Stirling Numbers of both kinds

Stirling numbers are very important and interesting topic of combinatorics and dp. Different Programming problems are quite easily solved by these concepts. There are 2 kinds of stirling number :

1st Kind: The stirling number of the first kind refers to the number of ways to arrange **n** objects into **k** cycles. In this context we will denote **S_1(n,k)** as a stirling number of first kind.

2nd Kind: The stirling number of the second kind refers to the number of ways to arrange **n** objects into **k** nonempty subsets. In this context we will denote **S_2(n,k)** as a stirling number of second kind.

Formula:

1st Kind: $S_1(n,k)=(n-1)*S_1(n-1,k)+ S_1(n-1,k-1)$

2nd Kind: $S_2(n,k)=k*S_2(n-1,k)+ S_2(n-1,k-1)$

Base case: If n=k then $S_1(n,k)=1$ and $S_2(n,k)=1$

Else If k=0 $S_1(n,k)=0$ and $S_2(n,k)=0$

List of Problems solved:

1. *Codechef_January 2011 Challenge_Problem:- Count Relations*
2. *Topcoder SRM 391 KeysInBoxes*
3. *UVA 1118 - Binary Stirling Numbers*
4. *UVA 10648 chocolatebox*

Description of the problems:

Topcoder SRM 391 KeysInBoxes

Problem Statement

There are **N** boxes numbered from 1 to **N** and **N** keys numbered from 1 to **N**. The i-th key can only be used to open the i-th box. Now, we randomly put exactly one key into each of the boxes. We assume that all configurations of keys in boxes occur with the same probability. Then we lock all the boxes. You have **M** bombs, each of which can be used to open one locked box. Once you open a locked box, you can get the key in it and perhaps open another

locked box with that key. Your strategy is to select a box, open it with a bomb, take the key and open all the boxes you can and then repeat with another bomb.
Return the probability that you can get all the keys. The return value must be a string formatted as "A/B" (quotes for clarity), representing the probability as a fraction. A and B must both be positive integers with no leading zeroes, and the greatest common divisor of A and B must be 1.

Definition

Class:	KeysInBoxes
Method:	getAllKeys
Parameters:	int, int
Returns:	String
Method signature:	String getAllKeys(int N, int M)

(be sure your method is public)

Constraints

- **N** will be between 1 and 20, inclusive.
- **M** will be between 1 and **N**, inclusive.

Examples

0)

```
2
1
Returns: "1/2"
```

When box 1 contains key 2, you can get all the keys.

1)

```
2
2
Returns: "1/1"
```

When **N**=**M**, you can always get all the keys.

2)

```
3
1
Returns: "1/3"
```

There are 6 possible configurations of keys in boxes. Using 1 bomb, you can open all the boxes in 2 of them:
box 1 - key 2, box 2 - key 3, box 3 - key 1;

box 1 - key 3, box 2 - key 1, box 3 - key 2.

3)

```
3
2
Returns: "5/6"
```

Now, when you have 2 bombs, you are only unable to get all the keys in the following configuration: box 1 - key 1, box 2 - key 2, box 3 - key 3.

4)

```
4
2
Returns: "17/24"
```

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.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 SZ(s)      ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)

string convertInt(LL number)
{
   stringstream ss;//create a stringstream
   ss << number;//add number to the stream
   return ss.str();//return a string with the contents of the
//stream
}
LL dp[30][30],fact[30];
```

```
LL gcd(LL a,LL b)
{
   if(b==0) return a;
   return gcd(b,a%b);
}

LL dprec(int n,int m)
{
   LL &ret=dp[n][m];
   if(ret!=-1) return ret;
   ret=0;
   if(n==m) return ret=1;
   if(m==0) return ret;
   ret=(n-1)*dprec(n-1,m)+dprec(n-1,m-1);
   return ret;
}

class KeysInBoxes
{
public:
     string getAllKeys(int n, int m)
     {
        mem(dp,-1);
        int i;
        fact[0]=1;
        for(i=1;i<=20;i++)
         fact[i]=i*fact[i-1];
      LL ans=0;
      for(i=1;i<=m;i++)
         ans+=dprec(n,i);
      LL d=gcd(ans,fact[n]);
      string man=convertInt(ans/d);
      man+="/";
      man+=convertInt(fact[n]/d);
      return man;
     }
};
```

Analysis:

It is a straight forward Stirling number problem of first kind. If we blow a box with a bomb we will get a key and then use that key to open another box and so on until we get the key of the

box which we blow. Here we can get all the keys with only **1** bomb or we may have to use all the **m** bombs. So the number of ways we can get all the keys is equal to the number of cyclic partition of **n** object where partition number can be **1 to m**. For counting the probability we simply divide the number of total number of ways of the partition by the total number of ways to place the keys in the box.

UVA 10648 chocolatebox

Recently one of my friend Tarik became a member of the food committee of an ACM regional competition. He has been given m distinguishable boxes, he has to put n types of chocolates in the boxes. The probability that one chocolate is placed in a certain box is 1 / m. What is the probability that one or more boxes are empty? At first he thought it as an easy task. But soon he found that it was much harder. So, he falls into a great trouble and asked you to help him in this task.

Input

Each line of the input contains two integers **n** indicating total number of distinguishable types of chocolate and **m**indicating total number of distinguishable boxes **(m <= n < 100)**. A single line containing ***-1*** denotes the end.

Output

For each of the cases you should calculate the probability corrected to seven decimal places. The output format is shown below.

Sample Input

50 12
50 12
-1

Sample Output

Case 1: 0.1476651
Case 2: 0.1476651

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.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 SZ(s)      ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)
double dp[110][110][110];

double dprec(int n,int a,int b)
{
    double &ret=dp[n][a][b];
    if(ret>-1) return ret;
    ret=0;
    if(n==0)
{
        if(a==0) return ret=1.0;
        return ret;
    }

    if(a==0) return 0.0;
    ret=(1.0*a*dprec(n-1,a,b))/(a+b);
    ret+=(1.0*(b+1)*dprec(n-1,a-1,b+1))/(a+b);
    return ret;
}


int main()
{
   int n, m;
   int cas=0;
   mem(dp,-1);
   while(cin>>n&&n!=-1)
   {
       cin>>m;
       double ans=dprec(n,m,0);
       ans=1.0-ans;
```

```
        printf("Case %d:",++cas); //no space
        printf("%.7lf\n",ans);
    }
    return 0;
}
```

Analysis:

The solution is mainly derived from the Stirling number of second kind. Here we will first count the probability of reverse case. That is the probability that no boxes will be empty. For this we have used 3 dimensional dp. The first denotes the remaining chocolate, second denote the number of box already filled and the third denote the number of empty boxes. If we analyze we can easily show that the recursive relation is just a more precise form of the recursive relation of stirling number of second kind. The final answer is 1-the dp ans.

UVA 1118 - Binary Stirling Numbers

The Stirling number of the second kind S(n, m) stands for the number of ways to partition a set of n things into m nonempty subsets. For example, there are seven ways to split a four-element set into two parts:

$$\{1, 2, 3\} \cup \{4\}, \{1, 2, 4\} \cup \{3\}, \{1, 3, 4\} \cup \{2\}, \{2, 3, 4\} \cup \{1\}$$
$$\{1, 2\} \cup \{3, 4\}, \{1, 3\} \cup \{2, 4\}, \{1, 4\} \cup \{2, 3\}.$$

There is a recurrence which allows to compute S(n, m) for all m and n.

$$S(0, 0) = 1; S(n, 0) = 0 \text{ for } n > 0; S(0, m) = 0 \text{ for } m > 0;$$
$$S(n, m) = m\, S(n - 1, m) + S(n - 1, m - 1), \text{ for } n, m > 0.$$

Your task is much "easier". Given integers n and m satisfying $1 <= m <= n$, compute the parity of S(n, m), i.e. S(n, m) mod 2.

Example

S(4, 2) mod 2 = 1.

Task

Write a program which for each data set:

- reads two positive integers n and m,
- computes S(n, m) mod 2,
- writes the result.

Input

The first line of the input contains exactly one positive integer d equal to the number of data sets, $1 \le d \le 200$. The data sets follow.

Line i + 1 contains the i-th data set - exactly two integers n_i and m_i separated by a single space, $1 \le m_i \le n_i \le 10^9$.

Output

The output should consist of exactly d lines, one line for each data set. Line i, $1 \le i \le d$, should contain 0 or 1, the value of $S(n_i, m_i) \bmod 2$.

Sample Input

```
1
4 2
```

Sample Output

```
1
```

Mycode:

```
#include <sstream>
#include <cstdio>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.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 SZ(s)       ((int)s.size())
#define LL           long long
#define mem(a,b)    memset(a,b,sizeof(a))
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define cntbit(mask) __builtin_popcount(mask)

int main()
{
   int t;
   cin>>t;
   while(t--)
   {
       int n,k;
       cin>>n>>k;
```

```
        if(k==0)
         {
             if(n==0)
              print1("1");
             else print1("0");
             continue;
         }

         int ans=(n-k)&((k-1)/2);
         if(ans) ans=0;
         else ans=1;
         print1(ans);
    }
    return 0;
}
```

Analysis:

It's just parity check of Stirling number of second kind. If we analyze we quickly get a formula which is **`ans=(n-k)&((k-1)/2)==0.`** We can easily prove it by induction.

Codechef_January 2011 Challenge_Problem:- Count Relations

Let A be a set of the first **N** positive integers :A={1,2,3,4.....N}
Let B be the set containing all the subsets of A.
Professor Eric is a mathematician who defined two kind of relations R1 and R2 on set B. The relations are defined as follows:

R1={ (x,y) : x and y belong to B **and** x is not a subset of y **and** y is not a subset of x **and** the intersection of x and y is equal to empty set }

R2={ (x,y) : x and y belong to B **and** x is not a subset of y **and** y is not a subset of x **and** the intersection of x and y is not equal to empty set }
Now given the number N,Professor Eric wants to know how many relations of kind R1 and R2 exists.Help him.

NOTE : (x,y) is the same as (y,x) ,i.e the pairs are **unordered**.

Input format:

The first line contains the number of test cases T.Each of the test case is denoted by a single integer N.

Output format:

Output T lines, one for each test case,containing two integers denoting the number of relations of kind R1 and R2 respectively, modulo 100000007.

Example

Sample Input:
3
1
2
3
Sample Output:
0 0
1 0
6 3
Constraints:
1 <= T <= 1000
1 <= N <= 10^18
Explanation:
Let A={1,2}
Then B={Phi,{1},{2},{1,2}}
Phi=Empty Set
So R1=Either {({1},{2})} or {({2},{1})}
and R2=No relation exists
So,there is 1 relation of kind R1 and 0 relation of kind R2.

Mycode:

```
#include <sstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <string.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 SZ(s)       ((int)s.size())
#define LL           long long
#define fr(i,a,b)   for(i=a;i<=b;i++)
#define mod          100000007

long long bigmod(long long b,long long p,long long m)
{   if(b==0) return 0;
```

```
    long long x,power;
    x=1;
    power=b%m;
    while(p)
    {
       if(p%2==1)
            x=(x*power)%m;
        power=(power*power)%m;
        p=p/2;
    }
    return x;
}

int main()
{
   int t;
   cin>>t;
   while(t--)
   {
       LL n;
       cin>>n;
       if(n<=3)
       {
          if(n==1)
            print2(0,0);
          else if(n==2)
            print2(1,0);
          else print2(6,3);
          continue;
       }
       LL ans1=0,ans2=0;
       ans1=bigmod(2,n-1,mod);
       ans1=(ans1+mod-1)%mod;
       LL make=(((bigmod(3,n,mod)+mod+3-
(3*bigmod(2,n,mod))%mod)%mod)*bigmod(2,mod-2,mod))%mod;
       ans1=(ans1+make)%mod;
       ans2=make;
       ans2=(ans2+((12*bigmod(4,n-4,mod))%mod))%mod;
       ans2=(ans2+((27*6*bigmod(4,n-4,mod))%mod))%mod;
       ans2=(ans2+((6*bigmod(2,n-1,mod))%mod))%mod;
       ans2=(ans2+((2*bigmod(4,n-4,mod))%mod))%mod;
       ans2=(ans2+mod-2)%mod;
       ans2=(ans2+mod-((6*bigmod(3,n-1,mod))%mod))%mod;
       ans2=(ans2+mod-((6*bigmod(2,2*n-5,mod))%mod))%mod;
       print2(ans1,ans2);
   }
   return 0;
}
```

Analysis:

The key concept behind this problem is to make partition of numbers in some groups. For the first part **R1** we have to first group n numbers into 2 and 3 nonempty sets. And for calculating **R2** we have to group n numbers into 3 and 4 nonempty sets. The following table describes the relationship.

	SET 1	**SET 2**	**SET 3**	**SET 4**
R1	The set X	The set Y	The set do not containing the elements of X and Y we can choose them in 3 ways from a given partition	Not needed
R2	The set X-Y	The set Y-X	The set do not containing the elements of X and Y we can choose them in 4 ways from a given partition	The set of the X **INTERSECT** Y. After selecting set 3 there are remaining 3 ways to choose from a given partition.

From the above table we can easily conclude the following relation
$N(R1)= S_2(N,2)+3*S_2(N,3)$
$N(R2)= 3*S_2(N,3)+12*S_2(N,4)$

The set 3 and 4 can be empty for R1 and R2 respectively so we have to separately count $S_2(N,2)$ and $S_2(N,3)$ for R1 and R2 respectively.

Now there is only one problem. The constraints is huge (**$N<=10^{18}$**), So a straight forward solution will give TLE for sure. So we have to find a closed form for every **K** and then use bigmod for getting the ans. So the complexity is **O(lgn)** and we don't need extra memory for that. The closed form is given below.

$S_2(N,2)= 2^{N-1}-1$
$S_2(N,3)=(3^N-3*2^N+3)/6$
$S_2(N,4)=4^{N-4}+(3^34^{N-4}-3^{N-1}-2^{2N-5}+2^{N-1}+(4^{N-4}-1)/3)/2$

These relations can be proved by algebraic equations and geometric progressions.