Codebook

1 Basic

1.1 vimrc

```
set nocompatible
filetype plugin indent on
set t_Co=256
set term=screen-256color
set number
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set wrap
set showcmd
colorscheme darkblue
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 == 'python')
    let exc = 'python "%"'
  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 '! clear && ' . cp_r . ' && ' . pause
endfunction

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

2 Number

2.1 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;
}
```

2.2 Modular Inverse

```
/*
 * 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;
}
```

2.3 Line Modular Equation

```
/*
 * 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;
}
```

2.4 Chinese Remainder Theorem

```
/*
 * 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;
}
```

2.5 C(N,M)

```
/* 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*(P-fact[n%P])%P;
    return res*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;
}
```

2.6 Phi

```
/*
 * gen phi from 1~MAXN
 * store answer in phi
 */
#define MAXN 100
int mindiv[MAXN], phi[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);
    }
}
```

2.7 Miller Rabin

```
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;
}
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);
}
bool MillerRabin(ll n){
    // test set
    vector<ll> a = {2, 325, 9375, 28178,  450775,
        9780504, 1795265022};
    for(int i=0;i<(int)a.size();i++)
        if(!PrimeTest(n, a[i], n-1)) return false;
    return true;
}
```

2.8 FFT

```
/*
```

```
    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;
}
```

2.9 Function

```
/*
 * 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;
```

```
template<class T>
double simpson(const T &f, double a, double b, double eps){
    return simpson(f, a, b, eps, simpson(f, a, b));
}
```

2.10 Equation

```
/*
 * 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;
}
```

2.11 Permutation

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

```
 * 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;
}
```

3 Matrix

3.1 Guass Elimination

```
/*
 * return guass eliminated matrix
 * r will be chenged to the number of the non-free
     variables
 * l[i] will be set to true if i-th variable is not
     free
 * ignore flag
 */
Matrix GuassElimination(int &r, vector<bool> &l, int
    flag=0) {
    l = vector<bool>(C);
    r = 0;
    Matrix res(*this);
    for(int i=0;i<res.C-flag;i++){
        for(int j=r;j<res.R;j++){
            if(fabs(res.at(j, i)) > EPS){
                swap(res.D[r], res.D[j]);
                break;
            }
        }
        if(fabs(res.at(r, i)) < EPS){
            continue;
        }
        for(int j=0;j<res.R;j++){
            if(j != r && fabs(res.at(j, i)) > EPS){
                double tmp = (double)res.at(j, i) / (
                    double)res.at(r, i);
                for(int k=0;k<res.C;k++){
                    res.at(j, k) -= tmp * res.at(r, k);
                }
            }
        }
        r++;
        l[i] = true;
    }
    return res;
}
```

3.2 Solve Matrix (Ax=B)

```
/*
 * Ax = b
 * it will return the answer(x)
 * if row != column or there is any free variable, it
     will return an empty vector
 */
vector<double> Solve(vector<double> a) {
    if(R != C) return vector<double>();
    vector<double> res(R);
    Matrix t(R, C+1);
    for(int i=0;i<R;i++){
        for(int j=0;j<C;j++)
            t.at(i, j) = at(i, j);
        t.at(i, C) = a[i];
    }
    int r = 0;
    vector<bool> l;
    t = t.GuassElimination(r, l, 1);
    if(r != R) return vector<double>();
    for(int i=0;i<C;i++){
        if(l[i])for(int j=0;j<R;j++){
            if(fabs(t.at(j, i)) > EPS)
                res[i] = t.at(j, C) / t.at(j, i);
        }
    }
    return res;
}
```

3.3 Inverse Matrix

```
/*
```

```
    Matrix t(R, R*2);
    for(int i=0;i<R;i++){
        for(int j=0;j<C;j++)
            t.at(i, j) = at(i, j);
        t.at(i, i+R) = 1;
    }
    int r = 0;
    vector<bool> l;
    t = t.GuassElimination(r, l, R);
    if(r != R)return Matrix();
    for(int i=0;i<C;i++){
        if(l[i])for(int j=0;j<R;j++){
            if(fabs(t.at(j, i)) > EPS){
                for(int k=0;k<C;k++)
                    t.at(j, C+k) /= t.at(j, i);
            }
        }
    }
    Matrix res(R, C);
    for(int i=0;i<R;i++)
        for(int j=0;j<C;j++)
            res.at(i, j) = t.at(i, j+C);
    return res;
}
```

4 Graph

4.1 Bridge And Cut

```
/* called by cut_bridge */
void _cut_bridge(int x, int f, int d){
    vis[x] = 1;
    dfn[x] = low[x] = d;
    int children = 0;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(e.to != f && vis[e.to] == 1)
            low[x] = min(low[x], dfn[e.to]);
        if(vis[e.to] == 0){
            _cut_bridge(e.to, x, d+1);
            children++;
            low[x] = min(low[x], low[e.to]);
            if((f == -1 && children > 1) || (f != -1 &&
                 low[e.to] >= dfn[x]))
                cut[x] = true;
            if(low[e.to] > dfn[x])
                bridge[x][e.to] = bridge[e.to][x] =
                    true;
        }
    }
}
/*
 * solve the cut and bridge
 * store answer in cut(vector<bool>) ans bridge(vector<
    vector<bool> >)
 * cut[i] == true iff i-th node is cut
 * bridge[i][j] == true iff edge between i-th ans j-th
    is bridge
 */
void cut_bridge(){
    vis = vector<int>(N+1, 0);
    dfn = low = vector<int>(N+1);
    cut = vector<bool>(N+1);
    bridge = vector<vector<bool> >(N+1, vector<bool>(N
        +1, false));
    for(int i=0;i<N;i++){
        if(!vis[i])
            _cut_bridge(i, -1, 0);
    }
}
```

4.2 BCC

```
/* called by BCC */
void _BBC(int x, int d){
    stk[++top] = x;
    dfn[x] = low[x] = d;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(dfn[e.to] == -1){
            _BBC(e.to, d+1);
            if(low[e.to] >= dfn[x]){
                vector<int> l;
                do{
                    l.push_back(stk[top]);
                    top--;
                }while(stk[top+1] != e.to);
                l.push_back(x);
                bcc.push_back(l);
            }
            low[x] = min(low[x], low[e.to]);
        }else low[x] = min(low[x], dfn[e.to]);
    }
}
/*
 * solve the biconnected components(BCC)
 * store answer in bcc(vector<vector<int> >)
 * bbc.size() is the number of BCC
 * bcc[i] is the sequence of a BCC
 */
void BCC(){
```

```
        if(dfn[i] == -1)
            _BBC(i, 0);
}
```

4.3 SCC

```
/* called by SCC */
void _SCC(int x, int d){
    stk[++top] = x;
    dfn[x] = low[x] = d;
    vis[x] = 1;
    for(int i=0;i<(int)vc[x].size();i++){
        Edge e = vc[x][i];
        if(dfn[e.to] != -1){
            if(vis[e.to] == 1)
                low[x] = min(low[x], dfn[e.to]);
        }else{
            _SCC(e.to, d+1);
            low[x] = min(low[x], low[e.to]);
        }
    }
    if(low[x] == dfn[x]){
        while(stk[top] != x){
            scc[stk[top]] = scc_cnt;
            vis[stk[top]] = 2;
            top--;
        }
        scc[stk[top]] = scc_cnt++;
        vis[stk[top]] = 2;
        top--;
    }
}
/*
 * solve the strongly connected component(SCC)
 * store answer in scc(vector<int>)
 * the value of scc[i] means the id of the SCC which i-th node in (id is based 0)
 * scc_cnt id the number of SCC
 */
void SCC(){
    dfn = low = vector<int>(N+1, -1);
    vis = vector<int>(N+1, 0);
    scc = vector<int>(N+1, 0);
    scc_cnt = 0;
    stk = vector<int>(N+1, -1);
    top = -1;
    for(int i=0;i<N;i++)
        if(dfn[i] == -1)
            _SCC(i, 0);
}
```

4.4 Two Sat

```
/*
 * called by TwoSat
 * get the value of i-th
 * 1 = true, 0 = false, -1 = undefined
 */
int TwoSatGet(int x){
    int r = x > N/2 ? x-N/2 : x;
    if(twosatans[r] == -1)
        return -1;
    return x > N/2 ? !twosatans[r] : twosatans[r];
}
/*
 * solve the 2SAT
 * return true if there exists a set of answer
 * store the answer in twosatans
 */
bool TwoSat(){
    SCC();
    twosatans = vector<int>(N/2+1, -1);
    for(int i=0;i<N/2;i++)
        if(scc[i] == scc[i+N/2])
            return false;
    vector<vector<int> > c(scc_cnt+1);
    for(int i=0;i<N;i++)
        c[scc[i]].push_back(i);
    int val = 1;
    for(int i=scc_cnt-1;i>=0;i--){
        for(int j=0;j<(int)c[i].size();j++){
            int x = c[i][j];
            if(TwoSatGet(x) == 0)
                val = 0;
            for(int k=0;k<(int)vc[x].size();k++)
                if(TwoSatGet(vc[x][k].to) == 0)
                    val = 0;
            if(!val)
                break;
        }
        for(int j=0;j<(int)c[i].size();j++){
            if(c[i][j] > N/2)
                twosatans[c[i][j]-N/2] = !val;
            else
                twosatans[c[i][j]] = val;
        }
    }
    return true;
}
```

5 Path

5.1 Kth Shortest

5.2 EulerCircuit

6 Flow

6.1 Dinic

6.2 StoerWanger

6.3 Mixed Euler

7 Match

7.1 KM

7.2 BiMatch

7.3 General Match

8 MST

8.1 Restricted MST

8.2 MDST

8.3 MRST