

1 General

2 Algorithms

3 Data Structures

4 String

5 Math

6 Graph

7 2D Geometry

8 3D Geometry

9 Optimization

1 General

run.sh

```
g++ -g -O2 -std=gnu++17 -static prog.cpp
./a.exe
```

test.sh

```
# compile and test all *.in and *.ans
g++ -g -O2 -std=gnu++17 -static prog.cpp
for i in *.in; do
f=${i%.in}
./a.exe < $i > "$f.out"
diff -b -q "$f.ans" "$f.out"
done
```

Header

```
// use better compiler options
#pragma GCC optimize("Ofast", "unroll-loops")
#pragma GCC target("avx2, fma")
// include everything
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <sys/resource.h>
using namespace std;
using namespace __gnu_cxx; // rope
using namespace __gnu_pbds; // tree/trie
// common defines
#define fastio
↳ ios_base::sync_with_stdio(0); cin.tie(0);
#define nostacklim rlimit
RZ; getrlimit(3, &RZ); RZ.rlim_cur =
↳ 1; setrlimit(3, &RZ);
#define DEBUG(v) cout<<"DEBUG: "<<#v<<" =
↳ "<<v<<"\n";
#define ll long long
#define ull unsigned ll
#define i128 __int128
#define u128 unsigned i128
#define ld long double
// global variables
mt19937 rng((uint32_t) chrono::steady
↳ _clock::now().time_since_epoch().count());
```

Fast IO

```
void readn(unsigned int& n) {
char c; n = 0;
while ((c = getchar_unlocked()) != ' ' && c != '\n')
n = n * 10 + c - '0';
}
void readn(int& n) {
char c; n = 0; int s = 1;
if ((c = getchar_unlocked()) == '-') s = -1;
else n = c - '0';
while ((c = getchar_unlocked()) != ' ' && c != '\n')
n = n * 10 + c - '0';
n *= s;
}
void readn(ld& n) {
char c; n = 0;
ld m = 0, o = 1; bool d = false; int s = 1;
if ((c = getchar_unlocked()) == '-') s = -1;
else if (c == '.') d = true;
else n = c - '0';
while ((c = getchar_unlocked()) != ' ' && c != '\n')
{
if (c == '.') d = true;
else if (d) { m = m * 10 + c - '0'; o *= 0.1; }
}
```

```
else n = n * 10 + c - '0';
}
n = s * (n + m * o);
}
void readn(double& n) {
ld m; readn(m); n = m;
}
void readn(float& n) {
ld m; readn(m); n = m;
}
void readn(string& s) {
char c; s = "";
while ((c = getchar_unlocked()) != ' ' && c != '\n')
s += c;
}
bool readline(string& s) {
char c; s = "";
while ((c = getchar_unlocked()) {
if (c == '\n') return true;
if (c == EOF) return false;
s += c;
}
return false;
}
void printn(unsigned int n) {
if (n / 10) printn(n / 10);
putchar_unlocked(n % 10 + '0');
}
void printn(int n) {
if (n < 0) { putchar_unlocked('-'); n = -n; }
printn((unsigned int) n);
}
```

2 Algorithms

Min/Max Subarray

```
// max - compare = a < b, reset = a < 0
// min - compare = a > b, reset = a > 0
// returns {sum, {start, end}}
pair<int, pair<int, int>
ContiguousSubarray(int* a, int size,
bool(*compare)(int, int),
bool(*reset)(int), int defbest = 0) {
int best = defbest, cur = 0, start = 0, end =
0, s = 0;
for (int i = 0; i < size; i++) {
cur += a[i];
if ((*compare)(best, cur)) { best = cur;
start = s; end = i; }
if ((*reset)(cur)) { cur = 0; s = i + 1; }
}
return {best, {start, end}};
}
```

Quickselect

```
#define QSNE -999999
int partition(int arr[], int l, int r)
{
int x = arr[r], i = l;
for (int j = l; j <= r - 1; j++)
if (arr[j] <= x)
swap(arr[j+1], arr[j]);
swap(arr[i], arr[r]);
return i;
}
// find k'th smallest element in unsorted
array, only if all distinct
int qselect(int arr[], int l, int r, int k)
{
if (!(k > 0 && k <= r - l + 1)) return QSNE;
swap(arr[l + rng() % (r - l + 1)], arr[r]);
int pos = partition(arr, l, r);
if (pos - l == k - 1) return arr[pos];
if (pos - l > k - 1) return qselect(arr, l, pos - 1, k);
return qselect(arr, pos + 1, r, k - pos + 1 - 1);
}
// TODO: compare against std::nth_element()
Saddleback Search
// search for v in 2d array arr[x][y], sorted
on both axis
pair<int, int> saddleback_search(int** arr,
int x, int y, int v) {
```

```
int i = x - 1, j = 0;
while (i >= 0 && j < y) {
if (arr[i][j] == v) return {i, j};
(arr[i][j] > v)? i--: j++;
}
return {-1, -1};
}
```

Ternary Search

```
// < max, > min, or any other unimodal func
#define TERNCOMP(a,b) (a)<(b)
int ternsearch(int a, int b, int (*f)(int)) {
while (b - a > 4) {
int m = (a + b) / 2;
if (TERNCOMP((*f)(m), (*f)(m + 1))) a = m;
else b = m + 1;
}
for (int i = a + 1; i <= b; i++)
if (TERNCOMP((*f)(a), (*f)(i)))
a = i;
return a;
}
#define TERNPREC 0.000001
double ternsearch(double a, double b, double
(*f)(double)) {
while (b - a > TERNPREC * 4) {
double m = (a + b) / 2;
if (TERNCOMP((*f)(m), (*f)(m + TERNPREC))) a
= m;
else b = m + TERNPREC;
}
for (double i = a + TERNPREC; i <= b; i +=
TERNPREC)
if (TERNCOMP((*f)(a), (*f)(i)))
a = i;
return a;
}
```

3 Data Structures

Fenwick Tree

```
// Fenwick tree, array of cumulative sums -
↳ O(log n) updates, O(log n) gets
struct Fenwick {
int n; ll* tree;
void update(int i, int val) {
++i;
while (i <= n) {
tree[i] += val;
i += i & (-i);
}
}
Fenwick(int size) {
n = size;
tree = new ll[n + 1];
for (int i = 1; i <= n; i++)
tree[i] = 0;
}
Fenwick(int* arr, int size) : Fenwick(size) {
for (int i = 0; i < n; i++)
update(i, arr[i]);
}
Fenwick() { delete[] tree; }
ll operator[] (int i) {
if (i < 0 || i > n) return 0;
ll sum = 0;
++i;
while (i > 0) {
sum += tree[i];
i = i & (-i);
}
return sum;
}
ll getRange(int a, int b) { return
operator[] (b) - operator[] (a - 1); }
};
```

Hashtable

```
// similar to unordered_map, but faster
struct chash {
const uint64_t C = (1ll)(2e18 * M_PI) + 71;
ll operator()(ll x) const { return
↳ __builtin_bswap64(x * C); }
};
```

```
int main() {
gp_hash_table<ll, int, chash>
↳ hashtable({}, {}, {}, {}, {1<16});
for (int i = 0; i < 100; i++)
hashtable[i] = 200 + i;
if (hashtable.find(10) != hashtable.end())
cout << hashtable[10];
}
```

Ordered Set

```
typedef tree<int, null_type, less<int>, rb_tree
↳ _tag, tree_order_statistics_node_update>
ordered_set;
int main()
{
ordered_set o_set;
o_set.insert(5); o_set.insert(1);
↳ o_set.insert(3);
// get second smallest element
cout << *(o_set.find_by_order(1)) << '\n';
// number of elements less than k=4
cout << o_set.order_of_key(4) << '\n';
}
```

Rope

```
// O(log n) insert, delete, concatenate
int main() {
// generate rope
rope<int> v;
for (int i = 0; i < 100; i++)
v.push_back(i);
// move range to front
rope<int> copy = v.substr(10, 10);
v.erase(10, 10);
v.insert(copy.mutable_begin(), copy);
// print elements of rope
for (auto it : v)
cout << it << " ";
}
```

Segment Tree

```
// max(a,b), min(a,b), a+b, a*b, gcd(a,b), a^b
struct SegmentTree {
typedef int T;
static constexpr T UNIT = INT_MIN;
T f(T a, T b) {
if (a == UNIT) return b;
if (b == UNIT) return a;
return max(a, b);
}
int n; vector<T> s;
SegmentTree(int n, T def=UNIT) : s(2*n, def),
↳ n(n) {}
SegmentTree(vector<T> arr) :
↳ SegmentTree(arr.size()) {
for (int i = 0; i < arr.size(); i++)
update(i, arr[i]);
}
void update(int pos, T val) {
for (s[pos] += val; pos /= 2;)
s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
}
T query(int b, int e) { // query [b, e)
T ra = UNIT, rb = UNIT;
for (b += n, e += n; b < e; b /= 2, e /= 2) {
if (b % 2) ra = f(ra, s[b++]);
if (e % 2) rb = f(s[e--], rb);
}
return f(ra, rb);
}
T get(int p) { return query(p, p + 1); }
};
```

```

Trie
typedef trie<string, null_type,
↳ trie_string_access_traits<,
↳ pat_trie_tag,
↳ trie_prefix_search_node_update> trie_type;
int main() {
    // generate trie
    trie_type trie;
    for (int i = 0; i < 20; i++)
        trie.insert(to_string(i)); // true if new,
↳ false if old
    // print things with prefix "1"
    auto range = trie.prefix_range("1");
    for (auto it = range.first; it !=
↳ range.second; it++)
        cout << *it << " ";
}

```

4 String

Aho Corasick

```

// range of alphabet for automata to consider
// MAXC = 26, OFFC = 'a' if only lowercase
const int MAXC = 256;
const int OFFC = 0;
struct aho_corasick {
    struct state
    {
        set<pair<int, int>> out;
        int fail; vector<int> go;
        state() : fail(-1), go(MAXC, -1) {}
    };
    vector<state> s;
    int id = 0;
    aho_corasick(string* arr, int size) : s(1) {
        for (int i = 0; i < size; i++) {
            int cur = 0;
            for (int c : arr[i]) {
                if (s[cur].go[c-OFFC] == -1) {
                    s[cur].go[c-OFFC] = s.size();
                    s.push_back(state());
                }
                cur = s[cur].go[c-OFFC];
            }
            s[cur].out.insert({arr[i].size(), id++});
        }
        for (int c = 0; c < MAXC; c++)
            if (s[0].go[c] == -1)
                s[0].go[c] = 0;
        queue<int> sq;
        for (int c = 0; c < MAXC; c++) {
            if (s[0].go[c] != 0) {
                s[s[0].go[c]].fail = 0;
                sq.push(s[0].go[c]);
            }
        }
        while (sq.size()) {
            int e = sq.front(); sq.pop();
            for (int c = 0; c < MAXC; c++) {
                if (s[e].go[c] != -1) {
                    int failure = s[e].fail;
                    while (s[failure].go[c] == -1)
                        failure = s[failure].fail;
                    failure = s[failure].go[c];
                    s[s[e].go[c]].fail = failure;
                    for (auto length : s[failure].out)
                        s[s[e].go[c]].out.insert(length);
                    sq.push(s[e].go[c]);
                }
            }
        }
    }
    // list of {start pos, pattern id}
    vector<pair<int, int>> search(string text)
    {
        vector<pair<int, int>> toret;
        int cur = 0;
        for (int i = 0; i < text.size(); i++) {
            while (s[cur].go[text[i]-OFFC] == -1)
                cur = s[cur].fail;
            cur = s[cur].go[text[i]-OFFC];
        }
    }
}

```

```

    if (s[cur].out.size())
        for (auto end : s[cur].out)
            toret.push_back({i - end.first + 1,
↳ end.second});
    return toret;
}

```

Boyer Moore

```

struct defint { int i = -1; };
vector<int> boyermoore(string txt, string pat)
{
    vector<int> toret; unordered_map<char,
↳ defint> badchar;
    int m = pat.size(), n = txt.size();
    for (int i = 0; i < m; i++) badchar[pat[i]].i
↳ = i;
    int s = 0;
    while (s <= n - m) {
        int j = m - 1;
        while (j >= 0 && pat[j] == txt[s + j]) j--;
        if (j < 0) {
            toret.push_back(s);
            s += (s + m < n) ? m - badchar[txt[s +
↳ m]].i : 1;
        } else
            s += max(1, j - badchar[txt[s + j]].i);
        return toret;
    }
}

```

English Conversion

```

const string ones[] = {"", "one", "two",
↳ "three", "four", "five", "six", "seven",
↳ "eight", "nine"};
const string teens[] = {"ten", "eleven",
↳ "twelve", "thirteen", "fourteen",
↳ "fifteen", "sixteen", "seventeen",
↳ "eighteen", "nineteen"};
const string tens[] = {"twenty", "thirty",
↳ "forty", "fifty", "sixty", "seventy",
↳ "eighty", "ninety"};
const string mags[] = {"thousand", "million",
↳ "billion", "trillion", "quadrillion",
↳ "quintillion", "sextillion",
↳ "septillion"};
string convert(int num, int carry) {
    if (num < 0) return "negative " +
↳ convert(-num, 0);
    if (num < 10) return ones[num];
    if (num < 20) return teens[num % 10];
    if (num < 100) return tens[(num / 10) - 2] +
↳ (num%10==0?" " : " ") + ones[num % 10];
    if (num < 1000) return ones[num / 100] +
↳ (num/100==0?" " : " ") + "hundred" +
↳ (num%100==0?" " : " ") + convert(num % 100,
↳ 0);
    return convert(num / 1000, carry + 1) + " " +
↳ mags[carry] + " " + convert(num % 1000,
↳ 0);
}
string convert(int num) {
    return (num == 0) ? "zero" : convert(num, 0);
}

```

Knuth Morris Pratt

```

vector<int> kmp(string txt, string pat) {
    vector<int> toret;
    int m = txt.length(), n = pat.length();
    int next[n + 1];
    for (int i = 0; i < n + 1; i++)
        next[i] = 0;
    for (int i = 1; i < n; i++) {
        int j = next[i + 1];
        while (j > 0 && pat[j] != pat[i])
            j = next[j];
        if (j > 0 || pat[j] == pat[i])
            next[i + 1] = j + 1;
    }
}

```

```

}
for (int i = 0, j = 0; i < m; i++) {
    if (txt[i] == pat[j]) {
        if (++j == n)
            toret.push_back(i - j + 1);
        else if (j > 0) {
            j = next[j];
        }
    }
    return toret;
}

```

Longest Common Prefix

```

string lcp(string* arr, int n) {
    if (n == 0) return "";
    sort(arr, arr + n);
    string r = ""; int v = 0;
    while (v < arr[0].length() && arr[0][v] ==
↳ arr[n-1][v])
        r += arr[0][v++];
    return r;
}

```

Longest Common Subsequence

```

string lcs(string a, string b) {
    int m = a.length(), n = b.length();
    int L[m+1][n+1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) L[i][j] = 0;
            else if (a[i-1] == b[j-1]) L[i][j] =
↳ L[i-1][j-1]+1;
            else L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }
    // return L[m][n]; // length of lcs
    string out = "";
    int i = m - 1, j = n - 1;
    while (i >= 0 && j >= 0) {
        if (a[i] == b[j]) {
            out = a[i-] + out;
            j--;
        }
        else if (L[i][j+1] > L[i+1][j]) i--;
        else j--;
    }
    return out;
}

```

Longest Common Substring

```

// l is array of palindrome length at that
↳ index
int manacher(string s, int* l) {
    int n = s.length() * 2;
    for (int i = 0, j = 0, k; i < n; i += k, j =
↳ max(j-k, 0)) {
        while (i >= j && i + j + 1 < n && s[(i-j)/2]
↳ == s[(i+j+1)/2]) j++;
        l[i] = j;
        for (k = 1; i >= k && j >= k && l[i-k] !=
↳ j-k; k++)
            l[i+k] = min(l[i-k], j-k);
    }
    return *max_element(l, l + n);
}

```

Subsequence Count

```

// "banana", "ban" > 3 (ban, ba..n, b..an)
ull subsequences(string body, string subs) {
    int m = subs.length(), n = body.length();
    if (m > n) return 0;
    ull** arr = new ull*[m+1];
    for (int i = 0; i <= m; i++) arr[i] = new
↳ ull[n+1];
    for (int i = 1; i <= m; i++) arr[i][0] = 0;
    for (int i = 0; i <= m; i++) arr[0][i] = 1;
    for (int i = 1; i <= m; i++)
        for (int j = i; j <= n; j++)

```

```

        arr[i][j] = arr[i][j-1] + ((body[j-1] ==
↳ subs[i-1])? arr[i-1][j-1] : 0);
    return arr[m][n];
}

```

5 Math

Catalan Numbers

```

ull* catalan = new ull[1000000];
void genCatalan(int n, int mod) {
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (int j = i - 1; j >= 0; j--) {
            catalan[i] += (catalan[j] * catalan[i-j-1])
↳ % mod;
            if (catalan[i] >= mod)
                catalan[i] -= mod;
        }
    }
}

```

// TODO: consider binomial coefficient method

Combinatorics (nCr, nPr)

```

// can optimize by precomputing factorials, and
↳ fact[n]/fact[n-r]
ull nPr(ull n, ull r) {
    ull v = 1;
    for (ull i = n-r+1; i <= n; i++)
        v *= i;
    return v;
}
ull nPr(ull n, ull r, ull m) {
    ull v = 1;
    for (ull i = n-r+1; i <= n; i++)
        v = (v * i) % m;
    return v;
}
ull nCr(ull n, ull r) {
    long double v = 1;
    for (ull i = 1; i <= r; i++)
        v = v * (n-r+i) / i;
    return (ull)(v + 0.001);
}
// requires modulo math
// can optimize by precomputing mfac and
↳ minv-mfac
ull nCr(ull n, ull r, ull m) {
    return mfac(n, m) * minv(mfac(k, m), m) % m *
↳ minv(mfac(n-k, m), m) % m;
}

```

Chinese Remainder Theorem

```

bool ecrt(ll* r, ll* m, int n, ll& re, ll& mo)
{
    ll x, y, d; mo = m[0]; re = r[0];
    for (int i = 1; i < n; i++) {
        d = egcd(mo, m[i], x, y);
        if ((r[i] - re) % d != 0) return false;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return true;
}

```

Count Digit Occurences

```

/*count(n,d) counts the number of occurences of
↳ a digit d in the range [0,n]*/
ll digit_count(ll n, ll d) {
    ll result = 0;
    while (n != 0) {
        result += ((n%10) == d ? 1 : 0);
        n /= 10;
    }
    return result;
}
ll count(ll n, ll d) {
    if (n < 10) return (d > 0 && n >= d);
    if ((n % 10) != 9) return digit_count(n, d) +
↳ count(n-1, d);
    return 10*count(n/10, d) + (n/10) + (d > 0);
}

```



```

    a[sat][sot] -= a[cur][sot] * num;
    cur++;
    break;
}
}
for (int j = cur; j < m; j++)
    if (!zero(a[j][n])) return vector<double>();
vector<double> ans(n,0);
for (int i = 0, sat = 0; i < n; i++)
    if (sat < m && !zero(a[sat][i]))
        ans[i] = a[sat][n] / a[sat+][i];
return ans;
}

```

6 Graph

```

struct edge {
    int u,v,w;
    edge (int u,int v,int w) : u(u),v(v),w(w) {}
    edge () : u(0), v(0), w(0) {}
};
bool operator < (const edge &e1, const edge
    &e2) { return e1.w < e2.w; }
bool operator > (const edge &e1, const edge
    &e2) { return e1.w > e2.w; }
struct subset { int p, rank; };

```

Eulerian Path

```

#define edge_list vector<edge>
#define adj_sets vector<set<int>>
struct EulerPathGraph {
    adj_sets graph; // actually indexes incident
    edges
    edge_list edges; int n; vector<int> indeg;
    EulerPathGraph(int n): n(n) {
        indeg = *(new vector<int>(n,0));
        graph = *(new adj_sets(n, set<int>()));
    }
    void add_edge(int u, int v) {
        graph[u].insert(edges.size());
        indeg[v]++;
        edges.push_back(edge(u,v,0));
    }
    bool eulerian_path(vector<int> &circuit) {
        if(edges.size()==0) return false;
        stack<int> st;
        int a[] = {-1, -1};
        for(int v=0;v<n;v++) {
            if(indeg[v]!=graph[v].size()) {
                bool b = indeg[v] > graph[v].size();
                if (abs(((int)indeg[v])-((int)graph[v]
                    .size())) > 1) return
                    false;
                if (a[b] != -1) return false;
                a[b] = v;
            }
        }
        int s = (a[0]!=-1 && a[1]!=-1 ? a[0] :
            (a[0]==-1 && a[1]==-1 ? edges[0].u : -1));
        if(s==-1) return false;
        while(!st.empty() || !graph[s].empty()) {
            if (graph[s].empty()) {
                circuit.push_back(s); s = st.top();
                st.pop();
            }
            else {
                int w = edges[*graph[s].begin()].v;
                graph[s].erase(graph[s].begin());
                st.push(s); s = w;
            }
        }
        circuit.push_back(s);
        return circuit.size()-1==edges.size();
    }
};

```

Minimum Spanning Tree

```

// returns vector of edges in the mst
// graph[i] = vector of edges incident to
    vertex i

```

```

// places total weight of the mst in &total
// if returned vector has size != n-1, there is
    no MST
vector<edge> mst(vector<vector<edge>> graph, ll
    &total) {
    total = 0;
    priority_queue<edge, vector<edge>,
        greater<edge>> pq;
    vector<edge> MST;
    bitset<20001> marked; // change size as
        needed
    marked[0] = 1;
    for (edge ep : graph[0]) pq.push(ep);
    while(MST.size()!=graph.size()-1 &&
        pq.size()!=0) {
        edge e = pq.top(); pq.pop();
        int u = e.u, v = e.v, w = e.w;
        if(marked[u] && marked[v]) continue;
        else if(marked[u]) swap(u, v);
        for(edge ep : graph[u]) pq.push(ep);
        marked[u] = 1;
        MST.push_back(e);
        total += e.w;
    }
    return MST;
}

```

Union Find

```

int uf_find(subset* s, int i) {
    if (s[i].p != i) s[i].p = uf_find(s, s[i].p);
    return s[i].p;
}
void uf_union(subset* s, int x, int y) {
    int xp = uf_find(s, x), yp = uf_find(s, y);
    if (s[xp].rank > s[yp].rank) s[yp].p = xp;
    else if (s[xp].rank < s[yp].rank) s[xp].p =
        yp;
    else { s[yp].p = xp; s[xp].rank++; }
}

```

7 2D Geometry

```

#define point complex<double>
#define sq(a) ((a)*(a))
#define cb(a) ((a)*(a)*(a))
double dot(point a, point b) { return
    real(conj(a)*b); }
double cross(point a, point b) { return
    imag(conj(a)*b); }
struct line { point a, b; };
struct circle { point c; double r; };
struct segment { point a, point b; };
struct triangle { point a, b, c; };
struct rectangle { point tl, br; };
struct convex_polygon {
    vector<point> points;
    convex_polygon(triangle a) {
        points.push_back(a.a);
        points.push_back(a.b);
        points.push_back(a.c);
    };
    convex_polygon(rectangle a) {
        points.push_back(a.tl);
        points.push_back({real(a.tl),
            imag(a.br)}});
        points.push_back(a.br);
        points.push_back({real(a.br),
            imag(a.tl)}});
    };
}
struct polygon {
    vector<point> points;
    polygon(triangle a) {
        points.push_back(a.a);
        points.push_back(a.b);
        points.push_back(a.c);
    };
    polygon(rectangle a) {

```

```

        points.push_back(a.tl);
        points.push_back({real(a.tl),
            imag(a.br)}});
        points.push_back(a.br);
        points.push_back({real(a.br),
            imag(a.tl)}});
    };
    polygon(convex_polygon a) {
        for (point v : a.points)
            points.push_back(v);
    };
    // line methods
    double lengthsq(line a) { return sq(real(a.a)
        - real(a.b)) + sq(imag(a.a) - imag(a.b)); }
    double length(line a) { return
        sqrt(lengthsq(a)); }
    // circle methods
    double circumference(circle a) { return 2 *
        a.r * M_PI; }
    double area(circle a) { return sq(a.r) * M_PI; }
    // rectangle methods
    double width(rectangle a) { return
        abs(real(a.br) - real(a.tl)); }
    double height(rectangle a) { return
        abs(imag(a.br) - real(a.tl)); }
    double diagonal(rectangle a) { return
        sqrt(sq(width(a)) + sq(height(a))); }
    double area(rectangle a) { return width(a) *
        height(a); }
    double perimeter(rectangle a) { return 2 *
        (width(a) + height(a)); }
}

```

Intersection

```

// -1 coincide, 0 parallel, 1 intersection
int intersection(line a, line b, point& p) {
    if (abs(cross(a.b - a.a, b.b - b.a)) > EPS) {
        p = cross(b.a - a.a, b.b - a.b) / cross(a.b
            - a.a, b.b - b.a) * (b - a) + a;
        return 1;
    }
    if (abs(cross(a.b - a.a, a.b - b.a)) > EPS)
        return 0;
    return -1;
}
// area of intersection
double intersection(circle a, circle b) {
    double d = abs(a.c - b.c);
    if (d <= b.r - a.r) return area(a);
    if (d <= a.r - b.r) return area(b);
    if (d >= a.r + b.r) return 0;
    double alpha = acos((sq(a.r) + sq(d) -
        sq(b.r)) / (2 * a.r * d));
    double beta = acos((sq(b.r) + sq(d) -
        sq(a.r)) / (2 * b.r * d));
    return sq(a.r) * (alpha - 0.5 * sin(2 *
        alpha)) + sq(b.r) * (beta - 0.5 * sin(2 *
        beta));
}
// -1 outside, 0 inside, 1 tangent, 2
    intersection
int intersection(circle a, circle b,
    vector<point>& inter) {
    double d2 = norm(b.c - a.c), rS = a.r + b.r,
        rD = a.r - b.r;
    if (d2 > sq(rS)) return -1;
    if (d2 < sq(rD)) return 0;
    double ca = 0.5 * (1 + rS * rD / d2);
    point z = point(ca, sqrt(sq(a.r) / d2 -
        sq(ca)));
    inter.push_back(a.c + (b.c - a.c) * z);
    if (abs(imag(z)) > EPS) inter.push_back(a.c +
        (b.c - a.c) * conj(z));
    return inter.size();
}
// points of intersection

```

```

vector<point> intersection(line a, circle c) {
    vector<point> inter;
    c.c -= a.a;
    a.b -= a.a;
    point m = a.b * real(c.c / a.b);
    double d2 = norm(m - c.c);
    if (d2 > sq(c.r)) return 0;
    double l = sqrt((sq(c.r) - d2) / norm(a.b));
    inter.push_back(a.a + m + l * a.b);
    if (abs(l) > EPS) inter.push_back(a.a + m - l
        * a.b);
    return inter;
}
// area of intersection
double intersection(rectangle a, rectangle b)
    {
    double x1 = max(real(a.tl), real(b.tl)), y1 =
        max(imag(a.tl), imag(b.tl));
    double x2 = min(real(a.br), real(b.br)), y2 =
        min(imag(a.br), imag(b.br));
    return (x2 <= x1 || y2 <= y1) ? 0 :
        (x2-x1)*(y2-y1);
}

```

8 3D Geometry

```

struct point3d {
    double x, y, z;
    point3d operator+(point3d a) const { return
        {x+a.x, y+a.y, z+a.z}; }
    point3d operator*(double a) const { return
        {x*a, y*a, z*a}; }
    point3d operator-() const { return {-x, -y,
        -z}; }
    point3d operator-(point3d a) const { return
        *this + -a; }
    point3d operator/(double a) const { return
        *this * (1/a); }
    double norm() { return x*x + y*y + z*z; }
    double abs() { return sqrt(norm()); }
    point3d normalize() { return *this /
        *this->abs(); }
};
double dot(point3d a, point3d b) { return
    a.x*b.x + a.y*b.y + a.z*b.z; }
point3d cross(point3d a, point3d b) { return
    {a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z,
    a.x*b.y - a.y*b.x}; }
struct line3d { point3d a, b; };
struct plane { double a, b, c, d; } // a*x +
    b*y + c*z + d = 0
struct sphere { point3d c; double r; };
#define sq(a) ((a)*(a))
#define cb(a) ((a)*(a)*(a))
double surface(circle a) { return 4 * sq(a.r)
    * M_PI; }
double volume(circle a) { return 4.0/3.0 *
    cb(a.r) * M_PI; }

```

9 Optimization

Snoob

```

// SameNumberOfOneBits, next permutation
int snoob(int a) {
    int b = a & -a, c = a + b;
    return c | ((a ^ c) >> 2) / b;
}
// example usage
int main() {
    char l1[] = {'1', '2', '3', '4', '5'};
    char l2[] = {'a', 'b', 'c', 'd'};
    int d1 = 5, d2 = 4;
    // prints 12345abcd, 1234a5bcd, ...
    int min = (1<d1)-1, max = min < d2;
    for (int i = min; i <= max; i = snoob(i)) {
        int p1 = 0, p2 = 0, v = i;
        while (p1 < d1 || p2 < d2) {
            cout << ((v & 1) ? l1[p1++] : l2[p2++]);
            v /= 2;
        }
    }
}

```

```
..}  
..cout << '\n';  
}  
}
```

Powers

```
bool isPowerOf2(ll a) {  
    return a > 0 && !(a & a-1);  
}  
bool isPowerOf3(ll a) {  
    return a>0&&!(12157665459056928801ull%a);  
}  
bool isPower(ll a, ll b) {  
    double x = log(a) / log(b);  
    return abs(x-round(x)) < 0.00000000001;  
}
```