ACM ICPC Team Reference

March 3, 2010

Algebra

```
typedef long long ll;

ll mdc(ll a, ll b) {
    ll c;
    while(b>0) {
        c=a%b;
        a=b;
        b=c;
    }
    return a;
}

ll mdc(ll a, ll b, ll c) {
    return a==0?mdc(b,c):mdc(mdc(a,b),c);
}


/* first*a + second*b = mdc(a, b) */
pair<ll, ll> mdc_ext(ll a, ll b) {
    if(b == 0) {
        return make_pair(1, 0);
    } else {
        pair<ll, ll> p = mdc_ext(b, a%b);
        return make_pair(p.second, p.first - p.second*(a/b));
    }
}

ll mmc(ll a, ll b) {
    return (a / mdc(a, b)) * b;
}

/* a^e */
ll pot(ll a, ll e) {
    ll resp = 1;
    for(;e>0;e/=2) {
        if(e%2==1)
            resp = (resp*a);
        a = (a*a);
    }
    return resp;
}

/* (a^e)%m */
ll pot(ll a, ll e, ll m) {
    ll resp = 1;
    for(;e>0;e/=2) {
        if(e%2==1)
            resp = (resp*a) % m;
        a = (a*a)%m;
    }
    return resp;
}

/* (a1*q^0 + a1*q^1 + .. + an*q^n) % mod */
ll soma_pg_mod(ll a1, ll q, ll n, ll mod) {
    if(n<=0) {
        return 0;
    } else if((n)%2==1) {
        return (soma_pg_mod((a1 * q) % mod, q, n-1, mod) + a1) % mod;
    } else {
        return ((1 + pot_mod(q, n/2, mod)) * soma_pg_mod(a1, q, n/2, mod)) % mod;
    }
}
```

RIT

```
#define MAX_ELEMENTS 100
int soma[MAX_ELEMENTS];
int bit_tam;


void init(int N) {
    bit_tam = N;
    memset(soma, 0, sizeof(soma[0])*bit_tam);
}

/* O(lg bit_tam) */
void add(int p, int amount) {
    int inicio=0, fim = bit_tam-1, meio;
    while(inicio <= fim) {
        meio = (inicio + fim) / 2;
        if(p <= meio) {
            soma[meio] += amount;
            fim = meio - 1;
        } else {
            inicio = meio + 1;
        }
    }
}

/* O(lg bit_tam)
 * returns sumation of [0,p] */
int get(int p) {
    int inicio=0, fim = bit_tam-1, meio, ret = 0;
    while(inicio <= fim) {
        meio = (inicio + fim) / 2;
        if(p < meio) {
            fim = meio - 1;
        } else {
            ret = (soma[meio] + ret) % MOD;
            inicio = meio + 1;
        }
    }
    return ret;
}

/* O(lg bit_tam)
 * returns smaller p that get(p)>=amount */
int find(int amount) {
    int inicio=0, fim = bit_tam-1, meio, ultimo = bit_tam-1;
    while(inicio <= fim) {
        meio = (inicio + fim) / 2;
        if(soma[meio] < amount) {
            amount -= soma[meio];
            inicio = meio + 1;
        } else {
            ultimo = meio;
            fim = meio - 1;
        }
    }
    return ultimo;
}
```

Combinatória

```
#define MOD 57

typedef long long ll;

ll table[MOD];

/* Computa fatoriais modulo MOD */
void init() {
    table[0]=1;
    for(int i=1;i<MOD;++i) {
        table[i] = (table[i-1] * i) % MOD;
    }

}

/* inverso multiplicativo no Zm */
ll inverse(ll a, ll m) {
    a = a%m;
    if(a == 1) return 1;
    return ((1-inverse(m, a)*m)/a+m)%m;
}

ll fat(int a) {
    ll resp = 1;
    for(int i = 2; i <= a; ++i) {
        resp *= i;
    }
    return resp;
}

ll fat(int a, ll m) {
    //return table[a];
    ll resp = 1;
    for(int i = 2; i <= a; ++i) {
        resp = (resp * i) % m;
    }
    return resp;
}

ll choose(ll a, ll b) {
    return fat(a) / fat(b) / fat(a-b);
}

ll choose(ll a, ll b, ll m) {
    if(b < 0 or a < b) {
        return 0;
    } else if(a < m) {
        return (fat(a,m)*inverse(fat(b,m)*fat(a-b,m),m)) % m;
    } else {
        return (choose(a%m, b%m, m) * choose(a/m, b/m, m)) % m;
    }
}
```

Crivo

```
#define MAX_NUMBERS 100000

bool crivo[MAX_NUMBERS];

int primos[MAX_NUMBERS];
int n_primos;

void init_crivo(int last) {
    memset(crivo, true, sizeof(crivo));
    crivo[0]=crivo[1]=false;
    int m = int(sqrt(last)+1);
    for(int i=2;i<=m;i++) {
        if(crivo[i]) {
            for(int j=i*i; j <= last; j+=i) {
                crivo[j]=false;
            }
        }
    }
    n_primos=0;
    for(int i=2;i<=last;++i) {
        if(crivo[i]) {
            primos[n_primos++]=i;
        }
    }
}
```

Determinante

```
#define MAX_TAM 20

double matriz[MAX_TAM][MAX_TAM];
int pmatriz[MAX_TAM][MAX_TAM];
int nmatriz[MAX_TAM];

double PD[1<<MAX_TAM];

bitset<1<<MAX_TAM> visitado;

int mat_tam;

void init(int N) {
    mat_tam = N;
    memset(nmatriz, 0, sizeof(nmatriz));
    visitado.reset();
}

void add(int l, int c, double v) {
    matriz[l][nmatriz[l]] = v;
    pmatriz[l][nmatriz[l]++]=c;
}

inline double det(int mask=0, int p=0) {
    if(p==N) return 1.0;
    double resp = 0.0;
    if(visitado[mask]) {
        return PD[mask];
    }
    visitado.set(mask);
    for(int i=0;i<nmatriz[p];++i) {
        if(((1<<pmatriz[p][i])&mask)==0) {
            int d = pmatriz[p][i];
            d -= __builtin_popcount(mask>>d);
            if(d%2==0) {
                resp += det(mask | (1<<pmatriz[p][i]),p+1) * matriz[p][i];
            } else {
                resp -= det(mask | (1<<pmatriz[p][i]),p+1) * matriz[p][i];
            }
        }
    }
    return PD[mask] = resp;
}
```

Dijkstra

```
#define MAXV 10000
#define MAXE 20000000

int grafo[MAXV];
int next[MAXE];
int vertice[MAXE];
int cost[MAXE];

int edge_count;

int menor[MAXV];

typedef pair<int, int> pii;

void add_edge(int u, int v, int c) {
    int e = edge_count++;
    next[e] = grafo[u];
    vertice[e] = v;
    cost[e] = c;
    grafo[u] = e;
}

int menor_caminho(int u, int v) {

    priority_queue<pii, vector<pii>, greater<pii> > fila;
    fila.push(pii(0, u));
    memset(menor, 0x7f, sizeof(menor));
    menor[u] = 0;

    while(not fila.empty()) {
        int w = fila.top().second;
        fila.pop();
        if(w == v) {
            break;
        }
        for(int e = grafo[w]; e != -1; e = next[e]) {
            int x = vertice[e];
            int tmp = menor[w] + cost[e];
            if(tmp < menor[x]) {
                menor[x] = tmp;
                fila.push(pii(tmp, x));
            }
        }
    }

    return menor[v];
}
```

Floyd

```
#define MAX_VERTICES 256
#define INF 0x7f7f7f7f

int dist[MAX_VERTICES][MAX_VERTICES];

void floyd(int size) {
    for(int k=0;k<size;++k)
        for(int u=0;u<size;++u)
            for(int v=0;v<size;++v) if(dist[u][k] != INF and dist[k][v] != INF)
                dist[u][v] = min(dist[u][v], dist[u][k] + dist[k][v]);
}
```

Fração

```
typedef long long ll;

ll mdc(ll a, ll b) {
    return (b==0)?a:mdc(b,a%b);
}
ll mmc(ll a, ll b) {
    return (a / mdc(a, b)) * b;
}
struct Fracao {
    ll a, b;
    Fracao() : a(0), b(1) { }
    Fracao(ll _a, ll _b=1) : a(_a), b(_b) {
        ll m = mdc(a, b);
        a /= m; b/= m;
        if(b<0) a=-a, b=-b;
        if(a==0) b=1;
    }
    Fracao operator+(const Fracao& f) const {
        ll m1 = mdc(b, f.b);
        return Fracao((f.b/m1)*a+(b/m1)*f.a,(b/m1)*f.b);
    }
    Fracao& operator+=(const Fracao& f) {
        *this = *this + f;
        return *this;
    }
    Fracao operator-(const Fracao& f) const {
        ll m1 = mdc(b, f.b);
        return Fracao((f.b/m1)*a-(b/m1)*f.a,(b/m1)*f.b);
    }
    Fracao& operator-=(const Fracao& f) {
        *this = *this - f;
        return *this;
    }
    Fracao operator*(const Fracao& f) const {
        ll m1 = mdc(a, f.b);
        ll m2 = mdc(b, f.a);
        return Fracao((a/m1)*(f.a/m2),(b/m2)*(f.b/m1));
    }
    Fracao& operator*=(const Fracao& f) {
        *this = *this * f;
        return *this;
    }
    Fracao operator/(const Fracao& f) const {
        ll m1 = mdc(a, f.a);
        ll m2 = mdc(b, f.b);
        return Fracao((a/m1)*(f.b/m2),(b/m2)*(f.a/m1));
    }
    Fracao& operator/=(const Fracao& f) {
        *this = *this / f;
        return *this;
    }
    bool operator==(ll n) const { return a==n and b==1; }
    bool operator!=(ll n) const { return a!=n or b!=1; }
    Fracao inv() const { return Fracao(b, a); }
    ll floor() const { return a/b; }
    ll ceil() const { return (a+b-1)/b; }
};

Fracao mmc(Fracao f1, Fracao f2) {
    ll m1 = mdc(f1.b, f2.b);
    ll m2 = mmc((f2.b/m1) * abs(f1.a), (f1.b/m1) * abs(f2.a));
    return Fracao(m2, (f1.b/m1) * f2.b);
}
```

Geometry

```
typedef long long ll;

#define ABS(x) ((x)>=0?(x):-(x))

struct geometry {
    ll x,y,w;
    geometry() : x(0), y(0), w(1) { }
    geometry(ll x, ll y, ll w=1) :x(x), y(y), w(w) {
    }
    ll escalar(const geometry& g) {return x*g.x+y*g.y+w*g.w;}
    geometry operator*(const geometry& r) {
        return geometry(y*r.w-w*r.y,w*r.x-x*r.w,x*r.y-y*r.x);
    }
    geometry operator+(const geometry& r) {
        return geometry(r.w*x+w*r.x,r.w*y+w*r.y,w*r.w);
    }
    geometry operator-(const geometry& r) {
        return geometry(r.w*x-w*r.x,r.w*y-w*r.y,w*r.w);
    }
    geometry perpendicular() {
        return geometry(-y,x,w);
    }
    geometry passa_por(const geometry& g) {
        return geometry(x*g.w,y*g.w,-x*g.x-y*g.y);
    }
    geometry operator*(ll k) {
        return geometry(x*k,y*k,w);
    }
    geometry operator/(ll k) {
        return geometry(x,y,w*k);
    }
    geometry inv() const {
        return geometry(-x,-y,-w);
    }
    geometry dist2(const geometry& g) {
        geometry tmp = *this-g;
        return geometry(tmp.x*tmp.x+tmp.y*tmp.y,0,tmp.w*tmp.w);
    }
    bool operator<(const geometry& g) const {
        geometry g1=*this;
        geometry g2=g;
        assert(g1.w>=0);
        assert(g2.w>=0);
        if(g1.w<g2.w) return true;
        if(g1.w>g2.w) return false;
        if(g1.x<g2.x) return true;
        if(g1.x>g2.x) return false;
        if(g1.y<g2.y) return true;
        if(g1.y>g2.y) return false;
        return false;
    }
    bool operator==(const geometry& g) const {
        return x==g.x and y==g.y and w==g.w;
    }
    geometry unique() const {
        geometry r = *this;
        ll m=mdc(ABS(x),ABS(y),ABS(w));
        if(this->w<0 or (this->w==0 and this->y<0) or
            (this->w==0 and this->y==0 and this->x<0))
            m=-m;
        r.x/=m;
        r.y/=m;
        r.w/=m;
        return r;
```

```
    }
    geometry norm() const {
        return geometry(x/w, y/w);
    }
};

ostream& operator <<(ostream& stream, const geometry& g) {
    return stream << g.x << ',' << g.y << ',' << g.w;
}
```

Matching

```
#define MAXV 1000
#define MAXE 10000

int grafo[MAXV];

int next[MAXE];
int vertice[MAXE];

int edge_count;

int match[MAXV];
int visited[MAXV];
int mark;

inline void init() {
    memset(grafo, -1, sizeof(grafo));
    edge_count = 0;
}

inline void add_edge(int u, int v) {
    int e1 = edge_count++;
    vertice[e1] = v;
    next[e1] = grafo[u];
    grafo[u] = e1;

    int e2 = edge_count++;
    vertice[e2] = u;
    next[e2] = grafo[v];
    grafo[v] = e2;
}

bool dfs(int v) {
    if(visited[v] == mark) return false;
    visited[v] = mark;
    for(int e = grafo[v]; e != -1; e = next[e]) {
        int u = vertice[e];
        if(match[u] == -1 or dfs(match[u])) {
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int max_matching() {
    memset(visited, 0, sizeof(visited));
    memset(match, -1, sizeof(match));
    mark = 1;
    int total = 0;
    for(int i = 0; i < N; ++i) if(match[i] == -1) {
        if(dfs(i)) {
            ++total;
            ++mark;
        }
    }
    return total;
}
```

Matriz

```
#include <iostream>
#include <cstring>

using namespace std;

#define MAX_TAM 64
#define MOD 123

#define EVAL(x) (x)
//#define EVAL(x) ((x) % MOD)

typedef int element_t;

typedef element_t vetor_t[MAX_TAM];
typedef element_t matriz_t[MAX_TAM][MAX_TAM];

void vet_cpy(vetor_t& r, const vetor_t& v, int size) {
    memcpy(r, v, sizeof(r[0])*size);
}

void mat_zero(matriz_t& r, int size) {
    for(int i=0;i<size;++i) {
        memset(r[i], 0, sizeof(r[0][0])*size);
    }
}

void mat_ident(matriz_t& r, int size) {
    mat_zero(r, size);
    for(int i=0;i<size;++i) {
        r[i][i] = 1;
    }
}

void mat_cpy(matriz_t& r, const matriz_t& m, int size) {
    for(int i=0;i<size;++i) {
        memcpy(r[i], m[i], sizeof(m[0][0])*size);
    }
}

void mat_add(matriz_t& r, const matriz_t& m, int size) {
    for(int i=0;i<size;++i) {
        for(int j=0;j<size;++j) {
            r[i][j] = EVAL(r[i][j] + m[i][j]);
        }
    }
}

void mat_add(matriz_t& r, const matriz_t& m1, const matriz_t& m2, int size) {
    mat_cpy(r, m1, size);
    mat_add(r, m2, size);
}

void mat_mul(matriz_t& r, const matriz_t& m1, const matriz_t& m2, int size) {
    for(int i=0;i<size;++i) {
        for(int j=0;j<size;++j) {
            r[i][j] = 0;
            for(int k=0;k<size;++k) {
                r[i][j] = EVAL(r[i][j] + m1[i][k] * m2[k][j]);
            }
        }
    }
}

void mat_mul(matriz_t& r, const matriz_t& m, int size) {
    matriz_t tmp;
    mat_mul(tmp, r, m, size);
    mat_cpy(r, tmp, size);
}

void vet_mul(vetor_t& r, const matriz_t& m, const vetor_t& v, int size) {
    for(int i=0;i<size;++i) {
        r[i] = 0;
        for(int j=0;j<size;++j) {
            r[i] = EVAL(r[i] + m[i][j] * v[j]);
        }
    }
}

void vet_mul(vetor_t& r, const matriz_t& m, int size) {
    vetor_t tmp;
    vet_mul(tmp, m, r, size);
    vet_cpy(r, tmp, size);
}

void mat_pow(matriz_t& r, int e, int size) {
    matriz_t b;
    mat_cpy(b, r, size);
    mat_ident(r, size);

    while(e>0) {
        if(e%2==1) {
            mat_mul(r, b, size);
        }
        mat_mul(b, b, size);
        e/=2;
    }
}

void mat_pow(matriz_t& r, const matriz_t& b, int e, int size) {
    mat_cpy(r, b, size);
    mat_pow(r, e, size);
}
```

Maxflow Dinitz

```
int grafo[MAXV];

int next[MAXE];
int vertice[MAXE];
int flow[MAXE];

int dist[MAXV];
int fila[MAXV];

bool bfs(int s, int d) {
    int ffila=0, ifila=0;
    memset(dist, -1, sizeof(dist));
    fila[ffila++] = s;
    dist[s] = 0;
    while(ifila != ffila) {
        int u = fila[ifila++];
        for(int e = grafo[u]; e != -1; e = next[e]) {
            int w = vertice[e];
            if(flow[e] > 0 and dist[w] == -1) {
                fila[ffila++] = w;
                dist[w] = dist[u] + 1;
            }
        }
    }
    return dist[d] != -1;
}

int nedge[MAXV];

int dfs(int v, int d, int mf = 1<<30) {
    if(v == d) return mf;
    int ret = 0;
    for(int& e = nedge[v]; e != -1; e = next[e]) {
        int w = vertice[e];
        if(flow[e] > 0 and dist[w] == dist[v] + 1) {
            int tmp = dfs(w, d, min(mf - ret, flow[e]));
            if(tmp > 0) {
                flow[e] -= tmp;
                flow[e^1] += tmp;
                ret += tmp;
                if(ret == mf) break;
            }
        }
    }
    return ret;
}

int max_flow(int s, int d) {
    int resp = 0;
    while(bfs(s, d)) {
        memcpy(nedge, grafo, sizeof(nedge));
        resp += dfs(s, d);
    }
    return resp;
}
```

Maxflow Edmonds Karp

```
int grafo[MAXV];

int next[MAXE];
int vertice[MAXE];
int flow[MAXE];

int edge_count;

int pai[MAXV];
int fila[MAXV];
int cflow[MAXV];

void _add_edge(int u, int v, int f) {
    int e = edge_count++;
    next[e] = grafo[u];
    vertice[e] = v;
    flow[e] = f;
    grafo[u] = e;
}

void add_edge(int u, int v, int f1, int f2 = 0) {
    _add_edge(u, v, f1);
    _add_edge(v, u, f2);
}

int max_flow(int source, int sink) {
    int resp = 0;
    while(1) {
        memset(pai, -1, sizeof(pai));
        int ifila = 0, ffila = 0;
        fila[ffila++] = source;
        pai[source] = -2;
        cflow[source] = 0x7fffffff;
        while(ifila != ffila and pai[sink] == -1) {
            int v = fila[ifila++];
            for(int e = grafo[v]; e != -1; e = next[e]) if(flow[e] > 0) {
                int u = vertice[e];
                if(pai[u] != -1) continue;
                pai[u] = e^1;
                fila[ffila++] = u;
                cflow[u] = min(cflow[v], flow[e]);
            }
        }
        if(pai[sink]==-1) break;
        int f = cflow[sink];
        for(int v = sink; pai[v] != -2; v = vertice[pai[v]]) {
            flow[pai[v]] += f;
            flow[pai[v]^1] -= f;
        }
        resp += f;
    }
    return resp;
}
```

Maxflow Mincost

```
#define INF 0x7f7f7f7f

#define MAXV 1000
#define MAXE 60000

int grafo[MAXV];

int next[MAXE];
int vertice[MAXE];
int flow[MAXE];
int cost[MAXE];

int fila[MAXV];
int nafila[MAXV];
int pai[MAXV];
int dist[MAXV];
int cflow[MAXV];

int edge_count;

void init() {
    memset(grafo, -1, sizeof(grafo));
    edge_count = -1;
}

void add_edge(int u, int v, int f1, int c) {
    int e1 = edge_count++;
    int e2 = edge_count++;

    next[e1] = grafo[u];
    vertice[e1] = v;
    flow[e1] = f1;
    cost[e1] = c;
    grafo[u] = e1;

    next[e2] = grafo[v];
    vertice[e2] = u;
    flow[e2] = 0;
    cost[e2] = -c;
    grafo[v] = e2;
}

int mincostflow(int source, int sink) {
    int resp = 0;
    while(1) {
        memset(nafila, 0, sizeof(nafila));
        memset(dist, 0x7f, sizeof(nafila));
        int ffila = 0, ifila = 0;
        fila[ffila++] = source;
        dist[source] = 0;
        nafila[source] = 1;
        cflow[source] = 1<<30;
        pai[source] = -1;
        while(ffila != ifila) {
            int u = fila[ifila];
            nafila[u] = 0;
            ifila = (ifila + 1)%MAXV;
            for(int e = grafo[u]; e != -1 ; e = next[e]) if(flow[e] > 0) {
                int v = vertice[e];
                int tmp = dist[u] + cost[e];
                if(tmp < dist[v]) {
                    dist[v] = tmp;
                    pai[v] = e^1;
                    cflow[v] = min(cflow[u], flow[e]);
                    if(nafila[v] == 0) {
                        fila[ffila] = v;
                        nafila[v] = 1;
                        ffila = (ffila + 1)%MAXV;
                    }
                }
            }
        }
        if(dist[sink] == INF) break;
        int f = cflow[sink];
        resp += dist[sink] * f;
        for(int u = sink; pai[u] != -1; u = vertice[pai[u]]) {
            flow[pai[u]^1] -= f;
            flow[pai[u]] += f;
        }
    }
    return resp;
}
```

Suffix Array

```
// Vetor de sufixos

// tam maximo do vetor de sufixos
// tome MAX maior que o tamanho do alfabeto
#define MAX 200010
//log do numero anterior
#define LMAX 20

char S[MAX]; //string que vai gerar o vetor
int n; // n = strlen(S);

int P[LMAX][MAX]; // P[j][i] eh o "balde" onde o sufixo comecado em i
                  // estarah quando estivermos considerando tam 2^j
int lim; //variavel interna
int sar[MAX]; // o vetor de sufixos

// struct usado numa lista ligada interna
struct node{
  int v;
  node *prox;
};

// lista livre
node buf[3*MAX];
int bufp;

node *lista[2][MAX];


inline int insere(int k, int i, int v)
{
  node *novo = &buf[bufp++];
  novo->v = v;
  novo->prox = lista[k][i];
  lista[k][i] = novo;
}

void monta()
{
  // mudar isso se o seu alfabeto for diferente
  for(int i = 0; i < n; i++)
    P[0][i] = S[i] - 'a' + 1;

  int pot = 1;
  bool para = false;
  int cnt = 30; //aqui so comecamos com as letras minusculas
  for(lim = 1; !para; lim++){

    for(int i = 0; i <= cnt; i++)
      lista[0][i] = lista[1][i] = NULL;
    bufp = 0;

    for(int i = 0; i < n; i++){
      if(i + pot >= n)
        insere(0, 0, i);
      else
        insere(0, P[lim-1][i+pot], i);
    }

    for(int i = cnt; i >= 0; i--)
      for(node *it = lista[0][i]; it != NULL; it = it->prox)
        insere(1, P[lim-1][it->v], it->v);

    para = true;
    int ncnt = 0;
    for(int i = 0; i <= cnt; i++){
      int ult = -1;
      for(node *it = lista[1][i]; it != NULL; it = it->prox){
        int at = ((it->v+pot>=n) ? 0 : P[lim-1][it->v + pot]);
        if(at != ult)
          ncnt++;
        else
          para = false;
        P[lim][it->v] = ncnt;
        ult = at;
      }
    }
    if(!para)
      cnt = ncnt;
    pot <<= 1;
  }

  int k = 0;
  for(int i = 0; i <= cnt; i++)
    for(node *it = lista[1][i]; it != NULL; it = it->prox)
      sar[k++] = it->v;
}


// devolve o tamanho do maior prefixo comum
// entre S[i .. n-1] e S[j .. n-1]
int iguais(int i, int j)
{
  int k = lim - 1;
  int r = 0;
  while(k >= 0 && i < n && j < n){
    if(P[k][i] == P[k][j]){
      r += (1 << k);
      i += (1 << k);
      j += (1 << k);
    }
    k--;
  }
  return r;
}
```

Mincut Stoer-Wagner

```
int grafo[MAXTAM][MAXTAM];
bool visitado[MAXTAM];
int cost[MAXTAM];

/* O(N^3) */
int mincut(int N) {
    int resp = INF;
    bzero(cost, sizeof(cost));
    for(;N>1;--N) {
        int cut = 0;
        bzero(visitado, sizeof(visitado[0])*N);
        int l1=0, l2=0;
        for(int i=0;i<N;++i) {
            int v=-1, c=-1;
            for(int u=0;u<N;++u) if(not visitado[u] and cost[u]>c)
                c=cost[v=u];
            visitado[v] = true;
            cut -= cost[v];
            cost[v]=0;
            for(int u=0;u<N;++u) if(not visitado[u]) {
                cost[u] += grafo[u][v];
                cut += grafo[u][v];
            }
            l2=l1; l1=v;
            if(i==N-2 and cut<resp)
                resp=cut;
        }
        grafo[l1][l2]=grafo[l2][l1]=0;
        for(int i=0;i<N;++i) {
            grafo[i][l1] = (grafo[l1][i] += grafo[l2][i]);
            grafo[l2][i] = grafo[i][l2] = 0;
        }
        for(int i=0;i<N;++i) {
            grafo[i][l2] = grafo[l2][i] = grafo[N-1][i];
        }
    }
    return resp;
}
```

Triângulo de pascal

```
#define PASCAL_LINES 100
#define PASCAL_COLUMNS 100

long long comb[PASCAL_LINES][PASCAL_COLUMNS];

void pascal(void) {
    comb[0][0]=1;
    for(int i=1;i<PASCAL_COLUMNS;++i)
        comb[0][i]=0;
    for(int i=1;i<PASCAL_LINES;++i) {
        comb[i][0]=1;
        for(int j=1;j<PASCAL_COLUMNS;++j)
            comb[i][j]=(comb[i-1][j-1]+comb[i-1][j]);
    }
}
```

Polinômio

```
struct Polinomio {
    double coefs[MAX_POL];

    Polinomio() { clear(); }

    double& operator[](int n) { return coefs[n]; }
    const double& operator[](int n) const { return coefs[n]; }

    void clear() { for(int i=0;i<MAX_POL;++i) coefs[i] = 0; }

    Polinomio operator+(const Polinomio& p) const {
        Polinomio tmp;
        for(int i=0;i<MAX_POL;++i) tmp[i] = coefs[i] + p[i];
        return tmp;
    }
    Polinomio& operator+=(const Polinomio& p) {
        *this = *this + p;
        return *this;
    }

    Polinomio operator-(const Polinomio& p) const {
        Polinomio tmp;
        for(int i=0;i<MAX_POL;++i) tmp[i] = coefs[i] - p[i];
        return tmp;
    }
    Polinomio& operator-=(const Polinomio& p) {
        *this = *this - p;
        return *this;
    }

    Polinomio operator*(const Polinomio& p) const {
        Polinomio tmp;
        for(int i=0;i<MAX_POL;++i) for(int j=0;i+j<MAX_POL;++j) {
            tmp[i+j] += coefs[i]*p[j];
        }
        return tmp;
    }

    pair<Polinomio, Polinomio> operator/(const Polinomio& p2) const {
        Polinomio p1 = *this, resp;
        int g1=p1.grau(), g2=p2.grau();
        for(;g1>=g2;--g1) {
            Polinomio tmp;
            resp[g1-g2] = tmp[g1-g2] = p1[g1] / p2[g2];
            tmp = tmp * p2;
            p1 -= tmp;
        }
        return make_pair(resp, p1);
    }
    Polinomio derivate() const {
        Polinomio tmp;
        for(int i=1;i<MAX_POL;++i) {
            tmp[i-1] = coefs[i]*i;
        }
        return tmp;
    }
    int grau() const {
        int g;
        for(g=MAX_POL-1;abs(coefs[g])<1e-8 and g>0;--g);
        return g;
    }
};
```

Prim

```
int grafo[MAXV];
int in_tree[MAXV];

int next[MAXE];
int vertice[MAXE];
int cost[MAXE];

typedef pair<int, int> pii;

int agm(int u, int v) {

    priority_queue<pii, vector<pii>, greater<pii> > fila;
    fila.push(pii(0, u));
    memset(menor, 0x7f, sizeof(menor));
    memset(in_tree, 0, sizeof(in_tree));
    menor[u] = 0;

    int cost = 0;

    while(not fila.empty()) {
        int w = fila.top().second;
        if(in_tree[w]) continue;
        in_tree[w] = 1;
        cost += menor[w];
        fila.pop();
        for(int e = grafo[w]; e != -1; e = next[e]) {
            int x = vertice[e];
            if(cost[e] < menor[x]) {
                menor[x] = cost[e];
                fila.push(pii(cost[e], x));
            }
        }
    }

    return cost;
}
```

Sistema linear

```
#define MAXTAM 32

double matriz[MAXTAM][MAXTAM];
double variables[MAXTAM];

void solve(int N) {
    for(int i=0;i<N;++i) {
        int m = i;
        for(int j=i+1;j<N;++j) {
            if(abs(matriz[j][i]) > abs(matriz[m][i]))
                m = j;
        }
        for(int j=i;j<=N;++j) {
            swap(matriz[i][j], matriz[m][j]);
        }
        for(int j=i+1;j<N;++j) {
            if(abs(matriz[j][i])<1e-9) continue;
            double f = matriz[j][i]/matriz[i][i];
            for(int k=i;k<=N;++k) {
                matriz[j][k] -= matriz[i][k] * f;
            }
        }
    }
    variables[N] = 1.0;
    for(int i=N-1;i>=0;--i) {
        double soma = 0.0;
        for(int j=i+1;j<=N;++j) {
            soma += matriz[i][j] * variables[j];
        }
        variables[i] = -soma / matriz[i][i];
    }
}
```

Union-Find

```
#define MAX_ELEMENTS 256

int sets[MAX_ELEMENTS];
int set_count;

void init(int element_count) {
    set_count = element_count;
    for(int i = 0; i < element_count; ++i) {
        sets[i] = i;
    }
}

int find(int a) { return (sets[a]==a)?(a):(sets[a]=find(sets[a])); }

bool merge(int a, int b) {
    a = find(a), b = find(b);
    if(a != b) {
        --set_count;
        sets[a] = b;
    }
    return a != b;
}
```

Raiz Polinomio

```
bool eh_raiz_inteira(const vector<int>& polinomio, int x) {
    ll tmp=0;
    if(x==0)
        return polinomio[0]==0;
    for(int i = 0;i<sz(polinomio);++i) {
        tmp += polinomio[i];
        if(tmp%x!=0)
            return false;
        tmp /= x;
    }
    return tmp==0;
}
```

KMP

```
void kmp_pre(int* table, char* needle) {
    table[0] = -1;

    for(int i = 1; needle[i]; ++i) {
        int p = table[i-1];
        while(p >= 0 and needle[i] != needle[p+1]) {
            p = table[p];
        }
        if(needle[i] == needle[p+1]) {
            table[i] = p + 1;
        }
        else table[i] = -1;
    }
}

void kmp(char* haystack, char* needle, int* table) {

    int p = -1;
    for(int i = 0; haystack[i]; ++i) {
        while(p >= 0 and needle[p+1] != haystack[i]) {
            p = table[p];
        }
        if(needle[p+1] == haystack[i]) {
            ++p;
        }
        if(needle[p+1] == 0) {
            printf("%d\n", i - p);
        }
    }
}
```

Pontes

```
#define MAX 50010

vector<int> G[MAX];
int n;

int lbl[MAX], low[MAX];
int tempo;

void dfs(int i, int p)
{
  lbl[i] = tempo++;
  low[i] = lbl[i];

  for(int k = 0; k < (int)G[i].size(); k++){
    int j = G[i][k];
    if(j == p) continue;
    if(lbl[j] == -1){
      dfs(j, i);
      if(low[j] >= lbl[j]){
        printf("A aresta entre %d e %d eh uma ponte.\n", i, j);
      }
      low[i] = min(low[i], low[j]);
    } else{
      low[i] = min(low[i], lbl[j]);
    }
  }
}

// Supoe que G seja um grafo conexo
void imprime_pontes()
{
  tempo = 0;
  memset(lbl, -1, sizeof lbl);
  dfs(0, -1);
}
```

Componentes fortemente conexas

```
//numero maximo de vertices
#define MAX 5010

// grafo original, o seu transposto e numero de vertices
vector<int> G[MAX];
vector<int> Ginv[MAX];
int n;


int numcomp;
int comp[MAX];

int tempo;
int final[MAX];
char vis[MAX];

void dfs1(int i)
{
  if(vis[i]) return;
  vis[i] = 1;
  for(int k = 0; k < (int)G[i].size(); k++)
    dfs1(G[i][k]);
  final[tempo++] = i;
}

void dfs2(int i, int cp)
{
  if(comp[i] != -1)
    return;
  comp[i] = cp;

  for(int k = 0; k < (int)Ginv[i].size(); k++)
    dfs2(Ginv[i][k], cp);
}

void encontra_componentes()
{
  memset(vis, 0, sizeof vis);
  memset(comp, -1, sizeof comp);


  tempo = 0;
  for(int i = 0; i < n; i++)
    dfs1(i);

  numcomp = 0;
  for(int i = n-1; i >= 0; i--){
    int j = final[i];
    if(comp[j] == -1)
      dfs2(j, numcomp++);
  }
}
```

Primos grandes

```
1000000000000000003 1000000000000000009 1000000000000000031
1000000000000000079 1000000000000000177 1000000000000000183
1000000000000000201 1000000000000000283 1000000000000000381
1000000000000000387 1000000000000000507 1000000000000000523
1000000000000000583 1000000000000000603 1000000000000000619
1000000000000000621 1000000000000000799 1000000000000000841
1000000000000000861 1000000000000000877 1000000000000000913
1000000000000000931 1000000000000000997 1000000000000001093
1000000000000001191 1000000000000001267 1000000000000001323
1000000000000001347 1000000000000001359 1000000000000001453
1000000000000001459 1000000000000001537 1000000000000001563
1000000000000001593 1000000000000001659 1000000000000001683
1000000000000001729 1000000000000001743 1000000000000001771
1000000000000001827 1000000000000001879 1000000000000001953
1000000000000002049 1000000000000002097 1000000000000002137
1000000000000002217 1000000000000002271 1000000000000002319
1000000000000002481 1000000000000002493 1000000000000002497
1000000000000002509 1000000000000002517 1000000000000002539
1000000000000002557 1000000000000002587 1000000000000002607
1000000000000002611 1000000000000002679 1000000000000002851
1000000000000002857 1000000000000002901 1000000000000002907
1000000000000002931 1000000000000002961 1000000000000003111
1000000000000003129 1000000000000003139 1000000000000003187
1000000000000003201 1000000000000003207 1000000000000003237
1000000000000003271 1000000000000003279 1000000000000003337
1000000000000003339 1000000000000003423 1000000000000003489
1000000000000003493 1000000000000003529 1000000000000003621
1000000000000003643 1000000000000003661 1000000000000003669
1000000000000003733 1000000000000003811 1000000000000003813
1000000000000003973 1000000000000004071 1000000000000004083
1000000000000004117 1000000000000004137 1000000000000004189
1000000000000004209 1000000000000004231 1000000000000004317
1000000000000004383 1000000000000004387 1000000000000004413
1000000000000004447
```

Primos

```
1000000007
1000000009 1000000021 1000000033
1000000087 1000000093 1000000097
1000000103 1000000123 1000000181
1000000207 1000000223 1000000241
1000000271 1000000289 1000000297
1000000321 1000000349 1000000363
1000000403 1000000409 1000000411
1000000427 1000000433 1000000439
1000000447 1000000453 1000000459
1000000483 1000000513 1000000531
1000000579 1000000607 1000000613
1000000637 1000000663 1000000711
1000000753 1000000787 1000000801
1000000829 1000000861 1000000871
1000000891 1000000901 1000000919
1000000931 1000000933 1000000993
1000001011 1000001021 1000001053
1000001087 1000001099 1000001137
1000001161 1000001203 1000001213
1000001237 1000001263 1000001269
1000001273 1000001279 1000001311
1000001329 1000001333 1000001351
1000001371 1000001393 1000001413
1000001447 1000001449 1000001491
1000001501 1000001531 1000001537
1000001539 1000001581 1000001617
1000001621 1000001633 1000001647
1000001663 1000001677 1000001699
1000001759 1000001773 1000001789
1000001791 1000001801 1000001803
1000001819 1000001857 1000001887
1000001917 1000001927 1000001957
1000001963 1000001969 1000002043
1000002044
```

Inteiros Grandes

```
/* ÚNMEROS GRANDES
 * Contem rotinas para çõoperaes com únmeros grandes.
 * Os únmeros ãsero representados em base 10000.
 * Os digitos menos significativos ãvo aparecer nas
 * primeiras posicoes do vetor.
 * Ex: 15000 => num[0] = 5000, num[1] = 1
 */
#include <cstdio>
#include <cstdlib>
#include <cstring>
#define MAX 100 /* (Numero de digitos)/4 */

/* STRING_TO_NUM
 * Converte um únmero de string para inteiro
 * âParmetros:
 * -entrada: úNmero em forma de string
 * -num: Vetor onde o únmero convertido áficar guardado
 * íSada:
 * -ndig: úNmero de çõposies ocupadas pelo únmero em num (base 10000)
 */
int stringToNum(char *entrada, int num[100]) {
  int i,j,k,tam;
  int ndig;

  tam = strlen(entrada);
  ndig = 0;

  for (i = tam-1; i >= 0; i--) {
    num[ndig] = 0;
    k = 1;
    for (j = 0; j < 4 && i >= 0; j++,i--) {
      num[ndig] = num[ndig] + k*(entrada[i]-'0');
      k *= 10;
    }
    i++;
    ndig++;
  }

  /* Tirar 0s à esquerda */
  while(ndig >= 1 && num[ndig-1] == 0)
    ndig--;

  /* Caso ópatolgico... */
  if (ndig == 0) {
    num[0] = 0;
    ndig = 1;
  }

  return ndig;
}

/* IMPRIME
 * Imprime um únmero grande
 * âParmetros:
 * -a: Vetor com o únmero grande
 * -tam: çõPosies de a ocupadas pelo únmero
 */
void imprime(int a[MAX], int tam) {
  int i = tam-1;
  printf("%d",a[i]);
  for (i--; i>= 0; i--) {
    printf("%04d",a[i]);
  }
  return;
}

/* SOMA
 * Calcula a soma de dois inteiros grandes
 * âParmetros:
 * -a,tama: Vetor com o º1 únmero e seu tamanho
 * -b,tamb: Vetor com o º2 únmero e seu tamanho
 * -c: Vetor que recebe o resultado
 * íSada:
 * -Devolve o tamanho do vetor c
 */
int soma(int a[MAX], int tama, int b[MAX], int tamb, int c[MAX]) {
  int i,j,cont = 0, tamc;

  for (i = 0; i < tama && i < tamb; i++) {
    c[i] = a[i]+b[i]+cont;
    cont = c[i]/10000;
    c[i] = c[i]%10000;
  }
  while (i < tama) {
    c[i] = a[i]+cont;
    cont = c[i]/10000;
    c[i] = c[i]%10000;
    i++;
  }
  while (i < tamb) {
    c[i] = b[i]+cont;
    cont = c[i]/10000;
    c[i] = c[i]%10000;
    i++;
  }
  if (cont)
    c[i++] = cont;
  tamc = i;
  return tamc;
}

/* SOMA2
 * Calcula a soma de dois inteiros grandes e guarda no primeiro
 * âParmetros:
 * -a,tama: Vetor com o º1 únmero e seu tamanho
 * -b,tamb: Vetor com o º2 únmero e seu tamanho
 * íSada:
 * -Devolve o tamanho do vetor c
 */
int soma2(int a[MAX], int tama, int b[MAX], int tamb) {
  int i,j,cont = 0, tamc;

  for (i = 0; i < tama && i < tamb; i++) {
    a[i] = a[i]+b[i]+cont;
    cont = a[i]/10000;
    a[i] = a[i]%10000;
  }
  while (i < tama) {
    a[i] = a[i]+cont;
    cont = a[i]/10000;
    a[i] = a[i]%10000;
    i++;
  }
  while (i < tamb) {
    a[i] = b[i]+cont;
    cont = a[i]/10000;
    a[i] = a[i]%10000;
    i++;
  }
  if (cont)
    a[i++] = cont;
```

```
  tamc = i;
  return tamc;
}



/* MULT
 * Calcula a úçãmltiplicao de dois inteiros grandes
 * âParmetros:
 * -a,ta: Vetor com o º1 únmero e seu tamanho
 * -b,tb: Vetor com o º2 únmero e seu tamanho
 * -c: Vetor que recebe o resultado
 * íSada:
 * -Devolve o tamanho do vetor c
 */
int mult(int a[MAX], int ta, int b[MAX], int tb, int c[MAX]) {
  int i, j, k, tc, aux,resto;

  /* Zera o vetor da resposta */
  tc = ta + tb + 1;
  for (i = 0; i < tc; i++)
    c[i] = 0;

  resto = 0;
  for (i = 0; i < ta; i++) {
    for (j = 0; j < tb; j++) {
      aux = a[i]*b[j] + resto + c[i+j];
      c[i+j] = (aux)%10000;
      resto = (aux)/10000;
    }
    j = i + j;
    while(resto) {
      aux = resto + c[j];
      c[j] = (aux)%10000;
      resto = (aux)/10000;
      j++;
    }
  }
  if (resto)
    c[tc-1] = resto;

  /* Retira os zeros à esquerda */
  i = tc - 1;
  while(c[i] == 0 && i > 0)
    i--;

  return i + 1;
}


/* DIVIDE
 * Calcula a ãdiviso de um inteiro grande por um inteiro pequeno
 * âParmetros:
 * -a,tama: Vetor com o únmero grande e seu tamanho
 * -divi: Inteiro pequeno que vai dividir o inteiro grande
 * íSada:
 * -Devolve o tamanha do vetor a, que guarda o resultado da ãdiviso
 */
int divide(int a[MAX], int tama, int divi) {
  int i,j,k,l,tam, resto, quoc;
  char num[10000], resp[10000];

  i = tama -1;
  sprintf(num,"%d",a[i]);
  tam = strlen(num);
  for (i--; i>= 0; i--) {
    sprintf(num+tam,"%04d",a[i]);
    tam = strlen(num);
  }

  resp[0] = 0;
  j = 0;
  resto = 0;
  quoc = num[0]-'0';
  i = 1;
  while(i < tam && quoc < divi) {
    quoc = 10*quoc + num[i]-'0';
    i++;
  }

  while(i <= tam) {
    resp[j] = ((quoc/divi))%10 + '0';
    j++;
    quoc = quoc%divi;
    quoc = 10*quoc + num[i]-'0';
    i++;
  }
  resp[j] = '\0';
  tama =  stringToNum(resp,a);
  return tama;
}



/* COMPARA
 * Compara dois únmeros grandes
 * âParmetros:
 * -a,tama: Vetor com o º1 únmero e seu tamanho
 * -b,tamb: Vetor com o º2 únmero e seu tamanho
 * íSada:
 * -a > b:  1
 * -a < b: -1
 * -a = b:  0
 */
int compara(int *a, int tama, int *b, int tamb) {
  int i;
  if (tama > tamb)
    return 1;
  else if (tama < tamb)
    return -1;
  for (i = tama-1; i>= 0; i--) {
    if (a[i] > b[i])
      return 1;
    else if (a[i] < b[i])
      return -1;
  }
  return 0;
}



/* SUBTRAI
 * Subtrai o primeiro únmero do segundo (o primeiro deve ser maior).
 * âParmetros:
 * -a,ta: Vetor com o º1 únmero e seu tamanho
 * -b,tb: Vetor com o º2 únmero e seu tamanho
 * -c: Vetor que recebe o resultado
 * íSada:
 * -Devolve o tamanho do vetor c
```

```
 */
int subtrai(int *a, int tama, int *b, int tamb,int *c) {
  int tamc;
  int i,j,cont = 0;
  for (i = j = 0; i < tama && j < tamb; i++,j++) {
    c[i] = a[i]-b[j]-cont;
    if (c[i] < 0) {
      cont = 1;
      c[i] += 10000;
    }
    else
      cont = 0;
  }
  while (i < tama) {
    c[i] = a[i]-cont;
    if (c[i] < 0) {
      cont = 1;
      c[i] += 10000;
    }
    i++;
  }
  tamc = i;
  while(tamc > 1 && c[tamc-1] == 0)
    tamc--;
  return tamc;
}
```

```
int main() {
  char entrada[10000];
  int n1[100], t1;
  int n2[100], t2;
  int n3[100], t3;
  int k;

  while(1) {
    /* Leitura dos únmeros e ãconverso */
    scanf("%s",entrada);
    t1 = stringToNum(entrada,n1);
    scanf("%s",entrada);
    t2 = stringToNum(entrada,n2);

    /* Soma dos únmeros */
    t3 = soma(n1,t1,n2,t2,n3);
    printf("A␣soma␣dos␣únmeros␣é␣");
    imprime(n3,t3);
    putchar('\n');

    /* Produto dos únmeros */
    t3 = mult(n1,t1,n2,t2,n3);
    printf("O␣produto␣dos␣únmeros␣é␣");
    imprime(n3,t3);
    putchar('\n');

    /* Compara os dois únmeros */
    k = compara(n1,t1,n2,t2);
    if (k == 1)
      printf("O␣primeiro␣é␣maior␣que␣o␣segundo\n");
    if (k == 0)
      printf("Os␣dois␣únmeros␣ãso␣iguais\n");
    if (k == -1)
      printf("O␣segundo␣é␣maior␣que␣o␣primeiro\n");

    /* çãSubtrao dos únmeros */
    if (k >= 0)
      t3 = subtrai(n1,t1,n2,t2,n3);
    else
      t3 = subtrai(n2,t2,n1,t1,n3);
    printf("A␣çdiferena␣dos␣únmeros␣é␣");
    imprime(n3,t3);
    putchar('\n');

    /* ãDiviso de únmeros */

    t1 = divide(n1,t1,n2[0]);

    printf("A␣ãdiviso␣dos␣únmeros␣é␣");
    imprime(n1,t1);
    putchar('\n');

  }
  return 0;
}
```