

# STANDARD CODE LIBRARY

---

*Chen Gang*  
School of Informaton Engineering  
Zhengzhou University

August 9, 2015

# Contents

<b>1</b>	<b>Graph Theorem</b>	<b>2</b>
1.1	Biconnected Component . . . . .	2
1.2	Strongly Connected Componenet . . . . .	2
1.3	Dijkstra . . . . .	3
1.4	Hungray . . . . .	3
1.5	Dinic . . . . .	4
1.6	Minimun Cost Maximun Flow . . . . .	5
<b>2</b>	<b>Data Structure</b>	<b>7</b>
2.1	Union-Find Set . . . . .	7
2.2	Hash Table . . . . .	7
2.3	Binary Indexed Tree . . . . .	7
2.4	Segment Tree . . . . .	8
2.5	KMP . . . . .	8
<b>3</b>	<b>Math</b>	<b>9</b>
3.1	Extended Eucild . . . . .	9
3.2	Mod Class C . . . . .	9
3.3	Guess . . . . .	9
3.4	FFT . . . . .	10
<b>4</b>	<b>Computational Geometry</b>	<b>11</b>
4.1	Intersection . . . . .	11
4.2	Point to Segment . . . . .	11
4.3	Point at Polygon . . . . .	11
4.4	Convex Hull . . . . .	11
<b>5</b>	<b>Others</b>	<b>12</b>
5.1	Big Number . . . . .	12
5.2	vimrc . . . . .	12

# 1 Graph Theorem

## 1.1 Biconnected Component

```
vi a[N+10], bcc[N+10];
int pre[N+10], bccno[N+10], low[N+10];
bool iscut[N+10];
int dfs_clock, bcc_cnt;
stack<E> s;

void dfs(int i, int fa){
    pre[i]=low[i]=++dfs_clock;
    int child=0;
    rep(k, sz(a[i])){
        int j=a[i][k];
        if (!pre[j]){
            s.push(E(i,j)), child++;
            dfs(j, i), checkmin(low[i], low[j]);
            if (low[j]>=pre[i]){
                iscut[i]=true, bcc_cnt++, bcc[bcc_cnt].clear();
                for(;;){
                    E e=s.top(); s.pop();
                    if (bccno[e.i]!=bcc_cnt) bcc[bcc_cnt].pb(e.i);
                    if (bccno[e.j]!=bcc_cnt) bcc[bcc_cnt].pb(e.j);
                    bccno[e.i]=bccno[e.j]=bcc_cnt;
                    if (e.i==i && e.j==j) break;
                }
            }
        }
        else if (j!=fa && pre[j]<pre[i])
            s.push(E(i,j)), checkmin(low[i], pre[j]);
    }
    if (fa<0 && child==1) iscut[i]=0;
}

void find_bcc(int n){
    dfs_clock=bcc_cnt=0;
    clr(pre, 0), clr(iscut, 0), clr(bccno, 0);
    repf(i, 1, n) if (!pre[i])
        dfs(i, -1);
}
```

## 1.2 Strongly Connected Component

```
vi a[N+10];
int dfn[N+10], low[N+10], num[N+10];
int belong[N+10], s[N+10];
bool inS[N+10];
int Idx, now;

void tar(int i){
    dfn[i]=low[i]=++now, s[++s[0]]=i, inS[i]=true;
    rep(k, sz(a[i])){
        int j=a[i][k];
        if (!dfn[j])
            tar(j), _checkmin(low[i], low[j]);
        else if (inS[j])
            _checkmin(low[i], dfn[j]);
    }
}
```

```

        if (low[i]==dfn[i]){
            Idx++;
            do{
                j=s[s[0]--];
                belong[j]=Idx, num[Idx]++;
                inS[j]=false;
            }
            while(j!=i);
        }
    }

void tar(){
    now=Idx=s[0]=0;
    clr(dfn, 0), clr(inS, 0), clr(num, 0);
    repf(i, 1, n) if (!dfn[i]) tar(i);
}

```

### 1.3 Dijkstra

```

struct P{
    int i, d;
    P(){}
    P(int i, int d):i(i),d(d){}
    bool operator < (const P&p)const{ return d>p.d; }
};

int d[N+10];
bool done[N+10];
vi a[N+10];

void dijkstra(int s){
    priority_queue<P> q; clr(d, -1), clr(done, 0);
    q.push(P(s, 0)), d[s]=0, done[s]=true;
    while (!q.empty()){
        P p=q.top(); q.pop();
        int i=p.i;
        if (done[i]) continue;
        done[i]=true;
        rep(k, sz(a[i])){
            j=a[i][k].to;
            if (d[j]==-1 || d[j]>d[i]+a[i][k].w){
                d[j]=d[i]+a[i][k].w;
                q.push(P(j, d[j]));
            }
        }
    }
}

```

### 1.4 Hungray

```

vi a[N+10];
int f[N+10], v[N+10];

bool find(int i){
    rep(j, sz(a[i])){
        int k=a[i][j];
        if (!v[k]){
            v[k]=true;
            if (!f[k] || find(f[k])){
                f[k]=i;
                return true;
            }
        }
    }
}

```

```

        }
    }
    return false;
}

int hungry(){
    int ret=0;
    clr(f, 0);
    repf(i, 1, n){
        clr(v, 0);
        if (find(i)) ret++;
    }
    return ret;
}

```

## 1.5 Dinic

```

#include <vector>
#include <queue>

struct e_t{
    int to, cap, rev;
    e_t(int to, int cap, int rev):to(to), cap(cap), rev(rev){}
};

template<int SZ>
class Dinic{
public:
    vector<e_t> a[SZ+10];
    int lev[SZ+10], done[SZ+10];
    int s, t;

    bool levelize(){
        queue<int> q;
        memset(lev, -1, sizeof(lev));
        q.push(s), lev[s]=0;
        while (!q.empty()){
            int i=q.front(); q.pop();
            rep(k, sz(a[i])){
                e_t e=a[i][k];
                if (!e.cap || lev[e.to]==-1)
                    continue;
                lev[e.to] = lev[i] + 1;
                q.push(e.to);
            }
        }
        return lev[t]!=-1;
    }

    int augment(int v, int f){
        if (v==t || !f) return f;
        for (; done[v] < sz(a[v]); ++done[v]){
            e_t &e = a[v][done[v]];
            if (lev[e.to] < lev[v] || !e.cap) continue;
            int t = augment(e.to, min(f, e.cap));
            if (t){
                e.cap -= t;
                a[e.to][e.rev].cap += t;
                return t;
            }
        }
        return 0;
    }
};

```

```

    }

    void clear(){
        rep(i, SZ) a[i].clear();
    }

    void add(int i, int j, int c){
        a[i].push_back(e_t(j, c, sz(a[j])));
        a[j].push_back(e_t(i, 0, sz(a[i])-1));
    }

    int maxFlow(){
        int tot=0, tmp;
        while (levelize()){
            memset(done, 0, sizeof(done));
            while ( (tmp = augment(s, INF)) )
                tot += tmp;
        }
        return tot;
    }
};

```

## 1.6 Minimun Cost Maximun Flow

```

struct e_t {int to, cap, rev, cost;};

template <int N>
class MCMF{
public:
    vector<e_t> a[N*5+10];
    int f[N*5+10], c[N*5+10];
    bool inQ[N*5+10];
    e_t *e[N*5+10];
    int s, t;

    void clear(){ rep(i, t+1) a[i].clear(); }

    void add(int i, int j, int c, int cost){
        a[i].pb((e_t){j, c, sz(a[j]), cost});
        a[j].pb((e_t){i, 0, sz(a[i])-1, -cost});
    }

    bool bellmanFord(int &flow, int &cost){
        queue<int> q;
        clr(f, 0), clr(c, 0x7f), clr(inQ, 0);

        q.push(s), f[s]=INF, c[s]=0, inQ[s]=1;
        while (!q.empty()){
            int i=q.front(); q.pop(); inQ[i]=0;
            rep(k, sz(a[i])){
                e_t & ei = a[i][k];
                if (ei.cap && c[ei.to]>c[i]+ei.cost){
                    f[ei.to]=min(f[i], ei.cap);
                    c[ei.to]=c[i]+ei.cost;
                    e[ei.to]=&ei;
                    if (!inQ[ei.to]) inQ[ei.to]=true, q.push(ei.to);
                }
            }
        }
        if (c[t]==0x7f7f7f7f) return false;
        flow+=f[t], cost+=c[t]*f[t];
    }
};

```

```

    int i=t;
    while (i!=s){
        e[i]->cap-=f[t];
        a[i][e[i]->rev].cap+=f[t];
        i=a[i][e[i]->rev].to;
    }
    return true;
}

void minCost(int &flow, int &cost){
    while (bellmanFord(flow, cost));
}

};

```

## 2 Data Structure

### 2.1 Union-Find Set

```
template<int SZ>
class UFS{
    int f[SZ+10];
public:
    void clear(){ rep(i, SZ+10) f[i]=i; }
    int find(int i){
        if (f[i]==i) return i;
        return f[i]=find(f[i]);
    }
    void unions(int i, int j){
        i=find(i), j=find(j);
        f[i]=j;
    }
};
```

### 2.2 Hash Table

```
char str[N+10][S+10];

template<int SZ>
struct Hash{
    int h[SZ+10];
    H(){ clr(h, -1); }
    int gao(char *s){
        int ret=0, n=strlen(s);
        rep(i, n) ret=(ret*131+s[i])%SZ;
        return ret;
    }
    int find(char *s){
        int k=gao(s);
        while (h[k]!=-1 && strcmp(str[h[k]], s)!=0)
            k=(k+1)%SZ;
        return h[k];
    }
    void ins(char *s, int i){
        int k=gao(s);
        while (h[k]!=-1 && strcmp(str[h[k]], s)!=0)
            k=(k+1)%SZ;
        h[k]=i;
    }
};
```

### 2.3 Binary Indexed Tree

```
template <int SZ>
struct BIT{
    int a[SZ+10];
public:
    void clear(){ clr(a, 0); }
    void ins(int x, int k){
        while (x<=n)
            a[x]+=k, x+=x&-x;
    }
    int qry(int x){
        int ret=0;
        while (x>0)
            ret+=a[x], x-=x&-x;
    }
};
```



```

        return ret;
    }
};

```

## 2.4 Segment Tree

```

#define lson i*2, x, z
#define rson i*2+1, z+1, y

template<int SZ>
class SegTree{
    int a[SZ*4+10], mod[SZ*4+10];
    void update(int i){
        a[i*2]=a[i*2+1]=mod[i*2]=mod[i*2+1]=mod[i];
        mod[i]=0;
    }
public:
    void clear(){ clr(a, 0), clr(mod, 0); }
    void ins(int i, int x, int y, int l, int r, int c){
        if (x==l && y==r){
            a[i]=c;
            mod[i]=c;
            return;
        }
        if (mod[i]) update(i);

        int z=mid(x,y);
        if (r<=z) ins(lson, l, r, c);
        else if (l>z) ins(rson, l, r, c);
        else{
            ins(lson, l, z, c);
            ins(rson, z+1, r, c);
        }
        a[i]=a[i*2] | a[i*2+1];
    }
    int query(int i, int x, int y, int l, int r){
        if (x==l && y==r) return a[i];
        if (mod[i]) update(i);

        int z=mid(x, y);
        if (r<=z) return query(lson, l, r);
        else if (l>z) return query(rson, l, r);
        else return query(lson, l, z) | query(rson, z+1, r);
    }
};

```

## 2.5 KMP

```

char s[N+10];
int f[N+10];

void getFail(char *s, int *f, int n){
    f[0]=f[1]=0;
    repf(i, 1, n-1){
        int j=f[i];
        while (j && s[i]!=s[j]) j=f[j];
        f[i+1]= s[i]==s[j]? j+1: 0;
    }
}

```

## 3 Math

### 3.1 Extended Eucild

```
template <class T>
T exgcd(T a, T b, T &x, T &y){
    if (b==0) return x=1, y=0, a;
    T ret=exgcd(b, a%b, x, y), t=x;
    x=y, y=t-a/b*y;
    return ret;
}
```

### 3.2 Mod Class C

```
template <class T>
T exgcd(T a, T b, T &x, T &y){
    if (b==0) return x=1, y=0, a;
    T ret=exgcd(b, a%b, x, y), t=x;
    x=y, y=t-a/b*y;
    return ret;
}

template<class T>
struct C{
    static const T M=1000000007;
    T x;
    C(){ }
    C(T _x){x=( _x%M+M)%M;}
    C anti()const{
        T _x, _y;
        exgcd(x, M, _x, _y);
        return C(_x);
    }
    C operator +(const C &c)const{ return C(x+c.x); }
    C operator -(const C &c)const{ return C(x-c.x); }
    C operator *(const C &c)const{ return C(x*c.x); }
    C operator /(const C &c)const{ return (*this)*c.anti(); }
    void out(){
        cout<<x<<endl;
    }
};

typedef C<long long> mType;
```

### 3.3 Guess

```
void Guess(int n ,int m){
    int k=0;
    rep(i, n){
        repf(j, k, m-1) if (a[j][i]){
            rep(1, n+1) swap(a[k][1], a[j][1]);
            break;
        }
        if (a[k][i]==0) continue;
        rep(j, m) if (j!=k && a[j][i]){
            int x=a[j][i], y=a[k][i];
            rep(1, n+1) a[j][1]= ((a[j][1]*y-a[k][1]*x)
                                %E+E)%E;
        }
        k++;
    }
}
```

```

    repf(i, k, m-1) if (a[i][n]){
        puts("Inconsistent data."); return;
    }
    if (k<n){
        puts("Multiple solutions."); return;
    }
    vi ans;
    rep(i, n) ans.pb( (a[i][n]*op[a[i][i]])%E);
    out(ans);
}

```

### 3.4 FFT

```

const double pi = acos(-1.0);
struct vir{
    double re, im;
    vir(){}
    vir(double re, double im):re(re),im(im){}
    vir operator +(const vir &b){ return vir(re+b.re,im+b.im);
    }
    vir operator -(const vir &b){ return vir(re-b.re,im-b.im);
    }
    vir operator *(const vir &b){ return vir(re*b.re-im*b.im,
        re*b.im+im*b.re); }
};

void brc(vir *y, int l){
    for (int i=1, j=l>>1; i<l-1; ++i){
        if (i<j) swap(y[i], y[j]);
        int k=l>>1;
        while (j>=k) j-=k, k>>=1;
        if (j<k) j+=k;
    }
}

void fft(vir *y, int l, int on){
    vir u, t;
    brc(y, l);
    for (int h=2; h<=l; h<<=1){
        vir wn(cos(on*2*pi/h), sin(on*2*pi/h));
        for (int j=0; j<l; j+=h){
            vir w(1,0);
            for (int k=j; k<j+h/2; ++k){
                u=y[k];
                t=w*y[k+h/2];
                y[k]=u+t;
                y[k+h/2]=u-t;
                w=w*wn;
            }
        }
    }
    if (on==-1)
        for (int i=0; i<l; ++i)
            y[i].re/=l;
}
// fft(y, l, 1);
// rep(i, l) y[i]=y[i]*y[i];
// fft(y, l, -1);

```

## 4 Computational Geometry

### 4.1 Intersection

```
bool Intersection(P p1, P p2, P p3, P p4, P &c){
    double d1=(p2-p1)*(p3-p1), d2=(p2-p1)*(p4-p1);
    double d3=(p4-p3)*(p1-p3), d4=(p4-p3)*(p2-p3);
    int s1=sgn(d1), s2=sgn(d2), s3=sgn(d3), s4=sgn(d4);
    if (s1*s2>0 || s3*s4>0) return false;
    c=P((p3.x*d2-p4.x*d1)/(d2-d1), (p3.y*d2-p4.y*d1)/(d2-d1));
    return true;
}
```

### 4.2 Point to Segment

```
double point2segment(P a, P b, P p){
    if (a==b) return (p-a).len();
    if (sgn((p-a)^(b-a))<0) return (p-a).len();
    else if (sgn((p-b)^(a-b))<0) return (p-b).len();
    else return fabs((p-a)*(a-b))/(a-b).len();
}
```

### 4.3 Point at Polygon

```
bool isPointInPolygon(P p, vp &a){
    int w=0;
    rep(i, n){
        int k=sgn((a[i+1]-a[i])*(p-a[i]));
        int d1=sgn(a[i].y-p.y);
        int d2=sgn(a[i+1].y-p.y);
        if (k>0 && d1<=0 && d2>0) w++;
        if (k<0 && d2<=0 && d1>0) w--;
    }
    if (w!=0) return 1;
    return 0;
}
```

### 4.4 Convex Hull

```
void ConvexHull(vp a, vp &b){
    int n = sz(a);
    sort(all(a));
    rep(i, n){
        while (sz(b)>1 && (b[sz(b)-1]-b[sz(b)-2])*(a[i]-b[sz(b)-2])
            <=0) b.pop_back();
        b.pb(a[i]);
    }
    int k=sz(b);
    repd(i, n-2, 0){
        while (sz(b)>k && (b[sz(b)-1]-b[sz(b)-2])*(a[i]-b[sz(b)-2])
            <=0) b.pop_back();
        b.pb(a[i]);
    }
    if (sz(b)>1) b.pop_back();
}
```

## 5 Others

### 5.1 Big Number

```
struct bigNum{
    static const int L=1000;
    int it[L+10];
    bigNum(){
        fill(it, 0), it[0]=1;
    }
    bigNum(int n){
        fill(it, 0);
        while (n){
            it[++it[0]]=n%10;
            n/=10;
        }
        if (!it[0]) it[0]=1;
    }
    bigNum operator +(const bigNum & b)const{
        bigNum ret;
        ret.it[0]=max(it[0], b.it[0])+1;
        repf(i, 1, ret.it[0]){
            ret.it[i]=it[i]+b.it[i];
            ret.it[i+1]+=ret.it[i]/10;
            ret.it[i]%=10;
        }
        while (ret.it[0]>1 && ret.it[ret.it[0]]==0) ret.it[0]--;
        return ret;
    }
    bigNum operator -(const bigNum & b)const{
        bigNum ret;
        ret.it[0]=it[0];
        repf(i, 1, ret.it[0]){
            ret.it[i]=it[i]-b.it[i];
            if (ret.it[i]<0)
                ret.it[i]+=10, ret.it[i+1]--;
        }
        while (ret.it[0]>1 && ret.it[ret.it[0]]==0) ret.it[0]--;
        return ret;
    }
    bigNum operator *(const bigNum & b)const{
        bigNum ret;
        ret.it[0]=it[0]+b.it[0];
        repf(i, 1, it[0]) repf(j, 1, b.it[0])
            ret.it[i+j-1]+=it[i]*b.it[j];
        repf(i, 1, ret.it[0])
            ret.it[i+1]+=ret.it[i]/10, ret.it[i]%=10;
        while (ret.it[0]>1 && ret.it[ret.it[0]]==0) ret.it[0]--;
        return ret;
    }
    void out(){
        repd(i, it[0], 1) printf("%d", it[i]);
        putchar('\n');
    }
};
```

### 5.2 vimrc

```
set mouse=a
```

```
set nu
set history=4000
set backspace=2
set sw=4
set ts=4
set cindent
syntax on

func! R()
    exec ":w"
    exec "!clear&&g++ %<.cpp -o %<"
    exec "!./%<"
endfunc

:map<F9> :call R()<CR>
```