Contents

1 judge-bash

```
#!/usr/bin/env bash
set -u

# Kiem tra cong cu can thiet (Linux)
for cmd in g++ timeout /usr/bin/time diff awk sed; do
  if ! command -v ${cmd%% *} >/dev/null 2>&1; then
    echo "Thieu cong cu: $cmd. Vui long cai dat truoc." >&2
    echo "Go y cai dat tren Ubuntu: sudo apt-get update && sudo apt-get install
         -y g++ time coreutils diffutils"
    exit 1
  fi
done

echo "=== Cau hinh ==="
read -rp "Duong dan file code mau (.cpp): " MODEL_CPP
read -rp "Duong dan file sinh input (.cpp): " GEN_CPP
read -rp "Duong dan file can check (.cpp): " SOL_CPP
read -rp "Gioi han bo nho (MB): " MEM_MB
read -rp "Gioi han thoi gian (vi du 0.5s): " TL_IN

# Chuan hoa thoi gian
if [[ "$TL_IN" =~ ^[0-9]+(\.[0-9]+)?s$ ]]; then
  TL="$TL_IN"
elif [[ "$TL_IN" =~ ^[0-9]+(\.[0-9]+)?$ ]]; then
  TL="${TL_IN}s"
else
  echo "Dinh dang thoi gian khong hop le. Vi du: 1, 0.5, 0.5s" >&2
  exit 1
fi

# Kiem tra file
for f in "$MODEL_CPP" "$GEN_CPP" "$SOL_CPP"; do
  [[ -f "$f" ]] || { echo "Khong tim thay file: $f" >&2; exit 1; }
done

WORKDIR="$(mktemp -d -t judge-XXXXXX)"
cleanup(){ rm -rf "$WORKDIR"; }
trap cleanup EXIT

MODEL_EXE="$WORKDIR/model"
GEN_EXE="$WORKDIR/gen"
SOL_EXE="$WORKDIR/sol"

INPUT_TXT="$WORKDIR/input.txt"
ANS_TXT="$WORKDIR/answer.txt"
OUT_TXT="$WORKDIR/output.txt"
METRICS_TXT="$WORKDIR/metrics.txt"
RUN_LOG="$WORKDIR/run.log"

echo
echo "=== Bien dich (Linux) ==="
echo "- Bien dich code mau..."
if ! g++ -std=c++17 -O2 -pipe "$MODEL_CPP" -o "$MODEL_EXE" 2>"$WORKDIR/
    compile_model.log"; then
  echo "Loi bien dich code mau. Xem $WORKDIR/compile_model.log" >&2; exit 1
fi
echo "- Bien dich sinh input..."
if ! g++ -std=c++17 -O2 -pipe "$GEN_CPP" -o "$GEN_EXE" 2>"$WORKDIR/compile_gen.
    log"; then
  echo "Loi bien dich sinh input. Xem $WORKDIR/compile_gen.log" >&2; exit 1
fi
echo "- Bien dich code can check..."
if ! g++ -std=c++17 -O2 -pipe "$SOL_CPP" -o "$SOL_EXE" 2>"$WORKDIR/compile_sol.
    log"; then
  echo "Loi bien dich code can check. Xem $WORKDIR/compile_sol.log" >&2; exit 1
fi

echo
echo "=== Sinh du lieu va dap an ==="
echo "- Chay file sinh input -> $INPUT_TXT"
if ! "$GEN_EXE" > "$INPUT_TXT" 2>"$WORKDIR/gen.log"; then
  echo "Loi khi chay file sinh input. Xem $WORKDIR/gen.log" >&2; exit 1
```

```
fi

echo "- Chay code mau -> $ANS_TXT"
if ! "$MODEL_EXE" < "$INPUT_TXT" > "$ANS_TXT" 2>"$WORKDIR/model.log"; then
  echo "Loi khi chay code mau. Xem $WORKDIR/model.log" >&2; exit 1
fi

echo
echo "=== Chay code can check voi gioi han Linux ==="
echo "Gioi han thoi gian: $TL"
echo "Gioi han bo nho: ${MEM_MB} MB"
: > "$METRICS_TXT"; : > "$RUN_LOG"

# Gioi han bo nho bang ulimit (KB)
MEM_KB_LIMIT=$(( MEM_MB * 1024 ))

# Thuc thi trong subshell de ulimit chi anh huong tien trinh con
(
  ulimit -v "$MEM_KB_LIMIT" 2>/dev/null || true
  /usr/bin/time -f "TIME:%e\nMEM_KB:%M" -o "$METRICS_TXT" \
    timeout --preserve-status "$TL" "$SOL_EXE" < "$INPUT_TXT" > "$OUT_TXT" 2> "
        $RUN_LOG"
)
EXEC_STATUS=$?

TIME_SEC=$(sed -n 's/^TIME:\(.*\)$/\1/p' "$METRICS_TXT")
MEM_KB=$(sed -n 's/^MEM_KB:\(.*\)$/\1/p' "$METRICS_TXT")
[[ -z "${TIME_SEC:-}" ]] && TIME_SEC="0"
[[ -z "${MEM_KB:-}" ]] && MEM_KB="0"
MEM_MB_USED=$(awk -v kb="$MEM_KB" 'BEGIN{printf "%.2f", kb/1024}')

if [[ $EXEC_STATUS -eq 124 ]]; then
  VERDICT="TLE"
else
  # Danh gia MLE: du do MEM_KB (RSS dinh cao) > gioi han nhap
  MLE_FLAG=$(awk -v used="$MEM_MB_USED" -v lim="$MEM_MB" 'BEGIN{if (used>lim)
      print 1; else print 0}')
  if [[ "$MLE_FLAG" -eq 1 ]]; then
    VERDICT="MLE"
  else
    if diff -q "$ANS_TXT" "$OUT_TXT" >/dev/null 2>&1; then
      VERDICT="AC"
    else
      VERDICT="WA"
    fi
  fi
fi

echo "=== Ket qua ==="
echo "Phan dinh: $VERDICT"
echo "Thoi gian su dung: ${TIME_SEC}s"
echo "Bo nho dinh cao: ${MEM_MB_USED} MB (gioi han ${MEM_MB} MB)"
echo
echo "Tep tam: $WORKDIR"
echo "Log:"
echo "  - $WORKDIR/compile_model.log"
echo "  - $WORKDIR/compile_gen.log"
echo "  - $WORKDIR/compile_sol.log"
echo "  - $WORKDIR/gen.log"
echo "  - $WORKDIR/model.log"
echo "  - $RUN_LOG"
```

2 judge-cpp

```
#include <bits/stdc++.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>

using namespace std;

// ---- Tien ich nho ----
static inline bool file_exists(const string& p){
    return ::access(p.c_str(), F_OK) == 0;
}

int compile_cpp(const string& src, const string& outExe, string logPath){
    string cmd = "g++ -std=c++17 -O2 -pipe \"" + src + "\" -o \"" + outExe + "
        \" 2>\"" + logPath + "\"";
    return ::system(cmd.c_str());
}

struct RunResult {
    int exit_code = -1;
    bool signaled = false;
    int term_signal = 0;
    bool tle = false;
    bool mle = false;
    double wall_time_sec = 0.0; // do bang CLOCK_MONOTONIC
    long long max_rss_kb = 0;   // tu getrusage
};

// Doc ru_maxrss tu getrusage(RUSAGE_CHILDREN) sau khi wait
static inline long long get_children_maxrss_kb(){
    struct rusage ru{};
    if (getrusage(RUSAGE_CHILDREN, &ru) == 0){
        return (long long)ru.ru_maxrss; // Linux: KB
    }
    return 0;
}

// Chay chuong trinh con duoi gioi han, do thoi gian, bo nho
RunResult run_with_limits(const string& exe,
                          const string& inFile,
                          const string& outFile,
```

```
                    double timeLimitSec, // wall time
                    long long memLimitMB){
    RunResult rr;
    // Mo file in/out
    int fin = ::open(inFile.c_str(), O_RDONLY);
    if (fin < 0){ perror("Mo input that bai"); }
    int fout = ::open(outFile.c_str(), O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fout < 0){ perror("Mo output that bai"); }

    // Gioi han bo nho (RLIMIT_AS) -> bytes
    rlimit mem{};
    mem.rlim_cur = mem.rlim_max = (rlim_t)memLimitMB * 1024ull * 1024ull;

    // Thoi diem bat dau
    timespec t0{}, t1{};
    clock_gettime(CLOCK_MONOTONIC, &t0);

    pid_t pid = fork();
    if (pid == -1){
        perror("fork that bai");
        if (fin>=0) close(fin);
        if (fout>=0) close(fout);
        return rr;
    }
    if (pid == 0){
        // Child: dat gioi han, chuyen huong IO, exec
        if (setrlimit(RLIMIT_AS, &mem) != 0){
            // khong dat duoc gioi han -> van thu chay
        }
        // Co the gioi han thoi luong CPU (nguyen giay) neu muon:
        // rlimit cpu{}; cpu.rlim_cur = cpu.rlim_max = (rlim_t)ceil(
            timeLimitSec);
        // setrlimit(RLIMIT_CPU, &cpu);

        if (fin >= 0) dup2(fin, STDIN_FILENO);
        if (fout >= 0) dup2(fout, STDOUT_FILENO);
        // Dong cac fd thua
        if (fin >= 0) close(fin);
        if (fout >= 0) close(fout);

        // Thuc thi
        execl(exe.c_str(), exe.c_str(), (char*)nullptr);
        // Neu exec loi
        _exit(127);
    }

    // Parent: theo doi thoi gian, kill neu qua han
    if (fin>=0) close(fin);
    if (fout>=0) close(fout);

    const int poll_ms = 5; // tan so kiem tra
    int status = 0;
    while (true){
        pid_t r = waitpid(pid, &status, WNOHANG);
        clock_gettime(CLOCK_MONOTONIC, &t1);
        double elapsed = (t1.tv_sec - t0.tv_sec) + (t1.tv_nsec - t0.tv_nsec)/1
            e9;
        if (r == 0){
            // con dang chay
            if (elapsed > timeLimitSec){
                rr.tle = true;
                kill(pid, SIGKILL);
                waitpid(pid, &status, 0);
                break;
            }
            // ngu 1 chut
            this_thread::sleep_for(chrono::milliseconds(poll_ms));
        } else if (r == pid){
            // da ket thuc
            break;
        } else {
            // loi
            break;
        }
    }

    // Cap nhat thoi gian
    clock_gettime(CLOCK_MONOTONIC, &t1);
    rr.wall_time_sec = (t1.tv_sec - t0.tv_sec) + (t1.tv_nsec - t0.tv_nsec)/1e9;

    // Trang thai thoat
    if (WIFEXITED(status)){
        rr.exit_code = WEXITSTATUS(status);
    } else if (WIFSIGNALED(status)){
        rr.signaled = true;
        rr.term_signal = WTERMSIG(status);
    }
    // Peak RSS (KB) cua tat ca child da doi
    rr.max_rss_kb = get_children_maxrss_kb();

    // Phan doan MLE: neu peak RSS > gioi han (tinh theo MB)
    double usedMB = rr.max_rss_kb / 1024.0;
    if (usedMB > memLimitMB) rr.mle = true;

    return rr;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string modelCpp, genCpp, solCpp;
    string timeStr;
    double timeLimit = 0.0;
    long long memLimitMB = 0;

    cout << "=== Cau hinh ===\n";
    cout << "Duong dan file code mau (.cpp): ";
    getline(cin, modelCpp);
    cout << "Duong dan file sinh input (.cpp): ";
```

```
    getline(cin, genCpp);
    cout << "Duong dan file can check (.cpp): ";
    getline(cin, solCpp);
    cout << "Gioi han bo nho (MB): ";
    {
        string tmp; getline(cin, tmp);
        memLimitMB = atoll(tmp.c_str());
    }
    cout << "Gioi han thoi gian (vi du 0.5s hoac 0.5): ";
    getline(cin, timeStr);
    {
        string s = timeStr;
        if (!s.empty() && s.back()=='s') s.pop_back();
        timeLimit = atof(s.c_str());
        if (timeLimit <= 0){
            cerr << "Thoi gian khong hop le.\n";
            return 1;
        }
    }

    // Kiem tra file ton tai
    if (!file_exists(modelCpp) || !file_exists(genCpp) || !file_exists(solCpp))
        {
        cerr << "Khong tim thay mot hoac nhieu file .cpp.\n";
        return 1;
    }

    // Thu muc tam
    char templ[] = "/tmp/judgeXXXXXX";
    char* wd = mkdtemp(templ);
    if (!wd){ perror("mkdtemp"); return 1; }
    string WORKDIR = wd;
    string modelExe = WORKDIR + "/model";
    string genExe   = WORKDIR + "/gen";
    string solExe   = WORKDIR + "/sol";
    string inputTxt = WORKDIR + "/input.txt";
    string ansTxt   = WORKDIR + "/answer.txt";
    string outTxt   = WORKDIR + "/output.txt";

    cout << "\n=== Bien dich ===\n";
    cout << "- Bien dich code mau...\n";
    if (compile_cpp(modelCpp, modelExe, WORKDIR+"/compile_model.log") != 0){
        cerr << "Loi bien dich code mau. Xem " << WORKDIR << "/compile_model.
            log\n";
        return 1;
    }
    cout << "- Bien dich sinh input...\n";
    if (compile_cpp(genCpp, genExe, WORKDIR+"/compile_gen.log") != 0){
        cerr << "Loi bien dich sinh input. Xem " << WORKDIR << "/compile_gen.
            log\n";
        return 1;
    }
    cout << "- Bien dich code can check...\n";
    if (compile_cpp(solCpp, solExe, WORKDIR+"/compile_sol.log") != 0){
        cerr << "Loi bien dich code can check. Xem " << WORKDIR << "/
            compile_sol.log\n";
        return 1;
    }

    cout << "\n=== Sinh du lieu va dap an ===\n";
    {
        // gen -> input.txt
        RunResult rg = run_with_limits(genExe, "/dev/null", inputTxt, 10.0,
            1024 /*1GB de rong rai*/);
        if (rg.exit_code != 0 || rg.signaled){
            cerr << "Loi khi chay file sinh input. Ma thoat: " << rg.exit_code
                << "\n";
            return 1;
        }
    }
    {
        // model < input -> answer.txt
        // cho model han 10s va 1024 MB cho an toan
        RunResult rm = run_with_limits(modelExe, inputTxt, ansTxt, 10.0, 1024);
        if (rm.exit_code != 0 || rm.signaled){
            cerr << "Loi khi chay code mau. Ma thoat: " << rm.exit_code << "\n"
                ;
            return 1;
        }
    }

    cout << "\n=== Chay code can check voi gioi han ===\n";
    cout << "Gioi han thoi gian: " << timeLimit << "s\n";
    cout << "Gioi han bo nho: " << memLimitMB << " MB\n";

    RunResult rs = run_with_limits(solExe, inputTxt, outTxt, timeLimit,
        memLimitMB);

    string verdict;
    if (rs.tle){
        verdict = "TLE";
    } else if (rs.mle){
        verdict = "MLE";
    } else if (rs.signaled){
        // Neu chet do SIGSEGV hoac khac, khong TLE/MLE, xem la WA (hoac RTE
            tuy y)
        verdict = "WA";
    } else {
        // So sanh ket qua
        // So sanh nhi phan: giong y he
        string diffCmd = "diff -q \"" + ansTxt + "\" \"" + outTxt + "\" > /dev/
            null 2>&1";
        int d = ::system(diffCmd.c_str());
        verdict = (d==0) ? "AC" : "WA";
    }

    cout << "=== Ket qua ===\n";
    cout << "Phan dinh: " << verdict << "\n";
    cout << "Thoi gian su dung: " << fixed << setprecision(3) << rs.
        wall_time_sec << "s\n";
```

```
    cout << "Bo nho dinh cao: " << (rs.max_rss_kb/1024.0) << " MB (gioi han "
        << memLimitMB << " MB)\n";
    cout << "\nTep tam: " << WORKDIR << "\n";
    cout << "Dang xuat:\n";
    cout << "  - " << WORKDIR << "/compile_model.log\n";
    cout << "  - " << WORKDIR << "/compile_gen.log\n";
    cout << "  - " << WORKDIR << "/compile_sol.log\n";

    return 0;
}
```

3 AhoCorasick

```
const int NODE = (int) 1e6 + 1;
const int NC = 26;

int nextNode[NODE][NC];
int chr[NODE];
int parent[NODE];
int prefix[NODE];
int numNodes;
set<int> match[NODE];

int getPrefix(int);

int go(int u, int c) {
    if (nextNode[u][c] != -1) return nextNode[u][c];
    if (u == 0) return 0;
    return nextNode[u][c] = go(getPrefix(u), c);
}

int getPrefix(int u) {
    if (prefix[u] != -1) return prefix[u];
    if (u == 0 || parent[u] == 0) return prefix[u] = 0;
    return prefix[u] = go(getPrefix(parent[u]), chr[u]);
}

void add(const string &s, int id) {
    int u = 0;
    for (int i = 0; i < (int) s.size(); ++i) {
        int c = s[i] - 'A';
        if (nextNode[u][c] == -1) {
            nextNode[u][c] = numNodes;
            fill(nextNode[numNodes], nextNode[numNodes] + NC, -1);
            chr[numNodes] = c;
            parent[numNodes] = u;
            prefix[numNodes] = -1;
            match[numNodes].clear();
            match[numNodes].insert(-1);
            ++numNodes;
        }
        u = nextNode[u][c];
    }
    match[u].insert(id);
}

set<int>& getMatch(int u) {
    if (match[u].count(-1) == 0) return match[u];
    const set<int> &foo = getMatch(getPrefix(u));
    match[u].insert(foo.begin(), foo.end());
    match[u].erase(-1);
    return match[u];
}

void init() {
    fill(nextNode[0], nextNode[0] + NC, -1);
    numNodes = 1;
}
```

4 BitsetAddition

```
// Bitset addition using long long
// Add two bitsets represented as arrays of long long
// Useful for subset sum DP optimization

template<int MAXN>
struct BitsetAdd {
    static const int BLOCK = 64;
    static const int SIZE = (MAXN + BLOCK - 1) / BLOCK;

    unsigned long long a[SIZE];

    BitsetAdd() {
        memset(a, 0, sizeof(a));
    }

    void set(int pos) {
        a[pos / BLOCK] |= (1ULL << (pos % BLOCK));
    }

    bool test(int pos) const {
        return (a[pos / BLOCK] >> (pos % BLOCK)) & 1;
    }

    void reset() {
        memset(a, 0, sizeof(a));
    }

    // Add value to all set bits (shift left by value positions)
    void add(int value) {
        if (value == 0) return;

        int block_shift = value / BLOCK;
        int bit_shift = value % BLOCK;

        if (bit_shift == 0) {
            // Simple block shift
            for (int i = SIZE - 1; i >= block_shift; --i) {
```

```
            a[i] = a[i - block_shift];
        }
        for (int i = 0; i < block_shift; ++i) {
            a[i] = 0;
        }
    } else {
        // Complex shift with carry
        for (int i = SIZE - 1; i > block_shift; --i) {
            a[i] = (a[i - block_shift] << bit_shift) |
                   (a[i - block_shift - 1] >> (BLOCK - bit_shift));
        }
        a[block_shift] = a[0] << bit_shift;
        for (int i = 0; i < block_shift; ++i) {
            a[i] = 0;
        }
    }
}

// OR operation
BitsetAdd operator|(const BitsetAdd &other) const {
    BitsetAdd result;
    for (int i = 0; i < SIZE; ++i) {
        result.a[i] = a[i] | other.a[i];
    }
    return result;
}

// OR assignment
BitsetAdd& operator|=(const BitsetAdd &other) {
    for (int i = 0; i < SIZE; ++i) {
        a[i] |= other.a[i];
    }
    return *this;
}

// AND operation
BitsetAdd operator&(const BitsetAdd &other) const {
    BitsetAdd result;
    for (int i = 0; i < SIZE; ++i) {
        result.a[i] = a[i] & other.a[i];
    }
    return result;
}

// XOR operation
BitsetAdd operator^(const BitsetAdd &other) const {
    BitsetAdd result;
    for (int i = 0; i < SIZE; ++i) {
        result.a[i] = a[i] ^ other.a[i];
    }
    return result;
}

// Count set bits
int count() const {
    int cnt = 0;
    for (int i = 0; i < SIZE; ++i) {
        cnt += __builtin_popcountll(a[i]);
    }
    return cnt;
}

// Find first set bit
int first() const {
    for (int i = 0; i < SIZE; ++i) {
        if (a[i]) {
            return i * BLOCK + __builtin_ctzll(a[i]);
        }
    }
    return -1;
}
};

// Example usage for subset sum
// BitsetAdd<100001> dp;
// dp.set(0);
// for (int i = 0; i < n; ++i) {
//     BitsetAdd<100001> tmp = dp;
//     tmp.add(arr[i]);
//     dp |= tmp;
// }
```

5 ConvexHull

```
struct Point {
    long long x, y;
    bool operator < (const Point &v) const {
        return x == v.x ? y < v.y : x < v.x;
    }
    long long cross(const Point &p, const Point &q) const {
        return (p.x - x) * (q.y - y) - (p.y - y) * (q.x - x);
    }
};

vector<Point> convexHull(vector<Point> p) {
    sort(p.begin(), p.end());
    int k = 0, n = p.size();
    vector<Point> poly (2 * n);
    for(int i = 0; i < n; ++i) {
        while(k >= 2 && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    for(int i = n-2, t = k+1; i >= 0; --i) {
        while(k >= t && poly[k-2].cross(poly[k-1], p[i]) < 0) --k;
        poly[k++] = p[i];
    }
    poly.resize(min(n, max(0, k - 1)));
    return poly;
}
```

6 DirectedMST

```
const int maxe = 100111, maxv = 100;

// Index from 0, running time O(E*V)
namespace chuliu {
    struct Cost;
    vector<Cost> costlist;

    struct Cost {
        int id, val, used, a, b, pos;
        Cost() { val = -1; used = 0; }
        Cost(int _id, int _val, bool temp) {
            a = b = -1; id = _id; val = _val; used = 0;
            pos = costlist.size(); costlist.push_back(*this);
        }
        Cost(int _a, int _b) {
            a = _a; b = _b; id = -1; val = costlist[a].val-costlist[b].val;
            used = 0; pos = costlist.size(); costlist.push_back(*this);
        }
        void push() {
            if (id == -1) {
                costlist[a].used += used;
                costlist[b].used -= used;
            }
        }
    };

    struct Edge {
        int u, v;
        Cost cost;
        Edge() {}
        Edge(int id, int _u, int _v, int c) {
            u = _u; v = _v; cost = Cost(id, c, 0);
        }
    } edge[maxe];

    int n, m, root, pre[maxv], node[maxv], vis[maxv], best[maxv];

    void init(int _n) {
        n = _n; m = 0;
        costlist.clear();
    }

    void add(int id, int u, int v, int c) {
        edge[m++] = Edge(id, u, v, c);
    }

    int mst(int root) {
        int ret = 0;
        while (true) {
            REP(i, n) best[i] = -1;
            REP(e, m) {
                int u = edge[e].u, v = edge[e].v;
                if ((best[v] == -1 || edge[e].cost.val < costlist[best[v]].val)
                     && u != v) {
                    pre[v] = u;
                    best[v] = edge[e].cost.pos;
                }
            }
            REP(i, n) if (i != root && best[i] == -1) return -1;
            int cntnode = 0;
            memset(node, -1, sizeof node); memset(vis, -1, sizeof vis);
            REP(i, n) if (i != root) {
                ret += costlist[best[i]].val;
                costlist[best[i]].used++;
                int v = i;
                while (vis[v] != i && node[v] == -1 && v != root) {
                    vis[v] = i;
                    v = pre[v];
                }
                if (v != root && node[v] == -1) {
                    for (int u = pre[v]; u != v; u = pre[u]) node[u] = cntnode;
                    node[v] = cntnode++;
                }
            }
            if (cntnode == 0) break;
            REP(i, n) if (node[i] == -1) node[i] = cntnode++;
            REP(e, m) {
                int v = edge[e].v;
                edge[e].u = node[edge[e].u];
                edge[e].v = node[edge[e].v];
                if (edge[e].u != edge[e].v) edge[e].cost = Cost(edge[e].cost.
                    pos, best[v]);
            }
            n = cntnode;
            root = node[root];
        }

        return ret;
    }

    vector<int> trace() {
        vector<int> ret;
        FORD(i, costlist.size()-1,0) costlist[i].push();
        REP(i, costlist.size()) {
            Cost cost = costlist[i];
            if (cost.id != -1 && cost.used > 0) ret.push_back(cost.id);
        }
        return ret;
    }
}
```

7 Euclid

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.
```

```
typedef vector<int> VI;
typedef pair<int,int> PII;

int mod(int a, int b) { // return a % b (positive value)
    return ((a%b)+b)%b;
}

int gcd(int a, int b) { // computes gcd(a,b)
    int tmp;
    while(b){a%=b; tmp=a; a=b; b=tmp;}
    return a;
}

int lcm(int a, int b) { // computes lcm(a,b)
    return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a/b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
    int x, y;
    VI solutions;
    int d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n/d), n));
    }
    return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
    int x, y;
    int d = extended_euclid(a, n, x, y);
    if (d > 1) return -1;
    return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid(x, y, s, t);
    if (a%d != b%d) return make_pair(0, -1);
    return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
    PII ret = make_pair(a[0], x[0]);
    for (int i = 1; i < (int) x.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
    int d = gcd(a,b);
    if (c%d) {
        x = y = -1;
    } else {
        x = c/d * mod_inverse(a/d, b/d);
        y = (c-a*x)/b;
    }
}

int main() {
    cout << gcd(14, 30) << endl; // 2
    int x, y, d = extended_euclid(14, 30, x, y);
    cout << d << " " << x << " " << y << endl; // 2 -2 1
    VI sols = modular_linear_equation_solver(14, 30, 100); // 95 45
    for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";
    cout << endl;
    cout << mod_inverse(8, 9) << endl; // 8
    int xs[] = {3, 5, 7, 4, 6};
    int as[] = {2, 3, 2, 3, 5};
    PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as, as+3));
    cout << ret.first << " " << ret.second << endl; // 23 56
    ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3, as+5));
    cout << ret.first << " " << ret.second << endl; // 11 12
    linear_diophantine(7, 2, 5, x, y);
    cout << x << " " << y << endl; // expected: 5 -15
}
```

8 EulerTotient

```
int phi[n];
for (int i = 0; i < n; i++) phi[i] = i;
```

```
for (int i = 1; i < n; i++)
        for (int j = 2 * i; j < n; j += i)
                phi[j] -= phi[i];
```

9 FFT

```
typedef complex<double> Complex;

template<class T> int size(const T &a) {
    return a.size();
}

unsigned roundUp(unsigned v) {
    --v;
    v |= v >> 1;
    v |= v >> 2;
    v |= v >> 4;
    v |= v >> 8;
    v |= v >> 16;
    return v + 1;
}

int reverse(int num, int lg) {
    int res = 0;
    for(int i = 0; i < lg; ++i) if(num & 1 << i)
        res |= 1 << (lg - i - 1);
    return res;
}

template<class T> ostream& operator << (ostream& out, const vector<T> &a) {
    for(int i = 0; i < size(a); ++i) {
        if(i > 0) out << ' ';
        out << a[i];
    }
    return out;
}

vector<Complex> fft(vector<Complex> a, bool invert) {
    int n = size(a), lg = 0;
    while(1 << lg < n) ++lg;
    vector<Complex> roots (n);
    for(int i = 0; i < n; ++i) {
        double alpha = 2 * M_PI / n * i * (invert ? -1 : 1);
        roots[i] = Complex(cos(alpha), sin(alpha));
    }
    for(int i = 0; i < n; ++i) {
        int rev = reverse(i, lg);
        if(i < rev) swap(a[i], a[rev]);
    }
    for(int len = 2; len <= n; len <<= 1)
        for(int i = 0; i < n; i += len)
            for(int j = 0; j < len >> 1; ++j) {
                Complex u = a[i + j], v = a[i + j + (len >> 1)] * roots[n / len
                     * j];
                a[i + j] = u + v;
                a[i + j + (len >> 1)] = u - v;
            }
    if(invert) for(int i = 0; i < n; ++i) a[i] /= n;
    return a;
}

vector<long long> multiply(const vector<int> &a, const vector<int> &b) {
    int n = roundUp(size(a) + size(b) - 1);
    vector<Complex> pa (n), pb (n);
    for(int i = 0; i < size(a); ++i) pa[i] = a[i];
    for(int i = 0; i < size(b); ++i) pb[i] = b[i];
    pa = fft(pa, false); pb = fft(pb, false);
    for(int i = 0; i < n; ++i) pa[i] *= pb[i];
    pa = fft(pa, true);
    vector<long long> res (n);
    for(int i = 0; i < n; ++i) res[i] = round(real(pa[i]));
    return res;
}
```

10 FFTMod

```
const int MODULO = 998244353;
const int ROOT = 3; // Primitive root

void fft(vector<int> &a, bool invert) {
    int n = a.size();
    assert((n & (n - 1)) == 0);
    int lg = __builtin_ctz(n);
    for (int i = 0; i < n; ++i) {
        int j = 0;
        for (int k = 0; k < lg; ++k) if ((i&1<<k)!=0) j |= 1 << (lg-k-1);
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len *= 2) {
        int wlen = power(ROOT, (MODULO - 1) / len);
        if (invert) wlen = inverse(wlen);
        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; ++j) {
                int u = a[i + j];
                int v = 1LL * a[i + j + len / 2] * w % MODULO;
                a[i + j] = (u + v) % MODULO;
                a[i + j + len / 2] = (u - v + MODULO) % MODULO;
                w = 1LL * w * wlen % MODULO;
            }
        }
    }
    if (invert) {
        int mul = inverse(n);
        for (auto &x : a) x = 1LL * x * mul % MODULO;
    }
}
```

11 FenwickTree

```
struct FenwickTree { // 1-based index
    vector<int> tmul, tadd;
    int n;

    FenwickTree(int _n): tmul (_n + 1), tadd (_n + 1), n (_n) {
    }

    void update_point(int x, int mul, int add) {
        while (1<=x && x<=n) {
            tmul[x]+=mul;
            tadd[x]+=add;
            x+=x&(-x);
        }
    }

    void update_range(int l, int r, int val) { // l to r (inclusive)
        update_point(l,val,-val*(l-1));
        update_point(r,-val,val*r);
    }

    int get(int x) {
        int mul = 0, add = 0, fst = x;
        while (1<=x && x<=n) {
            mul+=tmul[x];
            add+=tadd[x];
            x=x&(x-1);
        }
        return (mul*fst+add);
    }

    int value(int x) {
        return (get(x)-get(x-1));
    }
};
```

12 GlobalMinCut

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

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

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
    int N = weights.size();
    VI used(N), cut, best_cut;
    int best_weight = -1;

    for (int phase = N-1; phase >= 0; phase--) {
        VI w = weights[0];
        VI added = used;
        int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last;
            last = -1;
            for (int j = 1; j < N; j++)
                if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
            if (i == phase-1) {
                for (int j=0; j<N; j++) weights[prev][j] += weights[last][j];
                for (int j=0; j<N; j++) weights[j][prev] = weights[prev][j];
                used[last] = true;
                cut.push_back(last);
                if (best_weight == -1 || w[last] < best_weight) {
                    best_cut = cut;
                    best_weight = w[last];
                }
            } else {
                for (int j = 0; j < N; j++)
                    w[j] += weights[last][j];
                added[last] = true;
            }
        }
    }
    return make_pair(best_weight, best_cut);
}
```

13 HexagonalGrid

```
int roundCount(int round) {
    return (6*round);
}
int roundSum(int round) {
    return (6*round*(round+1)/2);
}
int findRound(int n) {
    int res=1;
    while (roundSum(res)<n) res++;
    return (res);
}
pair<int,int> cord(int n) {
    if (n==0) return (make_pair(0,0));
    int c=findRound(n);
    int prev=roundSum(c-1);
    if (n<=prev+c) return (make_pair(c,n-prev));
    if (n<=prev+2*c) return (make_pair(prev+2*c-n,c));
```

```
    if (n<=prev+3*c) return (make_pair(prev+2*c-n,prev+3*c-n));
    if (n<=prev+4*c) return (make_pair(-c,prev+3*c-n));
    if (n<=prev+5*c) return (make_pair(n-prev-5*c,-c));
    return (make_pair(n-prev-5*c,n-prev-6*c));
}
bool inRound(int x,int y,int c) {
    if (0<=y && y<=c && x==c) return (true);
    if (0<=x && x<=c && y==c) return (true);
    if (0<=y && y<=c && y-x==c) return (true);
    if (-c<=y && y<=0 && x==-c) return (true);
    if (-c<=x && x<=0 && y==-c) return (true);
    if (0<=x && x<=c && x-y==c) return (true);
    return (false);
}
int findRound(int x,int y) {
    int res=1;
    while (!inRound(x,y,res)) res++;
    return (res);
}
int number(int x,int y) {
    if (x==0 && y==0) return (0);
    int c=findRound(x,y);
    int prev=roundSum(c-1);
    if (1<=y && y<=c && x==c) return (prev+y);
    if (0<=x && x<=c && y==c) return (prev+2*c-x);
    if (0<=y && y<=c && y-x==c) return (prev+2*c-x);
    if (-c<=y && y<=0 && x==-c) return (prev+3*c-y);
    if (-c<=x && x<=0 && y==-c) return (prev+5*c+x);
    return (prev+5*c+x);
}
```

14 Manacher

```
vector<int> manacher(const string &os) {
    int n = os.size();
    string s;
    for(int i = 0; i < n; ++i) {
        s += os[i];
        if(i != n - 1) s += '$';
    }
    int mx = 0, id = 0;
    n = s.size();
    vector<int> p (n);
    for(int i = 0; i < n; ++i) {
        p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
        while(p[i] <= i && i + p[i] < n && s[i - p[i]] == s[i + p[i]]) ++p[i];
        if(i + p[i] > mx) {
            mx = i + p[i];
            id = i;
        }
    }
    return p;
}
```

15 PersistentSegmentTree

```
// Persistent Segment Tree
// Supports point update and range query with version control
// Space: O(N + Q*log(N)) where Q is number of updates
// Time: O(log(N)) per operation

struct PersistentSegmentTree {
    struct Node {
        int val;
        int left, right;
        Node() : val(0), left(-1), right(-1) {}
        Node(int v, int l, int r) : val(v), left(l), right(r) {}
    };

    vector<Node> tree;
    vector<int> roots;
    int n;

    PersistentSegmentTree(int _n) : n(_n) {
        tree.reserve(2 * n + 1000000); // Reserve space for nodes
        roots.push_back(build(0, n - 1));
    }

    PersistentSegmentTree(vector<int> &a) : n(a.size()) {
        tree.reserve(2 * n + 1000000);
        roots.push_back(build(a, 0, n - 1));
    }

    int build(int l, int r) {
        int node = tree.size();
        tree.push_back(Node());
        if (l == r) {
            tree[node].val = 0;
        } else {
            int mid = (l + r) / 2;
            tree[node].left = build(l, mid);
            tree[node].right = build(mid + 1, r);
            tree[node].val = tree[tree[node].left].val + tree[tree[node].right
                ].val;
        }
        return node;
    }

    int build(vector<int> &a, int l, int r) {
        int node = tree.size();
        tree.push_back(Node());
        if (l == r) {
            tree[node].val = a[l];
        } else {
            int mid = (l + r) / 2;
            tree[node].left = build(a, l, mid);
            tree[node].right = build(a, mid + 1, r);
            tree[node].val = tree[tree[node].left].val + tree[tree[node].right
```

```
            ].val;
    }
    return node;
}

// Update position pos to value val, creates new version
int update(int node, int l, int r, int pos, int val) {
    int new_node = tree.size();
    tree.push_back(tree[node]); // Copy current node

    if (l == r) {
        tree[new_node].val = val;
    } else {
        int mid = (l + r) / 2;
        if (pos <= mid) {
            tree[new_node].left = update(tree[node].left, l, mid, pos, val)
                ;
        } else {
            tree[new_node].right = update(tree[node].right, mid + 1, r, pos
                , val);
        }
        tree[new_node].val = tree[tree[new_node].left].val + tree[tree[
            new_node].right].val;
    }
    return new_node;
}

void update(int pos, int val) {
    int new_root = update(roots.back(), 0, n - 1, pos, val);
    roots.push_back(new_root);
}

// Query sum in range [ql, qr] for specific version
int query(int node, int l, int r, int ql, int qr) {
    if (node == -1 || qr < l || r < ql) return 0;
    if (ql <= l && r <= qr) return tree[node].val;

    int mid = (l + r) / 2;
    return query(tree[node].left, l, mid, ql, qr) +
           query(tree[node].right, mid + 1, r, ql, qr);
}

int query(int version, int ql, int qr) {
    return query(roots[version], 0, n - 1, ql, qr);
}

// Get value at position pos for specific version
int get(int node, int l, int r, int pos) {
    if (l == r) return tree[node].val;
    int mid = (l + r) / 2;
    if (pos <= mid) return get(tree[node].left, l, mid, pos);
    else return get(tree[node].right, mid + 1, r, pos);
}

int get(int version, int pos) {
    return get(roots[version], 0, n - 1, pos);
}

int version_count() {
    return roots.size();
}
};

// Example usage for k-th smallest in range [l, r]:
// Build PST on sorted positions
// For each element, update its position in sorted order
// Query difference between version[r+1] and version[l]
// Binary search on answer

// K-th smallest number in range [l, r] using PST
struct KthSmallest {
    PersistentSegmentTree pst;
    int n;

    KthSmallest(vector<int> &a) : n(a.size()), pst(n) {
        vector<pair<int, int>> sorted_a;
        for (int i = 0; i < n; ++i) {
            sorted_a.push_back({a[i], i});
        }
        sort(sorted_a.begin(), sorted_a.end());

        for (int i = 0; i < n; ++i) {
            int pos = sorted_a[i].second;
            pst.update(pos, 1);
        }
    }

    // Find k-th smallest (1-indexed) in range [l, r]
    int kth_smallest(int l, int r, int k) {
        return query_kth(pst.roots[0], pst.roots[r - l + 1], 0, n - 1, k);
    }

    int query_kth(int node_l, int node_r, int l, int r, int k) {
        if (l == r) return l;

        int mid = (l + r) / 2;
        int left_count = pst.tree[pst.tree[node_r].left].val -
                         pst.tree[pst.tree[node_l].left].val;

        if (k <= left_count) {
            return query_kth(pst.tree[node_l].left, pst.tree[node_r].left, l,
                mid, k);
        } else {
            return query_kth(pst.tree[node_l].right, pst.tree[node_r].right,
                             mid + 1, r, k - left_count);
        }
    }
};
```

16 Simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD &A, const VD &b, const VD &c) :
        m(b.size()), n(c.size()), B(m), N(n + 1), D(m + 2, VD(n + 2)) {
            for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A
                [i][j];
            for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n +
                1] = b[i]; }
            for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m + 1][n] = 1;
        }

    void Pivot(int r, int s) {
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] / D[r][s];
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] /= D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] /= -D[r][s];
        D[r][s] = 1.0 / D[r][s];
        swap(B[r], N[s]);
    }

    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || (D[x][j] == D[x][s] && N[j]
                     < N[s])) s = j;
            }
            if (D[x][s] > -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                     ((D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i
                    ] < B[r])) r = i;
            }
            if (r == -1) return false;
            Pivot(r, s);
        }
    }

    DOUBLE Solve(VD &x) {
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -EPS) {
            Pivot(r, n);
            if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<
                DOUBLE>::infinity();
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                    if (s == -1 || D[i][j] < D[i][s] || (D[i][j] == D[i][s] &&
                        N[j] < N[s])) s = j;
                Pivot(i, s);
            }
        }
        if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
        x = VD(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};

int main() {

    const int m = 4;
    const int n = 3;
    DOUBLE _A[m][n] = {
        { 6, -1, 0 },
        { -1, -5, 0 },
        { 1, 5, 1 },
        { -1, -5, -1 }
    };
    DOUBLE _b[m] = { 10, -4, 5, -5 };
    DOUBLE _c[n] = { 1, -1, 0 };

    VVD A(m);
```

```
    VD b(_b, _b + m);
    VD c(_c, _c + n);
    for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

    LPSolver solver(A, b, c);
    VD x;
    DOUBLE value = solver.Solve(x);

    cerr << "VALUE: " << value << endl; // VALUE: 1.29032
    cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
    for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
    cerr << endl;
    return 0;
}
```

17 SparseTable2D

```
// 2D Sparse Table for Range Minimum/Maximum Query
// Preprocessing: O(N*M*log(N)*log(M))
// Query: O(1)

template<typename T>
struct SparseTable2D {
    vector<vector<vector<vector<T>>>> st;
    vector<int> log_table;
    int n, m;

    // Function to combine two values (min, max, gcd, etc.)
    T combine(T a, T b) {
        return min(a, b); // Change to max(a, b) for RMQ
    }

    void build_log(int maxn) {
        log_table.resize(maxn + 1);
        log_table[1] = 0;
        for (int i = 2; i <= maxn; ++i) {
            log_table[i] = log_table[i / 2] + 1;
        }
    }

    SparseTable2D(vector<vector<T>> &a) {
        n = a.size();
        m = a[0].size();

        build_log(max(n, m));

        int log_n = log_table[n] + 1;
        int log_m = log_table[m] + 1;

        st.assign(log_n, vector<vector<vector<T>>>(log_m,
                  vector<vector<T>>(n, vector<T>(m))));

        // Copy original array
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                st[0][0][i][j] = a[i][j];
            }
        }

        // Build for rows (k1 varies, k2 = 0)
        for (int k1 = 1; k1 < log_n; ++k1) {
            for (int i = 0; i + (1 << k1) <= n; ++i) {
                for (int j = 0; j < m; ++j) {
                    st[k1][0][i][j] = combine(st[k1-1][0][i][j],
                                              st[k1-1][0][i + (1 << (k1-1))][j
                                                  ]);
                }
            }
        }

        // Build for columns (k1 varies, k2 varies)
        for (int k1 = 0; k1 < log_n; ++k1) {
            for (int k2 = 1; k2 < log_m; ++k2) {
                for (int i = 0; i + (1 << k1) <= n; ++i) {
                    for (int j = 0; j + (1 << k2) <= m; ++j) {
                        st[k1][k2][i][j] = combine(st[k1][k2-1][i][j],
                                                   st[k1][k2-1][i][j + (1 << (
                                                       k2-1))]);
                    }
                }
            }
        }
    }

    // Query rectangle [r1, r2] x [c1, c2] (0-indexed, inclusive)
    T query(int r1, int c1, int r2, int c2) {
        int k1 = log_table[r2 - r1 + 1];
        int k2 = log_table[c2 - c1 + 1];

        int len1 = 1 << k1;
        int len2 = 1 << k2;

        T res = st[k1][k2][r1][c1];
        res = combine(res, st[k1][k2][r2 - len1 + 1][c1]);
        res = combine(res, st[k1][k2][r1][c2 - len2 + 1]);
        res = combine(res, st[k1][k2][r2 - len1 + 1][c2 - len2 + 1]);

        return res;
    }
};

// Example usage:
// vector<vector<int>> a(n, vector<int>(m));
// // Fill array a
// SparseTable2D<int> st(a);
// int result = st.query(r1, c1, r2, c2);
```

18 SplayTree

```
struct Node {
    Node * child[2], * parent;
    bool reverse;
    int value, size;
    long long sum;
};

Node * nil, * root;

void initTree() {
    nil = new Node();
    nil->child[0] = nil->child[1] = nil->parent = nil;
    nil->value = nil->size = nil->sum = 0;
    nil->reverse = false;
    root = nil;
}

void pushDown(Node * x) {
    if(x == nil) return;
    if(x->reverse) {
        swap(x->child[0], x->child[1]);
        x->child[0]->reverse = !x->child[0]->reverse;
        x->child[1]->reverse = !x->child[1]->reverse;
        x->reverse = false;
    }
}

void update(Node * x) {
    pushDown(x->child[0]); pushDown(x->child[1]);
    x->size = x->child[0]->size + x->child[1]->size + 1;
    x->sum = x->child[0]->sum + x->child[1]->sum + x->value;
}

void setLink(Node * x, Node * y, int d) {
    x->child[d] = y;
    y->parent = x;
}

int getDir(Node * x, Node * y) {
    return x->child[0] == y ? 0 : 1;
}

void rotate(Node * x, int d) {
    Node * y = x->child[d], * z = x->parent;
    setLink(x, y->child[d ^ 1], d);
    setLink(y, x, d ^ 1);
    setLink(z, y, getDir(z, x));
    update(x); update(y);
}

void splay(Node * x) {
    while(x->parent != nil) {
        Node * y = x->parent, * z = y->parent;
        int dy = getDir(y, x), dz = getDir(z, y);
        if(z == nil) rotate(y, dy);
        else if(dy == dz) rotate(z, dz), rotate(y, dy);
        else rotate(y, dy), rotate(z, dz);
    }
}

Node * nodeAt(Node * x, int pos) {
    while(pushDown(x), x->child[0]->size != pos)
        if(pos < x->child[0]->size) x = x->child[0];
        else pos -= x->child[0]->size + 1, x = x->child[1];
    return splay(x), x;
}

void split(Node * x, int left, Node * &t1, Node * &t2) {
    if(left == 0) t1 = nil, t2 = x;
    else {
        t1 = nodeAt(x, left - 1);
        t2 = t1->child[1];
        t1->child[1] = t2->parent = nil;
        update(t1);
    }
}

Node * join(Node * x, Node * y) {
    if(x == nil) return y;
    x = nodeAt(x, x->size - 1);
    setLink(x, y, 1);
    update(x);
    return x;
}
```

19 SqrtMod

```
// Jacobi Symbol (m/n), m, n0  and n is odd
// (m/n) ==1 x^2 == m (mod n) solvable, -1 unsolvable
#define NEGPOW(e) ((e) % 2 ? -1 : 1)
int jacobi(int a, int m) {
   if (a == 0) return m == 1 ? 1 : 0;
   if (a % 2)  return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
   else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}
// No solution when: n(p-1)/2 = -1 mod p
int sqrtMod(int n, int p) { //find x: x2 = n (mod p) p is prime
   int S, Q, W, i, m = invMod(n, p);
   for (Q = p - 1, S = 0; Q % 2 == 0; Q /= 2, ++S);
   do { W = rand() % p; } while (W == 0 || jacobi(W, p) != -1);
   for (int R = powMod(n, (Q+1)/2, p), V = powMod(W, Q, p); ;) {
      int z = R * R * m % p;
      for (i = 0; i < S && z % p != 1; z *= z, ++i);
      if (i == 0) return R;
      R = (R * powMod(V, 1 << (S-i-1), p)) % p;
   }
```

```
}
```

20 SuffixArrayDC3

```
#include <bits/stdc++.h>
#define FOR(i,a,b) for (int i=(a),_b=(b);i<=_b;i=i+1)
#define REP(i,n) for (int i=0,_n=(n);i<_n;i=i+1)
#define MASK(i) (1LL<<(i))
#define BIT(x,i) (((x)>>(i))&1)
#define tget(i) BIT(t[(i) >> 3], (i) & 7)
#define tset(i, b) { if (b) t[(i) >> 3] |= MASK((i) & 7); else t[(i) >> 3] &= ~
    MASK((i) & 7); }
#define chr(i) (cs == sizeof(int) ? ((int *)s)[i] : ((unc *)s)[i])
#define isLMS(i) ((i) > 0 && tget(i) && !tget((i) - 1))

typedef unsigned char unc;
class SuffixArray {
    public:
    int *sa, *lcp, *rank, n;
    unc *s;
    void getbuckets(unc s[], vector<int> &bkt, int n, int k, int cs, bool end)
        {
        FOR(i, 0, k) bkt[i] = 0;
        REP(i, n) bkt[chr(i)]++;
        int sum = 0;
        FOR(i, 0, k) {
            sum += bkt[i];
            bkt[i] = end ? sum : sum - bkt[i];
        }
    }
    void inducesal(vector<unc> &t, int sa[], unc s[], vector<int> &bkt, int n,
        int k, int cs, bool end) {
        getbuckets(s, bkt, n, k, cs, end);
        REP(i, n) {
            int j = sa[i] - 1;
            if (j >= 0 && !tget(j)) sa[bkt[chr(j)]++] = j;
        }
    }
    void inducesas(vector<unc> &t, int sa[], unc s[], vector<int> &bkt, int n,
        int k, int cs, bool end) {
        getbuckets(s, bkt, n, k, cs, end);
        FORD(i, n - 1, 0) {
            int j = sa[i] - 1;
            if (j >= 0 && tget(j)) sa[--bkt[chr(j)]]=j;
        }
    }
    void build(unc s[], int sa[], int n, int k, int cs) {
        int j;
        vector<unc> t = vector<unc>(n / 8 + 1, 0);
        tset(n - 2, 0);
        tset(n - 1, 1);
        FORD(i, n - 3, 0) tset(i, chr(i) < chr(i+1) || (chr(i) == chr(i+1) &&
            tget(i+1)));
        vector<int> bkt = vector<int> (k + 1, 0);
        getbuckets(s, bkt, n, k, cs, true);
        REP(i, n) sa[i] = -1;
        REP(i, n) if (isLMS(i)) sa[--bkt[chr(i)]] = i;
        inducesal(t, sa, s, bkt, n, k, cs, false);
        inducesas(t, sa, s, bkt, n, k, cs, true);
        bkt.clear();
        int n1 = 0;
        REP(i, n) if (isLMS(sa[i])) sa[n1++] = sa[i];
        FOR(i, n1, n - 1) sa[i] = -1;
        int name = 0;
        int prev = -1;
        REP(i, n1) {
            int pos = sa[i];
            bool diff = false;
            REP(d, n) {
                if (prev < 0 || chr(prev + d) != chr(pos + d) || tget(prev + d)
                     != tget(pos + d)) {
                    diff = true;
                    break;
                }
                else if (d > 0 && (isLMS(prev + d) || isLMS(pos + d))) break;
            }
            if (diff) {
                name++;
                prev = pos;
            }
            sa[n1 + pos / 2] = name - 1;
        }
        j = n - 1;
        FORD(i, n - 1, n1) if (sa[i] >= 0) sa[j--] = sa[i];
        int *sa1 = sa;
        int *s1 = sa + n - n1;
        if (name < n1) build((unc *)s1, sa1, n1, name-1, sizeof(int));
        else REP(i, n1) sa1[s1[i]] = i;
        bkt.assign(k + 1, 0);
        getbuckets(s, bkt, n, k, cs, true);
        j = 0;
        REP(i, n) if (isLMS(i)) s1[j++] = i;
        REP(i, n1) sa1[i] = s1[sa1[i]];
        FOR(i, n1, n - 1) sa[i] = -1;
        FORD(i, n1 - 1, 0) {
            j = sa[i];
            sa[i] = -1;
            sa[--bkt[chr(j)]] = j;
        }
        inducesal(t, sa, s, bkt, n, k, cs, false);
        inducesas(t, sa, s, bkt, n, k, cs, true);
        bkt.clear();
        t.clear();
    }
    void calc_lcp(void) {
        FOR(i,1,n) rank[sa[i]] = i;
        int h = 0;
        REP(i, n) if (rank[i] < n) {
            int j = sa[rank[i] + 1];
```

```
            while (s[i + h] == s[j + h]) h++;
            lcp[rank[i]] = h;
            if (h > 0) h--;
        }
    }
    SuffixArray() {
        n = 0;
        sa = lcp = rank = NULL;
        s=NULL;
    }
    SuffixArray(string ss) {
        n = ss.size();
        sa = new int[n + 7];
        lcp = new int [n + 7];
        rank = new int [n + 7];
        s = (unc *)ss.c_str();
        build(s, sa, n + 1, 256, sizeof(char));
        calc_lcp();
    }
};

//Sorted suffices are SA[1] to SA[N]. The values of SA[1], SA[2], ..., SA[N]
    are 0, 1, ..., N - 1
//The longest common prefix of SA[i] and SA[i + 1] is LCP[i]

int main(void) {
    string s = "mississippi";
    SuffixArray suffixArray(s);
    FOR(i, 1, 11) printf("%d %s %d\n", suffixArray.sa[i], s.substr(suffixArray.
        sa[i]).c_str(), suffixArray.lcp[i]);
}
```

21 SuffixArrayPrefixDoubling

```
struct SuffixArray {
    string a;
    int N, m;
    vector<int> SA, LCP, x, y, w, c;

    SuffixArray(string _a, int m) : a(" " + _a), N(a.length()), m(m),
            SA(N), LCP(N), x(N), y(N), w(max(m, N)), c(N) {
        a[0] = 0;
        DA();
        kasaiLCP();
        #define REF(X) { rotate(X.begin(), X.begin()+1, X.end()); X.pop_back();
             }
        REF(SA); REF(LCP);
        a = a.substr(1, a.size());
        for(int i = 0; i < (int) SA.size(); ++i) --SA[i];
        #undef REF
    }

    inline bool cmp (const int a, const int b, const int l) { return (y[a] == y
        [b] && y[a + l] == y[b + l]); }

    void Sort() {
        for(int i = 0; i < m; ++i) w[i] = 0;
        for(int i = 0; i < N; ++i) ++w[x[y[i]]];
        for(int i = 0; i < m - 1; ++i) w[i + 1] += w[i];
        for(int i = N - 1; i >= 0; --i) SA[--w[x[y[i]]]] = y[i];
    }

    void DA() {
        for(int i = 0; i < N; ++i) x[i] = a[i], y[i] = i;
        Sort();
        for(int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
            for(p = 0, i = N - j; i < N; i++) y[p++] = i;
            for (int k = 0; k < N; ++k) if (SA[k] >= j) y[p++] = SA[k] - j;
            Sort();
            for(swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i < N; ++i)
                x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1 : p++;
        }
    }

    void kasaiLCP() {
        for (int i = 0; i < N; i++) c[SA[i]] = i;
        for (int i = 0, j, k = 0; i < N; LCP[c[i++]] = k)
            if (c[i] > 0) for (k ? k-- : 0, j = SA[c[i] - 1]; a[i + k] == a[j +
                 k]; k++);
            else k = 0;
    }
};

int main() {
    SuffixArray sa ("mississippi", 256);
    for (int i = 0; i < sa.N - 1; ++i) cout << sa.SA[i] << ' '; cout << '\n';
    for (int i = 0; i < sa.N - 1; ++i) cout << sa.LCP[i] << ' '; cout << '\n';
    // 10 7 4 1 0 9 8 6 3 5 2
    // 0 1 1 4 0 0 1 0 2 1 3
    return 0;
}
```

22 SuffixAutomaton

```
const int MAX_N = int(1e5) + 4;
const int MAX_SAM = 2 * MAX_N;
// a node in the "directed acyclic word graph" (or simply "DAWG")
struct State {
    // len: length of the path from root to this node (number of edges)
    // link: link to a node which is a suffix of this state
    // nexts: the node which is adjanced with this node by an edge ('a'..'z',
        '0'..'9', ...)
    int len, link;
    map<int, int> nexts;
    State() {
        len = 0;
        link = -1;
        nexts.clear();
```

```
    }
    void operator = (const State &other) {
        len = other.len;
        link = other.link;
        nexts = other.nexts;
    }
    bool hasNext(int x) {
        return nexts.find(x) != nexts.end();
    }
};
void sam_init() {
    nSAM = 1;       // number of nodes
    last = 0;       // id of the last node (start from 0)
    sam[0] = State();   // this is the root node
    f[0] = 0;           // for some applications
}
void sam_extend(int x) {
    int cur = nSAM++;       // id of new node
    sam[cur] = State();
    sam[cur].len = sam[last].len + 1;
    f[cur] = 1;
    int p = last;
    for (; p != -1 && !sam[p].hasNext(x); p = sam[p].link)
        sam[p].nexts[x] = cur;
    if (p == -1) sam[cur].link = 0;
    else {
        int q = sam[p].nexts[x];
        if (sam[q].len == sam[p].len + 1) sam[cur].link = q;
        else {
            int clone = nSAM++;     // create a clone node of q
            sam[clone] = sam[q];
            sam[clone].len = sam[p].len + 1;
            f[clone] = 0;
            for (; p != -1 && sam[p].nexts[x] == q; p = sam[p].link)
                sam[p].nexts[x] = clone;
            sam[cur].link = sam[q].link = clone;
        }
    }
    last = cur;
}
// // APLICATIONS
// // we should do the topo sort (by length) before implement other features
// for (int i = 0; i <= n; ++i) c[i] = 0;
// for (int i = 0; i < nSAM; ++i) ++c[sam[i].len];
// for (int i = 1; i <= n; ++i) c[i] += c[i-1];
// for (int i = 0; i < nSAM; ++i) id[--c[sam[i].len]] = i;
// // number of occurrents of state u, which corresponding with
// // number of occurrents of each substrings from root to u
// for (int i = nSAM-1; i >= 0; --i) {
//  int u = id[i];
//  f[sam[u].link] += f[u];
// }
// // number of ways to go from root to state u, which corresponding with
// // number of different substrings end at u.
// // If we sort these strings increasing by their lengths,
// // then the i-th string is a suffix of the (i+1)-th string.
// fill(g, 0);
// g[0] = 1;
// for (int i = 0; i < nSAM; ++i) {
//  int u = id[i];
//  tr(sam[u].nexts, it)
//      g[it->second] += g[u];
// }
// // number of substrings which have state u as its prefix
// for (int i = nSAM-1; i >= 0; --i) {
//  int u = id[i];
//  f[u] = 1;
//  tr(sam[u].nexts, it)
//      f[u] += [it->second];
// }
```

23 Trie

```
// Trie (Prefix Tree) implementation
// Supports insert, search, and prefix matching
// Time: O(L) per operation where L is string length
// Space: O(ALPHABET_SIZE * N * L) where N is number of strings

const int ALPHABET_SIZE = 26;

struct TrieNode {
    TrieNode* children[ALPHABET_SIZE];
    bool isEndOfWord;
    int count; // Number of words ending at this node
    int prefixCount; // Number of words with this prefix

    TrieNode() {
        isEndOfWord = false;
        count = 0;
        prefixCount = 0;
        for (int i = 0; i < ALPHABET_SIZE; i++) {
            children[i] = nullptr;
        }
    }
};

struct Trie {
    TrieNode* root;

    Trie() {
        root = new TrieNode();
    }

    // Insert a word into the trie
    void insert(const string& word) {
        TrieNode* node = root;
        for (char c : word) {
            int index = c - 'a'; // Change to c - 'A' for uppercase
            if (node->children[index] == nullptr) {
```

```
            node->children[index] = new TrieNode();
        }
        node = node->children[index];
        node->prefixCount++;
    }
    node->isEndOfWord = true;
    node->count++;
}

// Search for exact word
bool search(const string& word) {
    TrieNode* node = root;
    for (char c : word) {
        int index = c - 'a';
        if (node->children[index] == nullptr) {
            return false;
        }
        node = node->children[index];
    }
    return node != nullptr && node->isEndOfWord;
}

// Check if any word starts with given prefix
bool startsWith(const string& prefix) {
    TrieNode* node = root;
    for (char c : prefix) {
        int index = c - 'a';
        if (node->children[index] == nullptr) {
            return false;
        }
        node = node->children[index];
    }
    return true;
}

// Count words with given prefix
int countWordsWithPrefix(const string& prefix) {
    TrieNode* node = root;
    for (char c : prefix) {
        int index = c - 'a';
        if (node->children[index] == nullptr) {
            return 0;
        }
        node = node->children[index];
    }
    return node->prefixCount;
}

// Delete a word from trie
bool deleteWord(const string& word) {
    return deleteHelper(root, word, 0);
}

bool deleteHelper(TrieNode* node, const string& word, int depth) {
    if (node == nullptr) return false;

    if (depth == word.length()) {
        if (!node->isEndOfWord) return false;
        node->isEndOfWord = false;
        node->count--;
        return isEmpty(node);
    }

    int index = word[depth] - 'a';
    if (deleteHelper(node->children[index], word, depth + 1)) {
        delete node->children[index];
        node->children[index] = nullptr;
        node->prefixCount--;
        return !node->isEndOfWord && isEmpty(node);
    }

    return false;
}

bool isEmpty(TrieNode* node) {
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        if (node->children[i] != nullptr) {
            return false;
        }
    }
    return true;
}

// Get all words with given prefix
void getAllWordsWithPrefix(const string& prefix, vector<string>& result) {
    TrieNode* node = root;
    for (char c : prefix) {
        int index = c - 'a';
        if (node->children[index] == nullptr) {
            return;
        }
        node = node->children[index];
    }
    getAllWordsHelper(node, prefix, result);
}

void getAllWordsHelper(TrieNode* node, string current, vector<string>&
    result) {
    if (node->isEndOfWord) {
        result.push_back(current);
    }
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        if (node->children[i] != nullptr) {
            getAllWordsHelper(node->children[i], current + char('a' + i),
                result);
        }
    }
}

// Find longest common prefix
```

```
    string longestCommonPrefix() {
        string prefix = "";
        TrieNode* node = root;

        while (node != nullptr && !node->isEndOfWord && countChildren(node) ==
            1) {
            for (int i = 0; i < ALPHABET_SIZE; i++) {
                if (node->children[i] != nullptr) {
                    prefix += char('a' + i);
                    node = node->children[i];
                    break;
                }
            }
        }
        return prefix;
    }

    int countChildren(TrieNode* node) {
        int count = 0;
        for (int i = 0; i < ALPHABET_SIZE; i++) {
            if (node->children[i] != nullptr) {
                count++;
            }
        }
        return count;
    }
};

// Alternative: Trie using map for dynamic alphabet
struct TrieNodeMap {
    map<char, TrieNodeMap*> children;
    bool isEndOfWord;
    int count;

    TrieNodeMap() : isEndOfWord(false), count(0) {}
};

struct TrieMap {
    TrieNodeMap* root;

    TrieMap() {
        root = new TrieNodeMap();
    }

    void insert(const string& word) {
        TrieNodeMap* node = root;
        for (char c : word) {
            if (node->children.find(c) == node->children.end()) {
                node->children[c] = new TrieNodeMap();
            }
            node = node->children[c];
        }
        node->isEndOfWord = true;
        node->count++;
    }

    bool search(const string& word) {
        TrieNodeMap* node = root;
        for (char c : word) {
            if (node->children.find(c) == node->children.end()) {
                return false;
            }
            node = node->children[c];
        }
        return node != nullptr && node->isEndOfWord;
    }

    bool startsWith(const string& prefix) {
        TrieNodeMap* node = root;
        for (char c : prefix) {
            if (node->children.find(c) == node->children.end()) {
                return false;
            }
            node = node->children[c];
        }
        return true;
    }
};

// Example usage:
// Trie trie;
// trie.insert("hello");
// trie.insert("world");
// bool found = trie.search("hello"); // true
// bool hasPrefix = trie.startsWith("hel"); // true
// int count = trie.countWordsWithPrefix("hel"); // 1
```

24 ZFunction

```
vector<int> calcZ(const string &s) {
    int n = s.size();
    vector<int> z (n);
    for (int i = 1, j = 0; i < n; ++i) {
        if (j + z[j] > i) z[i] = min(j + z[j] - i, z[i - j]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (j + z[j] <= i || i + z[i] > j + z[j]) j = i;
    }
    return z;
}
```

25 kdTree

```
// ------------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
```

```
//  - handles nearest-neighbor query in O(lg n) if points are well
//    distributed
//  - worst case for nearest-neighbor may be linear in pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
// ------------------------------------------------------------------

#include <bits/stdc++.h>
using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b) {
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b) {
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b) {
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b) {
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox {
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < (int) v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode {
    bool leaf;      // true if this is a leaf node (has one point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp) {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        } else {
            // split on x if the bbox is wider than high (not best heuristic
                ...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
```

```
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree {
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p) {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//            if (p == node->pt) return sentry;
//            else
              return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        } else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};
```

```
// --------------------------------------------------------------------------
// some basic test code here

int main() {
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// --------------------------------------------------------------------------
```

$$\pi(x) = \lfloor x \rfloor - \sum_{i=1}^{a} \left\lfloor \frac{x}{p_i} \right\rfloor + \sum_{1 \le i \le j \le a} \left\lfloor \frac{x}{p_i p_j} \right\rfloor - \ldots + \frac{1}{2}(b+a-2)(b-a+1) - \sum_{a < i \le b} \pi\left(\frac{x}{p_i}\right) - \sum_{i=a+1}^{c} \sum_{j=i}^{b_i} \left[\pi\left(\frac{x}{p_i p_j}\right) - (j-1)\right], a = \pi\left(x^{1/4}\right), b = \pi\left(x^{1/2}\right), b_i = \pi\left(\sqrt{x/p_i}\right), c = \pi\left(x^{1/3}\right)$$

$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} = \frac{2(2n+1)}{n+2} C_n$

$C = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440$

Number of permutations of length n with k cycles:

$$s(n+1, k) = ns(n, k) + s(n, k-1)$$

Number of ways to partition a set of n labelled objects into k nonempty subsets:

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n = kS(n-1, k) + S(n, k-1)$$

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \frac{1}{252n^6} + \ldots$$

$\frac{1}{2(n+1)} < H_n - \ln n - \gamma < \frac{1}{2n}; \frac{1}{24(n+1)^2} < H_n - \ln\left(n + \frac{1}{2}\right) - \gamma < \frac{1}{24n^2}$

$\gamma = 0.57721566490153286060651209008240243104215933593992$

Sphere: $V = \frac{4}{3}\pi r^3; A = 4\pi r^2$

$V = \frac{\pi h}{6}\left(3a^2 + h^2\right); A = 2\pi r h = 2\pi r^2 (1 - \cos\theta) = \pi\left(a^2 + h^2\right); r = \frac{a^2 + h^2}{2h}$

SphericalCap.png

Maximum Flows with Edge Demands: $c'(s' \to v) = \sum_{u \in V} d(u \to v)$, $c'(v \to t') = \sum_{w \in V} d(v \to w)$, $c'(u \to v) = c(u \to v) - d(u \to v)$, $c'(t \to s) = \infty$. If feasible: $c_f(u \to v) = c(u \to v) - f(u \to v)$ if $u \to v \in E$; $f(v \to u) - d(v \to u)$ if $v \to u \in E$, 0 otherwise.