Codebook

1 Basic

1.1 .vimrc

```
set nocompatible              " be iMproved, required
filetype off                  " required

set rtp+=~/.vim/bundle/vundle/
call vundle#rc()
Plugin 'gmarik/vundle'
Plugin 'tpope/vim-fugitive'
Plugin 'L9'
Plugin 'Lokaltog/vim-easymotion'
Plugin 'rstacruz/sparkup', {'rtp': 'vim/'}
Plugin 'valloric/YouCompleteMe'
Plugin 'scrooloose/nerdtree'
Plugin 'jistr/vim-nerdtree-tabs'
Plugin 'bling/vim-airline'
Plugin 'terryma/vim-multiple-cursors'
filetype plugin indent on     " required

"airline config
set laststatus=2
let g:airline_powerline_fonts=1
let g:airline#extensions#tabline#enabled=1
let g:airline#extensions#tabline#buffer_nr_show=1

"youcompltememe config
let g:ycm_global_ycm_extra_conf = '~/.vim/.
    ycm_extra_conf.py'
let g:ycm_enable_diagnostic_signs = 0
let g:ycm_key_invoke_completion = '<c-\>'
set completeopt=menuone

"vim-easymotion config
map  / <Plug>(easymotion-sn)
omap / <Plug>(easymotion-tn)
map  n <Plug>(easymotion-next)
map  N <Plug>(easymotion-prev)

set t_Co=256
set term=screen-256color
set number
map <F5> :NERDTreeTabsToggle <CR>

set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab


set wrap
set showcmd
colorscheme torte
map  <F2> :w <CR> :call OP() <CR>
map! <F2> <ESC> :w <CR> :call OP() <CR> <ESC>
map  <F9> :w <CR> :call CP_R() <CR> <ESC>
map! <F9> <ESC> :w <CR> :call CP_R() <CR> <ESC>
map  <HOME> ^
map! <HOME> <ESC>^i
map  <ESC>OH <HOME>
map! <ESC>OH <HOME>
map  <END> $
map  <ESC>OF <END>
map! <ESC>OF <ESC><END>a
function CP_R()

  if( &ft == 'cpp')
    let cpl = 'g++ -w -o "%:r.exe" -std=c++11 "%"' |
        let exc = '"./%:r.exe"'
  elseif( &ft == 'c')
    let cpl = 'gcc -w -o "%:r" -std=c99 "%"' | let exc
        = '"./%:r"'
  elseif( &ft == 'java')
    let cpl = 'javac "%"' | let exc = 'java "%:r"'
  elseif( &ft == 'python')
    let exc = 'python "%"'
  elseif( &ft == 'sh')
    let exc = 'sh "%"'
  endif

```

```
  let pause = 'printf "Press any key to continue..." && read -n 1 && exit'
  if !exists('exc')
    echo 'Can''t compile this filetype...'
    return
  endif
  if exists('cpl')
    let cp_r = cpl . ' && time ' . exc
  else
    let cp_r = 'time ' . exc
  endif
  "execute '!$COLORTERM -x bash -c ''' . cp_r . ';' . pause . ';exec bash'''
    execute '! clear && ' . cp_r . ' && ' . pause
endfunction

function OP()
  execute '!$COLORTERM -x gedit ' . "%" . ";"
endfunction
```

2 Number

```
#include <stdio.h>
#include <vector>
#include <math.h>
#include <complex>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long ll;
#define EPS 1e-12
#define INF 1e15
typedef complex<double> Complex;
const double pi = acos(-1);

/* extended GCD */
ll ext_gcd(ll a,ll b,ll &x,ll &y){
    ll d=a;
    if(b!=0ll){
        d=ext_gcd(b,a%b,y,x);
        y-=(a/b)*x;
    }
    else x=1ll,y=0ll;
    return d;
}

/*
 * ax = b (mod n)
 * return a set of answer(vector<ll>)
 */
vector<ll> line_mod_equation(ll a,ll b,ll n){
    ll x, y, d;
    d = ext_gcd(a, n, x, y);
    vector<ll> ans;
    if(b%d==0ll){
        x = (x%n + n) % n;
        ans.push_back((x*(b/d))%(n/d));
        for(ll i=1;i<d;i++)
            ans.push_back((ans[0]+i*n/d)%n);
    }
    return ans;
}

/*
 * find the inverse of n modular p
 */
ll mod_inverse(ll n, ll p){
    ll x, y;
    ll d = ext_gcd(n, p, x, y);
    return (p+x%p) % p;
}

/* P is the modular number */
#define P 24851
int fact[P+1];
/* called by Cmod */
int mod_fact(int n,int &e){
    e = 0;
    if(n == 0) return 1;
    int res = mod_fact(n/P, e);
    e += n / P;
    if((n/P) % 2 == 0)
        return res * (fact[n%P]%P);
    return res * ((P-fact[n%P])%P);
}
/*
 * return C(n, m) mod P
 */
int Cmod(int n,int m){
    /* this section only need to be done once */
    fact[0] = 1;
    for(int i=1;i<=P;i++){
        fact[i] = fact[i-1] * i%P;
    }
    /* end */
    int a1, a2, a3, e1, e2, e3;
    a1 = mod_fact(n, e1);
    a2 = mod_fact(m, e2);
    a3 = mod_fact(n-m, e3);
    if(e1 > e2 + e3)return 0;
    return a1 * mod_inverse(a2 * (a3%P), P) % P;
}

/*
 * solve the chinese remainder theorem(CRT)
 * if a.size() != m.size(), return -1
 * return the minimun positive answer of CRT
 * x = a[i] (mod m[i])
 */
int CRT(vector<int> a, vector<int> m) {
    if(a.size() != m.size()) return -1;
    int M = 1;
    for(int i=0;i<(int)m.size();i++)
        M *= m[i];
    int res = 0;
    for(int i=0;i<(int)a.size();i++)
        res = (res + (M/m[i])*mod_inverse(M/m[i], m[i])
            *a[i]) % M;
    return (res + M) % M;
}

ll mul_mod(ll a, ll b, ll m){
    return b?(mul_mod((a*2)%m,b/2,m)+(b&1?a:0))%m:0;
}
/* fast exponential */
ll pow_mod(ll x, ll N, ll M) {
    ll res = 1;
    x %= M;
    while(N){
        if(N&1ll) res = mul_mod(res, x, M);
        x = mul_mod(x, x, M);
        N >>= 1;
    }
    return res;
}

/* called by MillerRabin */
bool PrimeTest(ll n, ll a, ll d) {
    if(n == 2 || n == a) return true;
    if((n&1) == 0) return false;
    while((d&1) == 0) d >>= 1;
    ll t = pow_mod(a, d, n);
    while((d!=n-1) && (t!=1) && (t!=n-1)){
        t = mul_mod(t, t, n);
        d <<= 1;
    }
    return (t==n-1) || ((d&1)==1);
}
/* return true if n is a prime */
bool MillerRabin(ll n){
    // test set
    vector<ll> a = {2, 7 ,61};
    for(int i=0;i<(int)a.size();i++)
        if(!PrimeTest(n, a[i], n-1)) return false;
    return true;
}

/*
 * gen phi from 1~MAXN
 * store answer in phi
 */
```

```cpp
#define MAXN 100
int mindiv[MAXN], phi[MAXN], sum[MAXN];
void genphi(){
    for(int i=1;i<MAXN;i++)
        mindiv[i] = i;
    for(int i=2;i*i<MAXN;i++)
        if(mindiv[i] == i)
            for(int j=i*i;j<MAXN;j+=i)
                mindiv[j] = i;
    phi[1] = 1;
    for(int i=2;i<MAXN;i++){
        phi[i] = phi[i/mindiv[i]];
        if((i/mindiv[i])%mindiv[i] == 0)
            phi[i] *= mindiv[i];
        else phi[i] *= (mindiv[i]-1);
    }
}

/*
 * class of polynomial function
 * coef is the coefficient
 * f(x) = sigma(c[i]*x^i)
 */
class Function {
public:
    vector<double> coef;
    Function(const vector<double> c=vector<double>()):
        coef(c){}
    double operator () (const double &rhs) const {
        double res = 0.0;
        double e = 1.0;
        for(int i=0;i<(int)coef.size();i++,e*=rhs)
            res += e * coef[i];
        return res;
    }

    Function derivative() const {
        vector<double> dc((int)this->coef.size()-1);
        for(int i=0;i<(int)dc.size();i++)
            dc[i] = coef[i+1] * (i+1);
        return Function(dc);
    }

    int degree() const {
        return (int)coef.size()-1;
    }
};

/*
 * calculate the integration of f(x) from a to b
 * divided into n piece
 * the bigger the n is, the more accurate the answer is
 */
template<class T>
double simpson(const T &f, double a, double b){
    double c = (a+b) / 2.0;
    return (f(a)+4.0*f(c)+f(b)) * (b-a) / 6.0;
}
template<class T>
double simpson(const T &f, double a, double b, double
    eps, double A){
    double c = (a+b) / 2.0;
    double L = simpson(f, a, c), R = simpson(f, c, b);
    if(fabs(A-L-R) <= 15.0*eps) return L + R + (A-L-R)
        / 15.0;
    return simpson(f, a, c, eps/2, L) + simpson(f, c, b
        , eps/2, R);
}
template<class T>
double simpson(const T &f, double a, double b, double
    eps){
    return simpson(f, a, b, eps, simpson(f, a, b));
}

/*
 * called by find
 * 1 = positive, -1 = negative, 0 = zero
 */
int sign(double x){
    return x < -EPS ? -1 : x > EPS;
}
/* called by equation */
template<class T>
double find(const T &f, double lo, double hi){
    int sign_lo, sign_hi;
    if((sign_lo=sign(f(lo))) == 0) return lo;
    if((sign_hi=sign(f(hi))) == 0) return hi;
    if(sign_hi * sign_lo > 0) return INF;
    while(hi-lo>EPS){
        double m = (hi+lo) / 2;
        int sign_mid = sign(f(m));
        if(sign_mid == 0) return m;
        if(sign_lo * sign_mid < 0)
            hi = m;
        else lo = m;
    }
    return (lo+hi) / 2;
}
/*
 * return a set of answer of f(x) = 0
 */
template<class T>
vector<double> equation(const T &f){
    vector<double> res;
    if(f.degree() == 1){
        if(sign(f.coef[1]))res.push_back(-f.coef[0]/f.
            coef[1]);
        return res;
    }
    vector<double> droot = equation(f.derivative());
    droot.insert(droot.begin(), -INF);
    droot.push_back(INF);
    for(int i=0;i<(int)droot.size()-1;i++){
        double tmp = find(f, droot[i], droot[i+1]);
        if(tmp < INF) res.push_back(tmp);
    }
    return res;
}

/*
 * called by FFT
 * build the sequence of a that used to calculate FFT
 * return a reversed sequence
 */
vector<Complex> reverse(vector<Complex> a){
    vector<Complex> res(a);
    for (int i=1,j=0;i<(int)res.size();i++){
        for(int k=((int)res.size())>>1;!((j^=k)&k);k
            >>=1);
        if(i > j) swap(res[i], res[j]);
    }
    return res;
}
/*
 * calculate the FFT of sequence
 * a.size() must be 2^k
 * flag = 1  -> FFT(a)
 * falg = -1 -> FFT-1(a)
 * return FFT(a) or FFT-1(a)
 */
vector<Complex> FFT(vector<Complex> a, int flag=1){
    vector<Complex> res = reverse(a);
    for(int k=2;k<=(int)res.size();k<<=1){
        double p0 = -pi / (k>>1) * flag;
        Complex unit_p0(cos(p0), sin(p0));
        for(int j=0;j<(int)res.size();j+=k){
            Complex unit(1.0, 0.0);
            for(int i=j;i<j+k/2;i++,unit*=unit_p0){
                Complex t1 = res[i], t2 = res[i+k/2] *
                    unit;
                res[i] = t1 + t2;
                res[i+k/2] = t1 - t2;
            }
        }
    }
    return res;
}

/*
 * return the sequence of x-th of n!
 * max(n) = 12
 * 0 of 3! -> 123
 * 5 of 3! -> 321
 */
```

```cpp
int factorial[] = {1, 1, 2, 6, 24, 120, 720, 5040,
    40320, 362880, 3628800, 39916800, 479001600};
vector<int> idx2permutation(int x, int n){
    vector<bool> used(n+1, false);
    vector<int> res(n);
    for(int i=0;i<n;i++){
        int tmp = x / factorial[n-i-1];
        int j;
        for(j=1;j<=n;j++)if(!used[j]){
            if(tmp == 0) break;
            tmp--;
        }
        res[i] = j, used[j] = true;
        x %= factorial[n-i-1];
    }
    return res;
}
/*
 * a is x-th og n!
 * return x(0~n!)
 * 123 of 3! -> 0
 * 321 of 3! -> 5
 */
int permutation2idx(vector<int> a){
    int res = 0;
    for(int i=0;i<(int)a.size();i++){
        int tmp = a[i] - 1;
        for(int j=0;j<i;j++)
            if(a[j] < a[i]) tmp--;
        res += factorial[(int)a.size()-i-1] * tmp;
    }
    return res;
}


int main(){
    printf("%d\n", MillerRabin(100000000000000009ll));
}
```