Fibonacci

const long long mod = 1e9 + 7;

long long fib[mx5];

void store(){

fib[0] = 0LL; fib[1] = 1LL;

register int i = 2;

while(i<mx5)

fib[i] = (fib[i-1] + fib[i-2])%mod, i++;

}

Stirling

int sss(int n, int k){

if ((k == 1) || (n == k)) return 1;

return sss (n-1, k-1) + k \* sss (n-1, k);

}

Catalan

/// memset dp -1 required

long long dp[100][100], n = 10;

int call (int x, int y){

if(x+y == (n<<1)) return 1;

if(dp[x][y] != -1) return dp[x][y];

int ans = 0;

if(x < n) ans += call(x+1, y);

if(y < x) ans += call(x, y+1);

return dp[x][y] = ans;

}

/// return nCr fastest way

long long findNCR(long long n, long long r)

{

if(r > n-r) r = n - r;

long long ans = 1, i;

for(i=0; i<r; i++){

ans \*= (n - i);

ans /= (i + 1);

}

return ans;

}

/// return catalan number from the formula

/// Cn = (2n) C (n) - (2n) C (n+1)

/// requirement nCr

long long catalan(int x){

return findNCR(x<<1, x) - findNCR(x<<1, x+1);

}

Derangement

/// derangement number with bit masking

#define ON(n, pos) n = n | (1<<pos)

#define OFF(n, pos) n = n & ~(1<<pos)

long long n = 8, total = (1 << n) - 1, mask;

const int mx = 11;

long long d[mx][1 << mx];

long long call(long long idx){

if(mask == total) return 1;

if(d[idx][mask] != -1) return d[idx][mask];

long long ans = 0;

for(int i=0; i<n; i++)

if(!(mask&(1<<i)) && (i != idx)){

ON(mask, i);

ans += call(idx + 1);

OFF(mask, i);

}

return d[idx][mask] = ans;

}

/// derangement number with formula

/// recursive dp

/// memset required with -1 value

long long dd[mx];

long long ddd(long long x){

if(x < 2) return 1-x;

if(dd[x] != -1) return dd[x];

return dd[x] = (x-1)\*ddd(x-2) + (x-1)\*ddd(x-1);

}