[伯努利数与自然数幂和](http://blog.csdn.net/acdreamers/article/details/38929067)

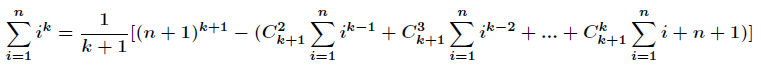
**分析：**其实求自然数的幂和方法有很多种，先来看看普通的递推求法，由于



     那么对于所有的累加得到



  进一步得到

  可以看出这是一个递推式，如果我们记



     那么得到如下递归式



     递归出口是



     为了提高效率，在递归的时候需要记忆化。由于要用到大数，用**Java**实现。 **代码：**

**import** java.math.\*;

1. **import** java.util.\*;
3. **public** **class** Main {
5. **public** **static** **final** **int** N = 105;
6. **public** **static** **final** BigInteger FLAG = (BigInteger.ZERO).subtract(BigInteger.ONE);
7. **public** **static** BigInteger[][] C = **new** BigInteger[N][N];
8. **public** **static** BigInteger[] ans = **new** BigInteger[N];
10. **public** **static** **void** Init(){
11. **for**(**int** i=0; i<N; i++){
12. C[i][0] = C[i][i] = BigInteger.ONE;
13. **if**(i == 0) **continue**;
14. **for**(**int** j=1; j<i; j++)
15. C[i][j] = C[i-1][j].add(C[i-1][j-1]);
16. }
17. }
19. **public** **static** BigInteger Solve(BigInteger n, **int** k){
20. **if**(ans[k].compareTo(FLAG) != 0){
21. **return** ans[k];
22. }
23. **if**(k == 1){
24. ans[k] = ((n.add(BigInteger.ONE)).multiply(n)).divide(BigInteger.valueOf(2));
25. **return** ans[k];
26. }
27. BigInteger tmp = BigInteger.ONE;
28. **for**(**int** i=0; i<k+1; i++){
29. tmp = tmp.multiply(n.add(BigInteger.ONE));
30. }
31. tmp = tmp.subtract(n.add(BigInteger.ONE));
32. BigInteger sum = BigInteger.ZERO;
33. **for**(**int** i=1; i<k; i++){
34. BigInteger t = C[k+1][i+1].multiply(Solve(n, k-i));
35. sum = sum.add(t);
36. }
37. ans[k] = (tmp.subtract(sum)).divide(BigInteger.valueOf(k+1));
38. **return** ans[k];
39. }
41. **public** **static** **void** main(String[] args){
42. Init();
43. Scanner cin = **new** Scanner(System.in);
44. **while**(cin.hasNext()){
45. BigInteger n = cin.nextBigInteger();
46. **int** k = cin.nextInt();
47. **for**(**int** i=0; i<N; i++){
48. ans[i] = FLAG;
49. }
50. System.out.println(Solve(n, k));
51. }
52. }
53. }

**分析：**本题题意就是求**自然数的幂和**，但是它的**case**比较多。对于求幂和本身就需要的时间复杂度，如果继

     续用上述方法来求自然数的幂和，**5000**个**case**会**TLE**，接下来介绍另一个求自然数幂和的方法，它是基于伯努利数的，公式描述如下



     可以看出只要我们预处理出每一项，就可以在线性时间内求得自然数的幂和。前面的倒数可以用递推法求逆元

     预处理，组合数也可以预处理，也可以先预处理，现在关键是如何预处理**伯努利数**。  **伯努利数**满足条件，且有



     那么继续得到



     这就是伯努利数的递推式，逆元部分同样可以预处理。

**代码：**

1. LL C[N][N];
2. LL B[N],Inv[N];
3. LL Tmp[N];
4. LL n;
6. **void** Init()
7. {
8. //预处理组合数
9. **for**(**int** i=0; i<N; i++)
10. {
11. C[i][0] = C[i][i] = 1;
12. **if**(i == 0) **continue**;
13. **for**(**int** j=1; j<i; j++)
14. C[i][j] = (C[i-1][j] % MOD + C[i-1][j-1] % MOD) % MOD;
15. }
16. //预处理逆元
17. Inv[1] = 1;
18. **for**(**int** i=2; i<N; i++)
19. Inv[i] = (MOD - MOD / i) \* Inv[MOD % i] % MOD;
20. //预处理伯努利数
21. B[0] = 1;
22. **for**(**int** i=1; i<N; i++)
23. {
24. LL ans = 0;
25. **if**(i == N - 1) **break**;
26. **for**(**int** j=0; j<i; j++)
27. {
28. ans += C[i+1][j] \* B[j];
29. ans %= MOD;
30. }
31. ans \*= -Inv[i+1];
32. ans = (ans % MOD + MOD) % MOD;
33. B[i] = ans;
34. }
35. }
37. LL Work(**int** k)
38. {
39. LL ans = Inv[k+1];
40. LL sum = 0;
41. **for**(**int** i=1; i<=k+1; i++)
42. {
43. sum += C[k+1][i] \* Tmp[i] % MOD \* B[k+1-i] % MOD;
44. sum %= MOD;
45. }
46. ans \*= sum;
47. ans %= MOD;
48. **return** ans;
49. }
51. **int** main()
52. {
53. **int** T;
54. Init();
55. scanf("%d", &T);
56. **while**(T--)
57. {
58. **int** k;
59. scanf("%I64d %d", &n, &k);
60. n %= MOD;
61. Tmp[0] = 1;
62. **for**(**int** i=1; i<N; i++)
63. Tmp[i] = Tmp[i-1] \* (n + 1) % MOD;
64. printf("%I64d\n", Work(k));
65. }
66. **return** 0;
67. }

**分析：**本题与上题不同的是值比较大，达到**50000**，如果采用同样的方法，会**TLE**的。那么必定要进行优化。

