Standard Code Library

Ceno Liu

June, 2013

Contents

1 Algorithms and Datastructures **2**
1.1 *High Precision in C Plus Plus . . . 2
1.2 *Fraction Class . . . 2
1.3 Splay Tree . . . 2
1.4 Link/Cut Tree . . . 4
1.5 Binary Heap . . . 5
1.6 Leftist Tree . . . 6
1.7 Treap . . . 7
1.8 Segment Tree . . . 8
1.9 Heavy-Light Decomposition . . . 9
1.10 KD Tree . . . 10
1.11 Manacher . . . 12
1.12 Z Algorithm . . . 12
1.13 Aho-Corasick Automaton . . . 13
1.14 Suffix Array . . . 13
1.15 Suffix Automaton . . . 14
1.16 *Dancing Links . . . 15

2 Graph Theory and Network Algorithms **16**
2.1 Dijkstra . . . 16
2.2 *Minimum Directed Spanning Tree . . . 16
2.3 KuhnMunkres . . . 16
2.4 Maximum Flow . . . 17
2.5 Minimum Cost Maximum Flow . . . 18
2.6 Strongly Connected Component . . . 20
2.7 *2-SAT . . . 20

3 Number Theory **21**
3.1 Chinese Remainder Theorem . . . 21
3.2 Pollard's Rho and Miller-Rabbin . . . 21
3.3 *Baby-step Giant-step . . . 23

4 Algebraic Algorithms **24**
4.1 *Linear Equations in Z_m . . . 24
4.2 *Linear Equations in R . . . 24
4.3 *Fast Fourier Transform . . . 24

5 Computational Geometry **25**
5.1 Basic Operations . . . 25
5.2 Convex Hull . . . 26
5.3 Convex Diameter . . . 27
5.4 Convex Cut . . . 27

6 Classic Problems **28**
6.1 Nim Game . . . 28

Chapter 1

Algorithms and Datastructures

1.1 *High Precision in C Plus Plus

Comming Soon

1.2 *Fraction Class

Comming Soon

1.3 Splay Tree

注意初始化内存池和 null 节点，以及根据需要修改 update 和 relax，区间必须是 1-based

```
const int MAX_NODE = 50000 + 10;
const int INF = 2000000000;

struct Node *null;

struct Node
{
    int rev, add;
    int val, maxv, size;
    Node *ch[2], *p;

    void set(Node *t, int _d) {
        ch[_d] = t;
        t->p = this;
    }
    int dir() {
        return this == p->ch[1];
    }
    void update() {
        maxv = max(max(ch[0]->maxv, ch[1]->maxv), val);
        size = ch[0]->size + ch[1]->size + 1;
    }
    void relax() {
        if (add) {
            ch[0]->appAdd(add);
            ch[1]->appAdd(add);
            add = 0;
        }
        if (rev) {
            ch[0]->appRev();
            ch[1]->appRev();
            rev = false;
        }
    }
    void appAdd(int x) {
```

```
        if (this == null) return;
        add += x;
        val += x;
        maxv += x;
    }
    void appRev() {
        if (this == null) return;
        rev ^= true;
        swap(ch[0], ch[1]);
    }
};

Node nodePool[MAX_NODE], *curNode;

Node *newNode(int val = 0)
{
    Node *t = curNode++;
    t->maxv = t->val = val;
    t->rev = t->add = 0;
    t->size = 1;
    t->ch[0] = t->ch[1] = t->p = null;
    return t;
}

struct Splay
{
    Node *root;

    Splay() {
        root = newNode();
        root->set(newNode(), 0);
        root->update();
    }

    Splay(int *a, int N) { //sequence is 1-based
        root = build(a, 0, N + 1);
    }

    Node* build(int *a, int l, int r) {
        if (l > r) return null;
        int mid = l + r >> 1;
        Node *t = newNode(a[mid]);
        t->set(build(a, l, mid - 1), 0);
        t->set(build(a, mid + 1, r), 1);
        t->update();
        return t;
    }

    void rot(Node *t)
    {
        Node *p = t->p; int d = t->dir();
        p->relax(); t->relax();
        if (p == root) root = t;
        p->set(t->ch[!d], d);
        p->p->set(t, p->dir());
        t->set(p, !d);
        p->update();
    }

    void splay(Node *t, Node *f = null)
    {
        for(t->relax(); t->p != f; ) {
            if (t->p->p == f) rot(t);
```

```
                else t->dir() == t->p->dir() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
            }
            t->update();
        }

        Node* getKth(int k) {
            Node *t = root;
            int tmp;
            for( ; ; ) {
                t->relax();
                tmp = t->ch[0]->size + 1;
                if (tmp == k) return t;
                if (tmp < k) {
                    k -= tmp;
                    t = t->ch[1];
                } else
                    t = t->ch[0];
            }
        }

        //make range[l,r] root->ch[1]->ch[0]
        //make range[x+1,x] to add something after position x
        void getRng(int l, int r) {
            r += 2;
            Node *p = getKth(l);
            Node *q = getKth(r);
            splay(p); splay(q, p);
        }

        void addRng(int l, int r, int x) {
            getRng(l, r);
            root->ch[1]->ch[0]->appAdd(x);
        }

        void revRng(int l, int r) {
            getRng(l, r);
            root->ch[1]->ch[0]->appRev();
        }

        int maxvRng(int l, int r) {
            getRng(l, r);
            return root->ch[1]->ch[0]->maxv;
        }
};

void initNull()
{
    curNode = nodePool;
    null = curNode++;
    null->size = 0;
    null->maxv = -INF;
}
```

1.4 Link/Cut Tree

根据需求修改 Node 中的 relax 和 update 函数，修改 access，以及 Node 的构造函数，注意初始化内存池和 null 节点

```
struct Node
{
    Node *ch[2], *p;
    int isroot;
    bool dir();
    void set(Node*, bool);
```

```
    void update();
    void relax();
} *null;

void rot(Node *t)
{
    Node *p = t->p; bool d = t->dir();
    p->relax(); t->relax();
    p->set(t->ch[! d], d);
    if (p->isroot) t->p = p->p, swap(p->isroot, t->isroot);
    else p->p->set(t, p->dir());
    t->set(p, ! d);
    p->update();
}

void Splay(Node *t)
{
    for(t->relax(); ! t->isroot; ) {
        if (t->p->isroot) rot(t);
        else t->dir() == t->p->dir() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
    }
    t->update();
}

void Access(Node *t)
{
    for(Node *s = null; t != null; s = t, t = t->p) {
        Splay(t);
        t->ch[1]->isroot = true;
        s->isroot = false;
        t->ch[1] = s;
        t->update();
    }
}
bool Node::dir()
{
    return this == p->ch[1];
}
void Node::set(Node *t, bool _d)
{
    ch[_d] = t; t->p = this;
}
void Node::Update()
{

}
void Node::Relax()
{
    if (this == Null) return;

}
```

1.5 Binary Heap

双射堆，ind[v] 表示标号为 v 的节点在堆中的位置

```
const int MAX_V = 100000 + 10;
struct Heap
{
    int tot;
    int a[MAX_V], h[MAX_V], ind[MAX_V];
    void exchange(int i, int j) {
        swap(h[i], h[j]);
```

```
            swap(ind[h[i]], ind[h[j]]);
        }
        inline int val(int x) {
            return a[h[x]];
        }
        void fixUp(int x) {
            if (x / 2 && val(x / 2) < val(x))
                exchange(x, x / 2), fixUp(x / 2);
        }
        void fixDown(int x) {
            int p = x * 2; if (p > tot) return;
            if (p < tot && val(p + 1) > val(p)) ++p;
            if (val(p) > val(x))
                exchange(p, x), fixDown(p);
        }
        void Update(int i, int x) {
            a[i] = x;
            fixUp(ind[i]);
            fixDown(ind[i]);
        }
        int top() {
            return h[1];
        }
        void pop() {
            exchange(1, tot);
            --tot;
            fixDown(1);
        }
        void insert(int i, int x) {
            ++tot;
            h[tot] = i;
            ind[i] = tot;
            a[i] = x;
            fixUp(tot);
        }
} H;
```

1.6 Leftist Tree

没写 delete 操作，注意初始化内存池和 null 节点

```
struct Node
{
    int dis, val;
    Node *ch[2];
} *null;

Node* merge(Node *u, Node *v)
{
    if (u == null) return v;
    if (v == null) return u;
    if (u->val < v->val) swap(u, v);
    u->ch[1] = merge(u->ch[1], v);
    if (u->ch[1]->dis > u->ch[0]->dis)
        swap(u->ch[1], u->ch[0]);
    u->dis = u->ch[1]->dis + 1;
    return u;
}

Node* newNode(int w)
{
    Node *t = totNode++;
    t->ch[0] = t->ch[1] = null;
```

```
    t->val = w; t->dis = 0;
    return t;
}
```

1.7 Treap

包含 build, insert 和 erase，执行时注意初始化内存池和 null 节点

```
struct Node *null;

struct Node
{
    int key, val, size;
    Node *ch[2];
    Node() {
        key = INT_MAX;
        val = size = 0;
    }
    Node(int _val) {
        size = 1;
        val = _val;
        key = bigRand();
        ch[0] = ch[1] = null;
    }
    int bigRand() {
        return rand() * RAND_MAX + rand();
    }
    void update() {
        size = ch[0]->size + ch[1]->size + 1;
    }
};

struct Treap
{
    Node *root;
    Treap() {
        root = null;
    }
    void rot(Node *&t, int d) {
        Node *p = t->ch[d]; t->ch[d] = p->ch[!d]; p->ch[!d] = t;
        t->update(); p->update();
        t = p;
    }

    void insert(Node *&t, int x) {
        if (t == null) {
            t = new Node(x);
            return;
        }
        int dir = x >= t->val;
        insert(t->ch[dir], x);
        if (t->ch[dir]->key < t->key)
            rot(t, dir);
        else
            t->update();
    }

    void erase(Node *&t, int x) {
        if (t == null)
            return;
        if (t->val == x) {
            int dir = t->ch[1]->key < t->ch[0]->key;
            if (t->ch[dir] == null) {
```

```
                delete t;
                t = null;
                return;
            }
            rot(t, dir);
            erase(t->ch[!dir], x);
            t->update();
            return;
        }
        bool dir = x > t->val;
        erase(t->ch[dir], x);
        t->update();
    }

    void insert(int x) {
        insert(root, x);
    }

    void erase(int x) {
        erase(root, x);
    }
};
```

1.8 Segment Tree

包含建树和区间操作样例，没有写具体操作

```
struct Tree
{
    int l, r;
    Tree *ch[2];
    Tree() {}
    Tree(int _l, int _r, int *sqn) {
        l = _l; r = _r;
        if (l + 1 == r)
            return;
        int mid = l + r >> 1;
        ch[0] = new Tree(l, mid, sqn);
        ch[1] = new Tree(mid, r, sqn);
    }

    void insert(int p, int x) {
        if (p < l || p >= r)
            return;
        //some operations
        if (l + 1 == r)
            return;
        ch[0]->insert(p, x);
        ch[1]->insert(p, x);
    }

    int query(int _l, int _r, int x) {
        if (_r <= l || _l >= r)
            return 0;
        if (_l <= l && _r >= r)
            // return information in [l, r)
        //merge ch[0]->query(_l, _r, x), ch[1]->query(_l, _r, x) and return
    }
};
```

1.9 Heavy-Light Decomposition

包含 BFS 剖分过程，线段树部分视题目而定

```
struct Tree()
{

};

int father[MAX_N], size[MAX_N], depth[MAX_N];
int bfsOrd[MAX_N], pathId[MAX_N], ordInPath[MAX_N], sqn[MAX_N];
Tree *root[MAX_N];

void doBfs(int s)
{
    int *que = bfsOrd;
    int qh = 0, qt = 0;
    father[s] = -1; depth[s] = 0;

    for(que[qt++] = s; qh < qt; ) {
        int u = que[qh++];
        foreach(iter, adj[u]) {
            int v = *iter;
            if (v == father[u])
                continue;
            father[v] = u;
            depth[v] = depth[u] + 1;
            que[qt++] = v;
        }
    }
}

void doSplit()
{
    for(int i = N - 1; i >= 0; --i) {
        int u = bfsOrd[i];
        size[u] = 1;
        foreach(iter, adj[u]) {
            int v = *iter;
            if (v == father[u])
                continue;
            size[u] += size[v];
        }
    }

    memset(pathId, -1, sizeof pathId);
    for(int i = 0; i < N; ++i) {
        int top = bfsOrd[i];
        if (pathId[top] != -1)
            continue;

        int cnt = 0;
        for(int u = top; u != -1; ) {
            sqn[cnt] = val[u];
            ordInPath[u] = cnt;
            pathId[u] = top;
            ++cnt;

            int next = -1;
            foreach(iter, adj[u]) {
                int v = *iter;
                if (v == father[u])
                    continue;
                if (next < 0 || size[next] < size[v])
```

```
                        next = v;
                    }
                    u = next;
                }

                root[top] = new Tree(0, cnt, sqn);
        }
}

void prepare()
{
        doBfs(0);
        doSplit();
}
```

1.10 KD Tree

读入 N 个点，输出距离每个点的最近点。

```
const int MAX_N = 100000 + 10;
const int MAX_NODE = 200000 + 10;
const LL INF = 2000000000000000020LL;

int N;

struct Point
{
        int x, y, id;
};

LL dis(const Point &a, const Point &b)
{
        return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y) * (a.y - b.y);
}

struct Node
{
        Point p;
        int maxX, minX, maxY, minY;
        int l, r, d;
        Node *ch[2];
};

LL ret;
LL ans[MAX_N];
Node *root;
Point p[MAX_N], queryPoint;
Node *totNode, nodePool[MAX_NODE];

int cmpx(const Point &a, const Point &b)
{
        return a.x < b.x;
}
int cmpy(const Point &a, const Point &b)
{
        return a.y < b.y;
}

Node* newNode(int l, int r, Point p, int deep)
{
        Node *t = totNode ++;
        t->l = l; t->r = r;
        t->p = p; t->d = deep;
```

```
    t->maxX = t->minX = p.x;
    t->maxY = t->minY = p.y;
    return t;
}

void updateInfo(Node *t, Node *p)
{
    t->maxX = max(t->maxX, p->maxX);
    t->maxY = max(t->maxY, p->maxY);
    t->minX = min(t->minX, p->minX);
    t->minY = min(t->minY, p->minY);
}

Node* build(int l, int r, int deep)
{
    if (l == r) return NULL;
    if (deep & 1) sort(p + l, p + r, cmpx);
    else sort(p + l, p + r, cmpy);
    int mid = (l + r) >> 1;
    Node *t = newNode(l, r, p[mid], deep & 1);
    if (l + 1 == r) return t;
    t->ch[0] = build(l, mid, deep + 1);
    t->ch[1] = build(mid + 1, r, deep + 1);
    if (t->ch[0]) updateInfo(t, t->ch[0]);
    if (t->ch[1]) updateInfo(t, t->ch[1]);
    return t;
}

void updateAns(Point p)
{
    ret = min(ret, dis(p, queryPoint));
}

LL calc(Node *t, LL d)
{
    LL tmp;
    if (d) {
        if (queryPoint.x >= t->minX && queryPoint.x <= t->maxX) tmp = 0;
        else tmp = min(abs(queryPoint.x - t->maxX), abs(queryPoint.x - t->minX));
    } else {
        if (queryPoint.y >= t->minY && queryPoint.y <= t->maxY) tmp = 0;
        else tmp = min(abs(queryPoint.y - t->maxY), abs(queryPoint.y - t->minY));
    }
    return tmp * tmp;
}

void query(Node *t)
{
    if (t == NULL) return;
    if (t->p.id != queryPoint.id) updateAns(t->p);
    if (t->l + 1 == t->r) return;
    LL dl = t->ch[0] ? calc(t->ch[0], t->d) : INF;
    LL dr = t->ch[1] ? calc(t->ch[1], t->d) : INF;
    if (dl < dr) {
        query(t->ch[0]);
        if (ret > dr) query(t->ch[1]);
    } else {
        query(t->ch[1]);
        if (ret > dl) query(t->ch[0]);
    }
}

void solve()
```

```
{
    scanf("%d", &N);
    for(int i = 0; i < N; ++i) {
        scanf("%d%d", &p[i].x, &p[i].y);
        p[i].id = i;
    }
    totNode = nodePool;
    root = build(0, N, 1);

    for(int i = 0; i < N; ++i) {
        queryPoint = p[i];
        ret = INF;
        query(root);
        ans[p[i].id] = ret;
    }
    for(int i = 0; i < N; ++i)
        printf("%I64d\n", ans[i]);
}

int main()
{
    int T; scanf("%d", &T);
    for( ; T--; )
        solve();
    return 0;
}
```

1.11 Manacher

len[i] means the max palindrome length centered i/2
eg: cs = "abbacabbaddabbaae"
len = 1 0 1 4 1 0 1 0 9 0 1 0 1 4 1 0 1 0 1 10 1 0 1 0 1 4 1 0 1 2 1 0 1 0

```
void palindrome(char cs[], int len[], int n) {
    for (int i = 0; i < n * 2; ++i) {
        len[i] = 0;
    }
    for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
        while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j + 1) / 2])
            j++;
        len[i] = j;
        for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++) {
            len[i + k] = min(len[i - k], j - k);
        }
    }
}
```

1.12 Z Algorithm

传入字符串 s 和长度 N，next[i]=LCP(s, s[i..N-1])

```
void z(char *s, int *next, int N)
{
    int j = 0, k = 1;
    while (j + 1 < N && s[j] == s[j + 1]) ++j;
    next[0] = N - 1; next[1] = j;
    for(int i = 2; i < N; ++i) {
        int far = k + next[k] - 1, L = next[i - k];
        if (L < far - i + 1) next[i] = L;
        else {
            j = max(0, far - i + 1);
```

```
                while (i + j < N && s[j] == s[i + j]) ++ j;
                next[i] = j; k = i;
            }
        }
}
```

1.13 Aho-Corasick Automaton

包含建 trie 和构造自动机的过程

```

struct acNode
{
    int id;
    acNode *ch[26], *fail;
} *totNode, *root, nodePool[MAX_V];

acNode* newNode()
{
    acNode *now = totNode ++;
    now->id = 0; now->fail = 0;
    memset(now->ch, 0, sizeof now->ch);
    return now;
}

void acInsert(char *c, int id)
{
    acNode *cur = root;
    while (*c) {
        int p = *c - 'A'; //change the index
        if (! cur->ch[p]) cur->ch[p] = newNode();
        cur = cur->ch[p];
        ++ c;
    }
    cur->id = id;
}

void getFail()
{
    acNode *cur;
    queue<acNode*> Q;
    for(int i = 0; i < 26; ++ i)
        if (root->ch[i]) {
            root->ch[i]->fail = root;
            Q.push(root->ch[i]);
        } else root->ch[i] = root;
    while (! Q.empty()) {
        cur = Q.front(); Q.pop();
        for(int i = 0; i < 26; ++ i)
            if (cur->ch[i]) {
                cur->ch[i]->fail = cur->fail->ch[i];
                Q.push(cur->ch[i]);
            } else cur->ch[i] = cur->fail->ch[i];
    }
}
```

1.14 Suffix Array

对于串 a 求 SA，长度为 N，M 为元素值范围，height[i]=LCP(suf[rank[i]],suf[rank[i]-1])

```
const int MAX_N = 1000000 + 10;

```

```
int rank[MAX_N], height[MAX_N];

int cmp(int *x, int a, int b, int d)
{
    return x[a] == x[b] && x[a + d] == x[b + d];
}

void doubling(int *a, int N, int M)
{
    static int sRank[MAX_N], tmpA[MAX_N], tmpB[MAX_N];
    int *x = tmpA, *y = tmpB;
    for(int i = 0; i < M; ++ i) sRank[i] = 0;
    for(int i = 0; i < N; ++ i) ++ sRank[x[i] = a[i]];
    for(int i = 1; i < M; ++ i) sRank[i] += sRank[i - 1];
    for(int i = N - 1; i >= 0; -- i) sa[-- sRank[x[i]]] = i;

    for(int d = 1, p = 0; p < N; M = p, d <<= 1) {
        p = 0; for(int i = N - d; i < N; ++ i) y[p ++] = i;
        for(int i = 0; i < N; ++ i)
            if (sa[i] >= d) y[p ++] = sa[i] - d;
        for(int i = 0; i < M; ++ i) sRank[i] = 0;
        for(int i = 0; i < N; ++ i) ++ sRank[x[i]];
        for(int i = 1; i < M; ++ i) sRank[i] += sRank[i - 1];
        for(int i = N - 1; i >= 0; -- i) sa[-- sRank[x[y[i]]]] = y[i];
        swap(x, y); x[sa[0]] = 0; p = 1;
        for(int i = 1; i < N; ++ i)
            x[sa[i]] = cmp(y, sa[i], sa[i - 1], d) ? p - 1 : p ++;
    }
}

void calcHeight()
{
    for(int i = 0; i < N; ++ i) rank[sa[i]] = i;
    int cur = 0;
    for(int i = 0; i < N; ++ i)
        if (rank[i]) {
            if (cur) cur --;
            for( ; a[i + cur] == a[sa[rank[i] - 1] + cur]; ++ cur);
            height[rank[i]] = cur;
        }
}
```

1.15 Suffix Automaton

......保重

```
struct State
{
    int val;
    State *suf, *go[26];
} *root, *last;

State statePool[MAX_N], *curState;

void extend(int w)
{
    State *p = last, *np = curState ++;
    np->val = p->val + 1;
    for( ; p && ! p->go[w]; p = p->suf)
        p->go[w] = np;
    if (! p)
        np->suf = root;
    else {
```

```
        State *q = p->go[w];
        if (q->val == p->val + 1)
            np->suf = q;
        else {
            State *nq = curState++;
            nq->val = p->val + 1;
            memcpy(nq->go, q->go, sizeof q->go);
            nq->suf = q->suf;
            q->suf = np->suf = nq;
            for( ; p && p->go[w] == q; p = p->suf)
                p->go[w] = nq;
        }
    }
    last = np;
}
```

1.16 *Dancing Links

Comming Soon

Chapter 2

Graph Theory and Network Algorithms

2.1 Dijkstra

求 s 到其他点的最短路

```
int used[MAX_N], dis[MAX_N];
void dijstra(int s) {
    fill(dis, dis + N, INF); dis[s] = 0;
    priority_queue<pair<int, int> > que;
    que.push(make_pair(-dis[s], s));
    while (!que.empty()) {
        int u = que.top().second; que.pop();
        if (used[u]) continue;
        used[u] = true;
        foreach(e, E[u])
            if (dis[u] + e->w < dis[e->t]) {
                dis[e->t] = dis[u] + e->w;
                que.push(make_pair(-dis[e->t], e->t));
            }
    }
}
```

2.2 *Minimum Directed Spanning Tree

Comming Soon

2.3 KuhnMunkres

求完备匹配的最大权匹配，建好的完全图用 w[][] 存储，点数为 N

```
const int MAX_N = 200 + 10;
const int INF = 1000000000;

int N, flag;
int w[MAX_N][MAX_N];
int fx[MAX_N], fy[MAX_N], lx[MAX_N], ly[MAX_N], slk[MAX_N], mat[MAX_N];

int DFS(int x) {
    fx[x] = flag;
    for(int i = 1; i <= N; ++i)
        if (lx[x] + ly[i] != w[x][i])
            slk[i] = min(slk[i], lx[x] + ly[i] - w[x][i]);
        else if (fy[i] != flag) {
            fy[i] = flag;
            if (mat[i] < 0 || DFS(mat[i])) {
                mat[i] = x;
                return true;
            }
```

```
        }
    return false;
}

int KM() {
    for(int i = 1; i <= N; ++ i) {
        mat[i] = -1;
        fx[i] = 0; fy[i] = 0;
        ly[i] = 0; lx[i] = -INF;
        for(int j = 1; j <= N; ++ j)
            lx[i] = max(lx[i], w[i][j]);
    }
    for(int now = 1; now <= N; ++ now) {
        ++ flag; for(int i = 1; i <= N; ++ i) slk[i] = INF;
        while (! DFS(now)) {
            int p(INF); for(int i = 1; i <= N; ++ i)
                if (fy[i] != flag) p = min(p, slk[i]);
            for(int i = 1; i <= N; ++ i) {
                if (fx[i] == flag) lx[i] -= p;
                if (fy[i] == flag) ly[i] += p;
                slk[i] = INF;
            }
            ++ flag;
        }
    }
    long long ans = 0;
    for(int i = 1; i <= N; ++ i) ans += lx[i], ans += ly[i];
    return ans;
}
```

2.4 Maximum Flow

iSAP 算法求 S 到 T 的最大流，点数为 cntN，边表存储在 *E[] 中

```
struct Edge
{
    int t, c;
    Edge *n, *r;
} *E[MAX_V], edges[MAX_M], *totEdge;

Edge* makeEdge(int s, int t, int c)
{
    Edge *e = totEdge ++;
    e->t = t; e->c = c; e->n = E[s];
    return E[s] = e;
}

void addEdge(int s, int t, int c)
{
    Edge *p = makeEdge(s, t, c), *q = makeEdge(t, s, 0);
    p->r = q; q->r = p;
}

int maxflow()
{
    static int  cnt     [MAX_V];
    static int  h       [MAX_V];
    static int  que     [MAX_V];
    static int  aug     [MAX_V];
    static Edge *cur    [MAX_V];
    static Edge *prev   [MAX_V];
    fill(h, h + cntN, cntN);
    memset(cnt, 0, sizeof cnt);
```

```
    int qt = 0, qh = 0; h[T] = 0;
    for(que[qt ++] = T; qh < qt; ) {
        int u = que[qh ++];
        ++ cnt[h[u]];
        for(Edge *e = E[u]; e; e = e->n)
            if (e->r->c && h[e->t] == cntN) {
                h[e->t] = h[u] + 1;
                que[qt ++] = e->t;
            }
    }
    memcpy(cur, E, sizeof E);
    aug[S] = INF; Edge *e;
    int flow = 0, u = S;
    while (h[S] < cntN) {
        for(e = cur[u]; e; e = e->n)
            if (e->c && h[e->t] + 1 == h[u])
                break;
        if (e) {
            int v = e->t;
            cur[u] = prev[v] = e;
            aug[v] = min(aug[u], e->c);
            if ((u = v) == T) {
                int by = aug[T];
                while (u != S) {
                    Edge *p = prev[u];
                    p->c -= by;
                    p->r->c += by;
                    u = p->r->t;
                }
                flow += by;
            }
        } else {
            if (!-- cnt[h[u]]) return flow;
            h[u] = cntN;
            for(e = E[u]; e; e = e->n)
                if (e->c && h[u] > h[e->t] + 1)
                    h[u] = h[e->t] + 1, cur[u] = e;
            ++ cnt[h[u]];
            if (u != S) u = prev[u]->r->t;
        }
    }
    return flow;
}
```

2.5 Minimum Cost Maximum Flow

注意图的初始化，费用和流的类型依题目而定

```
int flow, cost;

struct Edge
{
    int t, c, w;
    Edge *n, *r;
} *totEdge, edges[MAX_M], *E[MAX_V];

Edge* makeEdge(int s, int t, int c, int w)
{
    Edge *e = totEdge ++;
    e->t = t; e->c = c; e->w = w; e->n = E[s];
    return E[s] = e;
}

```

```
void addEdge(int s, int t, int c, int w)
{
    Edge *st = makeEdge(s, t, c, w), *ts = makeEdge(t, s, 0, -w);
    st->r = ts; ts->r = st;
}

int SPFA()
{
    static int que[MAX_V];
    static int aug[MAX_V];
    static int in[MAX_V];
    static int dist[MAX_V];
    static Edge *prev[MAX_V];
    int qh = 0, qt = 0;

    int u, v;
    fill(dist, dist + cntN, INF); dist[S] = 0;
    fill(in, in + cntN, 0); in[S] = true;
    que[qt ++] = S; aug[S] = INF;
    for( ; qh != qt; ) {
        u = que[qh]; qh = (qh + 1) % MAX_N;
        for(Edge *e = E[u]; e; e = e->n) {
            if (! e->c) continue;
            v = e->t;
            if (dist[v] > dist[u] + e->w) {
                dist[v] = dist[u] + e->w;
                aug[v] = min(aug[u], e->c);
                prev[v] = e;
                if (! in[v]) {
                    in[v] = true;
                    if (qh != qt && dist[v] <= dist[que[qh]]) {
                        qh = (qh - 1 + MAX_N) % MAX_N;
                        que[qh] = v;
                    } else {
                        que[qt] = v;
                        qt = (qt + 1) % MAX_N;
                    }
                }
            }
        }
        in[u] = false;
    }

    if (dist[T] == INF) return false;
    cost += dist[T] * aug[T];
    flow += aug[T];
    for(u = T; u != S; ) {
        prev[u]->c -= aug[T];
        prev[u]->r->c += aug[T];
        u = prev[u]->r->t;
    }
    return true;
}

int minCostFlow()
{
    flow = cost = 0;
    while(SPFA());
    return cost;
}
```

2.6 Strongly Connected Component

N 个点的图求 SCC，totID 为时间标记，top 为栈顶，totCol 为强联通分量个数，注意初始化

```
int totID, totCol;
int col[MAX_N], low[MAX_N], dfn[MAX_N];
int top, stack[MAX_N], instack[MAX_N];

int tarjan(int u)
{
    low[u] = dfn[u] = ++totID;
    instack[u] = true; stack[++top] = u;

    int v;
    foreach(it, adj[u]) {
        v = it->first;
        if (dfn[v] == -1)
            low[u] = min(low[u], tarjan(v));
        else if (instack[v])
            low[u] = min(low[u], dfn[v]);
    }

    if (low[u] == dfn[u]) {
        do {
            v = stack[top--];
            instack[v] = false;
            col[v] = totCol;
        } while(v != u);
        ++totCol;
    }
    return low[u];
}

void solve()
{
    totID = totCol = top = 0;
    fill(dfn, dfn + N, 0);
    for(int i = 0; i < N; ++i)
        if (!dfn[i])
            tarjan(i);
}
```

2.7 *2-SAT

Comming Soon

Chapter 3

Number Theory

3.1 Chinese Remainder Theorem

包括扩展欧几里得，求逆元，和保证除数互质条件下的 CRT

```
LL x, y;
void exGcd(LL a, LL b)
{
    if (b == 0) {
        x = 1;
        y = 0;
        return;
    }
    exGcd(b, a % b);
    LL k = y;
    y = x - a / b * y;
    x = k;
}

LL inversion(LL a, LL b)
{
    exGcd(a, b);
    return (x % b + b) % b;
}

LL CRT(vector<LL> m, vector<LL> a)
{
    int N = m.size();
    LL M = 1, ret = 0;
    for(int i = 0; i < N; ++i)
        M *= m[i];

    for(int i = 0; i < N; ++i) {
        ret = (ret + (M / m[i]) * a[i] % M * inversion(M / m[i], m[i])) % M;
    }
    return ret;
}
```

3.2 Pollard's Rho and Miller-Rabbin

大数分解和素性判断

```
typedef long long LL;

LL modMul(LL a, LL b, LL P)
{
    LL ret = 0;
    for( ; a; a >>= 1) {
        if (a & 1) {
```

```
                ret += b;
                if (ret >= P) ret -= P;
            }
            b <<= 1;
            if (b >= P) b -= P;
        }
        return ret;
}

LL modPow(LL a, LL u, LL P)
{
    LL ret = 1;
    for( ; u; u >>= 1, a = modMul(a, a, P))
        if (u & 1) ret = modMul(ret, a, P);
    return ret;
}

int millerRabin(LL N)
{
    if (N == 2) return true;
    LL t = 0, u = N - 1, x, y, a;
    for( ; !(u & 1); ++t, u >>= 1) ;
    for(int k = 0; k < 10; ++k) {
        a = rand() % (N - 2) + 2;
        x = modPow(a, u, N);
        for(int i = 0; i < t; ++i, x = y) {
            y = modMul(x, x, N);
            if (y == 1 && x > 1 && x < N - 1) return false;
        }
        if (x != 1) return false;
    }
    return true;
}

LL gcd(LL a, LL b)
{
    return !b ? a : gcd(b, a % b);
}

LL pollardRho(LL N)
{
    LL i = 1, x = rand() % N;
    LL y = x, k = 2, d = 1;
    do {
        d = gcd(x - y + N, N);
        if (d != 1 && d != N) return d;
        if (++i == k) y = x, k <<= 1;
        x = (modMul(x, x, N) - 1 + N) % N;
    } while (y != x);
    return N;
}

void getFactor(LL N)
{
    if (N < 2) return;
    if (millerRabin(N)) {
        //do some operations
        return;
    }
    LL x = pollardRho(N);
    getFactor(x);
    getFactor(N / x);
}
```

3.3 *Baby-step Giant-step

Comming soon

Chapter 4

Algebraic Algorithms

4.1 *Linear Equations in Z_m

Comming Soon

4.2 *Linear Equations in R

Comming Soon

4.3 *Fast Fourier Transform

Comming Soon

Chapter 5

Computational Geometry

5.1 Basic Operations

平面几何基本操作，之后的几个都需要先敲它

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <climits>
#include <numeric>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
#define REP(i,n) for(int i=0;i<n;++i)
using namespace std;

const double EPS = 1e-8;
inline int sign(double a) {
    return a < -EPS ? -1 : a > EPS;
}

struct Point {
    double x, y;
    Point() {
    }
    Point(double _x, double _y) :
            x(_x), y(_y) {
    }
    Point operator+(const Point&p) const {
        return Point(x + p.x, y + p.y);
    }
    Point operator-(const Point&p) const {
        return Point(x - p.x, y - p.y);
    }
    Point operator*(double d) const {
        return Point(x * d, y * d);
    }
    Point operator/(double d) const {
        return Point(x / d, y / d);
    }
    bool operator<(const Point&p) const {
        int c = sign(x - p.x);
        if (c)
            return c == -1;
        return sign(y - p.y) == -1;
    }
    double dot(const Point&p) const {
        return x * p.x + y * p.y;
    }
    double det(const Point&p) const {
        return x * p.y - y * p.x;
```

```
    }
    double alpha() const {
        return atan2(y, x);
    }
    double distTo(const Point&p) const {
        double dx = x - p.x, dy = y - p.y;
        return hypot(dx, dy);
    }
    double alphaTo(const Point&p) const {
        double dx = x - p.x, dy = y - p.y;
        return atan2(dy, dx);
    }
    void read() {
        scanf("%lf%lf", &x, &y);
    }
    double abs() {
        return hypot(x, y);
    }
    double abs2() {
        return x * x + y * y;
    }
    void write() {
        cout << "(" << x << "," << y << ")" << endl;
    }
};

#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))

#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))

Point isSS(Point p1, Point p2, Point q1, Point q2) {
    double a1 = cross(q1,q2,p1), a2 = -cross(q1,q2,p2);
    return (p1 * a2 + p2 * a1) / (a1 + a2);
}

double calcArea(const vector<Point>&ps) {
    int n = ps.size();
    double ret = 0;
    for (int i = 0; i < n; ++i) {
        ret += ps[i].det(ps[(i + 1) % n]);
    }
    return ret / 2; //maybe need abs(ret)
}
```

5.2 Convex Hull

Montone Chain

```
vector<Point> convexHull(vector<Point> ps) {
    int n = ps.size();
    if (n <= 1)
        return ps;
    sort(ps.begin(), ps.end());
    vector<Point> qs;
    for (int i = 0; i < n; qs.push_back(ps[i++])) {
        while (qs.size() > 1 && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
            qs.pop_back();
    }
    for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i--])) {
        while (qs.size() > t && crossOp(qs[qs.size()-2],qs.back(),ps[i]) <= 0)
            qs.pop_back();
    }
    qs.pop_back();
```

```
    return qs;
}
```

5.3 Convex Diameter

```
double convexDiameter(const vector<Point>&ps) {
    int n = ps.size();
    int is = 0, js = 0;
    for (int i = 1; i < n; ++i) {
        if (ps[i].x > ps[is].x)
            is = i;
        if (ps[i].x < ps[js].x)
            js = i;
    }
    double maxd = ps[is].distTo(ps[js]);
    int i = is, j = js;
    do {
        if ((ps[(i + 1) % n] - ps[i]).det(ps[(j + 1) % n] - ps[j]) >= 0)
            (++j) %= n;
        else
            (++i) %= n;
        maxd = max(maxd, ps[i].distTo(ps[j]));
    } while (i != is || j != js);
    return maxd;
}
```

5.4 Convex Cut

```
vector<Point> convexCut(const vector<Point>&ps, Point q1, Point q2) {
    vector<Point> qs;
    int n = ps.size();
    for (int i = 0; i < n; ++i) {
        Point p1 = ps[i], p2 = ps[(i + 1) % n];
        int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
        if (d1 >= 0)
            qs.push_back(p1);
        if (d1 * d2 < 0)
            qs.push_back(isSS(p1, p2, q1, q2));
    }
    return qs;
}
```

Chapter 6

Classic Problems

6.1 Nim Game

对于 N 堆石子，每人轮流取
1）每堆石子个数为 1，根据奇偶性直接判胜负
2）有一堆个数大于 1，先手必胜（直接根据奇偶性调整留一个还是取光）
3）Nim 游戏：每堆个数任意，xor 和为 0 则为必败态，否则必胜，证明略
4）Moore's Nim K 游戏：从最少 1 堆最多 K 堆中取任意数量的石子，结论：把所有堆的石子个数按二进制表示，如果任意一位一的个数总和都为 K+1 的倍数则为必败态，否则为必胜态，显然 Nim 是 K=1 的特殊情况。
5）anti-nim 游戏：取到最后一个石子为败，结论：必胜态当且仅当：1）所有堆石子数都为 1 且游戏的 SG 值为 0,2）存在某堆石子数大于 1 且游戏的 sg 值不为 0