University of Calgary Team Reference Document 2015

February 23, 2015

Contents

1 General Tips

- For `g++`, `#include <bits/stdc++.h>` includes all standard headers.
- The constant π is usually built-in as `M_PI`.

2 Geometry

2.1 Basic 2D Geometry

Basic definitions

```
const double EP = 1e-9; // do not use for angles
typedef complex<double> PX;
const PX BAD(1e100,1e100);
```

Cross/dot product, same slope test

```
double cp(PX a, PX b) {return (conj(a)*b).imag();}
double dp(PX a, PX b) {return (conj(a)*b).real();}
bool ss(PX a, PX b) {return fabs(cp(a,b)) < EP;}
```

Orientation: -1=CW, 1=CCW, 0=colinear

```
// Can be used to check if a point is on a line (0)
int ccw(PX a, PX b, PX c) {
    double r = cp(b-a, c-a);
    if (fabs(r) < EP) return 0;
    return r > 0 ? 1 : -1;
}
```

Check if x is on line segment from p_1 to p_2

```
bool onSeg(PX p1, PX p2, PX x) {
    return fabs(abs(p2-p1)-abs(x-p1)-abs(x-p2))<EP;
}
```

Point to line distance (x to $p + \vec{v}t$)

```
double ptToLine(PX p, PX v, PX x) {
    // Closest point on line: p + v*dp(v, x-p)
    return fabs(cp(v, x-p) / abs(v));
}
```

Intersection point of two lines

```
PX lineIntersect(PX p1, PX v1, PX p2, PX v2) {
    // If colinear, pick random point (p1)
    if (ss(v1, v2)) return ss(v1, p2-p1) ? p1 : BAD;
    return p1 + (cp(p2-p1,v2)/cp(v1,v2))*v1;
}
```

Intersection point of two line segments

```
PX segIntersect(PX p1, PX p2, PX q1, PX q2) {
    // Handle special cases for colinear
    if (onSeg(p1, p2, q1)) return q1;
    if (onSeg(p1, p2, q2)) return q2;
    if (onSeg(q1, q2, p1)) return p1;
    if (onSeg(q1, q2, p2)) return p2;
    PX ip = lineIntersect(p1, p2-p1, q1, q2-q1);
    return (onSeg(p1, p2, ip) && onSeg(q1, q2, ip))
        ? ip : BAD;
}
```

Area of polygon (including concave)

```
double area(vector<PX> const& P) {
    double a = 0.0;
    for (int i = 0; i < P.size(); i++)
        a += cp(P[i], P[(i+1)%P.size()]);
    return 0.5 * fabs(a);
}
```

Check if point is within convex polygon

```
// P must be a convex polygon sorted CCW
bool ptInConvexPolygon(vector<PX> const& P, PX p) {
    for (int i = 0; i < P.size(); i++)
        // Use == -1 to include edges of polygon
        if (ccw(P[i], P[(i+1)%P.size()], p) != 1)
            return false;
    return true;
}
```

2.2 Basic 3D Geometry

TODO

2.3 Convex Hull

Graham's scan. Complexity: $O(n \log n)$

```
vector<PX> pts;
void convexHull() {
    if (pts.empty()) return;
    int fi = 0;
    for (int i = 1; i < pts.size(); i++)
        if (pts[i].imag() + EP < pts[fi].imag() ||
           (fabs(pts[i].imag() - pts[fi].imag())<EP &&
           pts[i].real() + EP < pts[fi].real())) fi = i;
    swap(pts[0], pts[fi]);
    sort(++pts.begin(), pts.end(), [](PX a, PX b) {
        PX v1 = a - pts[0], v2 = b - pts[0];
        double a1 = arg(v1), a2 = arg(v2);
        // Use smaller epsilon for angles
        if (fabs(a1 - a2) > 1e-14) return a1 < a2;
        return abs(v1) < abs(v2);
    });
    int M = 2;
    for (int i = 2; i < pts.size(); i++) {
        while (M > 1 && ccw(pts[M-2], pts[M-1], pts[i]) <= 0) M--;
        swap(pts[i], pts[M++]);
    }
    if (M < pts.size()) pts.resize(M);
}
```

Notes:

- All intermediate colinear points and duplicate points are discarded
- If all points are colinear, the algorithm will output the two endpoints of the line
- Works with any number of points including 0, 1, 2
- Works with line segments colinear to the starting point

Example usage

```
pts.clear();
pts.emplace_back(0.0, 0.0); // put all the points in
convexHull();
// pts now contains the convex hull in CCW order, starting from lowest y point
```

3 Graphs

3.1 2-SAT

Kosaraju's algorithm. Complexity: $O(V+E)$

```
typedef vector<int> VI;
typedef vector<VI> VVI;

VVI adj, adjRev;
VI sccNum, sccStack, truthValues;

int VAR(int i) {return 2*i;}
int NOT(int i) {return i^1;}
int NVAR(int i) {return NOT(VAR(i));}
void addCond(int c1, int c2) {
    adj[NOT(c1)].push_back(c2);
    adjRev[c2].push_back(NOT(c1));
    adj[NOT(c2)].push_back(c1);
    adjRev[c1].push_back(NOT(c2));
}
void init2SAT(int numVars) {
    adj.clear(); adj.resize(2*numVars);
    adjRev.clear(); adjRev.resize(2*numVars);
}
void dfs(int i, int s, VVI& adj) {
    if (sccNum[i]) return;
    sccNum[i] = s;
    for (int j : adj[i]) dfs(j, s, adj);
    sccStack.push_back(i);
}
bool run2SAT() {
    sccStack.clear();
    sccNum.clear(); sccNum.resize(adj.size());
    for (int i = 0; i < adj.size(); i++) {
        dfs(i, 1, adj);
    }
    sccNum.clear(); sccNum.resize(adj.size());
    for (int s=1,i=sccStack.size()-1; i >= 0; i--) {
        int c = sccStack[i];
        if (sccNum[c]) continue;
        dfs(c, s++, adjRev);
    }
    truthValues.clear();
    truthValues.resize(adj.size()/2);
    for (int i = 0; i < adj.size(); i += 2) {
        if (sccNum[i] == sccNum[i+1]) return false;
        truthValues[i/2] = sccNum[i] > sccNum[i+1];
    }
    return true;
}
```

Example usage

```
init2SAT(N); // variables from 0 to N-1
addCond(VAR(4), NVAR(0)); // v4 or not v0
if (run2SAT()) {
    // there is a solution
    // truth values are in truthValues[0 to N-1]
}
```

3.2 Floyd-Warshall

Complexity: $O\left(V^3\right)$

```
let dist[V][V] be initialized to
    dist[v][v] = 0
    dist[u][v] = weight of edge else infinity
for k from 1 to V
    for i from 1 to V
        for j from 1 to V
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] = dist[i][k] + dist[k][j]
```

3.3 Articulation Points and Bridges

Graph does not need to be connected. Tested only on bidirectional (undirected) graphs. Complexity: $O(V+E)$

```
typedef vector<int> VI;
typedef vector<VI> VVI;

VVI adj;
VI dfs_low, dfs_num;
int cnt;

void dfs(int i, int r, int p) { // (current, root, parent)
    if (dfs_num[i] != -1) return;
    dfs_low[i] = dfs_num[i] = cnt++;
    int ap = i != r; // number of disconnected
                     // components if vertex is removed
    for (int j : adj[i]) if (j != p) {
        if (dfs_num[j] == -1) {
            dfs(j, r, i);
            if (dfs_low[j] >= dfs_num[i]) ap++;
            if (dfs_low[j] > dfs_num[i]) {
                // (i,j) is a bridge
                // each pair will only occur once
            }
            dfs_low[i] = min(dfs_low[i], dfs_low[j]);
        } else {
            dfs_low[i] = min(dfs_low[i], dfs_num[j]);
        }
    }
    if (ap >= 2) {
        // i is an articulation point
        // each vertex will only occur once
    }
}
```

Example usage:

```
// N is number of vertices
cnt = 0;
adj.assign(N, VI()); // fill adj
dfs_num.assign(N, -1);
dfs_low.resize(N); // initialization not necessary
for (int n = 0; n < N; n++) dfs(n, n, -1);
```

3.4 Bellman-Ford

Consider terminating the loop if no weight was modified in the loop. Complexity: $O(VE)$

```
let weight[V] = all infinity except weight[source] = 0
let parent[V] = all null

loop V-1 times
    for each edge (u,v) with weight w
        if weight[u] + w < weight[v]
            weight[v] = weight[u] + w
            parent[v] = u

// detecting negative weight cycles
for each edge (u,v) with weight w
    if weight[u] + w < weight[v]
    then graph has negative weight cycle
```

3.5 Eulerian Path/Cycle

TODO

3.6 Max Bipartite Matching

Matches M applicants to N jobs.
Complexity: $O\left(V^3\right)$

```
bool adj[M][N];
int matchR[N], seen[N];

bool bpm(int u) {
    for (int v = 0; v < N; v++) {
        if (adj[u][v] && !seen[v]) {
            seen[v] = true;
            if (matchR[v] < 0
            || bpm(matchR[v])) {
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}
```

Example usage:

```
// adj must have all edges
memset(matchR, -1, sizeof matchR);
for (int u = 0; u < M; u++) {
    memset(seen, 0, sizeof seen);
    if (bpm(u)) then there is a matching
}
```

3.7 Stable Marriage/Matching

Only tested with equal numbers of men and women. Complexity: $O\left(MW\right)$

```
typedef vector<int> VI;
vector<VI> mPref, wPref;
VI wPartner;
void stableMarriage() {
    int M = mPref.size();
    VI pr(M), fm(M);
    iota(begin(fm), end(fm), 0);
    wPartner.assign(wPref.size(), -1);
    while (!fm.empty()) {
        int m = fm.back();
        int w = mPref[m][pr[m]++];
        if (wPartner[w] == -1 || wPref[w][m]
        < wPref[w][wPartner[w]]) {
            fm.pop_back();
            if (wPartner[w] != -1)
                fm.push_back(wPartner[w]);
            wPartner[w] = m;
        }
    }
}
```

Example usage:

```
mPref.clear(); wPref.clear();

// Man 0 ranks women 2, 0, 1 (best to worst)
mPref.push_back(VI{2,0,1});

// Woman 0 ranks men 1, 2, 0 (best to worst)
wPref.push_back(VI{2,0,1});

stableMarriage(); // matching is in wPartner
```

3.8 Max Flow (with Min Cut)

Edmonds-Karp algorithm.
Complexity: $O\left(\min\left(VE^2, fE\right)\right)$ where f is the maximum flow
Note: The worst case performance is very rare and only happens with a specially crafted test case. Problem authors usually do not penalize you for using Edmonds-Karp over more efficient algorithms.

```
typedef long long LL;
typedef pair<int,LL> PT;

vector<vector<int>> adj;
vector<int> parent;
map<int, map<int, LL>> cap; // or LL[][] if V is not too large

LL totalflow;
bool runMaxFlow(int source, int sink) {
    parent.assign(adj.size(), -1);
    parent[source] = -2; // use different value to recognize min cut
    queue<PT> bfs;
    bfs.emplace(source, 1LL<<60); // must be larger than max flow
    while (!bfs.empty()) {
        PT t = bfs.front();
        bfs.pop();
        for (int j : adj[t.first]) {
            if (cap[t.first][j] == 0 || parent[j] != -1) continue;
            parent[j] = t.first;
            LL f = min(t.second, cap[t.first][j]);
            if (j == sink) {
                while (j != source) {
                    int p = parent[j];
                    cap[p][j] -= f;
                    cap[j][p] += f;
                    j = p;
                }
                totalflow += f;
                return true;
            }
            bfs.emplace(j, f);
        }
    }
    return false;
}
void initMaxFlow(int nodes) {
    totalflow = 0;
    adj.clear(); adj.resize(nodes);
    cap.clear(); // or memset if cap is an array
}
void addEdge(int a, int b, LL w) {
    adj[a].push_back(b);
    adj[b].push_back(a); // even without bidirectional edges
    cap[a][b] = w;
    //cap[b][a] = w; // if you want bidirectional edges
}
```

Example usage

```
initMaxFlow(desired number of nodes); // nodes from 0 to N-1
addEdge(0, 3, 123); // adds edge fron 0 to 3 with capacity 123
while (runMaxFlow(source, sink)) {}
// The max flow is now in totalflow
// The min cut: Nodes where parent[i] == -1 belong to the T
// component, otherwise S
```

3.9 Min Cost Max Flow

Edmonds-Karp with Bellman-Ford algorithm. Complexity: $O\left(\min\left(V^2E^2, fVE\right)\right)$ where f is the maximum flow

Note: The worst case performance is very rare and only happens with a specially crafted test case. Problem authors usually do not penalize you for using Edmonds-Karp with Bellman-Ford over more efficient algorithms. The Bellman-Ford part of this code is designed to exit early whenever possible.

```
const int NODES = 101 // maximum number of nodes
typedef long long LL;
typedef pair<int,int> PT;

vector<vector<int>> adj;
LL cap[NODES][NODES], cost[NODES][NODES], flow[NODES][NODES];

LL totalflow, totalcost;
bool runMCMF(int source, int sink) {
    vector<LL> mf(NODES), weight(NODES, 1LL<<60); // must be larger than longest path
    vector<int> parent(NODES, -1);
    weight[source] = 0;
    mf[source] = 1LL<<60; // value must be larger than max flow
    for (int i = 0, lm = 0; i < NODES-1 && lm == i; i++) {
        for (int u = 0; u < NODES; u++) {
            for (int v : adj[u]) {
                if (!cap[u][v] && !flow[v][u]) continue;
                LL w = (flow[v][u]) ? -cost[v][u] : cost[u][v];
                if (weight[u] + w < weight[v]) {
                    weight[v] = weight[u] + w;
                    parent[v] = u;
                    mf[v] = min(mf[u], (flow[v][u]) ? flow[v][u] : cap[u][v]);
                    lm = i+1;
                }
            }
        }
    }
    LL f = mf[sink];
    if (!f) return false;
    for (int j = sink; j != source;) {
        int p = parent[j];
        if (flow[j][p]) {
            cap[j][p] += f;
            flow[j][p] -= f;
        } else {
            cap[p][j] -= f;
            flow[p][j] += f;
        }
        totalcost += f * (weight[j] - weight[p]);
        j = p;
    }
    totalflow += f;
    return true;
}
void initMCMF() {
    totalflow = totalcost = 0;
    adj.clear(); adj.resize(NODES);
    memset(cap, 0, sizeof cap);
    memset(cost, 0, sizeof cost);
    memset(flow, 0, sizeof flow);
}
void addEdge(int a, int b, LL w, LL c) {
    adj[a].push_back(b);
    adj[b].push_back(a); // this line is necessary even without bidirectional edges
    cap[a][b] = w; // set cap[b][a] and cost[b][a] to the same to get bidirectional edges
    cost[a][b] = c;
}
```

Example usage

```
initMCMF();
addEdge(0, 3, 123, 5); // adds edge fron 0 to 3 with capacity 123 and cost 5
while (runMCMF(source, sink)) {}
// The max flow is now in totalflow and total cost in totalcost
```

4 Sequences and Strings

4.1 AVL Tree

Creating your own BST can be useful in certain situations; e.g. to find the kth element in a set in $O(\log n)$.

```
struct node {
    node *l, *r;
    int nodes, height, val;
    node(int val)
    : l(0), r(0), nodes(1), height(1), val(val) {}
} *root;

int height(node *n) {return (n) ? n->height : 0;}
int nodes(node *n) {return (n) ? n->nodes : 0;}
int gb(node *n)
    {return (n) ? height(n->l) - height(n->r) : 0;}

void updHeight(node *n) {
    n->height = max(height(n->l), height(n->r)) + 1;
    n->nodes = nodes(n->l) + nodes(n->r) + 1;
}

void leftRotate(node **n) {
    node *nr = (**n).r;
    (**n).r = nr->l;
    nr->l = *n;
    *n = nr;
    updHeight((**n).l);
    updHeight(*n);
}
void rightRotate(node **n) {
    node *nr = (**n).l;
    (**n).l = nr->r;
    nr->r = *n;
    *n = nr;
    updHeight((**n).r);
    updHeight(*n);
}

void fix(node **n) {
    if (!*n) return;
    updHeight(*n);
    if (gb(*n) > 1) {
        if (gb((**n).l) < 0) leftRotate(&(**n).l);
        rightRotate(n);
    } else if (gb(*n) < -1) {
        if (gb((**n).r) > 0) rightRotate(&(**n).r);
        leftRotate(n);
    }
}

void insert(node **n, int val) {
    if (!*n) *n = new node(val);
    else if (val < (**n).val) insert(&(**n).l, val);
    else if (val > (**n).val) insert(&(**n).r, val);
    fix(n);
}

int predec(node **n) {
    int ret;
    if ((**n).r) ret = predec(&(**n).r);
    else {
        node *x = *n;
        *n = x->l;
        ret = x->val;
        delete x;
    }
    fix(n);
    return ret;
}

void remove(node **n, int val) {
    if (!*n) return;
    if (val < (**n).val) remove(&(**n).l, val);
    else if (val > (**n).val) remove(&(**n).r, val);
    else if ((**n).l) (**n).val = predec(&(**n).l);
    else {
        node *x = *n;
        *n = x->r;
        delete x;
    }
    fix(n);
}
```

Example: in-order traversal

```
void inorder(node *n) {
    if (!n) return;
    inorder(n->l);
    cout << n->val << endl;
    inorder(n->r);
}
```

Example: get kth element in set (zero-based)

```
int kth(node *n, int k) {
    if (!n) return 2000000000;
    if (k < nodes(n->l)) return kth(n->l, k);
    else if (k > nodes(n->l))
        return kth(n->r, k - nodes(n->l) - 1);
    return n->val;
}
```

Example: count number of elements strictly less than x

```
int count(node *n, int x) {
    if (!n) return 0;
    if (x <= n->val) return count(n->l, x);
    return 1 + nodes(n->l) + count(n->r, x);
}
```

4.2 KMP

Knuth-Morris-Pratt algorithm. Complexity: $O(m + n)$

This function returns a vector containing the zero-based index of the start of each match of K in S. It works with strings, vectors, and pretty much any array-indexed data structure that has a size method. Matches may overlap.

For GNU C++, `strstr()` uses KMP, but `string.find()` in C++ and `String.indexOf()` in Java do not.

```
template<class T>
vector<int> KMP(T const& S, T const& K) {
    vector<int> b(K.size() + 1, -1);
    vector<int> matches;

    // Preprocess
    for (int i = 1; i <= K.size(); i++) {
        int pos = b[i - 1];
        while (pos != -1 && K[pos] != K[i - 1]) pos = b[pos];
        b[i] = pos + 1;
    }

    // Search
    int sp = 0, kp = 0;
    while (sp < S.size()) {
        while (kp != -1 && (kp == K.size() || K[kp] != S[sp])) kp = b[kp];
        kp++; sp++;
        if (kp == K.size()) matches.push_back(sp - K.size());
    }

    return matches;
}
```

4.3 Longest Common Subsequence

Note that if characters are never repeated in at least one string, LCS can be reduced to LIS. Complexity: $O(nm)$

```
template<class T>
int LCS(T const& A, T const& B) {
    int dp[][] = {}; // set size appropriately
    for (int a = 0; a < A.size(); a++) {
        for (int b = 0; b < B.size(); b++) {
            if (a) dp[a][b] = max(dp[a][b], dp[a-1][b]);
            if (b) dp[a][b] = max(dp[a][b], dp[a][b-1]);
            if (A[a] == B[b])
                dp[a][b] = max(dp[a][b], ((a && b) ? dp[a-1][b-1] : 0) + 1);
        }
    }
    return dp[A.size() - 1][B.size() - 1];
}
```

4.4 Longest Increasing Subsequence

Complexity: $O(n \log k)$ where k is the length of the LIS

```
vector<int> L; // L[x] = smallest end of length x LIS
for each x in sequence {
    auto it = lower_bound(L.begin(), L.end(), x);
    if (it == L.end()) L.push_back(x); else *it = x;
}
// Length of LIS is L.size()
```

4.5 Fenwick Tree / Binary Indexed Tree

This implements a D-dimensional Fenwick tree with indexes $[1, N-1]$. Complexity: $O\left(\log^D N\right)$ per operation

```
template<int N, int D=1>
class FenwickTree {
    vector<int> tree;
    int isum(int ps) {return tree[ps];}
    template<class... T>
    int isum(int ps, int n, T... tail) {
        int a = 0;
        while (n) {
            a += isum(ps*N + n, tail...);
            n -= (n & -n);
        }
        return a;
    }
    void iupd(int u, int ps) {tree[ps] += u;}
    template<class... T>
    void iupd(int u, int ps, int n, T... tail) {
        while (n < N) { // TODO: check cond
            iupd(u, ps*N + n, tail...);
            n += (n & -n);
        }
    }
public:
    FenwickTree() : tree(pow(N, D)) {}
    template<class... T, class = class enable_if<sizeof...(T)==D>::type>
    int sum(T... v) {return isum(0, v...);}
    template<class... T, class = class enable_if<sizeof...(T)==D>::type>
    void upd(int u, T... v) {iupd(u, 0, v...);}
};
```

Example usage

```
FenwickTree<130> t; // creates 1D fenwick tree with indexes [1,129]
t.upd(5, 7); // adds 5 to index 7
t.sum(14); // gets sum of all points [1, 14]

FenwickTree<130, 3> t; // creates 3D fenwick tree with indexes [1,129]
t.upd(5, 7, 8, 9); // adds 5 to the point (7, 8, 9)
t.sum(14, 15, 16); // gets sum of all points [(1, 1, 1), (14, 15, 16)]
```

Simple 1D tree (remember, first index is 1)

```
typedef long long LL;
const int N = 100002;
LL f1[N], f2[N];

LL sum(LL *f, int n) {
    LL a = 0;
    while (n) {
        a += f[n];
        n -= (n & -n);
    }
    return a;
}
```

```
void upd(LL *f, int n, LL v) {
    while (n < N) {
        f[n] += v;
        n += (n & -n);
    }
}

// only required for range queries
// with range updates
LL rsum(int n) {
    return sum(f1, n) * n - sum(f2, n);
}
```

To get sum from $[p, q]$:

```
rsum(q) - rsum(p-1)
```

To add v to $[p, q]$:

```
upd(f1, p, v);
upd(f1, q+1, -v);
upd(f2, p, v*(p-1));
upd(f2, q+1, -v*q);
```

4.6 Sparse Table

Solves static range min/max query with $O(n \log n)$ preprocessing and $O(1)$ per query. This code does range minimum query.

```
int N, A[1000000], spt[1000000][19]; // spt[N][floor(log2(N))]

void sptBuild() {
    for (int n = 0; 1<<n <= N; n++)
        for (int i = 0; i+(1<<n) <= N; i++)
            spt[i][n] = (n) ? min(spt[i][n-1],
                                  spt[i+(1<<(n-1))][n-1]) : A[i];
}

int sptQuery(int i, int j) {
    int n = 31 - __builtin_clz(j-i+1); // floor(log2(j-i+1))
    return min(spt[i][n], spt[j+1-(1<<n)][n]);
}
```

Example usage

```
N = 10; // size of array
A = {1, 5, -3, 7, -2, 1, 6, -8, 4, -2};
sptBuild();
sptQuery(0, 9); // returns -8
sptQuery(1, 1); // return 5
sptQuery(1, 4); // returns -3
sptQuery(5, 8); // returns -8
```

4.7 Segment Tree

The size of the segment tree should be 4 times the data size. Building is $O(n)$. Querying and updating is $O(\log n)$.

4.7.1 Example 1 (no range updates)

This segment tree finds the maximum subsequence sum in an arbitrary range.

```
int A[50000];

struct node {
    int bestPrefix, bestSuffix, bestSum, sum;
    void merge(node& ls, node& rs) {
        bestPrefix
            = max(ls.bestPrefix, ls.sum + rs.bestPrefix);
        bestSuffix
            = max(rs.bestSuffix, rs.sum + ls.bestSuffix);
        bestSum
            = max(ls.bestSuffix + rs.bestPrefix,
              max(ls.bestSum, rs.bestSum));
        sum = ls.sum + rs.sum;
    }
} seg[200000];

void segBuild(int n, int l, int r) {
    if (l == r) {
        seg[n].bestPrefix = seg[n].bestSuffix
            = seg[n].bestSum = seg[n].sum = A[l];
        return;
    }
    int m = (l+r)/2;
    segBuild(2*n+1, l, m);
    segBuild(2*n+2, m+1, r);
    seg[n].merge(seg[2*n+1], seg[2*n+2]);
}

node segQuery(int n, int l, int r, int i, int j) {
    if (i <= l && r <= j) return seg[n];
    int m = (l+r)/2;
    if (m < i) return segQuery(2*n+2, m+1, r, i, j);
    if (m >= j) return segQuery(2*n+1, l, m, i, j);
    node ls = segQuery(2*n+1, l, m, i, j);
    node rs = segQuery(2*n+2, m+1, r, i, j);
    node a;
    a.merge(ls, rs);
    return a;
}

void segUpdate(int n, int l, int r, int i) {
    if (i < l || i > r) return;
    if (i == l && l == r) {
        seg[n].bestPrefix = seg[n].bestSuffix
            = seg[n].bestSum = seg[n].sum = A[l];
        return;
    }
    int m = (l+r)/2;
    segUpdate(2*n+1, l, m, i);
    segUpdate(2*n+2, m+1, r, i);
    seg[n].merge(seg[2*n+1], seg[2*n+2]);
}
```

4.7.2 Example 2 (with range updates)

This segment tree stores a series of booleans and allows swapping all booleans in any range.

```
struct node {
    int sum;
    bool inv;
    void apply(int x) {
        sum = x - sum;
        inv = !inv;
    }
    void split(node& ls, node& rs, int l, int m, int r) {
        if (inv) {
            ls.apply(m-l+1);
            rs.apply(r-m);
            inv = false;
        }
    }
    void merge(node& ls, node& rs) {
        sum = ls.sum + rs.sum;
    }
} seg[200000];

node segQuery(int n, int l, int r, int i, int j) {
    if (i <= l && r <= j) return seg[n];
    int m = (l+r)/2;
    seg[n].split(seg[2*n+1],seg[2*n+2],l,m,r);
    if (m < i) return segQuery(2*n+2, m+1, r, i, j);
    if (m >= j) return segQuery(2*n+1, l, m, i, j);
    node ls = segQuery(2*n+1, l, m, i, j);
    node rs = segQuery(2*n+2, m+1, r, i, j);
    node a;
    a.merge(ls, rs);
    return a;
}

void segUpdate(int n, int l, int r, int i, int j) {
    if (i > r || j < l) return;
    if (i <= l && r <= j) {
        seg[n].apply(r-l+1);
        return;
    }
    int m = (l+r)/2;
    seg[n].split(seg[2*n+1],seg[2*n+2],l,m,r);
    segUpdate(2*n+1, l, m, i, j);
    segUpdate(2*n+2, m+1, r, i, j);
    seg[n].merge(seg[2*n+1], seg[2*n+2]);
}
```

Example usage:

```
N = size of list;
segBuild(0, 0, N-1);
segQuery(0, 0, N-1, i, j); // queries range [i, j]
segUpdate(0, 0, N-1, i, j); // updates range [i, j] (you may need to add parameters)
```

4.8 Suffix Array

4.8.1 Notes

- Terminating character ($) is not required (unlike CP book), but it is useful to compute the longest common substring of multiple strings
- Use slow version if possible as it is shorter

4.8.2 Initialization

Complexity: $O\left(n \log^2 n\right)$

```
typedef vector<int> VI;

VI sa, ra, lcp;
string s;

void saInit() {
    int l = s.size();
    sa.resize(l);
    iota(sa.begin(), sa.end(), 0);
    ra.assign(s.begin(), s.end());
    for (int k = 1; k < l; k *= 2) {
        // To use radix sort, replace sort() with:
        // csort(l, k); csort(l, 0);
        sort(sa.begin(), sa.end(), [&](int a, int b){
            if (ra[a] != ra[b]) return ra[a] < ra[b];
            int ak = a+k < l ? ra[a+k] : -1;
            int bk = b+k < l ? ra[b+k] : -1;
            return ak < bk;
        });
        VI ra2(l); int x = 0;
        for (int i = 1; i < l; i++) {
            if (ra[sa[i]] != ra[sa[i-1]] ||
                sa[i-1]+k >= l ||
                ra[sa[i]+k] != ra[sa[i-1]+k]) x++;
            ra2[sa[i]] = x;
        }
        ra = ra2;
    }
}
```

4.8.3 Initialization (slow)

Complexity: $O\left(n^2 \log n\right)$

```
void saInit() {
    int l = s.size();
    sa.resize(l);
    iota(sa.begin(), sa.end(), 0);
    sort(sa.begin(), sa.end(), [](int a, int b) {
        return s.compare(a, -1, s, b, -1) < 0;
    });
}
```

4.8.4 Example suffix array

i	sa[i]	lcp[i]	Suffix
0	0	0	abacabacx
1	4	4	abacx
2	2	1	acabacx
3	6	2	acx
4	1	0	bacabacx
5	5	3	bacx
6	3	0	cabacx
7	7	1	cx
8	8	0	x

4.8.5 Longest Common Prefix array

Complexity: $O(n)$

```
void saLCP() {
    int l = s.size();
    lcp.resize(l);
    VI p(l), rsa(l);
    for (int i = 0; i < l; i++) {
        p[sa[i]] = (i) ? sa[i-1] : -1;
        rsa[sa[i]] = i;
    }
    int x = 0;
    for (int i = 0; i < l; i++) {
        // Note: The $ condition is optional and is
        // useful for finding longest common substring
        while (p[i] != -1 && p[i]+x < l &&
            s[i+x] == s[p[i]+x] && s[i+x] != '$') x++;
        lcp[rsa[i]] = x;
        if (x) x--;
    }
}
```

4.8.6 String matching

Returns a vector containing the zero-based index of the start of each match of `m` in `s`. Complexity: $O(m \log n)$

```
VI saFind(string const& m) {
    auto r = equal_range(sa.begin(), sa.end(), -1,
    [&](int i, int j) {
        int a = 1;
        if (i == -1) {swap(i, j); a = -1;}
        return a*s.compare(i, m.size(), m) < 0;
    });
    VI occ(r.first, r.second);
    sort(occ.begin(), occ.end()); // optional
    return occ;
}
```

4.8.7 Optional counting sort

Improves `saInit()` performance to $O(n \log n)$
Usually not necessary, about 4x speed up on a 1M string

```
void csort(int l, int k) {
    int m = max(300, l+1);
    VI c(m), sa2(l);
    for (int i = 0; i < l; i++) c[i+k<l ? ra[i+k]+1 : 0]++;
    for (int s = 0, i = 0; i < m; i++) {
        swap(c[i], s); s += c[i];
    }
    for (int i = 0; i < l; i++)
        sa2[c[sa[i]+k<l ? ra[sa[i]+k]+1 : 0]++] = sa[i];
    sa = sa2;
}
```

4.8.8 Example usage

```
s = "abacabacx";
saInit(); // Now sa[] is filled
saLCP();  // Now lcp[] is filled
```

5 Math and Other Algorithms

5.1 Exponentiation by Squaring

Computes x^n. Complexity: $O(\log n)$ assuming multiplication and division are constant time.

```
result = 1
while n is nonzero
    if n is odd
        result *= x
        n-= 1
    x *= x
    n /= 2
```

5.2 Extended Euclidean

Complexity: $O(\log(\min(a, b)))$

```
int x, y, d;
void gcd(int a, int b) {
    if (b == 0) {x = 1; y = 0; d = a; return;}
    gcd(b, a % b);
    x -= y * (a / b);
    swap(x, y);
}
```

Finds $d = \gcd(a, b)$ and solves the equation $ax + by = d$. The equation $ax + by = c$ has a solution iff c is a multiple of $d = \gcd(a, b)$. If (x, y) is a solution, all other solutions have the form $(x + k\frac{b}{d}, y - k\frac{a}{d}), k \in \mathbb{Z}$.

5.3 Fast Fourier Transform

Cooley-Tukey algorithm. Complexity: $O(n \log n)$

```
typedef complex<double> PX;
typedef valarray<PX> VPX;

void fft(VPX& p, double c=2.0) {
    size_t n = p.size();
    if (n == 1) return;
    VPX g = p[slice(0, n/2, 2)], h = p[slice(1, n/2, 2)];
    fft(g, c); fft(h, c);
    PX x0 = polar(1.0, c*M_PI/n), x = 1.0;
    for (size_t i = 0; i < n; i++) {
        p[i] = (i<n/2) ? g[i]+x*h[i] : g[i-n/2]+x*h[i-n/2];
        x *= x0;
    }
}

void ifft(VPX& p) {fft(p, -2.0); p /= p.size();}
```

Example: fast polynomial multiplication

```
VPX polymul(VPX const& p1, VPX const& p2) {
    size_t pn = p1.size() + p2.size() - 1, n = pn;
    // round up n to nearest power of 2
    if (n & (n-1)) n = 1 << (32 - __builtin_clz(n));
    VPX p1e(n), p2e(n);
    copy(begin(p1), end(p1), begin(p1e));
    copy(begin(p2), end(p2), begin(p2e));
    fft(p1e); fft(p2e);
    p1e *= p2e;
    ifft(p1e);
    VPX p(pn);
    copy_n(begin(p1e), pn, begin(p));
    return p;
}
```

The discrete Fourier transform transforms a sequence of N complex numbers $x_0, x_1, \cdots, x_{N-1}$ into an N-periodic sequence of complex numbers:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}$$

Multiplying the individual terms of the DFT gives the convolution (polynomial multiplication):

$$(f*g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

$$x_N * y = \mathrm{DFT}^{-1}\left[\mathrm{DFT}\{x\} \cdot \mathrm{DFT}\{y\}\right]$$

5.4 Sieve and Prime Factorization

Sieve: $O(n \log \log n)$, prime factorization: $O\left(\frac{\sqrt{n}}{\log n}\right)$

```
const int MAX_P = 182; // greater than square root of max n to factorize

vector<int> primes;

void sieve() {
    bool np[MAX_P] = {1,1};
    for (int i = 2; i < MAX_P; i++) {
        if (np[i]) continue;
        primes.push_back(i);
        for (int j = 2*i; j < MAX_P; j += i) np[j] = true;
    }
}

// Returns 12 = [<2, 2>, <3, 1>]
vector<pair<int, int>> primeFactorize(int n) {
    vector<pair<int, int>> f;
    for (int p : primes) {
        if (p*p > n) break;
        int a = 0;
        while (n % p == 0) {n /= p; a++;}
        if (a) f.emplace_back(p, a);
    }
    if (n != 1) f.emplace_back(n, 1);
    return f;
}
```

5.5 Union-Find Disjoint Sets

Complexity: $O(1)$ per operation. Note: $O(\log n)$ if one of union-by-rank or path compression is omitted

```
vector<int> ds, dr;
int findSet(int i) {return ds[i] == i ? i : (ds[i] = findSet(ds[i]));}
void unionSet(int i, int j) {
    int x = findSet(i), y = findSet(j);
    if (dr[x] < dr[y]) ds[x] = y;
    else if (dr[x] > dr[y]) ds[y] = x;
    else {ds[x] = y; dr[y]++;}
}
bool sameSet(int i, int j) {return findSet(i) == findSet(j);}
```

Example initialization:

```
dr.assign(N, 0);
ds.resize(N);
iota(begin(ds), end(ds), 0);
```

5.6 Simplex

Complexity: $O(m \log n)$ on average

```
const int MAXM = 100, MAXN = 100;
const double EPS = 1e-9, INF = 1.0/0.0;
double A[MAXM][MAXN], X[MAXN];
int basis[MAXM], out[MAXN];

void pivot(int m, int n, int a, int b) {
    int i, j;
    for (i = 0; i <= m; i++) if (i != a)
        for (j = 0; j <= n; j++) if (j != b)
            A[i][j] -= A[a][j] * A[i][b] / A[a][b];
    for (j = 0; j <= n; j++) if (j != b)
        A[a][j] /= A[a][b];
    for (i = 0; i <= m; i++) if (i != a)
        A[i][b] = -A[i][b]/A[a][b];

    A[a][b] = 1/A[a][b];

    i = basis[a];
    basis[a] = out[b];
    out[b] = i;
}

double simplex(int m, int n) {
    int i, j, ii, jj;

    for (j = 0; j <= n; j++) {
        A[0][j] *= -1;
        out[j] = j;
    }
    for (i = 0; i <= m; i++) basis[i] = -i;

    for (;;) {
        for (i = ii = 1; i <= m; i++)
            if (A[i][n] < A[ii][n] || (A[i][n]==A[ii][n] && basis[i]<basis[ii])) ii = i;
        if (A[ii][n] >= -EPS) break;
        for (j = jj = 0; j < n; j++)
            if (A[ii][j] < A[ii][jj]-EPS || (A[ii][j] < A[ii][jj]+EPS && out[i]<out[j])) jj=j;
        if (A[ii][jj] >= -EPS) return -INF;
        pivot(m,n,ii,jj);
    }
    for (;;) {
        for (j = jj = 0; j < n; j++)
            if (A[0][j] < A[0][jj] || (A[0][j] == A[0][jj] && out[j] < out[jj])) jj = j;
        if (A[0][jj] > -EPS) break;

        for (i=1,ii=0; i <= m; i++)
            if (A[i][jj] > EPS && (!ii || A[i][n]/A[i][jj] < A[ii][n]/A[ii][jj]-EPS ||
                (A[i][n]/A[i][jj] < A[ii][n]/A[ii][jj]+EPS && basis[i]<basis[ii]))) ii = i;
        if (A[ii][jj] <= EPS) return INF;
        pivot(m,n,ii,jj);
    }
    for (j = 0; j < n; j++) X[j] = 0;
    for (i = 1; i <= m; i++) if (basis[i] >= 0) X[basis[i]] = A[i][n];
    return A[0][n];
}
```

Notes:

- m = number of inequalities
- n = number of variables
- $A[m+1][n+1]$ array of coefficients
- Row 0 is the objective function
- Rows 1 to m are less-than inequalities
- Columns 0 to $n-1$ are inequality coefficients
- Column n is the inequality constant (0 for objective function)
- $X[n]$ are result variables
- Returns maximum value of objective function ($-$INF for infeasible, INF for unbounded)

Example usage:

```
memset(A, 0, sizeof A);
A[0] = {1,5,7};
A[1] = {2,4,5,12};
A[2] = {7,2,1,42};
double ans = simplex(2, 3);
double x1 = X[0]; //etc
```

Maximize $x_1 + 5x_2 + 7x_3 = \text{ans}$, where
$2x_1 + 4x_2 + 5x_3 \leq 12$
$7x_1 + 2x_2 + x_3 \leq 42$

6 Tricks for Bit Manipulation

6.1 GCC Builtins and Other Tricks

For these builtins, you can append `l` or `ll` to the function names to get the `long` or `long long` version.

`int __builtin_ffs(int x)`	Returns one plus the index of the least significant 1-bit of x. Returns 0 if $x = 0$.
`int __builtin_clz(unsigned int x)`	Returns the number of leading 0-bits in x, starting at the most significant bit position. If $x = 0$, the result is undefined.
`int __builtin_ctz(unsigned int x)`	Returns the number of trailing 0-bits in x, starting at the most significant bit position. If $x = 0$, the result is undefined.
`int __builtin_clrsb(int x)`	Returns the number of leading redundant sign bits in x, i.e. the number of bits following the most significant bit that are identical to it. There are no special cases for 0 or other values.
`int __builtin_popcount(unsigned int x)`	Returns the number of 1-bits in x. (Slow on x86 without SSE4 flag)
`int __builtin_parity(unsigned int x)`	Returns the parity of x, i.e. the number of 1-bits in x modulo 2.
`uintN_t __builtin_bswapN(uintN_t x)`	Returns x with the order of the bytes reversed. $N = 16, 32, 64$
`(x & (x - 1)) == 0`	Checks if x is a power of 2 (only one bit set). Note: 0 is edge case.
`(x + y - 1) / y`	Finds $\left\lceil \frac{x}{y} \right\rceil$ (positive integers only)

6.2 Lexicographically Next Bit Permutation

Suppose we have a pattern of N bits set to 1 in an integer and we want the next permutation of N 1 bits in a lexicographical sense. For example, if N is 3 and the bit pattern is 00010011, the next patterns would be 00010101, 00010110, 00011001, 00011010, 00011100, 00100011, and so forth. The following is a fast way to compute the next permutation.

```
unsigned int v; // current permutation of bits
unsigned int w; // next permutation of bits

unsigned int t = v | (v - 1); // t gets v's least significant 0 bits set to 1
// Next set to 1 the most significant bit to change,
// set to 0 the least significant ones, and add the necessary 1 bits.
w = (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
```

6.3 Loop Through All Subsets

For example, if **bs** $= 10110$, loop through **bt** $= 10100, 10010, 10000, 00110, 00100, 00010$

```
for (int bt = (bs-1) & bs; bt; bt = (bt-1) & bs) {
    int bu = bt ^ bs; // contains the opposite subset of bt (e.g. if bt = 10000, bu = 00110)
}
```

6.4 Parsing and Printing __int128

GCC supports `(unsigned) __int128` type on most platforms (notable exception is Windows). However, it does not currently support printing and parsing of those types.

```
string printint128(__int128 a) { // prints as decimal
    if (!a) return "0";
    string s;
    while (a) {
        s = char(llabs(a % 10) + '0') + s;
        if (-10 < a && a < 0) s = '-' + s;
        a /= 10;
    }
    return s;
}

__int128 parseint128(string s) { // parses decimal number
    __int128 a = 0, sgn = 1;
    for (char c : s) {
        if (c == '-') sgn *= -1; else a = a * 10 + sgn * (c - '0');
    }
    return a;
}
```

7 Math Formulas and Theorems

Chinese remainder theorem	Suppose $n_1 \cdots n_k$ are positive integers that are pairwise coprime. Then, for any series of integers $a_1 \cdots a_k$, there are an infinite number of solutions x where $$\begin{cases} x & = a_1 \pmod{n_1} \\ & \cdots \\ x & = a_k \pmod{n_k} \end{cases}$$ All solutions x are congruent modulo $N = n_1 \cdots n_k$.
Fermat's last theorem	No three positive integers a, b, and c can satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2.
Fermat's little theorem	For any prime p and integer a, $a^p \equiv a \pmod{p}$. If a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$ and a^{p-2} is the modular inverse of a modulo p.
Triangles	$A = \frac{1}{2}bh = \frac{1}{2}ab\sin\gamma$