

Yerevan State University ACM ICPC Team Notebook (Forked from Stanford)

Contents

1	Dynamic Programming	1
1.1	Convex Hull Trick	1
1.2	ConvexHull Hull Trick (Integers)	1
1.3	Divide and Conquer Optimization	1
1.4	Knuth Optimization (Type 1)	2
1.5	Knuth Optimization (Type 2)	2
2	Data Structures	2
2.1	2D Dynamic Segment Tree	2
2.2	Centroid Decomposition	2
2.3	DSU With Rollbacks	3
2.4	Heavy-Light Decomposition	3
2.5	Persistent Segment Tree With Range Updates	3
2.6	Treap With Implicit Key	4
2.7	Persistent Treap	4
3	Geometry	4
3.1	3D Convex Hull	4
3.2	Closest Point Pair	5
3.3	Dynamic Upper Convex Hull	5
3.4	Farthest Point Pair	6
3.5	Half Plane Intersection (Randomized Incremental Algorithm)	6
3.6	Minkowski Sum of Convex Polygons	7
3.7	Pair of Intersecting Segments	7
3.8	Minimum Enclosing Circle (Randomized Incremental Algorithm)	8
4	Graphs	9
4.1	Maximum Flow (Dinic)	9
4.2	Facet Finding	9
4.3	LCA O(1)	9
4.4	Min Cost Max Flow (Dijkstra with Potentials)	10
4.5	Offline LCA (Tarjan)	10
4.6	Online LCA	10
4.7	Rooted Tree Isomorphism	10
4.8	Vertex Cover (Types of Vertices)	11
5	Math	11
5.1	Chinese Remainder Theorem	11
5.2	Discrete Logarithm and Square Root	11
5.3	Extended Euclid Algorithm	11
5.4	Fast Fourier Transform	12
5.5	Generator of \mathbb{Z}_p	12
5.6	Linear Time Precalculation of Inverses	12
6	Strings	12
6.1	Aho-Corasick	12
6.2	Discrete Manacer	12
6.3	Palindromic Tree	13
6.4	Prefix Function	13
6.5	Suffix Array (Implementation 1)	13
6.6	Suffix Array (Implementation 2)	13
6.7	Suffix Automaton	13
6.8	Suffix Tree	14
6.9	Z Function	14

1 Dynamic Programming

1.1 Convex Hull Trick

```
const int N = 100007;
const LD eps = 1e-9;
struct Line {
    LD a, b;
    Line(LD a = 0.0, LD b = 0.0) : a(a), b(b) {}
    bool operator < (const Line &o) const {
        return a > o.a;
    }
} st[N], mas[N];
int top, n;

inline LD cross(const Line &u, const Line &v) {
    return (v.b - u.b) / (u.a - v.a);
}

void add(const Line &L) {
    while (top > 1 && cross(st[top - 2], st[top - 1]) > cross(st[
        top - 1], L) - eps) {
        --top;
    }
    st[top++] = L;
}

LD query(LD x) {
    int l = 0;
    int r = top - 1;
    LD lx, rx;
    int pos = -1;
    while (true) {
        int m = (l + r) / 2;
        LD x1 = -1e18, x2 = 1e18;
        if (m != 0) {
            x1 = cross(st[m - 1], st[m]);
        }
        if (m != top - 1) {
            x2 = cross(st[m], st[m + 1]);
        }
        if (x > x1 - eps && x < x2 + eps) {
            pos = m;
            lx = x1;
            rx = x2;
            break;
        }
        else if (x < x1 - eps) {
            r = m - 1;
        }
        else {
            l = m + 1;
        }
    }
    assert(pos != -1);
    Line &L = st[pos];
    return L.a * x + L.b;
}
```

1.2 ConvexHull Hull Trick (Integers)

```
struct convex_hull {
    int top, pointer;

    struct unix {
        long long k, b;
    } mas[N];

    void init() {
        top = 0;
        pointer = 0;
    }

    void add(long long k, long long b) {
        k = -k; b = -b;
        unix cur;
        cur.k = k;
        cur.b = b;
        while (top > 2) {
            unix u, v;
            u = mas[top - 1];
            v = mas[top - 2];
            assert(k >= u.k);
            assert(u.k >= v.k);
```

```
if ((LD)(v.b - u.b) / (u.k - v.k) > (LD)(u.b - b) /
    (k - u.k))
    --top;
else
    break;
}
mas[top++] = cur;
}
/*
// O(logN) general case
long long get(long long x) {
    if (top == 0) return INF;
    int ina = 0, inb = top - 1, ans;
    while (ina < inb) {
        int mid = (ina + inb) / 2;
        long long p, q;
        if (mid == 0) {
            p = -INF;
            q = 1;
        }
        else {
            p = mas[mid - 1].b - mas[mid].b;
            q = mas[mid].k - mas[mid - 1].k;
        }
        if ((LD)p / q <= x) {
            ans = mid;
            ina = mid + 1;
        }
        else
            inb = mid - 1;
    }
    long long pat = mas[ans].k * x + mas[ans].b;
    return -pat;
}
}
// Amortized O(1), if queries are sorted
long long get(long long x) {
    if (top == 0) return INF;
    pointer = min(pointer, top - 1);
    while (pointer < top - 1) {
        long long p, q;
        q = mas[pointer + 1].k - mas[pointer].k;
        p = mas[pointer].b - mas[pointer + 1].b;
        if ((LD)p / q <= x)
            ++pointer;
        else
            break;
    }
    long long pat = mas[pointer].k * x + mas[pointer].b;
    return -pat;
}
} hull;
```

1.3 Divide and Conquer Optimization

```
// dp[i][j] = min_{k<j} { dp[i-1][k] + cost(k, j) }
// if opt[i][j] <= opt[i][j+1]

int dp[N][N];

int cost(int l, int r) {
    // ...
}

void calcdp(int k, int l, int r, int optl, int opt) {
    if (l > r) return;

    int m = (l + r) / 2;
    int best = INF;
    int pos = -1;

    for (int j = optl; j <= min(m, opt); ++j) {
        int val = dp[k - 1][j - 1] + cost(j, m);
        if (val < best) {
            best = val;
            pos = j;
        }
    }

    dp[k][m] = best;

    calcdp(k, l, m - 1, optl, pos);
    calcdp(k, m + 1, r, pos, opt);
}

scanf("%d", &n);
```

```

for (int i = 1; i <= n; ++i) {
    scanf("%d", &a[i]);
}
// init
for (int k = 1; k <= n; ++k) {
    // init ?
    calcdp(k, 1, n, 1, n);
}

```

1.4 Knuth Optimization (Type 1)

```

// dp[i][j] = Min_{k < j} {dp[i-1][k] + cost(k, j)}
// if opt[i-1][j] <= opt[i][j] <= opt[i][j+1]

int dp[N][N];
int opt[N][N];

int cost(int l, int r) {
    // ...
}

void solve() {
    FOR(k, N) FOR(i, N) {
        dp[k][i] = INF;
    }
    dp[0][0] = 0;
    for (int i = 0; i <= n; ++i) {
        opt[0][i] = 1;
    }
    for (int k = 1; k <= n; ++k) {
        dp[k][0] = 0;
        opt[k][0] = 1;
        opt[k][n+1] = n;
        for (int i = n; i >= 1; --i) {
            for (int j = opt[k-1][i]; j <= opt[k][i+1]; ++j) {
                int val = dp[k-1][j-1] + cost(j, i);
                if (val < dp[k][i]) {
                    dp[k][i] = val;
                    opt[k][i] = j;
                }
            }
        }
    }
}

```

1.5 Knuth Optimization (Type 2)

```

// dp[i][j] = Min_{i < k < j} { dp[i][k] + dp[k+1][j] + cost(i, j) }
// if opt[l][r-1] <= opt[l][r] <= opt[l+1][r]

LL dp[N][N];
int opt[N][N];

int cost(int l, int r) {
    // ...
}

void solve() {
    for (int l = n-1; l >= 0; --l) {
        dp[l][l] = 0;
        opt[l][l] = 1;

        if (l+1 < n) {
            dp[l][l+1] = cost(l, l+1);
            opt[l][l+1] = 1;
        }

        for (int r = l+2; r < n; ++r) {
            dp[l][r] = INF;

            int L = opt[l][r-1];
            int R = opt[l+1][r];

            for (int j = L; j <= R; ++j) {
                int val = dp[l][j] + dp[j+1][r] + getsum(l, r);
                if (val < dp[l][r]) {
                    dp[l][r] = val;
                }
            }
        }
    }
}

```

2 Data Structures

2.1 2D Dynamic Segment Tree

```

struct intNode {
    intNode *l, *r;
    int tl, tr;
    LL g;

    intNode(int tl = 0, int tr = 0, LL g = 0, intNode *l = NULL,
            intNode *r = NULL) :
        tl(tl), tr(tr), g(g), l(l), r(r) {}
};

typedef intNode* intPnode;

void intCreate(intPnode &T, int tl, int tr) {
    if (!T) T = new intNode(tl, tr);
}

LL intValue(intPnode &T) {
    return T == NULL ? 0 : T->g;
}

LL intGet(intPnode &T, int tl, int tr, int l, int r) {
    if (T == NULL) return 0;
    //intCreate(T, tl, tr); // eats more useless memory
    if (tl == l && tr == r) return intValue(T);
    int tm = (tl + tr) / 2;
    if (r <= tm) return intGet(T->l, tl, tm, l, r);
    if (l > tm) return intGet(T->r, tm + 1, tr, l, r);
    return gcd(intGet(T->l, tl, tm, l, tm), intGet(T->r, tm + 1,
        tr, tm + 1, r));
}

void intUpdate(intPnode &T, int tl, int tr, int pos, LL val, bool
    multipleLines = false, intPnode lson = NULL, intPnode rson
    = NULL) {
    intCreate(T, tl, tr);
    if (tl == tr) {
        if (!multipleLines) T->g = val;
        else {
            LL left = intGet(lson, 0, N, tl, tr);
            LL right = intGet(rson, 0, N, tl, tr);
            T->g = gcd(left, right);
        }
        return;
    }
    int tm = (tl + tr) / 2;
    if (pos <= tm) intUpdate(T->l, tl, tm, pos, val,
        multipleLines, lson, rson);
    else intUpdate(T->r, tm + 1, tr, pos, val, multipleLines,
        lson, rson);
    T->g = gcd(intValue(T->l), intValue(T->r));
}

struct extNode {
    extNode *l, *r;
    int tl, tr;
    intNode *root;

    extNode(int tl = 0, int tr = 0, intNode *root = NULL, extNode
        *l = NULL, extNode *r = NULL) :
        tl(tl), tr(tr), root(root), l(l), r(r) {}
};

typedef extNode* extPnode;

void extCreate(extPnode &T, int tl, int tr) {
    if (!T) T = new extNode(tl, tr);
}

LL extGet(extPnode &T, int tl, int tr, int lx, int rx, int ly,
    int ry) {
    if (T == NULL) return 0;
    //extCreate(T, tl, tr); // eats more useless memory
    if (tl == lx && tr == rx) return intGet(T->root, 0, N, ly, ry
        );
    int tm = (tl + tr) / 2;
}

```

```

if (rx <= tm) return extGet(T->l, tl, tm, lx, rx, ly, ry);
if (lx > tm) return extGet(T->r, tm + 1, tr, lx, rx, ly, ry);
return gcd(extGet(T->l, tl, tm, lx, tm, ly, ry), extGet(T->r,
    tm + 1, tr, tm + 1, rx, ly, ry));
}

void extUpdate(extPnode &T, int tl, int tr, int x, int y, LL val)
{
    extCreate(T, tl, tr);
    if (tl == tr) {
        intUpdate(T->root, 0, N, y, val);
        return;
    }
    int tm = (tl + tr) / 2;
    if (x <= tm) extUpdate(T->l, tl, tm, x, y, val);
    else extUpdate(T->r, tm + 1, tr, x, y, val);
    intUpdate(T->root, 0, N, y, val, true, T->l ? T->l->root :
        NULL, T->r ? T->r->root : NULL);
}

```

2.2 Centroid Decomposition

```

const int N = 1000007;

struct CentroidDecomposition {
    vector<int> G[N];
    bool used[N];
    int size[N]; // subtree size
    int maxi[N]; // max subtree size

    struct node {
        int v;
        vector<node*> to;
        unordered_map<int, int> id; // child id of a vertex in
            subtree
        vector<int> nodes; // nodes in subtree

        // keep additional data here

        node(int v = 0) : v(v) {}
    };

    typedef node* pnode;
    pnode root;

    void dfs(int u, int par, vector<int> &mas) {
        mas.pb(u);
        size[u] = 1;
        maxi[u] = 1;
        for (int to : G[u]) {
            if (to != par && !used[to]) {
                dfs(to, u, mas);
                size[u] += size[to];
                maxi[u] = max(maxi[u], size[to]);
            }
        }
    }

    void buildData(pnode root) {
        root->nodes.pb(root->v);
        for (int i = 0; i < sz(root->to); ++i) {
            pnode to = root->to[i];
            for (int u : to->nodes) {
                root->nodes.pb(u);
                root->id[u] = i;
            }
        }
    }

    pnode build(int u) {
        vector<int> mas;
        dfs(u, -1, mas);
        int n = sz(mas);
        int best = N, pos = -1;
        for (int u : mas) {
            maxi[u] = max(maxi[u], n - size[u]);
            if (maxi[u] < best) {
                best = maxi[u];
                pos = u;
            }
        }
        u = pos;
        used[u] = true;
        node *root = new node(u);
        for (int to : G[u])
            if (!used[to]) root->to.pb(build(to));
        buildData(root);
        return root;
    }
}

```

```

}
} cd;

// build
for (int i = 0; i < n - 1; ++i) {
    int u, v;
    scanf("%d%d", &u, &v);
    cd.G[u].pb(v);
    cd.G[v].pb(u);
}
cd.root = cd.build(1);

```

2.3 DSU With Rollbacks

```

struct DSU_withRollbacks {
    int *par, *size;
    int n, ncomps;

    enum { PAR, SIZE, NCOMPS };

    struct Change {
        int type, index, oldValue;
        Change() {}
        Change(int type, int index, int oldValue) : type(type),
            index(index), oldValue(oldValue) {}
    } *changes;
    int top;

    void init(int n) {
        this->n = n;
        par = new int[n];
        size = new int[n];
        changes = new Change[2 * n];
        ncomps = n;
        top = 0;
        FOR(i, n) {
            par[i] = i;
            size[i] = 1;
        }

        int get(int i) {
            if (par[i] == i) return i;
            return get(par[i]);
        }

        void unite(int i, int j) {
            i = get(i);
            j = get(j);
            if (i == j) return;
            if (size[i] < size[j]) swap(i, j);

            changes[top++] = Change(PAR, j, par[j]);
            changes[top++] = Change(SIZE, i, size[i]);
            changes[top++] = Change(NCOMPS, -1, ncomps);

            size[i] += size[j];
            par[j] = i;
            --ncomps;
        }

        int snapshot() {
            return top;
        }

        void toSnapshot(int snap) {
            while (top != snap) {
                --top;
                if (changes[top].type == PAR) {
                    par[changes[top].index] = changes[top].oldValue;
                }
                if (changes[top].type == SIZE) {
                    size[changes[top].index] = changes[top].oldValue;
                }
                if (changes[top].type == NCOMPS) {
                    ncomps = changes[top].oldValue;
                }
            }
        }
};

```

2.4 Heavy-Light Decomposition

```

const int N = 100007;
int n, dep[N], par[N], size[N];
vector<int> G[N];
bool heavy[N];

vector<vector<int>> > chains;
int id[N], pos[N];

void dfs(int u, int p = 0, int d = 0) {
    par[u] = p;
    dep[u] = d;
    size[u] = 1;
    int maxi = 0, pos = 0;
    for (int i = 0; i < sz(G[u]); ++i) {
        int to = G[u][i];
        if (to != p) {
            dfs(to, u, d + 1);
            if (size[to] > maxi) {
                maxi = size[to];
                pos = to;
            }
            size[u] += size[to];
        }
    }
    heavy[pos] = true;

    int makeHLD(int u) {
        bool f = false;
        for (int i = 0; i < sz(G[u]); ++i) {
            int to = G[u][i].first;
            if (to == par[u]) continue;
            int ind = makeHLD(to);
            if (ind != -1) {
                chains[ind].pb(u);
                id[u] = ind;
                f = true;
            }
        }
        if (!f) {
            id[u] = sz(chains);
            chains.pb(vector<int>());
            chains[id[u]].pb(u);
        }
        if (heavy[u]) return id[u];
        return -1;
    }

    struct segTree {
        int n, *T;

        void init(int s) {
            n = s;
            T = new int[4 * s + 2];
            memset(T, 0, 4 * (4 * s + 2));
        }

        // ...

        int getLca(int u, int v) {
            while (id[u] != id[v]) {
                if (dep[chains[id[u]][0]] < dep[chains[id[v]][0]]) v = par[
                    chains[id[v]][0]];
                else u = par[chains[id[u]][0]];
            }
            return dep[u] < dep[v] ? u : v;
        }

        int go(int u, int v) {
            int ret = -1000000000;
            while (id[u] != id[v]) {
                ret = max(ret, S[id[u]].get(1, 0, S[id[u]].n - 1, pos[u]
                    ));
                u = par[chains[id[u]][0]];
            }
            if (u == v) return ret;
            ret = max(ret, S[id[u]].get(1, 0, S[id[u]].n - 1, pos[v] + 1,
                pos[u]));
            return ret;
        }
    };

    // build

```

```

dfs(1);
chains.clear();
makeHLD(1);
for (int i = 0; i < sz(chains); ++i) reverse(all(chains[i]));
for (int i = 0; i < sz(chains); ++i)
    for (int j = 0; j < sz(chains[i]); ++j)
        pos[chains[i][j]] = dep[chains[i][j]] - dep[chains[i][0]];

S.clear();
S.resize(sz(chains));
for (int i = 0; i < sz(chains); ++i) {
    S[i].init(sz(chains[i]));
    for (int j = 0; j < S[i].n; ++j)
        S[i].update(1, 0, S[i].n - 1, j, w[chains[i][j]]);
}

```

2.5 Persistent Segment Tree With Range Updates

```

struct node {
    node *l, *r;
    int sum, add;

    node(int sum = 0, int add = 0) : sum(sum), add(add), l(NULL),
        r(NULL) {} // creates a leaf

    node(node *left, node *right) // creates a node
    {
        add = 0;
        sum = 0;
        l = left;
        r = right;
    }
};

typedef node* pnode;

inline int get(pnode T, int tl, int tr)
{
    if (!T)
        return 0;
    return T->add * (tr - tl + 1) + T->sum;
}

inline void fix(pnode &T, int tl, int tr)
{
    int tm = (tl + tr) / 2;
    T->sum = get(T->l, tl, tm) + get(T->r, tm + 1, tr);
}

pnode push(pnode &T, int tl, int tr)
{
    pnode ret = new node();
    if (T->l)
    {
        ret->l = new node(T->l->l, T->l->r);
        ret->l->add = T->l->add;
        ret->l->sum = T->l->sum;
        ret->l->add += T->add;
    }
    if (T->r)
    {
        ret->r = new node(T->r->l, T->r->r);
        ret->r->add = T->r->add;
        ret->r->sum = T->r->sum;
        ret->r->add += T->add;
    }
    ret->sum = T->sum + (tr - tl + 1) * T->add;
    ret->add = 0;
    return ret;
}

pnode build(int a[], int tl, int tr)
{
    if (tl == tr)
    {
        return new node(a[tl], 0);
    }
    int tm = (tl + tr) / 2;
    pnode ret = new node(build(a, tl, tm), build(a, tm + 1, tr));
    fix(ret, tl, tr);
    return ret;
}

```

```

pnode rsq(pnode &T, int tl, int tr, int l, int r, int &s)
{
    pnode ret = push(T, tl, tr);
    if (tl == l && tr == r)
    {
        s = ret->sum;
        return ret;
    }
    int tm = (tl + tr) / 2;
    if (r <= tm)
    {
        ret->l = rsq(ret->l, tl, tm, l, r, s);
    }
    else if (l > tm)
    {
        ret->r = rsq(ret->r, tm + 1, tr, l, r, s);
    }
    else
    {
        int s1, s2;
        ret->l = rsq(ret->l, tl, tm, l, tm, s1);
        ret->r = rsq(ret->r, tm + 1, tr, tm + 1, r, s2);
        s = s1 + s2;
    }
    fix(T, tl, tm);
    return ret;
}

pnode update(pnode &T, int tl, int tr, int l, int r, int delta)
{
    if (tl == l && tr == r)
    {
        pnode ret = new node(T->l, T->r);
        ret->add = T->add + delta;
        ret->sum = T->sum;
        return ret;
    }
    int tm = (tl + tr) / 2;
    pnode ret = push(T, tl, tr);
    if (r <= tm)
    {
        ret = new node(update(ret->l, tl, tm, l, r, delta), ret->r);
    }
    else if (l > tm)
    {
        ret = new node(ret->l, update(ret->r, tm + 1, tr, l, r, delta));
    }
    else
    {
        ret = new node(update(ret->l, tl, tm, l, tm, delta), update(ret->r, tm + 1, tr, tm + 1, r, delta));
    }
    fix(ret, tl, tr);
    return ret;
}

```

2.6 Treap With Implicit Key

```

const int N = 1000007;
struct node {
    int pr, size;
    int val;
    node *l, *r;

    node(int pr = 0, int val = 0, node *l = NULL, node *r = NULL) {
        pr(pr), l(l), r(r), size(1), val(val) {}
    }
} *root;
typedef node* pnode;
int cur, prio[N];

inline int getSize(pnode T) {
    return (T ? T->size : 0);
}

inline void fix(pnode &T) {
    if (!T) return;
    T->size = getSize(T->l) + getSize(T->r) + 1;
}

void merge(pnode A, pnode B, pnode &T) {
    if (!A || !B) {

```

```

        T = (A ? A : B);
    }
    else if (A->pr > B->pr) {
        merge(A->r, B, A->r);
        T = A;
    }
    else {
        merge(A, B->l, B->l);
        T = B;
    }
    fix(T);
}

void split(pnode T, int x, pnode &LT, pnode &RT, int add = 0) {
    if (!T) {
        LT = RT = 0;
        return;
    }
    int curx = add + getSize(T->l);
    if (x <= curx) {
        split(T->l, x, LT, T->l, add);
        RT = T;
    }
    else {
        split(T->r, x, T->r, RT, add + 1 + getSize(T->l));
        LT = T;
    }
    fix(LT);
    fix(RT);
}

```

2.7 Persistent Treap

```

struct node {
    node *l, *r;
    bool rev;
    int size;
    char data;

    node(char data = 'a') : data(data) {
        l = r = NULL;
        rev = false;
        size = 1;
    }
};

typedef node* pnode;

const int N = 200007;
char st[N];

pnode copyNode(pnode other) {
    if (other == NULL) return NULL;
    pnode ret = new node();
    ret->l = other->l;
    ret->r = other->r;
    ret->rev = other->rev;
    ret->size = other->size;
    ret->data = other->data;
    return ret;
}

int getSize(pnode T) {
    return T ? T->size : 0;
}

void fix(pnode T) {
    if (T) T->size = 1 + getSize(T->l) + getSize(T->r);
}

void push(pnode &T) {
    if (!T->rev) return;
    T->l = copyNode(T->l);
    T->r = copyNode(T->r);
    swap(T->l, T->r);
    T->rev = false;
    if (T->l) T->l->rev ^= 1;
    if (T->r) T->r->rev ^= 1;
}

void split(pnode T, int cnt, pnode &L, pnode &R) {
    if (!T) {
        L = R = NULL;
        return;
    }
    pnode cur = copyNode(T);
    push(cur);
    if (getSize(cur->l) >= cnt) {

```

```

        split(cur->l, cnt, L, cur->l);
        R = cur;
    }
    else {
        split(cur->r, cnt - 1 - getSize(cur->l), cur->r, R);
        L = cur;
    }
    fix(L);
    fix(R);
}

void merge(pnode L, pnode R, pnode &T) {
    if (!L || !R) {
        T = copyNode(L ? L : R);
        return;
    }
    int lsize = getSize(L);
    int rsize = getSize(R);
    if (rand() % (lsize + rsize) < lsize) {
        T = copyNode(L);
        push(T);
        merge(T->r, R, T->r);
    }
    else {
        T = copyNode(R);
        push(T);
        merge(L, T->l, T->l);
    }
    fix(T);
}

void print(pnode T) {
    if (!T) return;
    push(T);
    print(T->l);
    putchar(T->data);
    print(T->r);
}

```

3 Geometry

3.1 3D Convex Hull

```

const int N = 1007;
const LD pi = acos(-1.0);
const LD eps = 1e-9;
const LD INF = 1e18;

struct pt {
    LD x, y, z;

    pt (LD x=0, LD y=0, LD z=0) : x(x), y(y), z(z) {}

    void read() {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        x = a;
        y = b;
        z = c;
    }

    void print() {
        cerr << "(" << x << " " << y << " " << z << " ) ";
    }

    pt operator + (const pt &o) const {
        return pt(x+o.x, y+o.y, z+o.z);
    }

    LD dist2(const pt &o) const {
        return sqrt((x-o.x)*(x-o.x) + (y-o.y)*(y-o.y));
    }

    pt operator - (const pt &o) const {
        return pt(x-o.x, y-o.y, z-o.z);
    }

    pt operator * (LD c) const {
        return pt(x*c, y*c, z*c);
    }

    pt operator / (LD c) const {
        return pt(x/c, y/c, z/c);
    }
}

```

```

LD len() const {
    return sqrt(1.0 / (x*x + y*y + z*z));
}

pt normalize() const {
    return *this * (1.0 / len());
}

pt to_len(LD need_len) const {
    return this->normalize() * need_len;
}

bool operator < (const pt &o) const {
    if (fabs(x-o.x) > eps) return x < o.x;
    if (fabs(y-o.y) > eps) return y < o.y;
    return z < o.z - eps;
}

LD det(LD a, LD b, LD c, LD d) {
    return a*d - b*c;
}

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
}

pt cross(pt u, pt v) {
    return pt(det(u.y, u.z, v.y, v.z),
             -det(u.x, u.z, v.x, v.z),
             det(u.x, u.y, v.x, v.y));
}

vector<pt> p;

struct plane {
    LD a, b, c, d;
    pt A, B, C;
    int i, j, k;

    plane(LD a=0, LD b=0, LD c=0, LD d=0): a(a), b(b), c(c), d(d) {
        // note A, B, C stay undefined
    }

    plane(int i, int j, int k): A(p[i]), B(p[j]), C(p[k]), i(i),
                                   j(j), k(k) {
        pt norm = cross(B-A, C-A).normalize();
        a = norm.x;
        b = norm.y;
        c = norm.z;
        d = -a * A.x - b * A.y - c * A.z;
    }
};

LD dist_from_plane(pt A, plane& t) {
    return (A.x * t.a + A.y * t.b + A.z * t.c + t.d) / sqrt(1.0 / (t.a * t.a + t.b * t.b + t.c * t.c));
}

LD sign(LD x) {
    if (x < -eps) return -1;
    if (x > eps) return 1;
    return 0;
}

bool inside(pt A, vector<plane>& faces, pt O) {
    for (auto t : faces) {
        int need_sign = sign(dist_from_plane(O, t));
        int cur_sign = sign(dist_from_plane(A, t));
        if (need_sign * cur_sign == -1) return false;
    }
    return true;
}

void dfs(int u, vector<vector<int>> &G, vector<bool>& used,
         vector<int>& order) {
    order.pb(u);
    used[u] = true;
    for (auto to : G[u]) {
        if (!used[to]) {
            dfs(to, G, used, order);
        }
    }
}

void solve() {
    int n;
    scanf("%d", &n);

```

```

p.resize(n);
FOR(i, n) {
    p[i].read();
}

vector<plane> faces;
FOR(i, 4) FOR(j, i) FOR(k, j) {
    faces.pb(plane(i, j, k));
}

pt O = (p[0] + p[1] + p[2] + p[3]) / 4.0; // point strictly inside

for (int ptr = 4; ptr < n; ++ptr) {
    auto P = p[ptr];
    if (inside(P, faces, O)) continue;

    vector<plane> new_faces;
    vector<plane> to_delete;

    FOR(i, sz(faces)) {
        int need_sign = sign(dist_from_plane(O, faces[i]));
        int cur_sign = sign(dist_from_plane(P, faces[i]));
        if (cur_sign * need_sign != -1) {
            new_faces.pb(faces[i]);
        }
        else {
            to_delete.pb(faces[i]);
        }
    }

    unordered_map<int, int> M;

    for (auto t : to_delete) {
        int i = t.i;
        int j = t.j;
        int k = t.k;
        M[min(i, j) * N + max(i, j)] += 1;
        M[min(i, k) * N + max(i, k)] += 1;
        M[min(k, j) * N + max(k, j)] += 1;
    }

    vector<vector<int>> G(n);
    int some_vertex = -1;
    for (auto pr : M) {
        if (pr.second >= 2) continue;
        int i = pr.first / N;
        int j = pr.first % N;
        some_vertex = i;
        G[i].pb(j);
        G[j].pb(i);
    }

    assert(some_vertex != -1);
    vector<bool> used(ptr, false);
    vector<int> order;
    dfs(some_vertex, G, used, order);

    vector<plane> to_add;
    FOR(i, sz(order)) {
        int j = (i+1)%sz(order);
        int u = order[i];
        int v = order[j];
        to_add.pb(plane(ptr, u, v));
    }

    for (auto t : to_add) {
        new_faces.pb(t);
    }
    faces = new_faces;
}

```

3.2 Closest Point Pair

```

struct point {
    double x, y;
    int ind;
    double operator + (const point &a) const {
        return sqrt((a.x - x) * (a.x - x) + (a.y - y) * (a.y - y));
    }
}

p[100007], temp[100007]; // temp for merge
double best = 1e18;
int n, ind1, ind2;

bool cmp1(const point &a, const point &b) // sort by x
{

```

```

    return (a.x < b.x || (a.x == b.x && a.y < b.y));
}

bool cmp2(const point &a, const point &b) // sort by y
{
    return (a.y < b.y || (a.y == b.y && a.x < b.x));
}

void merge(int l, int r) // merge for merge_sort by y
{
    int m = (l + r) / 2, i = l, j = m + 1, k;
    for (k = l; k <= r; ++k) {
        if (i > m) {
            temp[k] = p[j++];
        }
        else if (j > r) {
            temp[k] = p[i++];
        }
        else if (cmp2(p[i], p[j])) {
            temp[k] = p[i++];
        }
        else {
            temp[k] = p[j++];
        }
    }
    for (k = l; k <= r; ++k) {
        p[k] = temp[k];
    }
}

void update(int i, int j) {
    double dis = p[i].x + p[j].x;
    if (dis < best) {
        best = dis;
        ind1 = p[i].ind;
        ind2 = p[j].ind;
        if (ind1 > ind2) {
            swap(ind1, ind2);
        }
    }
}

void solve(int l, int r) {
    double midx;
    int i, j, m;
    if (r - l <= 3) {
        for (i = l; i <= r; ++i) {
            for (j = i + 1; j <= r; ++j) {
                update(i, j);
            }
        }
        sort(p + l, p + r + 1, cmp2);
        return;
    }
    m = (l + r) / 2;
    midx = p[m].x;
    solve(l, m);
    solve(m + 1, r);
    merge(l, r);

    static int t[100007];
    int tsz = 0;
    for (i = l; i <= r; ++i) {
        if (fabs(p[i].x - midx) < best) {
            j = tsz;
            while (j >= 0 && p[i].y - p[t[j]].y < best) {
                update(i, t[j]);
                j--;
            }
            t[tsz++] = i;
        }
    }
}

```

```

// run
scanf("%d", &n);
sort(p, p + n, cmp1);
solve(0, n - 1);

```

3.3 Dynamic Upper Convex Hull

```

const int N = 100007;
int n, x[N], c[N], t[N];
LL dp[N];

```

```

const LD eps = 1e-12;
struct segment {
    LL k, b; // y = kx + b
    LD x1, x2; // x1 < x2

    segment(LL k = 0, LL b = 0, LD x1 = -1e7, LD x2 = +1e7) : k(k), b(b), x1(x1), x2(x2) {}

    bool operator < (const segment &o) const {
        if (k != o.k) return k < o.k;
        return b < o.b;
    }
};

struct segment2 {
    LL k, b;
    LD x;

    segment2(LL k = 0, LL b = 0, LD x = 0.0) : k(k), b(b), x(x) {}

    bool operator < (const segment2 &o) const {
        if (fabs1(x - o.x) < eps) return k < o.k;
        return x < o.x;
    }
};

set<segment2> hull2;
set<segment> hull;

int cmp(LL k, LL b, const segment &s) {
    LD ylline = k * s.x1 + b;
    LD y2lline = k * s.x2 + b;
    LD y1seg = s.k * s.x1 + s.b;
    LD y2seg = s.k * s.x2 + s.b;
    if (ylline < y1seg + eps && y2lline < y2seg + eps) return -1;
    if (ylline + eps > y1seg && y2lline + eps > y2seg) return +1;
    return 0;
}

inline void addSegment(segment s) {
    hull.insert(s);
    hull2.insert(segment2(s.k, s.b, s.x1));
}

inline void deleteSegment(segment s) {
    hull.erase(s);
    hull2.erase(segment2(s.k, s.b, s.x1));
}

void addLine(LL k, LL b) {
    //lines.pb(mp(k, b));

    segment cur(k, b);
    if (hull.empty()) {
        addSegment(cur);
        return;
    }
    if (hull.count(cur)) return;
    auto it = hull.lower_bound(cur);
    // go left
    while (!hull.empty() && it != hull.begin()) {
        --it;
        if (cmp(k, b, *it) == 1) deleteSegment(*it);
        else break;
        it = hull.lower_bound(cur);
    }
    it = hull.lower_bound(cur);
    // go right
    while (!hull.empty() && it != hull.end()) {
        if (cmp(k, b, *it) == 1) deleteSegment(*it);
        else break;
        it = hull.lower_bound(cur);
    }
    if (hull.empty()) {
        addSegment(cur);
        return;
    }
    // intersect with left and right
    it = hull.lower_bound(cur);
    if (it == hull.begin()) {
        if (cmp(k, b, *it) == -1) return;
        else {
            LD x = (LD)(b - it->b) / (it->k - k);
            segment tmp = *it;
            tmp.x1 = x;
            deleteSegment(*it);
            addSegment(tmp);
            cur.x2 = x;
            addSegment(cur);
        }
    }
    else if (it == hull.end()) {
        --it;

```

```

        if (cmp(k, b, *it) == -1) return;
        LD x = (LD)(b - it->b) / (it->k - k);
        segment tmp = *it;
        tmp.x2 = x;
        deleteSegment(*it);
        addSegment(tmp);
        cur.x1 = x;
        addSegment(cur);
    }
    else {
        auto lit = it; --lit;
        auto rit = it;
        if (cmp(k, b, *lit) == -1 || cmp(k, b, *rit) == -1) return;
        LD x1 = (LD)(b - lit->b) / (lit->k - k);
        LD x2 = (LD)(b - rit->b) / (rit->k - k);
        segment t1 = *lit;
        segment t2 = *rit;
        deleteSegment(t1);
        deleteSegment(t2);
        t1.x2 = x1;
        t2.x1 = x2;
        cur.x1 = x1;
        cur.x2 = x2;
        addSegment(t1);
        addSegment(cur);
        addSegment(t2);
    }
}

LL getMax(LL x) {
    segment2 t(-(1LL << 58), 0, x);
    auto it = hull2.lower_bound(t);
    --it;
    return x * it->k + it->b;
}

```

3.4 Farthest Point Pair

```

struct pt {
    LD x, y;

    pt(LD x=0, LD y=0) : x(x), y(y) {}
};

void read() {
    int a, b;
    scanf("%d%d", &a, &b);
    x = a;
    y = b;
}

pt operator + (const pt &o) const {
    return pt(x+o.x, y+o.y);
}

pt operator - (const pt &o) const {
    return pt(x-o.x, y-o.y);
}

LD dist(const pt &o) const {
    return (x-o.x)*(x-o.x) + (y-o.y)*(y-o.y);
}

pt prod(LD c) const {
    return pt(c*x, c*y);
}

bool operator < (const pt &o) const {
    static const LD eps=1e-5;
    if (fabs1(x-o.x) > eps) return x < o.x;
    return y < o.y - eps;
}

} hull[N];
int n, nhull;

LD cross(pt u, pt v) {
    return u.x * v.y - u.y * v.x;
}

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y;
}

bool ccw(pt A, pt B, pt C) {
    static const LD eps = 1e-5;

```

```

        return cross(B-A, C-B) > eps;
    }

void convex_hull(pt *A, int n) {
    // build convex hull, write answer in (hull, nhull)
}

LD dist_from_line(pt A, pt B, pt C) {
    pt u = B-A;
    swap(u.x, u.y);
    u.x = -u.x;
    return fabs1(dot(u, C-A));
}

LD get_distance(pt *A, int n) {
    convex_hull(A, n);

    LD ret = 0.0;
    int n = nhull;
    pt *A = hull;

    if (n <= 3) {
        FOR(i, n) FOR(j, i) {
            ret = max(ret, A[i].dist(A[j]));
        }
        return ret;
    }

    int r = 1;
    while(dist_from_line(A[0], A[1], A[(r+1)%n]) > dist_from_line(A[0], A[1], A[r])) r = (r+1) % n;
    ret = max(ret, A[r].dist(A[0]));

    for (int i = 1; i < n; ++i) {
        while(dist_from_line(A[i], A[(i+1)%n], A[(r+1)%n]) > dist_from_line(A[i], A[(i+1)%n], A[r])) r = (r+1) % n;
        ret = max(ret, A[r].dist(A[i]));
    }
    return ret;
}

```

3.5 Half Plane Intersection (Randomized Incremental Algorithm)

```

const LD eps = 1e-8;

struct pt {
    LD x, y;
    pt (LD x=0, LD y=0) : x(x), y(y) {}
};

void read() {
    int a, b;
    scanf("%d%d", &a, &b);
    x=a;
    y=b;
}

pt operator + (const pt &o) const {
    return pt(x+o.x, y+o.y);
}

pt operator - (const pt &o) const {
    return pt(x-o.x, y-o.y);
}

} p[N];
int n;

struct Plane {
    LD a, b, c;
    int s;

    Plane(LD aa=0, LD bb=0, LD cc=0, int s=0) : s(s) {
        LD norm = sqrt1(aa*aa+bb*bb);
        a = aa / norm;
        b = bb / norm;
        c = cc / norm;
    }
};

LD dot(pt u, pt v) {
    return u.x * v.x + u.y * v.y;
}

```

```

LD sign(LD x) {
    if (x < -eps) return -1;
    if (x > +eps) return 1;
    return 0;
}

int get_sign(pt A, Plane p) {
    return sign(p.a * A.x + p.b * A.y + p.c);
}

struct HalfPlaneIntersection {
    vector<Plane> planes;

    void init() {
        planes.clear();
        planes.resize(0);
    }

    void add(Plane p) {
        if (p.s == 1) {
            p.c -= eps;
        }
        else {
            p.c += eps;
        }
        planes.pb(p);
    }

    bool check() {
        random_shuffle(all(planes));
        planes.pb(Plane(1, 0, INF, 1));
        planes.pb(Plane(1, 0, -INF, -1));
        planes.pb(Plane(0, 1, INF, 1));
        planes.pb(Plane(0, 1, -INF, -1));
        reverse(all(planes));

        pt cur(-INF, 0);
        for (int iter = 4; iter < sz(planes); ++iter) {
            auto p = planes[iter];

            if (get_sign(cur, p) == -p.s) {
                pt dir = pt(-p.b, p.a);
                vector<pt> A, B;

                FOR(i, iter) {
                    auto q = planes[i];

                    LD det = p.a * q.b - p.b * q.a;
                    if (fabs1(det) < eps) continue;
                    LD detx = q.c * p.b - p.c * q.b;
                    LD dety = p.c * q.a - q.c * p.a;
                    pt X = pt(detx/det, dety/det);

                    if (get_sign(X + dir, q) == q.s) {
                        A.pb(X);
                    }
                    else {
                        B.pb(X);
                    }
                }

                // A and B are not empty (note 4 initial half
                // planes)
                pt amax = A[0];
                for (auto P : A) {
                    if (dot(P, dir) > dot(amax, dir)) amax = P;
                }

                pt bmax = B[0];
                for (auto P : B) {
                    if (dot(P, dir) < dot(bmax, dir)) bmax = P;
                }

                if (amax.x < bmax.x) cur = amax;
                else cur = bmax;

                FOR(i, iter+1) {
                    if (get_sign(cur, planes[i]) == -planes[i].s)
                        return false;
                }
            }
        }

        return true;
    }
} hp;

bool check(int k) {
    hp.init();

```

```

FOR(i, n) {
    int j=(i+k)%n;

    LD a = p[i].y - p[j].y;
    LD b = p[j].x - p[i].x;
    LD c = -a * p[i].x - b * p[i].y;

    int z=(i-1+n)%n;
    int s = sign(a*p[z].x + b*p[z].y + c);

    hp.add(Plane(a, b, c, s));
}
return hp.check();
}

const LD pi = acos1(-1.0);
const LD eps = 1e-11;
const int N = 250007;

#define vect pt
struct pt {
    LL x, y;
    pt() : x(0), y(0) {}
    pt(LL x, LL y) : x(x), y(y) {}

    pt operator + (const pt &o) const {
        return pt(x + o.x, y + o.y);
    }

    pt operator - (const pt &o) const {
        return pt(x - o.x, y - o.y);
    }

    bool operator < (const pt &o) const {
        if (x != o.x) return x < o.x;
        return y < o.y;
    }

    LD len() const {
        return sqrt1((LD)x * x + (LD)y * y);
    }

    pt rotate() const {
        return pt(y, -x);
    }
};

vector<pt> mas[N];
LL ans = 0;

LD get_angle(LD ux, LD uy, LD x, LD y) {
    LD l1 = sqrt1(ux * ux + uy * uy);
    LD l2 = sqrt1(x * x + y * y);
    if (l1 < eps || l2 < eps) return 2 * pi;
    LD t = (ux * x + uy * y) / l1 / l2;
    if (t < -1.0 + eps) return pi;
    if (t > 1.0 - eps) return 0;
    return acos1(t);
}

void minkowski_sum(int na, pt *A, int nb, pt *B, int &nc, pt *C)
{
    int posa = 0;
    FOR(i, na) {
        if (A[i].x > A[posa].x || (A[i].x == A[posa].x && A[i].y
            > A[posa].y)) posa = i;
    }
    int posb = 0;
    FOR(i, nb) {
        if (B[i].x > B[posb].x || (B[i].x == B[posb].x && B[i].y
            > B[posb].y)) posb = i;
    }
    nc = 0;
    C[nc++] = A[posa] + B[posb];
    LD rot = 0.0;
    while (true) {
        int nposa = (posa + 1) % na;
        int nposb = (posb + 1) % nb;
        vect ua = vect(A[nposa].x - A[posa].x, A[nposa].y - A[
            posa].y).rotate();
        vect ub = vect(B[nposb].x - B[posb].x, B[nposb].y - B[
            posb].y).rotate();

        if (na == 3 && nb == 2 && rot > 0) {

```

3.6 Minkowski Sum of Convex Polygons

```

        cerr << ua.x << " " << ua.y << " " << cos1(rot) <<
            sin1(rot) << endl;
    }
    LD anga = get_angle(0.0 + ua.x, 0.0 + ua.y, cos1(rot),
        sin1(rot));
    LD angb = get_angle(0.0 + ub.x, 0.0 + ub.y, cos1(rot),
        sin1(rot));

    if (anga < angb) {
        rot += anga;
        posa = nposa;
    }
    else {
        rot += angb;
        posb = nposb;
    }
    if (rot > 2 * pi - eps) break;
    C[nc++] = A[posa] + B[posb];
}

pt tmp[N], H[N];
pt A[N], B[N], C[N];
int na, nb, nc;

LL cross(vect a, vect b) {
    return a.x * b.y - a.y * b.x;
}

bool ccw(pt A, pt B, pt C) {
    return cross(B - A, C - B) > 0;
}

void convex_hull(int &na, pt *A) {
    if (na <= 2) return;
    int nh = 0;
    sort(A, A + na);
    int ptr = 0;
    FOR(i, na) {
        while (ptr >= 2 && !ccw(tmp[ptr - 2], tmp[ptr - 1], A[i])
            ) --ptr;
        tmp[ptr++] = A[i];
    }
    FOR(i, ptr - 1) H[nh++] = tmp[i];
    ptr = 0;
    for (int i = na - 1; i >= 0; --i) {
        while (ptr >= 2 && !ccw(tmp[ptr - 2], tmp[ptr - 1], A[i])
            ) --ptr;
        tmp[ptr++] = A[i];
    }
    FOR(i, ptr - 1) H[nh++] = tmp[i];
    na = nh;
    FOR(i, na) A[i] = H[i];
}

void solvefor(int l, int r) {
    if (l == r) return;

    int m = (l + r) / 2;
    solvefor(l, m);
    solvefor(m + 1, r);

    na = 0;
    for (int i = l; i <= m; ++i) {
        for (pt P : mas[i]) A[na++] = P;
    }
    nb = 0;
    for (int i = m + 1; i <= r; ++i) {
        for (pt P : mas[i]) B[nb++] = pt(-P.x, -P.y);
    }
    convex_hull(na, A);
    convex_hull(nb, B);

    if (na == 0 || nb == 0) return;

    minkowski_sum(na, A, nb, B, nc, C);
    FOR(i, nc) {
        ans = max(ans, C[i].x * C[i].x + C[i].y * C[i].y);
    }
}

```

3.7 Pair of Intersecting Segments

```

const LD pi = acos1(-1.0);
const LD eps = 1e-7;
struct pt {
    LD x, y;

```

```

pt(LD x = 0.0, LD y = 0.0) : x(x), y(y) {}

void rotate(LD ang) {
    LD fi = atan2l(y, x);
    LD len = sqrtl(x * x + y * y);
    fi += ang;
    x = len * cosl(fi);
    y = len * sinl(fi);
}

vector<pair<pt, pt>> seg;
int n;

bool inSegment(pt &M, pt &A, pt &B) {
    return min(A.x, B.x) < M.x + eps && M.x < max(A.x, B.x) + eps
    && min(A.y, B.y) < M.y + eps && M.y < max(A.y, B.y) + eps;
}

bool intersects(LD a, LD b, LD c, LD d) {
    if (a > b) swap(a, b);
    if (c > d) swap(c, d);
    return max(a, c) < min(b, d) + eps;
}

bool intersects(pt &A, pt &B, pt &C, pt &D) {
    LD a1 = A.y - B.y;
    LD b1 = B.x - A.x;
    LD c1 = -a1 * A.x - b1 * A.y;

    LD a2 = C.y - D.y;
    LD b2 = D.x - C.x;
    LD c2 = -a2 * C.x - b2 * C.y;

    LD det = a1 * b2 - a2 * b1;

    if (fabsl(det) < eps) {
        if (fabsl(c1 * b2 - c2 * b1) < eps && fabsl(c1 * a2 - c2 * a1) < eps)
            return intersects(A.x, B.x, C.x, D.x) && intersects(A.y, B.y, C.y, D.y);
        return false;
    }
    else {
        pt M;
        M.x = (b1 * c2 - b2 * c1) / det;
        M.y = (a2 * c1 - a1 * c2) / det;
        return inSegment(M, A, B) && inSegment(M, C, D);
    }
}

inline bool intersects(int i, int j) {
    return intersects(seg[i].first, seg[i].second, seg[j].first, seg[j].second);
}

LD sweepX;
struct segment {
    int id;
    segment(int id = 0) : id(id) {}
    bool operator < (const segment &o) const {
        pt& A = seg[id].first;
        pt& B = seg[id].second;
        pt& C = seg[o.id].first;
        pt& D = seg[o.id].second;
        /*
        ete unenq vertical hatvacner, hetevyal kerpov enq y hashvum
        aglorithmy jisht e ashxatum naev
        vertical uxixneri depqum, vorpes y yntrum enq kamayakan,
        orinak hatvacaci araji keti Y
        bolor verticalnery skzbbic insert en arvum, heto deleta
        TESTED
        ete ayspes aneng, nergevum(144-159) petq che pttel ketery
        LD y1 = fabsl(B.x - A.x) > eps ? A.y + (B.y - A.y) * (
        sweepX - A.x) / (B.x - A.x) : A.y;
        LD y2 = fabsl(D.x - C.x) > eps ? C.y + (D.y - C.y) * (
        sweepX - C.x) / (D.x - C.x) : C.y;
        */
        LD y1 = A.y + (B.y - A.y) * (sweepX - A.x) / (B.x - A.x);
        LD y2 = C.y + (D.y - C.y) * (sweepX - C.x) / (D.x - C.x);
        if (fabsl(y1 - y2) < eps) return id < o.id;
        return y1 < y2;
    }
};

set<segment> status;

// main
scanf("%d", &n);
for (int i = 0; i < n; ++i) {
    int ax, ay, bx, by;
    scanf("%d%d%d%d", &ax, &ay, &bx, &by);
    seg.pb(mp(pt(ax, ay), pt(bx, by)));
}

```

```

}

while (true) {
    LD rotAngle = rand() / (LD)RAND_MAX * 2.0 * pi;
    bool ok = true;
    for (int i = 0; i < n; ++i) {
        seg[i].first.rotate(rotAngle);
        seg[i].second.rotate(rotAngle);
        if (fabsl(seg[i].first.x - seg[i].second.x) < eps) {
            ok = false;
            break;
        }
    }
    if (ok) break;
}

vector<pair<LD, int>> events;

for (int i = 0; i < n; ++i) {
    if (seg[i].first.x > seg[i].second.x) swap(seg[i].first, seg[i].second);

    events.pb(mp(seg[i].first.x, -i - 1));
    events.pb(mp(seg[i].second.x, +i + 1));
}

sort(all(events));
for (int i = 0; i < 2 * n; ++i) {
    int ty = events[i].second;
    sweepX = events[i].first;
    if (ty < 0) // open
    {
        int id = -ty - 1;
        auto it = status.lower_bound(segment(id));
        if (it != status.begin()) {
            auto oit = it;
            --oit;
            if (intersects(oit->id, id)) {
                printf("YES\n%d %d\n", id + 1, oit->id + 1);
                return 0;
            }
        }
        if (it != status.end() && intersects(id, it->id)) {
            printf("YES\n%d %d\n", id + 1, it->id + 1);
            return 0;
        }
        status.insert(segment(id));
    }
    else // close
    {
        int id = ty - 1;
        auto it = status.lower_bound(segment(id));
        if (it != status.begin() && it != status.end()) {
            auto lit = it;
            --lit;
            auto rit = it;
            ++rit;
            if (rit != status.end() && intersects(lit->id, rit->id)) {
                printf("YES\n%d %d\n", lit->id + 1, rit->id + 1);
                return 0;
            }
        }
        status.erase(segment(id));
    }
}

puts("NO");

```

3.8 Minimum Enclosing Circle (Randomized Incremental Algorithm)

```

// One can do 2 nested ternary searches: O(N * (logN)^2)
struct MEC {

    bool inside_circle(pt A, pt C, LD R) {
        return A.dist2(C) < R + eps;
    }

    LD cross(pt u, pt v) {
        return u.x * v.y - u.y * v.x;
    }

    pt intersect_bisectors(pt A, pt B, pt C, pt D) {
        pt AB = (A+B) / 2.0;
        pt CD = (C+D) / 2.0;

        pt n1 = A - B;
        pt n2 = C - D;

        LD a1 = n1.x;

```

```

        LD b1 = n1.y;
        LD c1 = -a1 * AB.x - b1 * AB.y;

        LD a2 = n2.x;
        LD b2 = n2.y;
        LD c2 = -a2 * CD.x - b2 * CD.y;

        LD det = a1 * b2 - a2 * b1;
        assert(fabsl(det) > eps);

        LD detx = c2 * b1 - c1 * b2;
        LD dety = c1 * a2 - c2 * a1;

        return pt(detx/det, dety/det);
    }

    void update_ans(pt O, vector<pt>& p, pt A, pt B, pt& C, LD& R)
    {
        LD cur = O.dist2(A);
        if (cur > R) return;
        for (auto E : p) {
            if (!inside_circle(E, O, cur)) return;
        }
        R = cur;
        C = O;
    }

    void consider(vector<pt> &p, vector<pt>& side, pt A, pt B, pt& C, LD& R)
    {
        pt mini(INF, INF, INF);
        pt maxi(-INF, -INF, -INF);

        for (auto E : side) {
            pt X = intersect_bisectors(A, E, A, B);
            if (X < mini) mini = X;
            if (maxi < X) maxi = X;
        }

        if (mini.x < INF/2) {
            update_ans(mini, p, A, B, C, R);
            update_ans(maxi, p, A, B, C, R);
        }
    }

    void two_points_known(vector<pt>& p, pt A, pt B, pt& C, LD& R)
    {
        pt dir = B-A;

        R=INF;
        update_ans((A+B)/2.0, p, A, B, C, R);

        vector<pt> lside, rside;
        for (auto E : p) {
            if (cross(dir, E-A) > +eps) lside.pb(E);
            if (cross(dir, E-A) < -eps) rside.pb(E);
        }
        consider(p, lside, A, B, C, R);
        consider(p, rside, A, B, C, R);
    }

    void one_point_known(vector<pt>& p, pt A, pt& C, LD& R) {
        random_shuffle(all(p));

        C = (p[0] + A) / 2.0;
        R = p[0].dist2(A) / 2.0;

        for (int i = 1; i < sz(p); ++i) {
            if (inside_circle(p[i], C, R)) continue;
            vector<pt> mas(p.begin(), p.begin()+i);
            two_points_known(mas, A, p[i], C, R);
        }
    }

    LD get(vector<pt> &p) {
        random_shuffle(all(p));
        pt C = p[0];
        LD R = 0.0;

        for (int i = 1; i < sz(p); ++i) {
            if (inside_circle(p[i], C, R)) continue;
            vector<pt> mas(p.begin(), p.begin()+i);
            one_point_known(mas, p[i], C, R);
        }

        return R;
    }
} mec;

```


4 Graphs

4.1 Maximum Flow (Dinic)

```
struct edge {
    int u, v, cap, flow;
    edge(int u = 0, int v = 0, int cap = 0, int flow = 0) : u(u),
        v(v), cap(cap), flow(flow) {}
};

int n, m, s, t, Q[N], ptr[N], dist[N];
vector<edge> edges;
vector<int> G[N];

void addEdge(int u, int v, int cap) {
    G[u].pb(sz(edges));
    edges.pb(edge(u, v, cap, 0));
    G[v].pb(sz(edges));
    edges.pb(edge(v, u, 0, 0));
}

bool bfs() {
    int l = 0;
    int r = 0;
    fill(dist, dist + N, -1);
    dist[s] = 0;
    Q[r++] = s;
    while (l < r) {
        int u = Q[l++];
        for (int i = 0; i < sz(G[u]); ++i) {
            int id = G[u][i];
            int to = edges[id].v;
            if (dist[to] != -1 || edges[id].flow == edges[id].cap) {
                continue;
            }
            dist[to] = dist[u] + 1;
            Q[r++] = to;
        }
    }
    return dist[t] != -1;
}

int dfs(int u, int flow = INF) {
    if (!flow) {
        return 0;
    }
    if (u == t) {
        return flow;
    }
    for (; ptr[u] < sz(G[u]); ++ptr[u]) {
        int id = G[u][ptr[u]];
        int to = edges[id].v;
        if (dist[to] != dist[u] + 1) {
            continue;
        }
        int pushed = dfs(to, min(flow, edges[id].cap - edges[id].flow));
        if (pushed) {
            edges[id].flow += pushed;
            edges[id ^ 1].flow -= pushed;
            return pushed;
        }
    }
    return 0;
}

LL dinic(int from, int to) {
    LL flow = 0;
    s = from;
    t = to;
    while (bfs()) {
        fill(ptr, ptr + N, 0);
        int pushed;
        while (pushed = dfs(s)) {
            flow += pushed;
        }
    }
    return flow;
}
```

4.2 Facet Finding

```
const LD eps = 1e-12;
struct pt {
    LD x, y;
    pt(LD x = 0.0, LD y = 0.0) : x(x), y(y) {}
};
typedef pt vect;
vector<pair<pt, pt> > lines;
vector<vector<pt> > in;
vector<vector<int> > G;
vector<pt> mas;
int n, nlines;
map<LL, LD> M;

bool cmp(const pt &A, const pt &B) {
    if (fabsl(A.x - B.x) < eps) return A.y + eps < B.y;
    return A.x + eps < B.x;
}

void unique(vector<pt> &mas) {
    if (sz(mas) == 0) return;
    sort(all(mas), cmp);
    vector<pt> tmp;
    tmp.pb(mas[0]);
    for (int i = 1; i < sz(mas); ++i)
        if (cmp(mas[i], mas[i - 1]) || cmp(mas[i - 1], mas[i])) tmp.pb(mas[i]);
    mas = tmp;
}

int find(pt &A) {
    for (int i = 0; i < sz(mas); ++i)
        if (!cmp(mas[i], A) && !cmp(A, mas[i])) return i;
    throw "no such point finded";
}

bool ccw(vect &u, vect &v) {
    return u.x * v.y - u.y * v.x > eps;
}

void dfs(int i, vect &u, vector<int> &facet) {
    if (i == facet.front()) return;
    facet.pb(i);
    vect best = u;
    int pos = -1;
    for (int j : G[i]) {
        vect v(mas[j].x - mas[i].x, mas[j].y - mas[i].y);
        if (ccw(best, v) && ccw(u, v)) best = v, pos = j;
    }
    if (pos == -1) facet.clear();
    else dfs(pos, best, facet);
}

int main() {
    #ifndef harhro94
    freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    #else
    #define task "areas"
    freopen(task".in", "r", stdin);
    freopen(task".out", "w", stdout);
    #endif

    int nlines;
    cin >> nlines;
    lines.resize(nlines);
    for (int i = 0; i < nlines; ++i)
        cin >> lines[i].first.x >> lines[i].first.y >> lines[i].second.x >> lines[i].second.y;
    in.resize(nlines);

    for (int i = 0; i < nlines; ++i) {
        for (int j = i + 1; j < nlines; ++j) {
            pt A = lines[i].first;
            pt B = lines[i].second;
            pt C = lines[j].first;
            pt D = lines[j].second;

            LD a1 = B.y - A.y;
            LD b1 = A.x - B.x;
            LD c1 = B.x * A.y - B.y * A.x;

            LD a2 = D.y - C.y;
            LD b2 = C.x - D.x;
            LD c2 = D.x * C.y - D.y * C.x;

            LD det = a1 * b2 - a2 * b1;

            if (fabsl(det) < eps) continue;

            LD detx = -c1 * b2 + c2 * b1;
            LD dety = -a1 * c2 + a2 * c1;
            mas.pb(pt(detx / det, dety / det));
            in[i].pb(mas.back());
            in[j].pb(mas.back());
        }
    }
    unique(mas);
    for (int i = 0; i < nlines; ++i) unique(in[i]);
    int n = sz(mas);
    G.resize(n);
    for (int i = 0; i < nlines; ++i) {
        int cnt = sz(in[i]);
        for (int j = 1; j < cnt; ++j) {
            int u = find(in[i][j]);
            int v = find(in[i][j - 1]);
            G[u].pb(v);
            G[v].pb(u);
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j : G[i]) {
            vect u(mas[j].x - mas[i].x, mas[j].y - mas[i].y);
            vector<int> facet;
            facet.pb(i);
            dfs(j, u, facet);
            if (sz(facet) == 0) continue;
            //cerr << "Facet finded !!!\n";
            //for (int p : facet) cerr << "\t" << mas[p].x << " " << mas[p].y << endl;
            LD S = 0.0;
            for (int u = 1; u < sz(facet); ++u) {
                pt B = mas[facet[u]];
                pt A = mas[facet[u - 1]];
                S += (A.y + B.y) * (B.x - A.x);
            }
            pt A = mas[facet.back()];
            pt B = mas[facet.front()];
            S += (A.y + B.y) * (B.x - A.x);
            S = fabsl(S) / 2.0;
            sort(all(facet));
            LL h = 0;
            for (int p : facet) h = 1000000007 * h + p;
            if (S > 1e-9) M[h] = S;
        }
    }
    vector<LD> S;
    for (pair<LL, LD> p : M) S.pb(p.second);
    cout << sz(S) << endl;
    sort(all(S));
    for (LD s : S) cout << fixed << setprecision(9) << s << endl;

    #ifndef harhro94
    cerr << fixed << setprecision(3) << "\nExecution time = " << clock() / 1000.0 << "\n";
    #endif
    return 0;
}
```

4.3 LCA O(1)

```
const int N = 100007;
int n, timer, tin[N], id[N], ptr, mas[N + N], pos[N], mlog[N + N], mini[20][N + N];
vector<int> G[N];

void dfs(int u, int p = -1) {
    id[timer] = u;
    tin[u] = timer++;
    pos[u] = ptr;
    mas[ptr++] = tin[u];
    for (int to : G[u]) {
        if (to != p) {
            dfs(to, u);
            mas[ptr++] = tin[u];
        }
    }
}

int lca(int u, int v) {
    int l = pos[u], r = pos[v];
    if (l > r) swap(l, r);
    int k = mlog[r - l + 1];
    int w = id[min(mini[k][l], mini[k][r - (1 << k) + 1])];
}
```

```

    return w;
}

dfs(0);
mlog[1] = 0;
for (int i = 2; i < ptr; ++i) mlog[i] = 1 + mlog[i >> 1];
for (int i = 0; i < ptr; ++i) mini[0][i] = mas[i];
for (int k = 0; k < 19; ++k) {
    for (int i = 0; i < ptr; ++i) {
        if (i + (1 << (k + 1)) - 1 >= ptr) break;
        mini[k + 1][i] = min(mini[k][i], mini[k][i + (1 << k)]);
    }
}

```

4.4 Min Cost Max Flow (Dijkstra with Potentials)

```

struct MCMF {
    vector<Edge> E;
    vector<int> G[N];
    int n, s, t;
    int Q[N], pre[N], preId[N];
    LL pi[N], dist[N];
    bool is[N];
    LL ansFlow, ansCost;

    void addEdge(int u, int v, LL cap, LL cost) {
        G[u].pb(sz(E));
        E.pb(Edge(v, cap, cost));
        G[v].pb(sz(E));
        E.pb(Edge(u, 0, -cost));
    }

    void init(int n, int s, int t) {
        this->n = n;
        this->s = s;
        this->t = t;
        assert(t < n);
        FOR(i, n) G[i].clear();
        E.clear();
        ansFlow = ansCost = 0;
    }

    void fordBellman() {
        FOR(i, n) {
            is[i] = false;
            dist[i] = INF;
        }
        int l = 0, r = 0;
        Q[r++] = s;
        dist[s] = 0;
        is[s] = true;
        while (l != r) {
            int u = Q[l++];
            if (l == N) l = 0;
            is[u] = false;
            for (int id : G[u]) {
                int to = E[id].to;
                if (E[id].cap - E[id].flow > 0 && dist[to] > dist[u] + E[id].cost) {
                    dist[to] = dist[u] + E[id].cost;
                    if (!is[to]) {
                        is[to] = true;
                        Q[r++] = to;
                        if (r == N) r = 0;
                    }
                }
            }
        }
    }

    bool dijkstra() {
        FOR(i, n) dist[i] = INF;
        dist[s] = 0;
        set<pair<LL, int>> S;
        S.insert(mp(0, s));
        while (!S.empty()) {
            int u = S.begin()->second;
            S.erase(S.begin());
            for (int id : G[u]) {
                int to = E[id].to;
                LL cost = E[id].cost + pi[u] - pi[to];
                if (E[id].cap - E[id].flow > 0 && dist[to] > dist[u] + cost) {
                    S.erase(mp(dist[to], to));

```

```

                pre[to] = u;
                preId[to] = id;
                dist[to] = dist[u] + cost;
                S.insert(mp(dist[to], to));
            }
        }
        // new_pi = pi + dist, easy to prove
        FOR(i, n) {
            if (dist[i] != INF) pi[i] += dist[i];
        }
        return dist[t] != INF;
    }

    void augment() {
        int u = t;
        LL minCap = INF;
        LL totalCost = 0;
        while (u != s) {
            int id = preId[u];
            minCap = min(minCap, E[id].cap - E[id].flow);
            totalCost += E[id].cost;
            u = pre[u];
        }

        ansFlow += minCap;
        ansCost += minCap * totalCost;

        u = t;
        while (u != s) {
            int id = preId[u];
            E[id].flow += minCap;
            E[id ^ 1].flow -= minCap;
            u = pre[u];
        }
    }

    pair<LL, LL> mcmf() {
        fordBellman();
        FOR(i, n) pi[i] = dist[i];
        while (dijkstra()) {
            augment();
        }
        return mp(ansFlow, ansCost);
    }
} mf;

```

4.5 Offline LCA (Tarjan)

```

int n, m, timer, par[N], cand[N], u1[M], u2[M], pos1[M], pos2[M],
lca[M];
vector<pair<int, int>> query[M];
vector<int> G[N];
bool used[N];

int getPar(int u) {
    return (par[u] == u ? u : (par[u] = getPar(par[u])));
}

void unite(int u, int v, int newCand) {
    u = getPar(u);
    v = getPar(v);
    if (rand() & 1) {
        swap(u, v);
    }
    par[u] = v;
    cand[v] = newCand;
}

void dfs(int u) {
    used[u] = true;
    par[u] = u;
    cand[u] = u;
    for (int to : G[u]) {
        if (!used[to]) {
            dfs(to);
            unite(to, u, u);
        }
    }
    for (int i = 0; i < sz(query[u]); ++i) {
        int v = query[u][i].first;
        if (used[v]) {
            query[u][i].second = cand[getPar(v)];
        }
    }
}

```

```

void process() {
    for (int i = 0; i < m; ++i) {
        int u = u1[i];
        int v = u2[i];
        pos1[i] = sz(query[u]);
        query[u].pb(mp(v, -1));
        pos2[i] = sz(query[v]);
        query[v].pb(mp(u, -1));
    }
    dfs(1);
    for (int i = 0; i < m; ++i) {
        int p1 = pos1[i];
        int p2 = pos2[i];
        int u = u1[i];
        int v = u2[i];
        if (query[u][p1].second != -1)
            lca[i] = query[u][p1].second;
        else if (query[v][p2].second != -1)
            lca[i] = query[v][p2].second;
        else
            assert(false);
    }
}

```

4.6 Online LCA

```

int n, m, par[N], dep[N], up[20][N];
vector<int> G[N];

void dfs(int u, int p = 1, int d = 0) {
    par[u] = p;
    dep[u] = d;
    for (int i = 0; i < sz(G[u]); ++i) {
        int to = G[u][i];
        if (to != p) dfs(to, u, d + 1);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int diff = dep[u] - dep[v];
    for (int i = 0; (1 << i) <= diff; ++i)
        if (((diff >> i) & 1) == 1) u = up[i][u];
    if (u == v) return u;
    for (int i = 19; i >= 0; --i)
        if (up[i][u] != up[i][v])
            u = up[i][u], v = up[i][v];
    return up[0][u];
}

// preprocessing
dfs(1);
for (int i = 1; i <= n; ++i) up[0][i] = par[i];
for (int k = 1; k < 20; ++k)
    for (int i = 1; i <= n; ++i)
        up[k][i] = up[k - 1][up[k - 1][i]];

```

4.7 Rooted Tree Isomorphism

```

// Slow string implementation
string dfsSlow(int root, int par, vector<int> *G) {
    vector<string> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfsSlow(to, root, G));
    }
    sort(all(childs));
    string cur = "(";
    for (auto s : childs) cur += s;
    cur += ")";
    return cur;
}

// fast algorithm, DETERMINISTIC
map<vector<int>, int> ID;
int curid = 0;

int getId(vector<int> &v) {
    if (ID.count(v)) return ID[v];
    ID[v] = curid++;
}

```

```

    return curid - 1;
}

int dfs(int root, int par, set<int> +G) {
    vector<int> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfs(to, root, G));
    }
    sort(all(childs));
    return getId(childs);
}

// faster algorithm: vector<int>'s in map is slow
// we will hash that vectors then add hashes to the map

map<LL, int> ID;
int curid = 0;

LL getVectorHash(vector<int> &v) {
    static const LL P = 1000003;
    LL h = 1;
    for (int u : v) h = h * P + u;
    return h;
}

int getId(vector<int> &v) {
    LL h = getVectorHash(v);
    if (ID.count(h)) return ID[h];
    ID[h] = curid++;
    return curid - 1;
}

int dfs(int root, int par, set<int> +G) {
    vector<int> childs;
    for (int to : G[root]) {
        if (to != par) childs.pb(dfs(to, root, G));
    }
    sort(all(childs));
    return getId(childs);
}

```

4.8 Vertex Cover (Types of Vertices)

```

const int N = 2007;
int n, m, k, mt[N];
int ptr[N], deg[N], G[N][N];
bool used[N];
char ans[N];
queue<int> Q;

bool kuhn(int u)
{
    used[u] = true;
    for (int i = 0; i < ptr[u]; ++i)
    {
        int to = G[u][i];
        if (mt[to] == -1 || (!used[mt[to]] && kuhn(mt[to])))
        {
            mt[to] = u;
            mt[u] = to;
            return true;
        }
    }
    return false;
}

// init some matching
for (int i = 1; i <= n + m; ++i)
    mt[i] = -1;

for (int i = 1; i <= n; ++i) deg[i] = ptr[i];

int size = 0;
FOR(iter, n)
{
    int u = -1;
    int best = N + N;
    for (int i = 1; i <= n; ++i)
    {
        if (deg[i] >= 1 && deg[i] < best)
        {
            best = deg[i];
            u = i;
        }
    }
    if (u == -1) break;
}

```

```

deg[u] = 0;
FOR(i, ptr[u])
{
    int to = G[u][i];
    if (mt[to] == -1)
    {
        mt[to] = u;
        mt[u] = to;
        ++size;
        FOR(j, ptr[to]) --deg[G[to][j]];
        break;
    }
}
}

// kuhn
for (int i = 1; i <= n; ++i)
{
    if (mt[i] == -1)
    {
        for (int j = 1; j <= n; ++j) used[j] = false;
        size += kuhn(i);
    }
}

// bfs from 2 unmatched sides
for (int i = 1; i <= n + m; ++i) ans[i] = 'E';
for (int i = 1; i <= n + m; ++i)
{
    if (mt[i] == -1)
    {
        ans[i] = 'N';
        Q.push(i);
    }
}
while (!Q.empty())
{
    int u = Q.front();
    Q.pop();
    if (ans[u] == 'N')
    {
        FOR(i, ptr[u])
        {
            int to = G[u][i];
            if (ans[to] == 'E')
            {
                ans[to] = 'A';
                Q.push(to);
            }
        }
    }
    else
    {
        int to = mt[u];
        if (ans[to] == 'N')
        {
            ans[to] = 'N';
            Q.push(to);
        }
    }
}

```

5 Math

5.1 Chinese Remainder Theorem

```

int power(int a, int b, int mod) {
    if (b == 0) return 1;
    if (b & 1) return (a * power(a, b - 1, mod)) % mod;
    int ret = power(a, b >> 1, mod);
    return (ret * ret) % mod;
}

int getInverse(int a, int mod) {
    a %= mod;
    return power(a, mod - 2, mod);
}

LL solve(vector<int> p, vector<int> r) {
    LL M = 1;
    int n = sz(p);
    for (int i = 0; i < n; ++i) M *= p[i];
    vector<vector<int>> inv(n, vector<int>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)

```

```

        if (i != j) inv[i][j] = getInverse(p[i], p[j]);
    vector<LL> x(n);
    for (int i = 0; i < n; ++i) {
        x[i] = r[i];
        for (int j = 0; j < i; ++j) {
            x[i] = (x[i] - x[j]) * inv[j][i];
            x[i] %= p[i];
            if (x[i] < 0) x[i] += p[i];
        }
    }
    LL ret = 0;
    LL cur = 1;
    for (int i = 0; i < n; ++i) {
        ret += x[i] * cur;
        cur *= p[i];
        ret %= M;
        cur %= M;
    }
    return ret;
}

```

5.2 Discrete Logarithm and Square Root

```

// g ^ x = a (mod p)
int discreteLog(int g, int a, int p) {
    int s = (int)sqrt(p + 0.0);
    vector<pair<int, int>> A, B;
    int gs = power(g, s, p), cur = 1;
    for (int t = 0; t * s < p - 1; ++t, cur = cur * gs % p) {
        A.pb(mp(cur, t));
    }
    int invg = power(g, p - 2, p);
    cur = a;
    for (int i = 0; i < s; ++i, cur = cur * invg % p) {
        B.pb(mp(cur, i));
    }
    sort(all(A));
    sort(all(B));
    int i = 0, j = 0;
    while (i < sz(A) && j < sz(B)) {
        if (A[i].first == B[j].first) return A[i].second * s + B[j].second;
        if (A[i].first < B[j].first) ++i;
        else ++j;
    }
    assert(false);
}

// to find square root
int g = findGenerator(p);
int r = discreteLog(g, a, p);
if (r & 1) puts("No root");
else {
    int x1 = power(g, r / 2, p);
    int x2 = p - x1;
    if (x1 > x2) swap(x1, x2);
    if (x1 == x2) {
        assert(p == 2);
        puts("1");
    }
    else printf("%d %d\n", x1, x2);
}

```

5.3 Extended Euclid Algorithm

```

int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int g = gcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return g;
}

```

5.4 Fast Fourier Transform

```
void DFT(comp *a, int n) {
    int bitlen = 0;
    while ((1 << bitlen) < n) ++bitlen;
    for (int i = 0; i < n; ++i) {
        int r = rev(i, bitlen);
        if (i < r) swap(a[i], a[r]);
    }
    for (int len = 2; len <= n; len <= 1) {
        int half = (len >> 1);
        comp wlen(cos(2 * pi / len), sin(2 * pi / len));
        for (int i = 0; i < n; i += len) {
            comp power = 1;
            for (int j = i, l = i, r = i + half; j < i + half; ++j,
                ++l, ++r, power *= wlen) {
                comp u = a[l], v = power * a[r];
                a[j] = u + v;
                a[j + half] = u - v;
            }
        }
    }

    void inverseDFT(comp *a, int n) {
        DFT(a, n);
        for (int i = 0; i < n; ++i) a[i] /= n;
        reverse(a + 1, a + n);
    }
}
```

5.5 Generator of \mathbb{Z}_p

```
vector<int> d[N];
int cur, used[N];
bool is[N];
int gen[N];

void sieve() {
    fill(is + 2, is + N, true);
    for (int i = 2; i < 200; ++i)
        if (is[i]) for (int j = i * i; j < N; j += i)
            is[j] = false;
    for (int i = 0; i < N; ++i)
        if (is[i]) for (int j = i; j < N; j += i) d[j].pb(i);
}

int power(int a, int n, int MOD) {
    int res = 1;
    while (n) {
        if (n & 1) res = res * a % MOD;
        a = a * a % MOD;
        n >>= 1;
    }
    return res;
}

int findGenerator(int p) {
    if (gen[p]) return gen[p];
    ++cur;
    while (true) {
        int g = rand() % (p - 1) + 1;
        if (used[g] == cur) continue;
        used[g] = cur;
        bool ok = true;
        for (int i = 0; i < sz(d[p - 1]); ++i) {
            int t = d[p - 1][i];
            if (power(g, (p - 1) / t, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return (gen[p] = g);
    }
}
```

5.6 Linear Time Precalculation of Inverses

```
/**/ O(MOD) methods to precalculate all inverses
a)
r[1] = 1;
for (int i=2; i<m; ++i)
    r[i] = (m - (m/i) * r[m/i] % m) % m;
```

```
b)
r[a+b] = r[a] * r[b] // O(1)
r[p] = power(p, MOD-2) // O(log(MOD))
total: O(MOD)
****/

const LL MOD = 1000000000 + 7LL;
const int N = 2000007;

LL f[N], invf[N];
int lp[N];
int pr_cnt, pr[N];

void sieve() {
    lp[1] = 1;
    for (int i = 2; i < N; ++i) {
        if (lp[i] == 0) {
            pr[pr_cnt++] = i;
            lp[i] = i;
        }
        for (int j = 0; j < pr_cnt && pr[j] <= lp[i] && pr[j] * i
            < N; ++j) {
            lp[pr[j] * i] = pr[j];
        }
    }
}
```

```
LL power(LL a, LL b) {
    if (b == 0) return 1;
    if (b & 1) return a * power(a, b-1) % MOD;
    LL r = power(a, b >> 1);
    return r * r % MOD;
}
```

```
inline LL inv(LL a) {
    return power(a, MOD-2);
}

inline LL C(int n, int k) {
    if (k < 0 || k > n) return 0;
    return f[n] * invf[k] % MOD * invf[n-k] % MOD;
}
```

```
void precalc() {
    f[0] = 1;
    for (int i = 1; i < N; ++i) {
        f[i] = i * f[i-1] % MOD;
    }
    sieve();
    for (int i = 1; i < N; ++i) {
        if (lp[i] == i) invf[i] = inv(i);
        else invf[i] = invf[i / lp[i]] * invf[lp[i]] % MOD;
    }
    invf[0] = 1;
    for (int i = 1; i < N; ++i) {
        invf[i] = invf[i] * invf[i-1] % MOD;
    }
}
```

6 Strings

6.1 Aho-Corasick

```
int code(char ch) {
    if (ch >= 'a' && ch <= 'z') return ch - 'a';
    if (ch >= 'A' && ch <= 'Z') return ch - 'A' + 26;
    assert(ch >= '0' && ch <= '9');
    return ch - '0' + 52;
}

const int A = 64;
const int N = 100007;

struct AC {
    int to[64][N], go[64][N];
    int par[N], dep[N], suff[N];
    int cur_node;
    char pch[N];
}
```

```
/**/ additional information
int next_term[N], min_len[N];
bool term[N];

AC() {
    memset(suff, -1, sizeof suff);
    memset(go, -1, sizeof go);
    memset(to, -1, sizeof to);
    cur_node = 1;
}
```

```
void add(string st) {
    int cur = 0;
    for (char symbol : st) {
        int ch = code(symbol);
        if (to[ch][cur] == -1) {
            to[ch][cur] = cur_node++;
        }
        int nxt = to[ch][cur];
        pch[nxt] = ch;
        par[nxt] = cur;
        dep[nxt] = dep[cur] + 1;
        cur = nxt;
    }
    term[cur] = true;
}
```

```
int get_suff(int node) {
    if (node == 0) return 0;
    if (par[node] == 0) return 0;
    if (suff[node] != -1) return suff[node];

    int ret = get_go(pch[node], get_suff(par[node]));
    suff[node] = ret;
    return ret;
}
```

```
int get_go(int ch, int node) {
    if (to[ch][node] != -1) return to[ch][node];
    if (go[ch][node] != -1) return go[ch][node];
    if (node == 0) return 0;
    int ret = get_go(ch, get_suff(node));
    go[ch][node] = ret;
    return ret;
}
```

```
void calc_term() {
    vector<pair<int, int>> mas;
    FOR(i, cur_node) {
        mas.pb(mp(dep[i], i));
    }
    sort(all(mas));
    for (auto p : mas) {
        int i = p.second;
        next_term[i] = -1;
        int to = get_suff(i);
        if (term[to]) next_term[i] = to;
        else next_term[i] = next_term[to];
    }

    for (auto p : mas) {
        int i = p.second;
        min_len[i] = N;
        if (term[i]) min_len[i] = dep[i];
        if (next_term[i] != -1) {
            min_len[i] = min(min_len[i], min_len[next_term[i]]);
        }
    }
}
```

6.2 Discrete Manacer

```
n = strlen(st);
int l = 0, r = -1;
for (int i = 0; i < n; ++i) {
    dl[i] = 1;
    if (i <= r) dl[i] = min(dl[l + (r - i)], r - i + 1);
    while (i + dl[i] < n && i - dl[i] >= 0 && st[i + dl[i]] ==
        st[i - dl[i]]) ++dl[i];
    if (i + dl[i] - 1 > r) {
        l = i - dl[i] + 1;
        r = i + dl[i] - 1;
    }
}
```

```

}
l = 0, r = -1;
for (int i = 0; i < n; ++i) {
    d2[i] = 0;
    if (i <= r) d2[i] = min(d2[l + (r - i) + 1], r - i + 1);
    while (i + d2[i] < n && i - d2[i] - 1 >= 0 && st[i + d2[i]]
        == st[i - d2[i] - 1]) ++d2[i];
    if (i + d2[i] - 1 > r) {
        l = i - d2[i] + 1 - 1;
        r = i + d2[i] - 1;
    }
}

```

6.3 Palindromic Tree

```

struct node {
    int len, suff;
    int to[2]; // size of the alphabet

    node() {
        to[0] = to[1] = -1;
        suff = -1;
        len = -1;
    }
} T[N];
int size = 2; // 0 -> -1 / 1 -> e
int maxPal = 0;
char st[N], ans[N];

bool addChar(int i, int c) {
    bool ret = false;
    while (true) {
        int curLen = T[maxPal].len;
        if (st[i] == st[i - curLen - 1]) {
            int v;
            if (T[maxPal].to[c] == -1) {
                v = size++;
                T[maxPal].to[c] = v;
                T[v].len = curLen + 2;
                ret = true;
            }
            else v = T[maxPal].to[c];
        }
        else v = T[maxPal].to[c];
        if (T[v].len == 1) T[v].suff = 1;
        else {
            while (true) {
                maxPal = T[maxPal].suff;
                if (st[i] == st[i - T[maxPal].len - 1]) {
                    T[v].suff = T[maxPal].to[c];
                    break;
                }
            }
            maxPal = v;
            break;
        }
        maxPal = T[maxPal].suff;
    }
    return ret;
}

// init
T[1].len = 0;
T[1].suff = 0; // e -> "-1"

```

6.4 Prefix Function

```

p[0] = 0;
for (int i = 1; i < n + m + 1; ++i) {
    int pos = p[i - 1];
    while (pos > 0 && s[pos] != s[i]) pos = p[pos - 1];
    if (s[pos] == s[i]) pos++;
    p[i] = pos;
}

```

6.5 Suffix Array (Implementation 1)

```

const int N = 20007;
const int LOG = 16;
char st[N];
int n;
int cnt[N], p[N], tp[N], c[N], tc[N];
int lcp[N], mlog[N], mini[LOG][N];

void build() {
    FOR(i, n) ++cnt[st[i]];
    FOR(i, N) {
        if (i) cnt[i] += cnt[i - 1];
    }
    for (int i = n - 1; i >= 0; --i) {
        p[--cnt[st[i]]] = i;
    }
    c[p[0]] = 0;
    for (int i = 1, cur = 0; i < n; ++i) {
        if (st[p[i]] != st[p[i - 1]]) ++cur;
        c[p[i]] = cur;
    }
    for (int h = 0; (1 << h) < n; ++h) {
        FOR(i, n) {
            tp[i] = p[i] - (1 << h);
            if (tp[i] < 0) tp[i] += n;
        }

        memset(cnt, 0, sizeof cnt);
        FOR(i, n) ++cnt[c[i]];
        FOR(i, N) {
            if (i) cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            p[--cnt[c[tp[i]]]] = tp[i];
        }

        tc[p[0]] = 0;
        for (int i = 1, cur = 0; i < n; ++i) {
            int m1 = (p[i] + (1 << h)) % n;
            int m2 = (p[i - 1] + (1 << h)) % n;
            if (c[p[i]] != c[p[i - 1]] || c[m1] != c[m2]) ++cur;
            tc[p[i]] = cur;
        }
        FOR(i, n) c[i] = tc[i];
    }
}

void calc_lcps() {
    int *pos = new int[n];
    FOR(i, n) pos[p[i]] = i;
    int cur = 0;
    FOR(i, n) {
        int ind = pos[i];
        if (ind == n - 1) {
            cur = 0;
            continue;
        }
        cur = max(0, cur - 1);
        while (p[ind] + cur < n && p[ind + 1] + cur < n && st[p[ind]
            ] + cur == st[p[ind + 1] + cur]) {
            ++cur;
        }
        lcp[ind] = cur;
    }
}

void build_sparse_table() {
    mlog[1] = 0;
    for (int i = 2; i < N; ++i) mlog[i] = 1 + mlog[i >> 1];
    FOR(i, n - 1) mini[0][i] = lcp[i];
    FOR(h, LOG - 1) {
        FOR(i, n - 1) {
            if (i + (1 << (h + 1)) > n - 1) break;
            mini[h + 1][i] = min(mini[h][i], mini[h][i + (1 << h)]);
        }
    }
}

int get_min(int i, int j) {
    if (i == j) return N;
    --j;
    int k = mlog[j - i + 1];
    return min(mini[k][i], mini[k][j - (1 << k) + 1]);
}

```

6.6 Suffix Array (Implementation 2)

```
const int N = 100007;
```

```

const int A = 256;
char st[N];
int n;

struct SA {
    int C[20][N], p[N], tp[N], cnt[N];

    void build() {
        FOR(i, A) cnt[i] = 0;
        FOR(i, n) ++cnt[st[i]];
        FOR(i, A) {
            if (i) cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            p[--cnt[st[i]]] = i;
        }
        C[0][p[0]] = 0;
        for (int i = 1, cur = 0; i < n; ++i) {
            if (st[p[i]] != st[p[i - 1]]) ++cur;
            C[0][p[i]] = cur;
        }
        for (int h = 1, lev = 0; h < n; h <= 1, ++lev) {
            FOR(i, n) tp[i] = (p[i] - h + n) % n;
            FOR(i, n) cnt[i] = 0;
            FOR(i, n) ++cnt[C[lev][i]];
            FOR(i, n) {
                if (i) cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i >= 0; --i) {
                int ind = tp[i];
                p[--cnt[C[lev][ind]]] = ind;
            }

            C[lev + 1][p[0]] = 0;
            for (int i = 1, cur = 0; i < n; ++i) {
                if (C[lev][p[i - 1]] != C[lev][p[i]] || C[lev][(p[i - 1]
                    + h) % n] != C[lev][(p[i] + h) % n]) ++cur;
                C[lev + 1][p[i]] = cur;
            }
        }
    }
} sa;

int getlcp(int i, int j) {
    int ans = 0;
    for (int h = 19; h >= 0; --h) {
        if ((1 << h) > n) continue;
        if (sa.C[h][i] == sa.C[h][j]) {
            i += (1 << h);
            j += (1 << h);
            ans += (1 << h);
        }
    }
    return ans;
}

int compare(int i, int j, int len) {
    for (int h = 19; h >= 0; --h) {
        if ((1 << h) > len) continue;
        if (sa.C[h][i] < sa.C[h][j]) return -1;
        if (sa.C[h][i] > sa.C[h][j]) return 1;
        i = (i + (1 << h)) % n;
        j = (j + (1 << h)) % n;
        len -= (1 << h);
    }
    return 0;
}

```

6.7 Suffix Automaton

```

const int N = 100007;
const int A = 26;
struct node {
    int len, link;
    int nxt[A];
} V[2 * N];
bool terminal[2 * N];
int size, last;
char *st;

void prepare() {
    size = 1;
    last = 0;
    V[0].link = -1;
    V[0].len = 0;
    for (int i = 0; i < 2 * N; ++i) {

```

```

        fill(V[i].nxt, V[i].nxt + A, -1);
    }
    fill(terminal, terminal + 2 * N, false);
}

void addchar(int c) {
    int cur = size++;
    V[cur].len = V[last].len + 1;
    int now = last;
    for (; now != -1 && V[now].nxt[c] == -1; now = V[now].link) {
        V[now].nxt[c] = cur;
    }
    if (now == -1) {
        V[cur].link = 0;
    }
    else {
        int to = V[now].nxt[c];
        if (V[to].len == V[now].len + 1) {
            V[cur].link = to;
        }
        else {
            int clone = size++;
            V[clone].len = V[now].len + 1;
            for (int i = 0; i < A; ++i) {
                V[clone].nxt[i] = V[to].nxt[i];
            }
            V[clone].link = V[to].link;
            for (; now != -1 && V[now].nxt[c] == to; now = V[now].link) {
                V[now].nxt[c] = clone;
            }
            V[cur].link = clone;
            V[to].link = clone;
        }
    }

    last = cur;
}

void build(char *str) {
    st = str;
    int n = strlen(st);
    prepare();
    for (int i = 0; i < n; ++i) {
        addchar(st[i] - 'a');
    }
}

void markTerminals() {
    for (int cur = last; cur != -1; cur = V[cur].link) {
        terminal[cur] = true;
    }
}

```

6.8 Suffix Tree

```

const int N = 300007;
char A[N], B[N], C[N], S[N];
int na, nb, nc, n;

struct node {
    int l, r, par, suff;
    char pch;
    map<char, int> nxt;

    node(int l = 0, int r = 0, int par = 0, char pch = 0, int suff = -1) :
        l(l), r(r), par(par), pch(pch), suff(suff) {}
}

```

```

    }

    int len() const {
        return r - l;
    }
    T[2 + N];

    struct pos {
        int v, p;
        pos(int v = 0, int p = 0) : v(v), p(p) {}
    } ptr;
    int size = 1;

    pos go(const pos &ptr, char c) {
        if (T[ptr.v].len() == ptr.p) {
            int v = ptr.v;
            if (T[v].nxt.count(c)) {
                int to = T[v].nxt[c];
                return pos(to, 1);
            }
            return pos(-1, -1);
        }
        if (S[T[ptr.v].l + ptr.p] == c) return pos(ptr.v, ptr.p + 1);
        return pos(-1, -1);
    }

    int split(const pos &ptr) {
        if (T[ptr.v].len() == ptr.p) return ptr.v;
        if (ptr.p == 0) return T[ptr.v].par;

        int mid = size++;
        int v = ptr.v;
        int par = T[v].par;
        char midch = S[T[v].l + ptr.p];

        //par
        T[par].nxt[T[v].pch] = mid;

        //mid
        T[mid].par = par;
        T[mid].pch = T[v].pch;
        T[mid].l = T[v].l;
        T[mid].r = T[v].l + ptr.p;
        T[mid].nxt[midch] = v;

        // v
        T[v].par = mid;
        T[v].pch = midch;
        T[v].l = T[mid].r;

        return mid;
    }

    pos fastGo(pos ptr, int l, int r) {
        while (l != r) {
            if (T[ptr.v].len() == ptr.p) {
                ptr.v = T[ptr.v].nxt[S[l]];
                ++l;
                ptr.p = 1;
                continue;
            }
            int step = min(T[ptr.v].len() - ptr.p, r - l);
            ptr.p += step;
            l += step;
        }
        if (ptr.p == 0) {
            ptr.p = T[ptr.v].len();
            ptr.v = T[ptr.v].par;
        }
        return ptr;
    }
}

```

```

pos getSuff(int v) {
    if (T[v].suff != -1) return pos(T[v].suff, T[T[v].suff].len());
    ;
    pos ret;
    if (T[v].par == 0) ret = fastGo(0, T[v].l + 1, T[v].r);
    else ret = fastGo(getSuff(T[v].par), T[v].l, T[v].r);
    assert(ret.p == T[ret.v].len());
    T[v].suff = ret.v;
    return ret;
}

void extend(int i) {
    char c = S[i];
    while (true) {
        pos goPos = go(ptr, c);
        if (goPos.v == -1) {
            int mid = split(ptr);
            int leaf = size++;
            T[mid].nxt[c] = leaf;
            T[leaf] = node(i, n, mid, c);
            if (ptr.v == 0) {
                ptr = pos(0, 0);
                break;
            }
            int par = T[mid].par;
            if (par == 0) ptr = fastGo(pos(0, 0), T[mid].l + 1, T[mid].r);
            else ptr = fastGo(getSuff(par), T[mid].l, T[mid].r);
        }
        else {
            ptr = goPos;
            break;
        }
    }
}

void print(int u) {
    for (auto p : T[u].nxt) {
        int to = p.second;
        cerr << "edge from " << u << " " << " to " << to << " by " << S[u].l << " " << S[to].l << endl;
        print(to);
    }
}

// build
T[0].suff = 0;
for (int i = 0; i < n; ++i) extend(i);
}

```

6.9 Z Function

```

n = (int)s.size();
z.resize(n, 0);
z[0] = 0;
for (i = 1; i < n; ++i) {
    if (r >= i) {
        z[i] = min(z[i - 1], r - i + 1);
    }
    while (z[i] + i < n && s[z[i] + i] == s[i + z[i]]) z[i]++;
    if (i + z[i] - 1 > r) {
        l = i;
        r = i + z[i] - 1;
    }
}
}

```

Theoretical Computer Science Cheat Sheet

Definitions		Series
$f(n) = O(g(n))$	iff \exists positive c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$.	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$
$f(n) = \Omega(g(n))$	iff \exists positive c, n_0 such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$.	In general:
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.	$\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.	$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a < \epsilon, \forall n \geq n_0$.	Geometric series:
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$.	$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad c < 1,$
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$.	$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad c < 1.$
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	Harmonic series:
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n iH_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}.$
$\binom{n}{k}$	Combinations: Size k sub-sets of a size n set.	$\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right).$
$[n]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\} = 1,$
14. $\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!,$	15. $\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)!H_{n-1},$	12. $\left\{ \begin{smallmatrix} n \\ 2 \end{smallmatrix} \right\} = 2^{n-1} - 1, \quad 13. \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right\},$
16. $\begin{bmatrix} n \\ n \end{bmatrix} = 1,$	17. $\begin{bmatrix} n \\ k \end{bmatrix} \geq \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\},$	
18. $\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix},$	19. $\left\{ \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right\} = \begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2},$	20. $\sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
22. $\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \rangle = \langle \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \rangle = 1,$	23. $\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle = \langle \begin{smallmatrix} n \\ n-1-k \end{smallmatrix} \rangle,$	24. $\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle = (k+1) \langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \rangle + (n-k) \langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \rangle,$
25. $\langle \begin{smallmatrix} 0 \\ k \end{smallmatrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\langle \begin{smallmatrix} n \\ 1 \end{smallmatrix} \rangle = 2^n - n - 1,$	27. $\langle \begin{smallmatrix} n \\ 2 \end{smallmatrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
28. $x^n = \sum_{k=0}^n \langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle \binom{x+k}{n},$	29. $\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	30. $m! \left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle \binom{k}{n-m},$
31. $\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \rangle = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\langle\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \rangle\rangle = 1,$	33. $\langle\langle \begin{smallmatrix} n \\ n \end{smallmatrix} \rangle\rangle = 0 \text{ for } n \neq 0,$
34. $\langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle = (k+1) \langle\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \rangle\rangle + (2n-1-k) \langle\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \rangle\rangle,$	35. $\sum_{k=0}^n \langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle = \frac{(2n)^n}{2^n},$	
36. $\left\{ \begin{smallmatrix} x \\ x-n \end{smallmatrix} \right\} = \sum_{k=0}^n \langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle \binom{x+n-1-k}{2n},$	37. $\left\{ \begin{smallmatrix} n+1 \\ m+1 \end{smallmatrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} (m+1)^{n-k},$	

Theoretical Computer Science Cheat Sheet

Identities Cont.

$$\begin{aligned}
 38. \quad \left[\begin{matrix} n+1 \\ m+1 \end{matrix} \right] &= \sum_k \left[\begin{matrix} n \\ k \end{matrix} \right] \binom{k}{m} = \sum_{k=0}^n \left[\begin{matrix} k \\ m \end{matrix} \right] n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \left[\begin{matrix} k \\ m \end{matrix} \right], & 39. \quad \left[\begin{matrix} x \\ x-n \end{matrix} \right] &= \sum_{k=0}^n \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \binom{x+k}{2n}, \\
 40. \quad \left\{ \begin{matrix} n \\ m \end{matrix} \right\} &= \sum_k \binom{n}{k} \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} (-1)^{n-k}, & 41. \quad \left[\begin{matrix} n \\ m \end{matrix} \right] &= \sum_k \left[\begin{matrix} n+1 \\ k+1 \end{matrix} \right] \binom{k}{m} (-1)^{m-k}, \\
 42. \quad \left\{ \begin{matrix} m+n+1 \\ m \end{matrix} \right\} &= \sum_{k=0}^m k \left\{ \begin{matrix} n+k \\ k \end{matrix} \right\}, & 43. \quad \left[\begin{matrix} m+n+1 \\ m \end{matrix} \right] &= \sum_{k=0}^m k(n+k) \left[\begin{matrix} n+k \\ k \end{matrix} \right], \\
 44. \quad \binom{n}{m} &= \sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \left[\begin{matrix} k \\ m \end{matrix} \right] (-1)^{m-k}, & 45. \quad (n-m)! \binom{n}{m} &= \sum_k \left[\begin{matrix} n+1 \\ k+1 \end{matrix} \right] \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (-1)^{m-k}, \quad \text{for } n \geq m, \\
 46. \quad \left\{ \begin{matrix} n \\ n-m \end{matrix} \right\} &= \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left[\begin{matrix} m+k \\ k \end{matrix} \right], & 47. \quad \left[\begin{matrix} n \\ n-m \end{matrix} \right] &= \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\}, \\
 48. \quad \left\{ \begin{matrix} n \\ \ell+m \end{matrix} \right\} \binom{\ell+m}{\ell} &= \sum_k \left\{ \begin{matrix} k \\ \ell \end{matrix} \right\} \left\{ \begin{matrix} n-k \\ m \end{matrix} \right\} \binom{n}{k}, & 49. \quad \left[\begin{matrix} n \\ \ell+m \end{matrix} \right] \binom{\ell+m}{\ell} &= \sum_k \left[\begin{matrix} k \\ \ell \end{matrix} \right] \left[\begin{matrix} n-k \\ m \end{matrix} \right] \binom{n}{k}.
 \end{aligned}$$

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n :

$$\sum_{i=1}^n 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n , then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two.

Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.

Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side “telescope”

$$\begin{aligned}
 1(T(n) - 3T(n/2)) &= n \\
 3(T(n/2) - 3T(n/4)) &= n/2 \\
 &\vdots \\
 3^{\log_2 n - 1}(T(2) - 3T(1)) &= 2
 \end{aligned}$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$\begin{aligned}
 n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\
 &= 2n(c^{\log_2 n} - 1) \\
 &= 2n(c^{(k-1)\log_2 n} - 1) \\
 &= 2n^k - 2n,
 \end{aligned}$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^i T_j.$$

Subtracting we find

$$\begin{aligned}
 T_{i+1} - T_i &= 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j \\
 &= T_i.
 \end{aligned}$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i .
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i .

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

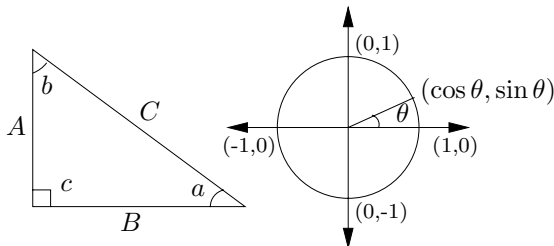
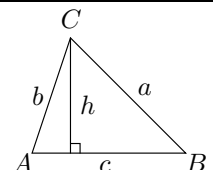
Expand this using partial fractions:

$$\begin{aligned}
 G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\
 &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\
 &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}.
 \end{aligned}$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet					
$\pi \approx 3.14159,$		$e \approx 2.71828,$	$\gamma \approx 0.57721,$	$\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803,$	$\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$
i	2^i	p_i	General	Probability	
1	2	2	Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):	Continuous distributions: If	
2	4	3	$B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$	$\Pr[a < X < b] = \int_a^b p(x) dx,$	
3	8	5	$B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	then p is the probability density function of X . If	
4	16	7	Change of base, quadratic formula:	$\Pr[X < a] = P(a),$	
5	32	11	$\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	then P is the distribution function of X . If P and p both exist then	
6	64	13	Euler's number e :	$P(a) = \int_{-\infty}^a p(x) dx.$	
7	128	17	$e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \cdots$	Expectation: If X is discrete	
8	256	19	$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	$E[g(X)] = \sum_x g(x) \Pr[X = x].$	
9	512	23	$\left(1 + \frac{1}{n}\right)^n < e < \left(1 + \frac{1}{n}\right)^{n+1}.$	If X continuous then	
10	1,024	29	$\left(1 + \frac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$	$E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$	
11	2,048	31	Harmonic numbers:	Variance, standard deviation:	
12	4,096	37	$1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$	$\text{VAR}[X] = E[X^2] - E[X]^2,$	
13	8,192	41	$\ln n < H_n < \ln n + 1,$	$\sigma = \sqrt{\text{VAR}[X]}.$	
14	16,384	43	$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$	For events A and B :	
15	32,768	47	Factorial, Stirling's approximation:	$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$	
16	65,536	53	$1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$	
17	131,072	59	$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$	iff A and B are independent.	
18	262,144	61	Ackermann's function and inverse:	$\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$	
19	524,288	67	$a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$	For random variables X and Y :	
20	1,048,576	71	$\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	$E[X \cdot Y] = E[X] \cdot E[Y],$	
21	2,097,152	73		if X and Y are independent.	
22	4,194,304	79		$E[X + Y] = E[X] + E[Y],$	
23	8,388,608	83		$E[cX] = c E[X].$	
24	16,777,216	89	Binomial distribution:	Bayes' theorem:	
25	33,554,432	97	$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$	$\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$	
26	67,108,864	101	$E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	Inclusion-exclusion:	
27	134,217,728	103	Poisson distribution:	$\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] +$	
28	268,435,456	107	$\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$	$\sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$	
29	536,870,912	109	Normal (Gaussian) distribution:	Moment inequalities:	
30	1,073,741,824	113	$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$	$\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$	
31	2,147,483,648	127	The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is	$\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$	
32	4,294,967,296	131	$nH_n.$	Geometric distribution:	
Pascal's Triangle				$\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$	
1				$E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$	
1 1					
1 2 1					
1 3 3 1					
1 4 6 4 1					
1 5 10 10 5 1					
1 6 15 20 15 6 1					
1 7 21 35 35 21 7 1					
1 8 28 56 70 56 28 8 1					
1 9 36 84 126 126 84 36 9 1					
1 10 45 120 210 252 210 120 45 10 1					

Theoretical Computer Science Cheat Sheet

Trigonometry	Matrices	More Trig.																								
<div></div> <p>Pythagorean theorem: $C^2 = A^2 + B^2.$</p> <p>Definitions:</p> $\sin a = A/C, \quad \cos a = B/C,$ $\csc a = C/A, \quad \sec a = C/B,$ $\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$ <p>Area, radius of inscribed circle:</p> $\frac{1}{2}AB, \quad \frac{AB}{A+B+C}.$ <p>Identities:</p> $\sin x = \frac{1}{\csc x}, \quad \cos x = \frac{1}{\sec x},$ $\tan x = \frac{1}{\cot x}, \quad \sin^2 x + \cos^2 x = 1,$ $1 + \tan^2 x = \sec^2 x, \quad 1 + \cot^2 x = \csc^2 x,$ $\sin x = \cos\left(\frac{\pi}{2} - x\right), \quad \sin x = \sin(\pi - x),$ $\cos x = -\cos(\pi - x), \quad \tan x = \cot\left(\frac{\pi}{2} - x\right),$ $\cot x = -\cot(\pi - x), \quad \csc x = \cot \frac{x}{2} - \cot x,$ $\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$ $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$ $\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$ $\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$ $\sin 2x = 2 \sin x \cos x, \quad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$ $\cos 2x = \cos^2 x - \sin^2 x, \quad \cos 2x = 2 \cos^2 x - 1,$ $\cos 2x = 1 - 2 \sin^2 x, \quad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$ $\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \quad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$ $\sin(x + y) \sin(x - y) = \sin^2 x - \sin^2 y,$ $\cos(x + y) \cos(x - y) = \cos^2 x - \sin^2 y.$ <p>Euler's equation:</p> $e^{ix} = \cos x + i \sin x, \quad e^{i\pi} = -1.$	<p>Multiplication:</p> $C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}.$ <p>Determinants: $\det A \neq 0$ iff A is non-singular.</p> $\det A \cdot B = \det A \cdot \det B,$ $\det A = \sum_{\pi} \prod_{i=1}^n \text{sign}(\pi) a_{i,\pi(i)}.$ <p>2×2 and 3×3 determinant:</p> $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$ $\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$ $= aei + bfg + cdh - ceg - fha - ibd.$ <p>Permanents:</p> $\text{perm } A = \sum_{\pi} \prod_{i=1}^n a_{i,\pi(i)}.$	<div></div> <p>Law of cosines: $c^2 = a^2 + b^2 - 2ab \cos C.$</p> <p>Area:</p> $A = \frac{1}{2}hc,$ $= \frac{1}{2}ab \sin C,$ $= \frac{c^2 \sin A \sin B}{2 \sin C}.$ <p>Heron's formula:</p> $A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$ $s = \frac{1}{2}(a + b + c),$ $s_a = s - a,$ $s_b = s - b,$ $s_c = s - c.$ <p>More identities:</p> $\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$ $\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$ $\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$ $= \frac{1 - \cos x}{\sin x},$ $= \frac{\sin x}{1 + \cos x},$ $\cot \frac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$ $= \frac{1 + \cos x}{\sin x},$ $= \frac{\sin x}{1 - \cos x},$ $\sin x = \frac{e^{ix} - e^{-ix}}{2i},$ $\cos x = \frac{e^{ix} + e^{-ix}}{2},$ $\tan x = -i \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$ $= -i \frac{e^{2ix} - 1}{e^{2ix} + 1},$ $\sin x = \frac{\sinh ix}{i},$ $\cos x = \cosh ix,$ $\tan x = \frac{\tanh ix}{i}.$																								
	<p>Hyperbolic Functions</p> <p>Definitions:</p> $\sinh x = \frac{e^x - e^{-x}}{2}, \quad \cosh x = \frac{e^x + e^{-x}}{2},$ $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{csch } x = \frac{1}{\sinh x},$ $\text{sech } x = \frac{1}{\cosh x}, \quad \coth x = \frac{1}{\tanh x}.$ <p>Identities:</p> $\cosh^2 x - \sinh^2 x = 1, \quad \tanh^2 x + \text{sech}^2 x = 1,$ $\coth^2 x - \text{csch}^2 x = 1, \quad \sinh(-x) = -\sinh x,$ $\cosh(-x) = \cosh x, \quad \tanh(-x) = -\tanh x,$ $\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$ $\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$ $\sinh 2x = 2 \sinh x \cosh x,$ $\cosh 2x = \cosh^2 x + \sinh^2 x,$ $\cosh x + \sinh x = e^x, \quad \cosh x - \sinh x = e^{-x},$ $(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$ $2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$																									
	<table><tr><th>θ</th><th>$\sin \theta$</th><th>$\cos \theta$</th><th>$\tan \theta$</th></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>$\frac{\pi}{6}$</td><td>$\frac{1}{2}$</td><td>$\frac{\sqrt{3}}{2}$</td><td>$\frac{\sqrt{3}}{3}$</td></tr><tr><td>$\frac{\pi}{4}$</td><td>$\frac{\sqrt{2}}{2}$</td><td>$\frac{\sqrt{2}}{2}$</td><td>1</td></tr><tr><td>$\frac{\pi}{3}$</td><td>$\frac{\sqrt{3}}{2}$</td><td>$\frac{1}{2}$</td><td>$\sqrt{3}$</td></tr><tr><td>$\frac{\pi}{2}$</td><td>1</td><td>0</td><td>∞</td></tr></table>	θ	$\sin \theta$	$\cos \theta$	$\tan \theta$	0	0	1	0	$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$	$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1	$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$	$\frac{\pi}{2}$	1	0	∞	<p>... in mathematics you don't understand things, you just get used to them.</p> <p>- J. von Neumann</p>
θ	$\sin \theta$	$\cos \theta$	$\tan \theta$																							
0	0	1	0																							
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$																							
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1																							
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$																							
$\frac{\pi}{2}$	1	0	∞																							
<p>v2.02 ©1994 by Steve Seiden sseiden@acm.org http://www.csc.lsu.edu/~seiden</p>																										

Theoretical Computer Science Cheat Sheet

Number Theory

The Chinese remainder theorem: There exists a number C such that:

$$C \equiv r_1 \pmod{m_1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$C \equiv r_n \pmod{m_n}$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x . If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \pmod{b}.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \pmod{p}.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \pmod{n}.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n}$$

$$+ O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3}$$

$$+ O\left(\frac{n}{(\ln n)^4}\right).$$

Graph Theory

Definitions:

Loop An edge connecting a vertex to itself.

Directed Each edge has a direction.

Simple Graph with no loops or multi-edges.

Walk A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$.

Trail A walk with distinct edges.

Path A trail with distinct vertices.

Connected A graph where there exists a path between any two vertices.

Component A maximal connected subgraph.

Tree A connected acyclic graph.

Free tree A tree with no root.

DAG Directed acyclic graph.

Eulerian Graph with a trail visiting each edge exactly once.

Hamiltonian Graph with a cycle visiting each vertex exactly once.

Cut A set of edges whose removal increases the number of components.

Cut-set A minimal cut.

Cut edge A size 1 cut.

k-Connected A graph connected with the removal of any $k-1$ vertices.

k-Tough $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq |S|$.

k-Regular A graph where all vertices have degree k .

k-Factor A k -regular spanning subgraph.

Matching A set of edges, no two of which are adjacent.

Clique A set of vertices, all of which are adjacent.

Ind. set A set of vertices, none of which are adjacent.

Vertex cover A set of vertices which cover all edges.

Planar graph A graph which can be embedded in the plane.

Plane graph An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5 .

Notation:

$E(G)$ Edge set

$V(G)$ Vertex set

$c(G)$ Number of components

$G[S]$ Induced subgraph

$\deg(v)$ Degree of v

$\Delta(G)$ Maximum degree

$\delta(G)$ Minimum degree

$\chi(G)$ Chromatic number

$\chi_E(G)$ Edge chromatic number

G^c Complement graph

K_n Complete graph

K_{n_1, n_2} Complete bipartite graph

$r(k, \ell)$ Ramsey number

Geometry

Projective coordinates: triples (x, y, z) , not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian Projective

$$(x, y) \quad (x, y, 1)$$

$$y = mx + b \quad (m, -1, b)$$

$$x = c \quad (1, 0, -c)$$

Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

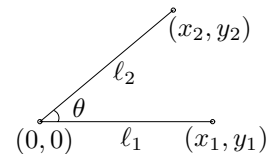
$$[|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p},$$

$$\lim_{p \rightarrow \infty} [|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p}.$$

Area of triangle (x_0, y_0) , (x_1, y_1) and (x_2, y_2) :

$$\frac{1}{2} \text{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1) :

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.

– Issac Newton

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \cdots}}}}$$

Gregory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x . We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D , divide N by D , obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D . Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)} \right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!} \left[\frac{d^k}{dx^k} \left(\frac{N(x)}{D(x)} \right) \right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Calculus

Derivatives:

$$1. \frac{d(cu)}{dx} = c \frac{du}{dx}, \quad 2. \frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}, \quad 3. \frac{d(uv)}{dx} = u \frac{dv}{dx} + v \frac{du}{dx},$$

$$4. \frac{d(u^n)}{dx} = nu^{n-1} \frac{du}{dx}, \quad 5. \frac{d(u/v)}{dx} = \frac{v \left(\frac{du}{dx} \right) - u \left(\frac{dv}{dx} \right)}{v^2}, \quad 6. \frac{d(e^{cu})}{dx} = ce^{cu} \frac{du}{dx},$$

$$7. \frac{d(c^u)}{dx} = (\ln c) c^u \frac{du}{dx}, \quad 8. \frac{d(\ln u)}{dx} = \frac{1}{u} \frac{du}{dx},$$

$$9. \frac{d(\sin u)}{dx} = \cos u \frac{du}{dx}, \quad 10. \frac{d(\cos u)}{dx} = -\sin u \frac{du}{dx},$$

$$11. \frac{d(\tan u)}{dx} = \sec^2 u \frac{du}{dx}, \quad 12. \frac{d(\cot u)}{dx} = \csc^2 u \frac{du}{dx},$$

$$13. \frac{d(\sec u)}{dx} = \tan u \sec u \frac{du}{dx}, \quad 14. \frac{d(\csc u)}{dx} = -\cot u \csc u \frac{du}{dx},$$

$$15. \frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}} \frac{du}{dx}, \quad 16. \frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}} \frac{du}{dx},$$

$$17. \frac{d(\arctan u)}{dx} = \frac{1}{1+u^2} \frac{du}{dx}, \quad 18. \frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2} \frac{du}{dx},$$

$$19. \frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}} \frac{du}{dx}, \quad 20. \frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx},$$

$$21. \frac{d(\sinh u)}{dx} = \cosh u \frac{du}{dx}, \quad 22. \frac{d(\cosh u)}{dx} = \sinh u \frac{du}{dx},$$

$$23. \frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u \frac{du}{dx}, \quad 24. \frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u \frac{du}{dx},$$

$$25. \frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u \frac{du}{dx}, \quad 26. \frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u \frac{du}{dx},$$

$$27. \frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}} \frac{du}{dx}, \quad 28. \frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}} \frac{du}{dx},$$

$$29. \frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2} \frac{du}{dx}, \quad 30. \frac{d(\operatorname{arcoth} u)}{dx} = \frac{1}{u^2-1} \frac{du}{dx},$$

$$31. \frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx}, \quad 32. \frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{|u|\sqrt{1+u^2}} \frac{du}{dx}.$$

Integrals:

$$1. \int cu \, dx = c \int u \, dx, \quad 2. \int (u+v) \, dx = \int u \, dx + \int v \, dx,$$

$$3. \int x^n \, dx = \frac{1}{n+1} x^{n+1}, \quad n \neq -1, \quad 4. \int \frac{1}{x} \, dx = \ln x, \quad 5. \int e^x \, dx = e^x,$$

$$6. \int \frac{dx}{1+x^2} = \arctan x, \quad 7. \int u \frac{dv}{dx} \, dx = uv - \int v \frac{du}{dx} \, dx,$$

$$8. \int \sin x \, dx = -\cos x, \quad 9. \int \cos x \, dx = \sin x,$$

$$10. \int \tan x \, dx = -\ln |\cos x|, \quad 11. \int \cot x \, dx = \ln |\cos x|,$$

$$12. \int \sec x \, dx = \ln |\sec x + \tan x|, \quad 13. \int \csc x \, dx = \ln |\csc x + \cot x|,$$

$$14. \int \arcsin \frac{x}{a} \, dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$
16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$
17. $\int \sin^2(ax) dx = \frac{1}{2a} (ax - \sin(ax) \cos(ax)),$
18. $\int \cos^2(ax) dx = \frac{1}{2a} (ax + \sin(ax) \cos(ax)),$
19. $\int \sec^2 x dx = \tan x,$
20. $\int \csc^2 x dx = -\cot x,$
21. $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
22. $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
23. $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
24. $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
25. $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
26. $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
27. $\int \sinh x dx = \cosh x,$
28. $\int \cosh x dx = \sinh x,$
29. $\int \tanh x dx = \ln |\cosh x|,$
30. $\int \coth x dx = \ln |\sinh x|,$
31. $\int \operatorname{sech} x dx = \arctan \sinh x,$
32. $\int \operatorname{csch} x dx = \ln \left| \tanh \frac{x}{2} \right|,$
33. $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2} x,$
34. $\int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2} x,$
35. $\int \operatorname{sech}^2 x dx = \tanh x,$
36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
38. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left(x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
41. $\int \sqrt{a^2 - x^2} dx = \frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8} (5a^2 - 2x^2) \sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|,$
45. $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
46. $\int \sqrt{a^2 \pm x^2} dx = \frac{x}{2} \sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left| x + \sqrt{a^2 \pm x^2} \right|,$
47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
49. $\int x \sqrt{a+bx} dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$
50. $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
51. $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
53. $\int x \sqrt{a^2 - x^2} dx = -\frac{1}{3} (a^2 - x^2)^{3/2},$
54. $\int x^2 \sqrt{a^2 - x^2} dx = \frac{x}{8} (2x^2 - a^2) \sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
56. $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
57. $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
60. $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
61. $\int \frac{dx}{x \sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

$$\begin{aligned}
 \text{62. } \int \frac{dx}{x\sqrt{x^2 - a^2}} &= \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0, & \text{63. } \int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} &= \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x}, \\
 \text{64. } \int \frac{x dx}{\sqrt{x^2 \pm a^2}} &= \sqrt{x^2 \pm a^2}, & \text{65. } \int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx &= \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3}, \\
 \text{66. } \int \frac{dx}{ax^2 + bx + c} &= \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases} \\
 \text{67. } \int \frac{dx}{\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases} \\
 \text{68. } \int \sqrt{ax^2 + bx + c} dx &= \frac{2ax + b}{4a} \sqrt{ax^2 + bx + c} + \frac{4ac - b^2}{8a} \int \frac{dx}{\sqrt{ax^2 + bx + c}}, \\
 \text{69. } \int \frac{x dx}{\sqrt{ax^2 + bx + c}} &= \frac{\sqrt{ax^2 + bx + c}}{a} - \frac{b}{2a} \int \frac{dx}{\sqrt{ax^2 + bx + c}}, \\
 \text{70. } \int \frac{dx}{x\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases} \\
 \text{71. } \int x^3 \sqrt{x^2 + a^2} dx &= \left(\frac{1}{3}x^2 - \frac{2}{15}a^2\right)(x^2 + a^2)^{3/2}, \\
 \text{72. } \int x^n \sin(ax) dx &= -\frac{1}{a}x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx, \\
 \text{73. } \int x^n \cos(ax) dx &= \frac{1}{a}x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx, \\
 \text{74. } \int x^n e^{ax} dx &= \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx, \\
 \text{75. } \int x^n \ln(ax) dx &= x^{n+1} \left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2} \right), \\
 \text{76. } \int x^n (\ln ax)^m dx &= \frac{x^{n+1}}{n+1} (\ln ax)^m - \frac{m}{n+1} \int x^n (\ln ax)^{m-1} dx.
 \end{aligned}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$

$$\mathbf{E} f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x) \delta x = F(x) + C.$$

$$\sum_a^b f(x) \delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$$\Delta(cu) = c\Delta u, \quad \Delta(u+v) = \Delta u + \Delta v,$$

$$\Delta(uv) = u\Delta v + \mathbf{E} v \Delta u,$$

$$\Delta(x^n) = nx^{n-1},$$

$$\Delta(H_x) = x^{-1},$$

$$\Delta(2^x) = 2^x,$$

$$\Delta(c^x) = (c-1)c^x,$$

$$\Delta \binom{x}{m} = \binom{x}{m-1}.$$

Sums:

$$\sum cu \delta x = c \sum u \delta x,$$

$$\sum (u+v) \delta x = \sum u \delta x + \sum v \delta x,$$

$$\sum u \Delta v \delta x = uv - \sum \mathbf{E} v \Delta u \delta x,$$

$$\sum x^n \delta x = \frac{x^{n+1}}{n+1}, \quad \sum x^{-1} \delta x = H_x,$$

$$\sum c^x \delta x = \frac{c^x}{c-1}, \quad \sum \binom{x}{m} \delta x = \binom{x}{m+1}.$$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1) \cdots (x-n+1), \quad n > 0,$$

$$x^{\underline{0}} = 1,$$

$$x^{\underline{n}} = \frac{1}{(x+1) \cdots (x+|n|)}, \quad n < 0,$$

$$x^{\underline{n+m}} = x^{\underline{n}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1) \cdots (x+n-1), \quad n > 0,$$

$$x^{\overline{0}} = 1,$$

$$x^{\overline{n}} = \frac{1}{(x-1) \cdots (x-|n|)}, \quad n < 0,$$

$$x^{\overline{n+m}} = x^{\overline{n}}(x+m)^{\overline{n}}.$$

Conversion:

$$x^{\underline{n}} = (-1)^n (-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$$

$$= 1/(x+1)^{\overline{-n}},$$

$$x^{\overline{n}} = (-1)^n (-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$$

$$= 1/(x-1)^{\underline{-n}},$$

$$x^n = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] x^k.$$

$x^1 =$	$x^{\underline{1}}$	$=$	$x^{\overline{1}}$
$x^2 =$	$x^{\underline{2}} + x^{\underline{1}}$	$=$	$x^{\overline{2}} - x^{\overline{1}}$
$x^3 =$	$x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}}$	$=$	$x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}}$
$x^4 =$	$x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}}$	$=$	$x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}}$
$x^5 =$	$x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}}$	$=$	$x^{\overline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}$
$x^{\overline{1}} =$	x^1	$x^{\underline{1}} =$	x^1
$x^{\overline{2}} =$	$x^2 + x^1$	$x^{\underline{2}} =$	$x^2 - x^1$
$x^{\overline{3}} =$	$x^3 + 3x^2 + 2x^1$	$x^{\underline{3}} =$	$x^3 - 3x^2 + 2x^1$
$x^{\overline{4}} =$	$x^4 + 6x^3 + 11x^2 + 6x^1$	$x^{\underline{4}} =$	$x^4 - 6x^3 + 11x^2 - 6x^1$
$x^{\overline{5}} =$	$x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1$	$x^{\underline{5}} =$	$x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \dots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned} \frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \dots = \sum_{i=0}^{\infty} x^i, \\ \frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \dots = \sum_{i=0}^{\infty} c^i x^i, \\ \frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \dots = \sum_{i=0}^{\infty} x^{ni}, \\ \frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \dots = \sum_{i=0}^{\infty} ix^i, \\ x^k \frac{d^n}{dx^n} \left(\frac{1}{1-x} \right) &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \dots = \sum_{i=0}^{\infty} i^n x^i, \\ e^x &= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\ \ln(1+x) &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\ \ln \frac{1}{1-x} &= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots = \sum_{i=1}^{\infty} \frac{x^i}{i}, \\ \sin x &= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\ \cos x &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\ \tan^{-1} x &= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\ (1+x)^n &= 1 + nx + \frac{n(n-1)}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\ \frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \binom{n+2}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\ \frac{x}{e^x - 1} &= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + \dots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\ \frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \dots = \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i, \\ \frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \dots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i, \\ \frac{1}{\sqrt{1-4x}} \left(\frac{1 - \sqrt{1-4x}}{2x} \right)^n &= 1 + (2+n)x + \binom{4+n}{2}x^2 + \dots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\ \frac{1}{1-x} \ln \frac{1}{1-x} &= x + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \dots = \sum_{i=1}^{\infty} H_i x^i, \\ \frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2 &= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{11}{24}x^4 + \dots = \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i}, \\ \frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \dots = \sum_{i=0}^{\infty} F_i x^i, \\ \frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} &= F_n x + F_{2n} x^2 + F_{3n} x^3 + \dots = \sum_{i=0}^{\infty} F_{ni} x^i. \end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$x A'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x) dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^i a_j$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^i a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

8. USEFUL INFORMATION		· sufficient: QI and $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$	
9. Misc			– Permutations <ul style="list-style-type: none">* Consider the cycles of the permutation
9.1. Debugging Tips.		<ul style="list-style-type: none">• Greedy• Randomized• Optimizations<ul style="list-style-type: none">– Use bitset (/64)– Switch order of loops (cache locality)• Process queries offline<ul style="list-style-type: none">– Mo’s algorithm• Square-root decomposition• Precomputation• Efficient simulation<ul style="list-style-type: none">– Mo’s algorithm– Sqrt decomposition– Store 2^k jump pointers• Data structure techniques<ul style="list-style-type: none">– Sqrt buckets– Store 2^k jump pointers– 2^k merging trick• Counting<ul style="list-style-type: none">– Inclusion-exclusion principle– Generating functions• Graphs<ul style="list-style-type: none">– Can we model the problem as a graph?– Can we use any properties of the graph?– Strongly connected components– Cycles (or odd cycles)– Bipartite (no odd cycles)<ul style="list-style-type: none">* Bipartite matching* Hall’s marriage theorem* Stable Marriage– Cut vertex/bridge– Biconnected components– Degrees of vertices (odd/even)– Trees<ul style="list-style-type: none">* Heavy-light decomposition* Centroid decomposition* Least common ancestor* Centers of the tree– Eulerian path/circuit– Chinese postman problem– Topological sort– (Min-Cost) Max Flow– Min Cut<ul style="list-style-type: none">* Maximum Density Subgraph– Huffman Coding– Min-Cost Arborescence– Steiner Tree– Kirchoff’s matrix tree theorem– Prüfer sequences– Lovász Toggle– Look at the DFS tree (which has no cross-edges)– Is the graph a DFA or NFA?<ul style="list-style-type: none">* Is it the Synchronizing word problem?• Mathematics<ul style="list-style-type: none">– Is the function multiplicative?– Look for a pattern	<ul style="list-style-type: none">– Functions<ul style="list-style-type: none">* Sum of piecewise-linear functions is a piecewise-linear function* Sum of convex (concave) functions is convex (concave)– Modular arithmetic<ul style="list-style-type: none">* Chinese Remainder Theorem* Linear Congruence– Sieve– System of linear equations– Values too big to represent?<ul style="list-style-type: none">* Compute using the logarithm* Divide everything by some large value– Linear programming<ul style="list-style-type: none">* Is the dual problem easier to solve?– Can the problem be modeled as a different combinatorial problem? Does that simplify calculations? <ul style="list-style-type: none">• Logic<ul style="list-style-type: none">– 2-SAT– XOR-SAT (Gauss elimination or Bipartite matching)• Meet in the middle• Only work with the smaller half ($\log(n)$)• Strings<ul style="list-style-type: none">– Trie (maybe over something weird, like bits)– Suffix array– Suffix automaton (+DP?)– Aho-Corasick– eerTree– Work with $S + S$• Hashing• Euler tour, tree to array• Segment trees<ul style="list-style-type: none">– Lazy propagation– Persistent– Implicit– Segment tree of X• Geometry<ul style="list-style-type: none">– Minkowski sum (of convex sets)– Rotating calipers– Sweep line (horizontally or vertically?)– Sweep angle– Convex hull• Fix a parameter (possibly the answer).• Are there few distinct values?• Binary search• Sliding Window (+ Monotonic Queue)• Computing a Convolution? Fast Fourier Transform• Computing a 2D Convolution? FFT on each row, and then on each column• Exact Cover (+ Algorithm X)• Cycle-Finding• What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?• Look at the complement problem
9.2. Solution Ideas.		<ul style="list-style-type: none">• Dynamic Programming<ul style="list-style-type: none">– Parsing CFGs: CYK Algorithm– Drop a parameter, recover from others– Swap answer and a parameter– When grouping: try splitting in two– 2^k trick– When optimizing<ul style="list-style-type: none">* Convex hull optimization<ul style="list-style-type: none">· $dp[i] = \min_{j < i} \{dp[j] + b[j] \times a[i]\}$· $b[j] \geq b[j + 1]$· optionally $a[i] \leq a[i + 1]$· $O(n^2)$ to $O(n)$* Divide and conquer optimization<ul style="list-style-type: none">· $dp[i][j] = \min_{k < j} \{dp[i - 1][k] + C[k][j]\}$· $A[i][j] \leq A[i][j + 1]$· $O(kn^2)$ to $O(kn \log n)$· sufficient: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$ (QI)* Knuth optimization<ul style="list-style-type: none">· $dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j] + C[i][j]\}$· $A[i][j - 1] \leq A[i][j] \leq A[i + 1][j]$· $O(n^3)$ to $O(n^2)$	

- Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/ Bucket sort

10. FORMULAS

- **Legendre symbol:** $(\frac{a}{b}) = a^{(b-1)/2} \pmod{b}$, b odd prime.
- **Heron’s formula:** A triangle with side lengths a, b, c has area $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{a+b+c}{2}$.
- **Pick’s theorem:** A polygon on an integer grid strictly containing i lattice points and having b lattice points on the boundary has area $i + \frac{b}{2} - 1$. (Nothing similar in higher dimensions)
- **Euler’s totient:** The number of integers less than n that are coprime to n are $n \prod_{p|n} \left(1 - \frac{1}{p}\right)$ where each p is a distinct prime factor of n .
- **König’s theorem:** In any bipartite graph $G = (L \cup R, E)$, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover. Let U be the set of unmatched vertices in L , and Z be the set of vertices that are either in U or are connected to U by an alternating path. Then $K = (L \setminus Z) \cup (R \cap Z)$ is the minimum vertex cover.
- A minumum Steiner tree for n vertices requires at most $n-2$ additional Steiner vertices.
- The number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.
- **Lagrange polynomial** through points $(x_0, y_0), \dots, (x_k, y_k)$ is $L(x) = \sum_{j=0}^k y_j \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x-x_m}{x_j-x_m}$
- **Hook length formula:** If λ is a Young diagram and $h_\lambda(i, j)$ is the hook-length of cell (i, j) , then then the number of Young tableaux $d_\lambda = n! / \prod h_\lambda(i, j)$.
- **Möbius inversion formula:** If $f(n) = \sum_{d|n} g(d)$, then $g(n) = \sum_{d|n} \mu(d) f(n/d)$. If $f(n) = \sum_{m=1}^n g(\lfloor n/m \rfloor)$, then $g(n) = \sum_{m=1}^n \mu(m) f(\lfloor \frac{n}{m} \rfloor)$.
- #primitive pythagorean triples with hypotenuse $< n$ approx $n/(2\pi)$.
- **Frobenius Number:** largest number which can’t be expressed as a linear combination of numbers a_1, \dots, a_n with non-negative coefficients. $g(a_1, a_2) = a_1 a_2 - a_1 - a_2$, $N(a_1, a_2) = (a_1 - 1)(a_2 - 1)/2$. $g(d \cdot a_1, d \cdot a_2, a_3) = d \cdot g(a_1, a_2, a_3) + a_3(d - 1)$. An integer $x > (\max_i a_i)^2$ can be expressed in such a way iff. $x \mid \gcd(a_1, \dots, a_n)$.

10.1. Physics.

- **Snell’s law:** $\frac{\sin \theta_1}{v_1} = \frac{\sin \theta_2}{v_2}$

10.2. **Markov Chains.** A Markov Chain can be represented as a weighted directed graph of states, where the weight of an edge represents the probability of transitioning over that edge in one timestep. Let $P^{(m)} = (p_{ij}^{(m)})$ be the probability matrix of transitioning from state i to state j in m timesteps, and note that $P^{(1)}$ is the adjacency matrix of the graph. **Chapman-Kolmogorov:** $p_{ij}^{(m+n)} = \sum_k p_{ik}^{(m)} p_{kj}^{(n)}$. It follows that $P^{(m+n)} = P^{(m)} P^{(n)}$ and $P^{(m)} = P^m$. If $p^{(0)}$ is the initial probability distribution (a vector), then $p^{(0)} P^{(m)}$ is the probability distribution after m timesteps.

The return times of a state i is $R_i = \{m \mid p_{ii}^{(m)} > 0\}$, and i is *aperiodic* if $\gcd(R_i) = 1$. A MC is aperiodic if any of its vertices is aperiodic. A MC is *irreducible* if the corresponding graph is strongly connected.

A distribution π is stationary if $\pi P = \pi$. If MC is irreducible then $\pi_i = 1/\mathbb{E}[T_i]$, where T_i is the expected time between two visits at i . π_j/π_i is the expected number of visits at j in between two consecutive visits at i . A MC is *ergodic* if $\lim_{m \rightarrow \infty} p^{(0)} P^m = \pi$. A MC is ergodic iff. it is irreducible and aperiodic.

A MC for a random walk in an undirected weighted graph (un-weighted graph can be made weighted by adding 1-weights) has $p_{uv} = w_{uv} / \sum_x w_{ux}$. If the graph is connected, then $\pi_u = \sum_x w_{ux} / \sum_v \sum_x w_{vx}$. Such a random walk is aperiodic iff. the graph is not bipartite.

An *absorbing* MC is of the form $P = \begin{pmatrix} Q & R \\ 0 & I_r \end{pmatrix}$. Let $N = \sum_{m=0}^\infty Q^m = (I_t - Q)^{-1}$. Then, if starting in state i , the expected number of steps till absorption is the i -th entry in $N1$. If starting in state i , the probability of being absorbed in state j is the (i, j) -th entry of NR .

Many problems on MC can be formulated in terms of a system of recurrence relations, and then solved using Gaussian elimination.

10.3. **Burnside’s Lemma.** Let G be a finite group that acts on a set X . For each g in G let X^g denote the set of elements in X that are fixed by g . Then the number of orbits

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

$$Z(S_n) = \frac{1}{n} \sum_{l=1}^n a_l Z(S_{n-l})$$

10.4. **Bézout’s identity.** If (x, y) is any solution to $ax + by = d$ (e.g. found by the Extended Euclidean Algorithm), then all solutions are given by

$$\left(x + k \frac{b}{\gcd(a, b)}, y - k \frac{a}{\gcd(a, b)}\right)$$

10.5. Misc.

10.5.1. *Determinants and PM.*

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n a_{i, \sigma(i)}$$

$$\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^n a_{i, \sigma(i)}$$

$$\begin{aligned} pf(A) &= \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \text{sgn}(\sigma) \prod_{i=1}^n a_{\sigma(2i-1), \sigma(2i)} \\ &= \sum_{M \in \text{PM}(n)} \text{sgn}(M) \prod_{(i,j) \in M} a_{i,j} \end{aligned}$$

10.5.2. *BEST Theorem.* Count directed Eulerian cycles. Number of OST given by Kirchoff’s Theorem (remove r/c with root) $\# \text{OST}(G, r) \cdot \prod_v (d_v - 1)!$

10.5.3. *Primitive Roots.* Only exists when n is $2, 4, p^k, 2p^k$, where p odd prime. Assume n prime. Number of primitive roots $\phi(\phi(n))$ Let g be primitive root. All primitive roots are of the form g^k where $k, \phi(p)$ are coprime.
 k -roots: $g^{i \cdot \phi(n)/k}$ for $0 \leq i < k$

10.5.4. *Sum of primes.* For any multiplicative f :

$$S(n, p) = S(n, p-1) - f(p) \cdot (S(n/p, p-1) - S(p-1, p-1))$$

10.5.5. *Floor.*

$$\lfloor \lfloor x/y \rfloor / z \rfloor = \lfloor x/(yz) \rfloor$$

$$x \% y = x - y \lfloor x/y \rfloor$$