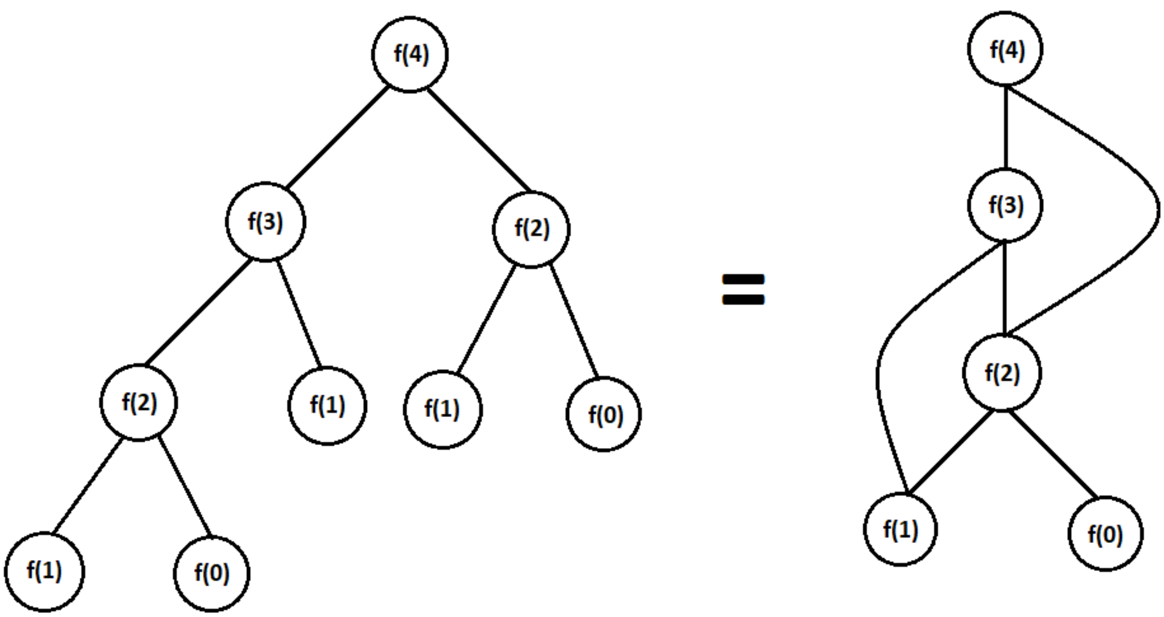
Introduction To DP

Factorial using recursion -

```
int fact(int n)
{
	if(n==0)
		return 1;
	return n*fact(n-1);
}
```

Fibonacci Numbers -

Recurrence - f(n) = f(n-1) + f(n-2)
Base cases - f(0) = 0, f(1)=1



Number of nodes = 2^n
Time Complexity = $O(2^n)$

Number of nodes = n
Time Complexity = O(n)

```
int fib[N];
int f(int n)
{
	if(n==0)
		return 0;
	if(n==1)
		return 1;
	if(fib[n]!=-1)
		return fib[n];
	fib[n] = f(n-1)+f(n-2);
	return fib[n];
}
int main()
{
	for(int i=0;i<N;i++)
		fib[i]=-1;
	cout << f(n) << ‘\n’
}
```

<u>Dynamic Programming</u>

States
Transitions
Base Case
Goal State

1. Tabulation or Iterative or Bottom-Up Approach

```
int main()
{
	int fib[n+1];
	fib[0]=0;
	fib[1]=1;
	for(int i=2;i<=n;i++)
		f[i]=f[i-1]+f[i-2];
```

```
        cout << f[n] << '\n';
}
```

2. Memoization or Recursive or Top-Down Approach

```
int fib[N];
int f(int n)
{
        if(n==0)
                return 0;
        if(n==1)
                return 1;
        if(fib[n]!=-1)
                return fib[n];
        fib[n] = f(n-1)+f(n-2);
        return fib[n];
}

int main()
{
        for(int i=0;i<N;i++)
                fib[i]=-1;
        cout << f(n) << '\n'
}
```

Types of DP Problems

1. Overlapping Subproblems
2. Optimal Substructure

Q) You have a staircase of N steps. In each turn, you can either climb 1 step or 2 steps. Find the number of ways to reach the top.

N=4
1+1+1+1
1+1+2
1+2+1
2+1+1
2+2
5 ways

N -> N-1 + 1
   -> N-2 + 2

States - s[i] = Number of ways to reach i
Transitions - s[i] = s[i-1] + s[i-2] (s[i] = s[i-a] + s[i-b])
Base Cases - s[1] = 1, s[2] = 2 (s[0] = 1)
Goal - s[N]

```
int main()
{
        int s[n+1];
        s[1] = 1;
        s[2] = 2;
        for(int i=3;i<=n;i++)
                s[i]=s[i-1]+s[i-2];
        cout << s[n] << '\n'
}
```

https://codeforces.com/group/8lnePmWc8m/contest/344525/problem/F
States - dp[i] = Maximum nuggets that can be collected till i-th cave
Transitions - dp[n] = max(dp[n-k],dp[n-k+1])+a[n]
Base Case - if(n<=0) return 0;
Goal - max(dp[1],dp[2],...,dp[n])

```
int nuggets(int n)
{
        if(n<=0)
                return 0;
        if(dp[n]!=-1)
                return dp[n];
        dp[n]=max(nuggets(n-k),nuggets(n-k+1))+a[n];
        return dp[n];
}

int main()
{
        for(int i=0;i<=n;i++)
                dp[i]=-1;
        int ans=0;
        for(int i=1;i<=n;i++)
                ans = max(ans,nuggets(i));
        cout << ans << '\n';
}
```

2D DP

Compute nCr modulo m
n<=1000

$$^{n}C_{r} = \frac{n!}{r!(n-r)!}$$

$$^{n}C_{r} = {}^{n-1}C_{r} + {}^{n-1}C_{r-1}$$

States = c[n][r] = Value of nCr mod m

Transitions - c[n][r] = (c[n-1][r] + c[n-1][r-1]) % m

Base Case - if(n<r) return 0; if(r==0) return 1;

Goal - c[n][r]

```
int main()
{
      int c[N][N];
      c[0][0]=1;
      for(int i=0;i<n;i++)
      {
            c[i][0]=1;
            for(int j=1;j<i;j++)
            {
                  c[i][j]=(c[i-1][j]+c[i-1][j-1])%m;
            }
            c[i][i]=1;
      }
      cout << c[n][r] << ‘\n’;
}
```

```
1 -> 1 1
2 -> 1 2 1
3 -> 1 3 3 1
```

Q. You are given an NxM grid. You can move only right and down. Find the number of ways to reach cell (N,M) from (1,1).

Example, if N=2, M=2

RD

DR

If N=3, M=3

RRDD

RDRD

RDDR

DRRD

DRDR

DDRR

{0,0} = 1		down ↓
	right ➡	i,j

States - dp[i][j] = Number of ways to reach cell (i,j) from (1,1)

Transitions - dp[i][j]=dp[i-1][j]+dp[i][j-1]

Base Case - dp[0][0]=1

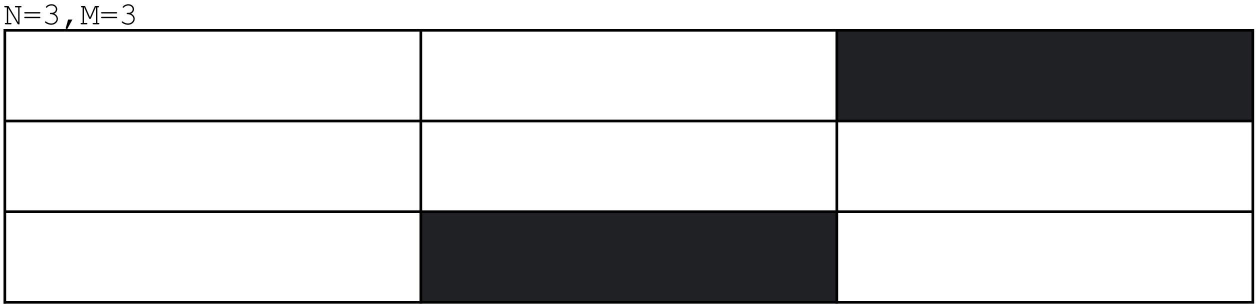
Goal - dp[n][m]

```
int main()
{
	int dp[n][m];
	dp[0][0]=1;
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<m;j++)
		{
```

```
                dp[i][j]=0;
                if(i>0)
                     dp[i][j]+=dp[i-1][j];
                if(j>0)
                     dp[i][j]+=dp[i][j-1];
           }
      }
      cout << dp[n][m] << '\n';
}
```

Q. You are given an NxM grid. You can move only right and down. Some cells are blocked while some are not blocked. You cannot visit blocked cells. Find the number of ways to reach cell (N,M) from (1,1).

N=3,M=3



RDRD

DRRD

States - dp[i][j] = Number of ways to reach cell (i,j) from (1,1)

Transitions - dp[i][j]=dp[i-1][j]+dp[i][j-1]

If blocked[i][j], dp[i][j]=0

Base Case - dp[0][0]=1

Goal - dp[n][m]

```
int main()
{
	int dp[n][m];
	dp[0][0]=1;
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<m;j++)
		{
			dp[i][j]=0;
			if(i>0)
				dp[i][j]+=dp[i-1][j];
			if(j>0)
				dp[i][j]+=dp[i][j-1];
			if(blocked[i][j])
				dp[i][j]=0;
		}
	}
	cout << dp[n][m] << '\n';
}
```

```
ai < 10^5
ai * aj = x^k k < 100
max distinct primes in any ai would be about 8 -9
ai = p1^a1 * p2^a2 *p3^a3 …
p1-> a1modk
p2 -> a2 modk
```

```
map<vector<pair<int,int>> ,int> mp;
n * 8* logn

for every aj:
    aj vector<pair<int,int>> b
    which is primes in aj along with its max power mod k
    vector<pair<int,int>> c;
    c = for every el in b : {el.first, (k - el.second)%k}
    ans += mp[c];
    mp[b]++;
```

x^k - (pi1^a1 pi2^a2 pi3^a3)^k = pi1^a1*k

k = 3

48 ->{{2, 4}, {3, 1}} ->{{2, 1}{3,1}}

2^(1 mod k) 2 ^(k-1 modk)

5 0, 1, 2, 3, 4

0 0 1-> 4 2-> 3

Problem M2

5

5 6 7 8 9 10 11 12 13 14 15 16 …

ans[j] += ans[i]

ans[i]

j = t*i

x = j , z = t

ans[x] += ans[i]

ans[j] += ans[i]

5 6 7 8 9 10 11 12 13 14 15 16 …

10/2 z = 2 x = 10

11/2 -> z = 2, x= 11

ans[10] += ans[5] and ans[11] += ans[5];

for floored division case, we assume that we have calculated ans[i]

we go through every multiple of i say j.

now we find what is the z for which this x (which is mainly j here) will give i which is

z = j/i;

Now all numbers from j upto j + z-1 will give floor division i when we use z.

Hence,

Let's say we have ans[i] calculated then,

```
for every multiple of i say j:
     z = j/i
     for all k from 0 to z-1:
          ans[j+k] += ans[i]         -----(P)
```

Now the operation (P) can be calculated more efficiently using prefix sum operation.

We simply maintain running pref[] array

and update it like this

for every multiple of i say j:

```
z = j/i;
pref[j] += ans[i];
pref[j+z] -= ans[i];        (Take care of the mod)
```

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


using namespace std;

#define int long long


void solve(){

    int n, mod;

    cin>>n>>mod;

    vector<int> ans(n+2);

    vector<int> pref(n+2);

    ans[0] = 1;

    int s = 0;

    for(int i=1;i< n;i++){

        pref[i] += pref[i-1];

        pref[i] %= mod;

        ans[i] +=  pref[i];

        ans[i] %= mod;

        if(i==1){

            ans[i] = 1;

        }
```

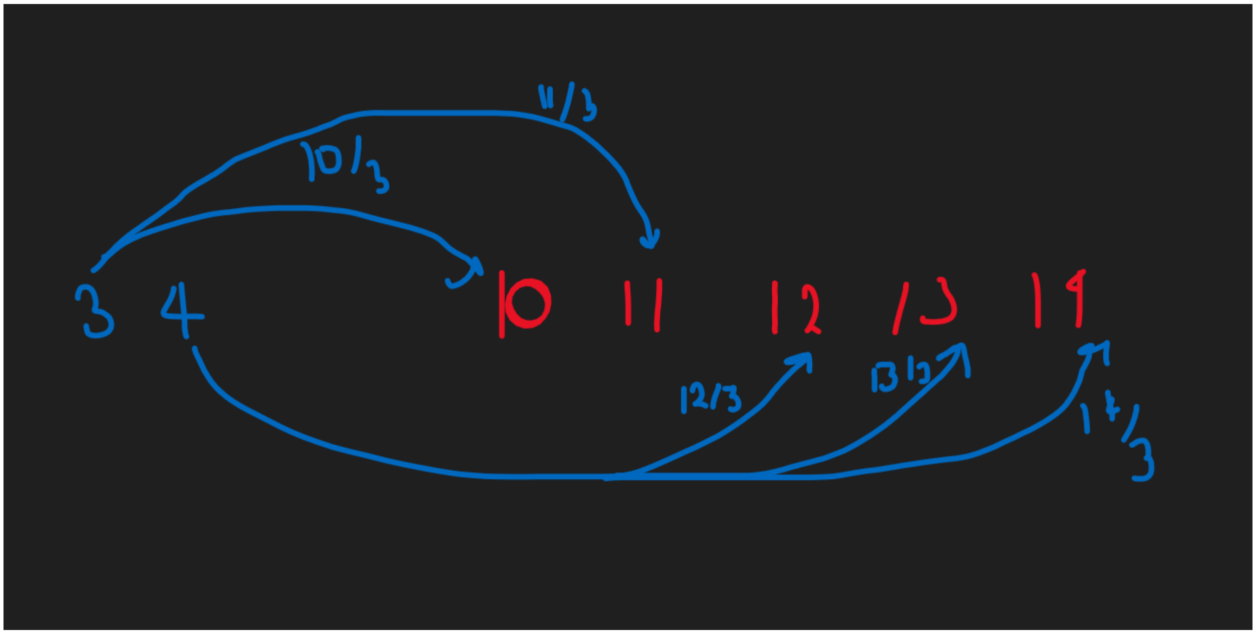
```
        ans[i] += s;
        ans[i] %= mod;
        s += ans[i];
        s %= mod;
        for(int j= i*2;j<= n;j+= i){
            int t = j/i;
            pref[j] += ans[i];
            pref[j] %= mod;
            pref[min(n+1, j+t)] += mod - ans[i];
            pref[min(n+1, j+t)] %= mod;
        }
        // x / t  = i
        // x = t * i = j
        // x + t-1 / t
        // t * i + 1/ t
        // t * i + 2 / t
        // i  t *i  t*i + t -1
    }
    pref[n] += pref[n-1];
    pref[n] %= mod;
    ans[n] += pref[n];
    ans[n] %= mod;
    ans[n] += s;
    ans[n] %= mod;
```

```
    cout<<ans[n]<<"\n";
}


int32_t main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);


    int t;
    t=1;
    int _=1;
    while(t--){
        // cout<<"Case #"<<_++<<": ";
        solve();
    }
    return 0;
}
```

5 ans[x] += ans[5]



Approach 2:

9 - 9/2, 9/3, 9/4, 9/5, 9/6, 9/7, 9/8, 9/9

- 4, 3, 2, 1, 1, 1, 1 ,1

8 - 4, 2, 2, 1, 1, 1, 1

12, 11

2 3 4 6 12

12 - 6, 4, 3, 2, 2, 1, 1, 1, 1, 1, 1

11 - 5, 3, 2, 2, 1, 1, 1, 1, 1, 1

n, n-1 - Factors of n

11 -> 12

```
2 -> 4, 6, 8, …

4 -> dp[4/2]-dp[3/2]

6 -> dp[6/2]-dp[5/2]
```