Contents

1 Contest Setup

1.1 vimrc

```
set number      " Show line numbers
set mouse=a     " Enable inaction via mouse
set showmatch       " Highlight matching brace
set cursorline      " Show underline
set cursorcolumn    " highlight vertical column

filetype on "enable file detection
syntax on   "syntax highlight

set autoindent      " Auto-indent new lines
set shiftwidth=4    " Number of auto-indent spaces
set smartindent     " Enable smart-indent
set smarttab        " Enable smart-tabs
set tabstop=4   " Number of spaces per Tab

" ---------Optional----------

set undolevels=10000    " Number of undo levels
set scrolloff=5     " Auto scroll

set hlsearch    " Highlight all search results
set smartcase   " Enable smart-case search
set ignorecase  " Always case-insensitive
set incsearch   " Searches for strings incrementally

highlight Comment ctermfg=cyan
set showmode

set encoding=utf-8
set fileencoding=utf-8
scriptencoding=utf-8
```

1.2 bashrc

```
alias g++="g++ -Wall -Wextra -std=c++11 -O2"
```

1.3 Grep Error and Warnings

```
g++ main.cpp 2>&1 | grep -E 'warning|error'
```

1.4 C++ template

```
#include <bits/stdc++.h>

using namespace std;

typedef long long int ll;
typedef pair<int, int> ii;

int main()
{
    return 0;
}
```

1.5 Java template

```
import java.io.*;
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        MyScanner sc = new MyScanner();
        out = new PrintWriter(new BufferedOutputStream(System.out));
        // Start writing your solution here.

        // Stop writing your solution here.
        out.close();
    }

    public static PrintWriter out;

    public static class MyScanner
    {
        BufferedReader br;
        StringTokenizer st;

        public MyScanner()
        {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        boolean hasNext()
        {
            while (st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                } catch (Exception e) {
                    return false;
                }
            }
            return true;
        }

        String next()
        {
            if (hasNext())
                return st.nextToken();
            return null;
        }

        int nextInt()
        {
            return Integer.parseInt(next());
        }

        long nextLong()
        {
            return Long.parseLong(next());
        }

        double nextDouble()
        {
            return Double.parseDouble(next());
        }

        String nextLine()
        {
            String str = "";
            try {
                str = br.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }
}
```

1.5.1 Java Issues

1. Random Shuffle before sorting: $Random\ rnd = new\ Random();\ rnd.nextInt();$
2. Use StringBuilder for large output
3. Java has strict parsing rules. e.g. using sc.nextInt() to read a long will result in RE

2 System Testing

1. Setup bashrc and vimrc
2. Look for compilation parameter and code it into bashrc
3. Test if c++ and java templates work properly on local and judge machine
4. Test "divide by 0" → RE/TLE?

5. Test heap size

3 Reminder

1. 隊友的建議，要認真聽！通常隊友的建議都會突破你盲點
2. Read the problem statements carefully. Input and output specifications and constraints are crucial!
3. Estimate the **time complexity** and **memory complexity** carefully.
4. Time penalty is 20 minutes per WA, **don't rush**!
5. Sample test cases must all be tested and passed before every submission!
6. Test the corner cases, such as 0, 1, -1. Test all edge cases of the input specification.
7. Bus error: the code has $scanf$, $fgets$ but have nothing to read! Check if you have early termination but didn't handle it properly.
8. Binary search? 數學算式移項合併後查詢?
9. Two Pointer <–> Binary Search
10. Directed graph connectivity -> DFS. Undirected graph -> Union Find
11. Check connectivity of the graph if the problem statement doesn't say anything
12. $longlong = int * int$ won't work!
13. Shifting for $longlongint$ should be something like $1LL << 35$
14. For continuous input problems, be sure to read in all input BEFORE terminating and start processing next the input.
15. Don't use anonymous struct
16. 因式分解

4 Topic list

1. enumeration
2. greedy
3. sorting, topological sort
4. binary search
5. 離散化
6. 矩陣快速冪
7. Pigeonhole
8. DFS 轉換成 RMQ

5 Useful code

5.1 Leap year

```
year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)
```

5.2 Fast Exponentiation $O(log(exp))$

Fermat's little theorem: 若 m 是質數，則 $a^{m-1} \equiv 1 \pmod m$

```
ll fast_pow(ll a, ll b, ll M) {
    ll ans = 1;
    ll base = a % M;
    while (b) {
        if (b & 1)
            ans = ans * base % M;
        base = base * base % M;
        b >>= 1;
    }
    return ans;
}
```


5.3 Mod Inverse

Case 1: $gcd(a, m) = 1$: ax + my = gcd(a, m) = 1 (use ext_gcd)

Case 2: $m\ is\ prime$: $a^{m-2} \equiv a^{-1} mod\ m$

5.4 GCD $O(log(a + b))$

注意負數的 case! C++ 是看被除數決定正負號的。

```
ll gcd(ll a, ll b)
{
    return b == 0 ? a : gcd(b, a % b);
}
```

5.5 Extended Euclidean Algorithm GCD $O(log(a + b))$

Bezout identity $ax + by = gcd(a, b)$, where $gcd(a, b)$ is the smallest positive integer that can be written as ax + by, and every integer of the form ax + by is a multiple of $gcd(a, b)$.

```
ll extgcd(ll a, ll b, ll& x, ll&y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    else {
        ll d = extgcd(b, a % b, y, x);
        y -= (a / b) * x;
        return d;
    }
}
```

5.6 Prime Generator

```
const ll MAX_NUM = 1e6; // 要是合數
bool is_prime[MAX_NUM];
vector<ll> primes;

void init_primes() {
    fill(is_prime, is_prime + MAX_NUM, true);
    is_prime[0] = is_prime[1] = false;
    for (ll i = 2; i < MAX_NUM; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (ll j = i * i; j < MAX_NUM; j += i)
                is_prime[j] = false;
        }
    }
}
```

5.7 C++ Reference

5.7.1 scanf/printf reference

5.7.2 Map

```
map<T1, T2> m; // iterable
void clear();
void erase(T1 key);
it find(T1 key); // <key, val>
void insert(pair<T1, T2> P);
T2 &[](T1 key); // if key not in map, new key will be inserted with default val
it lower_bound(T1 key); // = m.end() if not found, *it = <key, val>
it upper_bound(T1 key); // = m.end() if not found, *it = <key, val>
```

5.7.3 Set

```
set<T> s; // iterable
void clear();
size_t count(T val); // number of val in set
void erase(T val);
it find(T val); // = s.end() if not found
void insert(T val);
it lower_bound(T val); // = s.end() if not found, *it = <key, val>
it upper_bound(T val); // = s.end() if not found, *it = <key, val>
```

5.7.4 Algorithm

```
// return if i is smaller than j
comp = [&](const T &i, const T &j) -> bool;
vector<T> v;
bool any_of(v.begin(), v.end(), [&](const T &i) -> bool);
bool all_of(v.begin(), v.end(), [&](const T &i) -> bool);
void copy(inp.begin(), in.end(), out.begin());
int count(v.begin(), v.end(), int val); // number of val in v
it unique(v.begin(), v.end());          // it - v.begin() = size
// after calling, v[nth] will be n-th smallest elem in v
void nth_element(v.begin(), nth_it, bin_comp);
void merge(in1.begin(), in1.end(), in2.begin(), in2.end(), out.begin(),
    comp);
// include union, intersection, difference, symmetric_difference(xor)
void set_union(in1.begin(), in1.end(), in2.begin(), in2.end(), out.
    begin(), comp);
bool next_permutation(v.begin(), v.end());
// v1, v2 need sorted already, whether v1 includes v2
bool inclues(v1.begin(), v1.end(), v2.begin(), v2.end());
it find(v.begin(), v.end(), T val); // = v.end() if not found
it search(v1.begin(), v1.end(), v2.begin(), v2.end());
it lower_bound(v.begin(), v.end(), T val);
it upper_bound(v.begin(), v.end(), T val);
bool binary_search(v.begin(), v.end(), T val); // exist in v ?
void sort(v.begin(), v.end(), comp);
void stable_sort(v.begin(), v.end(), comp);
```

5.7.5 String

5.7.6 Priority Queue

```
bool cmp(ii a, ii b)
{
    if(a.first == b.first)
        return a.second > b.second;
    return b.first > a.first;
}

priority_queue< ii, vector<ii>, function<bool(ii, ii)> > pq(cmp);
```

6 Search

6.1 Ternary Search

6.2 折半完全列舉

能用 vector 就用 vector

6.3 Two-pointer 爬行法 (右跑左追)

7 Basic data structure

7.1 1D BIT

```
// BIT is 1-based
const int MAX_N = 20000; //這個記得改!
ll bit[MAX_N + 1];

ll sum(int i) {
    int s = 0;
    while (i > 0) {
        s += bit[i];
        i -= (i & -i);
    }
    return s;
}

void add(int i, ll x) {
    while (i <= MAX_N) {
        bit[i] += x;
        i += (i & -i);
    }
}
```

7.2 2D BIT

```
// BIT is 1-based
const int MAX_N = 20000, MAX_M = 20000; //這個記得改!
ll bit[MAX_N + 1][MAX_M + 1];

```


4

```cpp
ll sum(int a, int b) {
    ll s = 0;
    for (int i = a; i > 0; i -= (i & -i))
        for (int j = b; j > 0; j -= (j & -j))
            s += bit[i][j];
    return s;
}

void add(int a, int b, ll x) {
    // MAX_N, MAX_M 須適時調整!
    for (int i = a; i <= MAX_N; i += (i & -i))
        for (int j = b; j <= MAX_M; j += (j & -j))
            bit[i][j] += x;
}
```

7.3 Union Find

```cpp
#define N 20000 // 記得改
struct UFDS {
    int par[N];

    void init() {
        memset(par, -1, sizeof(par));
    }

    int root(int x) {
        return par[x] < 0 ? x : par[x] = root(par[x]);
    }

    void merge(int x, int y) {
        x = root(x);
        y = root(y);

        if (x != y) {
            if (par[x] > par[y])
                swap(x, y);
            par[x] += par[y];
            par[y] = x;
        }
    }
}
```

7.4 Segment Tree

```cpp
const int MAX_N = 100000;
const int MAX_NN = (1 << 20); // should be bigger than MAX_N

int N;
ll inp[MAX_N];

int NN;
ll seg[2 * MAX_NN - 1];
ll lazy[2 * MAX_NN - 1];
// lazy[u] != 0 : the subtree of u (u not included) is not up-to-date

void seg_gather(int u)
{
    seg[u] = seg[u * 2 + 1] + seg[u * 2 + 2];
}

void seg_push(int u, int l, int m, int r)
{
    if (lazy[u] != 0) {
        seg[u * 2 + 1] += (m - l) * lazy[u];
        seg[u * 2 + 2] += (r - m) * lazy[u];

        lazy[u * 2 + 1] += lazy[u];
        lazy[u * 2 + 2] += lazy[u];
        lazy[u] = 0;
    }
}

void seg_init()
{
    NN = 1;
    while (NN < N)
        NN *= 2;

    memset(seg, 0, sizeof(seg)); // val that won't affect result
    memset(lazy, 0, sizeof(lazy)); // val that won't affect result
    memcpy(seg + NN - 1, inp, sizeof(ll) * N); // fill in leaves
}

void seg_build(int u)
{
    if (u >= NN - 1) { // leaf
        return;
    }

    seg_build(u * 2 + 1);
    seg_build(u * 2 + 2);
    seg_gather(u);
}

void seg_update(int a, int b, int delta, int u, int l, int r)
{
    if (l >= b || r <= a) {
        return;
    }

    if (a <= l && r <= b) {
        seg[u] += (r - l) * delta;
        lazy[u] += delta;
        return;
    }

    int m = (l + r) / 2;
    seg_push(u, l, m, r);
    seg_update(a, b, delta, u * 2 + 1, l, m);
    seg_update(a, b, delta, u * 2 + 2, m, r);
```

```cpp
    seg_gather(u);
}

ll seg_query(int a, int b, int u, int l, int r)
{
    if (l >= b || r <= a) {
        return 0;
    }

    if (a <= l && r <= b) {
        return seg[u];
    }

    int m = (l + r) / 2;
    seg_push(u, l, m, r);
    ll ans = 0;
    ans += seg_query(a, b, u * 2 + 1, l, m);
    ans += seg_query(a, b, u * 2 + 2, m, r);
    seg_gather(u);

    return ans;
}
```

7.5 Sparse Table

```cpp
struct {
    int sp[MAX_LOG_N][MAX_N]; // MAX_LOG_N = ceil(lg(MAX_N))

    void build(int inp[], int n)
    {
        for (int j = 0; j < n; j++)
            sp[0][j] = inp[j];

        for (int i = 1; (1 << i) <= n; i++)
            for (int j = 0; j + (1 << i) <= n; j++)
                sp[i][j] = min(sp[i-1][j],sp[i-1][j+(1 << (i - 1))]);
    }

    int query(int l, int r) // [l, r)
    {
        int k = floor(log2(r - l));
        return min(sp[k][l], sp[k][r - (1 << k)]);
    }
} sptb;
```

8 Dynamic Programming

9 Tree

9.1 LCA

9.2 Tree Centroid

9.3 Treap

10 Graph

10.1 Articulation point / edge

10.2 CC

10.2.1 BCC vertex

10.2.2 BCC edge

10.2.3 SCC

10.3 Shortest Path

Time complexity notations: V = vertex, E = edge

10.3.1 Dijkatra (next-to-shortest path)

```cpp
struct Edge {
    int to, cost;
};

typedef pair<int, int> P; // <d, v>
const int INF = 0x3f3f3f3f;

int N, R;
vector<Edge> g[5000];

int d[5000];
int sd[5000];

int solve() {
    fill(d, d + N, INF);
    fill(sd, sd + N, INF);
    priority_queue< P, vector<P>, greater<P> > pq;

    d[0] = 0;
    pq.push(P(0, 0));

    while (!pq.empty()) {
        P p = pq.top(); pq.pop();
        int v = p.second;

        if (sd[v] < p.first) // 比次短距離還大，沒用，跳過
            continue;

```

```
        for (size_t i = 0; i < g[v].size(); i++) {
            Edge& e = g[v][i];
            int nd = p.first + e.cost;
            if (nd < d[e.to]) { // 更新最短距離
                swap(d[e.to], nd);
                pq.push(P(d[e.to], e.to));
            }
            if (d[e.to] < nd && nd < sd[e.to]) { // 更新次短距離
                sd[e.to] = nd;
                pq.push(P(sd[e.to], e.to));
            }
        }
    }

    return sd[N-1];
}
```

10.3.2 SPFA

```
#define st first
#define nd second

typedef pair<int, int> pii; // <d, v>
struct Edge {
    int to, w;
};

const int MAX_V = ...;
const int INF = 0x3f3f3f3f;

int V, S; // V, Source
vector<Edge> g[MAX_V];
int d[MAX_V];
int cnt[MAX_V];

bool spfa() { // 回傳有無負環
    fill(d, d + V, INF);
    fill(cnt, cnt + V, 0);
    priority_queue< pii, vector<pii>, greater<pii> > pq;

    d[S] = 0;
    pq.push(pii(0, S));
    cnt[S] = 1;

    while (!pq.empty()) {
        pii top = pq.top(); pq.pop();
        int u = top.nd;

        if (d[u] < top.st) continue;

        // for (const Edge& e : g[u]) {
        for (size_t i = 0; i < g[u].size(); i++) {
            const Edge& e = g[u][i];
            if (d[e.to] > d[u] + e.w) {
                d[e.to] = d[u] + e.w;
                pq.push(pii(d[e.to], e.to));

                cnt[e.to]++;
                if (cnt[e.to] >= V)
                    return true;
            }
        }
    }

    return false;
}
```

10.3.3 Bellman-Ford $O(VE)$

```
vector<pair<ii, int>> edge; // store graph by edge: ((u, v), w)

void BellmanFord()
{
    ll d[n]; // n: total nodes
    fill(d, d + n, INT_MAX);
    d[0] = 0; // src is 0
    bool loop = false;
    for (int i = 0; i <= n; i++) {
        // Do n - 1 times. If the n-th time still has relaxation, loop exists
        bool hasChange = false;
        for (int j = 0; j < (int)edge.size(); j++) {
            int u = edge[j].first.first, v = edge[j].first.second, w = edge[j].second;
            if (d[u] + w < d[v]) {
                hasChange = true;
                d[v] = d[u] + w;
            }
        }

        if (i == n && hasChange == true)
            loop = true;
        else if (hasChange == false)
            break;
    }
}
```

10.3.4 Floyd-Warshall $O(V^3)$

The graph is stored using adjacency matrix. The initial state is $diagnal = 0$ and $others = INF$. (If INF is int, use long long for the matrix)
If diagonal numbers are negative $\leftarrow$ cycle .

```
for(int k = 0; k < N; k++)
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
```

10.4 MST

10.4.1 Kruskal

1. Store the graph by $(weight, (from, to))$
2. Sort the graph by $weight$
3. Start from the smallest weight, and keep adding edges that won't form a cycle with the current MST set
4. Early termination condition: $n - 1$ edges has been added, NOT size of the union-find set

10.4.2 Prim

11 Flow

11.1 Max Flow (Dinic)

```
struct Edge {
    int to, cap, rev;
    Edge(int a, int b, int c) {
        to = a;
        cap = b;
        rev = c;
    }
};

const int INF = 0x3f3f3f3f;
const int MAX_V = 20000 + 10;
// vector<Edge> g[MAX_V];
vector< vector<Edge> > g(MAX_V);
int level[MAX_V];
int iter[MAX_V];

inline void add_edge(int u, int v, int cap) {
    g[u].push_back((Edge){v, cap, (int)g[v].size()});
    g[v].push_back((Edge){u, 0, (int)g[u].size() - 1});
}

void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> q;

    level[s] = 0;
    q.push(s);

    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (int i = 0; i < int(g[v].size()); i++) {
            const Edge& e = g[v][i];
            if (e.cap > 0 && level[e.to] < 0) {
                level[e.to] = level[v] + 1;
                q.push(e.to);
            }
        }
    }
}

int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int& i = iter[v]; i < int(g[v].size()); i++) {
        Edge& e = g[v][i];
        if (e.cap > 0 && level[v] < level[e.to]) {
            int d = dfs(e.to, t, min(f, e.cap));
            if (d > 0) {
                e.cap -= d;
                g[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) { // dinic
    int flow = 0;
    for (;;) {
        bfs(s);
        if (level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        int f;
        while ((f = dfs(s, t, INF)) > 0) {
            flow += f;
        }
    }
}
```

11.2 Min Cost Flow

```
#define st first
#define nd second

typedef pair<double, int> pii;
const double INF = 1e10;

struct Edge {
    int to, cap;
    double cost;
    int rev;
};

const int MAX_V = 2 * 100 + 10;
int V;
vector<Edge> g[MAX_V];
double h[MAX_V];
double d[MAX_V];
int prevv[MAX_V];
int preve[MAX_V];
// int match[MAX_V];

void add_edge(int u, int v, int cap, double cost) {
    g[u].push_back((Edge){v, cap, cost, (int)g[v].size()});
```

```cpp
    g[v].push_back((Edge){u, 0, -cost, (int)g[u].size() - 1});
}

double min_cost_flow(int s, int t, int f) {
    double res = 0;
    fill(h, h + V, 0);
    fill(match, match + V, -1);
    while (f > 0) {
        // dijkstra 找最小成本增廣路徑
        // without h will reduce to SPFA = O(V*E)
        fill(d, d + V, INF);
        priority_queue< pii, vector<pii>, greater<pii> > pq;

        d[s] = 0;
        pq.push(pii(d[s], s));

        while (!pq.empty()) {
            pii p = pq.top(); pq.pop();
            int v = p.nd;
            if (d[v] < p.st) continue;
            for (size_t i = 0; i < g[v].size(); i++) {
                const Edge& e = g[v][i];
                if (e.cap > 0 && d[e.to] > d[v] + e.cost + h[v] - h[e.to]) {
                    d[e.to] = d[v] + e.cost + h[v] - h[e.to];
                    prevv[e.to] = v;
                    preve[e.to] = i;
                    pq.push(pii(d[e.to], e.to));
                }
            }
        }

        // 找不到增廣路徑
        if (d[t] == INF) return -1;

        // 維護 h[v]
        for (int v = 0; v < V; v++)
            h[v] += d[v];

        // 找瓶頸
        int bn = f;
        for (int v = t; v != s; v = prevv[v])
            bn = min(bn, g[prevv[v]][preve[v]].cap);

        // // find match
        // for (int v = prevv[t]; v != s; v = prevv[prevv[v]]) {
        //     int u = prevv[v];
        //     match[v] = u;
        //     match[u] = v;
        // }

        // 更新剩餘圖
        f -= bn;
        res += bn * h[t]; // SPFA: res += bn * d[t]
        for (int v = t; v != s; v = prevv[v]) {
            Edge& e = g[prevv[v]][preve[v]];
            e.cap -= bn;
            g[v][e.rev].cap += bn;
        }
    }
    return res;
}
```

11.3 Bipartite Matching

```cpp
const int MAX_V = ...;
int V;
vector<int> g[MAX_V];
int match[MAX_V];
bool used[MAX_V];

void add_edge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
}

// 回傳有無找到從 v 出發的增廣路徑
//（首尾都為未匹配點的交錯路徑）
// [待確認] 每次遞迴都找一個末匹配點 v 及匹配點 u
bool dfs(int v) {
    used[v] = true;
    for (size_t i = 0; i < g[v].size(); i++) {
        int u = g[v][i], w = match[u];
        // 尚未配對或可從 w 找到增廣路徑（即路徑繼續增長）
        if (w < 0 || (!used[w] && dfs(w))) {
            // 交錯配對
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int bipartite_matching() { // 匈牙利演算法
    int res = 0;
    memset(match, -1, sizeof(match));
    for (int v = 0; v < V; v++) {
        if (match[v] == -1) {
            memset(used, false, sizeof(used));
            if (dfs(v)) {
                res++;
            }
        }
    }
    return res;
}
```

12 String

12.1 Rolling Hash

1. Use two rolling hashes if needed.
2. The prime for pre-calculation can be 137 and 257, for modulo can be $1e9 + 7$ and $0xdefaced$

```
#define N 1000100
#define B 137
#define M 1000000007

typedef long long ll;

char inp[N];
int len;
ll p[N], h[N];

void init()
{ // build polynomial table and hash value
    p[0] = 1; // b to the ith power
    for (int i = 1; i <= len; i++) {
        h[i] = (h[i - 1] * B % M + inp[i - 1]) % M; // hash value
        p[i] = p[i - 1] * B % M;
    }
}

ll get_hash(int l, int r) // [l, r] of the inp string array
{
    return ((h[r + 1] - (h[l] * p[r - l + 1])) % M + M) % M;
}
```

12.2 KMP

```
void fail()
{
    int len = strlen(pat);

    f[0] = 0;
    int j = 0;
    for (int i = 1; i < len; i++) {
        while (j != 0 && pat[i] != pat[j])
            j = f[j - 1];

        if (pat[i] == pat[j])
            j++;

        f[i] = j;
    }
}

int match()
{
    int res = 0;
    int j = 0, plen = strlen(pat), tlen = strlen(text);

    for (int i = 0; i < tlen; i++) {
        while (j != 0 && text[i] != pat[j])
            j = f[j - 1];

        if (text[i] == pat[j]) {
            if (j == plen - 1) { // find match
                res++;
                j = f[j];
            } else {
                j++;
            }
        }
    }

    return res;
}
```

12.3 Z Algorithm

12.4 Trie

注意 count 的擺放位置，視題意可以擺在迴圈外

```
struct Node {
    int cnt;
    Node* nxt[2];
    Node() {
        cnt = 0;
        fill(nxt, nxt + 2, nullptr);
    }
};

const int MAX_Q = 200000;
int Q;

int NN = 0;
Node data[MAX_Q * 30];
Node* root = &data[NN++];

void insert(Node* u, int x) {
    for (int i = 30; i >= 0; i--) {
        int t = ((x >> i) & 1);
        if (u->nxt[t] == nullptr) {
            u->nxt[t] = &data[NN++];
        }

        u = u->nxt[t];
        u->cnt++;
    }
}

void remove(Node* u, int x) {
    for (int i = 30; i >= 0; i--) {
        int t = ((x >> i) & 1);
        u = u->nxt[t];
        u->cnt--;
    }
}
```

```

int query(Node* u, int x) {
    int res = 0;
    for (int i = 30; i >= 0; i--) {
        int t = ((x >> i) & 1);
        // if it is possible to go the another branch
        // then the result of this bit is 1
        if (u->nxt[t ^ 1] != nullptr && u->nxt[t ^ 1]->cnt > 0) {
            u = u->nxt[t ^ 1];
            res |= (1 << i);
        }
        else {
            u = u->nxt[t];
        }
    }
    return res;
}
```

12.5 Suffix Array

13 Matrix

13.1 高斯消去法

13.2 高斯喬登

14 Geometry

1. Keep things in integers as much as possible!
2. Try not to divide
3. If you have decimals, if they are fixed precision, you can usually just multiply all the input and use integers instead

14.1 EPS

$= 0$: $fabs \leq eps$
< 0: $< -eps$
> 0: $> +eps$

14.2 Template

```
// if the points are given in doubles form, change the code accordingly

typedef long long ll;

typedef pair<ll, ll> pt;  // points are stored using long long
typedef pair<pt, pt> seg; // segments are a pair of points

#define x first
#define y second

#define EPS 1e-9

pt operator+(pt a, pt b)
{
    return pt(a.x + b.x, a.y + b.y);
}

pt operator-(pt a, pt b)
{
    return pt(a.x - b.x, a.y - b.y);
}

pt operator*(pt a, int d)
{
    return pt(a.x * d, a.y * d);
}

ll cross(pt a, pt b)
{
    return a.x * b.y - a.y * b.x;
}

int ccw(pt a, pt b, pt c)
{
    ll res = cross(b - a, c - a);
    if (res > 0) // left turn
        return 1;
    else if (res == 0) // straight
        return 0;
    else // right turn
        return -1;
}

double dist(pt a, pt b)
{
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

bool zero(double x)
{
    return fabs(x) <= EPS;
}

bool overlap(seg a, seg b)
{
    return ccw(a.x, a.y, b.x) == 0 && ccw(a.x, a.y, b.y) == 0;
}

bool intersect(seg a, seg b)
{
    if (overlap(a, b) == true) { // non-proper intersection
        double d = 0;
        d = max(d, dist(a.x, a.y));
        d = max(d, dist(a.x, b.x));
        d = max(d, dist(a.x, b.y));
        d = max(d, dist(a.y, b.x));
        d = max(d, dist(a.y, b.y));
        d = max(d, dist(b.x, b.y));
```

```cpp

        // d > dist(a.x, a.y) + dist(b.x, b.y)
        if (d - (dist(a.x, a.y) + dist(b.x, b.y)) > EPS)
            return false;
        return true;
    }
    //                        |
    // Equal sign for -----| case
    // non qeual sign => proper intersection
    if (ccw(a.x, a.y, b.x) * ccw(a.x, a.y, b.y) <= 0 &&
        ccw(b.x, b.y, a.x) * ccw(b.x, b.y, a.y) <= 0)
        return true;
    return false;
}

double area(vector<pt> pts)
{
    double res = 0;
    int n = pts.size();
    for (int i = 0; i < n; i++)
        res += (pts[i].y + pts[(i + 1) % n].y) * (pts[(i + 1) % n].x - pts[i].x);
    return res / 2.0;
}

vector<pt> halfHull(vector<pt> &points)
{
    vector<pt> res;

    for (int i = 0; i < (int)points.size(); i++) {
        while ((int)res.size() >= 2 &&
               ccw(res[res.size() - 2], res[res.size() - 1], points[i]) < 0)
            res.pop_back(); // res.size() - 2 can't be assign before size() >= 2
        // check, bitch

        res.push_back(points[i]);
    }

    return res;
}

vector<pt> convexHull(vector<pt> &points)
{
    vector<pt> upper, lower;

    // make upper hull
    sort(points.begin(), points.end());

    upper = halfHull(points);
    // make lower hull
    reverse(points.begin(), points.end());
    lower = halfHull(points);

    // merge hulls
    if ((int)upper.size() > 0) // yes sir~
        upper.pop_back();
    if ((int)lower.size() > 0)
        lower.pop_back();

    vector<pt> res(upper.begin(), upper.end());
    res.insert(res.end(), lower.begin(), lower.end());

    return res;
}

bool completelyInside(vector<pt> &outer, vector<pt> &inner)
{
    int even = 0, odd = 0;
    for (int i = 0; i < (int)inner.size(); i++) {
        // y = slope * x + offset
        int cntIntersection = 0;
        ll slope = rand() % INT_MAX + 1;
        ll offset = inner[i].y - slope * inner[i].x;

        ll farx = 111111 * (slope >= 0 ? 1 : -1);
        ll fary = farx * slope + offset;
        seg a = seg(pt(inner[i].x, inner[i].y), pt(farx, fary));
        for (int j = 0; j < (int)outer.size(); j++) {
            seg b = seg(outer[j], outer[(j + 1) % (int)outer.size()]);

            if ((b.x.x * slope + offset == b.x.y) ||
                (b.y.x * slope + offset == b.y.y)) { // on-line
                i--;
                break;
            }

            if (intersect(a, b) == true)
                cntIntersection++;
        }

        if (cntIntersection % 2 == 0) // outside
            even++;
        else
            odd++;
    }

    return odd == (int)inner.size();
}

// srand(time(NULL))
// rand()
```

15 Math

15.1 Euclid's formula (Pythagorean Triples)

$a = p^2 - q^2$
$b = 2pq$ **(always even)**
$c = p^2 + q^2$

15.2 Difference between two consecutive numbers' square is odd

$(k+1)^2 - k^2 = 2k + 1$

15.3 Summation

$\sum_{k=1}^{n} 1 = n$
$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4}$

15.4 FFT

```
typedef unsigned int ui;
typedef long double ldb;
const ldb pi = atan2(0, -1);

struct Complex {
    ldb real, imag;
    Complex(): real(0.0), imag(0.0) {;}
    Complex(ldb a, ldb b) : real(a), imag(b) {;}
    Complex conj() const {
        return Complex(real, -imag);
    }
    Complex operator + (const Complex& c) const {
        return Complex(real + c.real, imag + c.imag);
    }
    Complex operator - (const Complex& c) const {
        return Complex(real - c.real, imag - c.imag);
    }
    Complex operator * (const Complex& c) const {
        return Complex(real*c.real - imag*c.imag, real*c.imag + imag*c.real);
    }
    Complex operator / (ldb x) const {
        return Complex(real / x, imag / x);
    }
    Complex operator / (const Complex& c) const {
        return *this * c.conj() / (c.real * c.real + c.imag * c.imag);
    }
};

inline ui rev_bit(ui x, int len){
    x = ((x & 0x55555555u) << 1)  | ((x & 0xAAAAAAAAu) >> 1);
    x = ((x & 0x33333333u) << 2)  | ((x & 0xCCCCCCCCu) >> 2);
    x = ((x & 0x0F0F0F0Fu) << 4)  | ((x & 0xF0F0F0F0u) >> 4);
    x = ((x & 0x00FF00FFu) << 8)  | ((x & 0xFF00FF00u) >> 8);
    x = ((x & 0x0000FFFFu) << 16) | ((x & 0xFFFF0000u) >> 16);
    return x >> (32 - len);
}

// flag = -1 if ifft else +1
void fft(vector<Complex>& a, int flag = +1) {
    int n = a.size(); // n should be power of 2

    int len = __builtin_ctz(n);
    for (int i = 0; i < n; i++) {
        int rev = rev_bit(i, len);

        if (i < rev)
            swap(a[i], a[rev]);
    }

    for (int m = 2; m <= n; m <<= 1) { // width of each item
        auto wm = Complex(cos(2 * pi / m), flag * sin(2 * pi / m));
        for (int k = 0; k < n; k += m) { // start idx of each item
            auto w = Complex(1, 0);
            for (int j = 0; j < m / 2; j++) { // iterate half
                Complex t = w * a[k + j + m / 2];
                Complex u = a[k + j];
                a[k + j] = u + t;
                a[k + j + m / 2] = u - t;
                w = w * wm;
            }
        }
    }

    if (flag == -1) { // if it's ifft
        for (int i = 0; i < n; i++)
            a[i].real /= n;
    }
}

vector<int> mul(const vector<int>& a, const vector<int>& b) {
    int n = int(a.size()) + int(b.size()) - 1;
    int nn = 1;
    while (nn < n)
        nn <<= 1;

    vector<Complex> fa(nn, Complex(0, 0));
    vector<Complex> fb(nn, Complex(0, 0));
    for (int i = 0; i < int(a.size()); i++)
        fa[i] = Complex(a[i], 0);
    for (int i = 0; i < int(b.size()); i++)
        fb[i] = Complex(b[i], 0);

    fft(fa, +1);
    fft(fb, +1);
    for (int i = 0; i < nn; i++) {
        fa[i] = fa[i] * fb[i];
    }
    fft(fa, -1);

    vector<int> c;
    for(int i = 0; i < nn; i++) {
        int val = int(fa[i].real + 0.5);
        if (val) {
            while (int(c.size()) <= i)
                c.push_back(0);
```

```cpp
                c[i] = 1;
            }
        }

        return c;
}
```

15.5 Combination

15.5.1 Pascal triangle

```cpp
#define N 210
ll C[N][N];

void Combination() {
    for(ll i=0; i<N; i++) {
        C[i][0] = 1;
        C[i][i] = 1;
    }

    for(ll i=2; i<N; i++) {
        for(ll j=1; j<=i; j++) {
            C[i][j] = (C[i-1][j] + C[i-1][j-1])%M; // if needed, mod it
        }
    }
}
```

15.5.2 線性

```cpp
ll binomialCoeff(ll n, ll k)
{
    ll res = 1;

    if ( k > n - k ) // Since C(n, k) = C(n, n-k)
        k = n - k;

    for (int i = 0; i < k; ++i) // n...n-k / 1...k
    {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}
```

15.6 重複組合

15.7 Chinese remainder theorem

```cpp
typedef long long ll;

struct Item {
    ll m, r;
};

ll extgcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    } else {
        ll d = extgcd(b, a % b, y, x);
        y -= (a / b) * x;
        return d;
    }
}

Item extcrt(const vector<Item> &v)
{
    ll m1 = v[0].m, r1 = v[0].r, x, y;

    for (int i = 1; i < int(v.size()); i++) {
        ll m2 = v[i].m, r2 = v[i].r;
        ll g = extgcd(m1, m2, x, y); // now x = (m/g)^(-1)

        if ((r2 - r1) % g != 0)
            return {-1, -1};

        ll k = (r2 - r1) / g * x % (m2 / g);
        k = (k + m2 / g) % (m2 / g); // for the case k is negative

        ll m = m1 * m2 / g;
        ll r = (m1 * k + r1) % m;

        m1 = m;
        r1 = (r + m) % m; // for the case r is negative
    }

    return (Item) {
        m1, r1
    };
}
```

15.8 2-Circle relations

$d =$ 圓心距, R, r 為半徑 $(R \geq r)$
內切: $d = R - r$
外切: $d = R + r$
內離: $d < R - r$
外離: $d > R + r$
相交: $d < R + r$ 且 $d > R - r$

15.9 Fun Facts

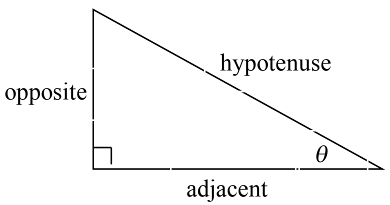
1. 如果 $\frac{b}{a}$ 是最簡分數, 則 $1 - \frac{b}{a}$ 也是
2.

Trig Cheat Sheet

Definition of the Trig Functions

Right triangle definition
For this definition we assume that

$0 < \theta < \frac{\pi}{2}$ or $0° < \theta < 90°$.

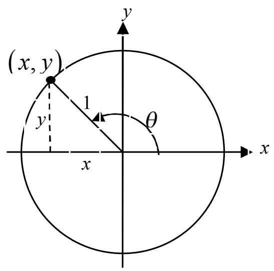


$\sin\theta = \frac{\text{opposite}}{\text{hypotenuse}}$ $\csc\theta = \frac{\text{hypotenuse}}{\text{opposite}}$

$\cos\theta = \frac{\text{adjacent}}{\text{hypotenuse}}$ $\sec\theta = \frac{\text{hypotenuse}}{\text{adjacent}}$

$\tan\theta = \frac{\text{opposite}}{\text{adjacent}}$ $\cot\theta = \frac{\text{adjacent}}{\text{opposite}}$

Unit circle definition
For this definition θ is any angle.



$\sin\theta = \frac{y}{1} = y$ $\csc\theta = \frac{1}{y}$

$\cos\theta = \frac{x}{1} = x$ $\sec\theta = \frac{1}{x}$

$\tan\theta = \frac{y}{x}$ $\cot\theta = \frac{x}{y}$

Facts and Properties

Domain
The domain is all the values of θ that can be plugged into the function.

$\sin\theta$, θ can be any angle
$\cos\theta$, θ can be any angle
$\tan\theta$, $\theta \neq \left(n+\frac{1}{2}\right)\pi,\ n = 0, \pm1, \pm2, \ldots$
$\csc\theta$, $\theta \neq n\pi,\ n = 0, \pm1, \pm2, \ldots$
$\sec\theta$, $\theta \neq \left(n+\frac{1}{2}\right)\pi,\ n = 0, \pm1, \pm2, \ldots$
$\cot\theta$, $\theta \neq n\pi,\ n = 0, \pm1, \pm2, \ldots$

Range
The range is all possible values to get out of the function.

$-1 \le \sin\theta \le 1$ $\csc\theta \ge 1$ and $\csc\theta \le -1$
$-1 \le \cos\theta \le 1$ $\sec\theta \ge 1$ and $\sec\theta \le -1$
$-\infty < \tan\theta < \infty$ $-\infty < \cot\theta < \infty$

Period
The period of a function is the number, T, such that $f(\theta + T) = f(\theta)$. So, if ω is a fixed number and θ is any angle we have the following periods.

$\sin(\omega\theta) \rightarrow T = \frac{2\pi}{\omega}$

$\cos(\omega\theta) \rightarrow T = \frac{2\pi}{\omega}$

$\tan(\omega\theta) \rightarrow T = \frac{\pi}{\omega}$

$\csc(\omega\theta) \rightarrow T = \frac{2\pi}{\omega}$

$\sec(\omega\theta) \rightarrow T = \frac{2\pi}{\omega}$

$\cot(\omega\theta) \rightarrow T = \frac{\pi}{\omega}$

© 2005 Paul Dawkins

Formulas and Identities

Tangent and Cotangent Identities

$\tan\theta = \frac{\sin\theta}{\cos\theta}$ $\cot\theta = \frac{\cos\theta}{\sin\theta}$

Reciprocal Identities

$\csc\theta = \frac{1}{\sin\theta}$ $\sin\theta = \frac{1}{\csc\theta}$

$\sec\theta = \frac{1}{\cos\theta}$ $\cos\theta = \frac{1}{\sec\theta}$

$\cot\theta = \frac{1}{\tan\theta}$ $\tan\theta = \frac{1}{\cot\theta}$

Pythagorean Identities

$\sin^2\theta + \cos^2\theta = 1$

$\tan^2\theta + 1 = \sec^2\theta$

$1 + \cot^2\theta = \csc^2\theta$

Even/Odd Formulas

$\sin(-\theta) = -\sin\theta$ $\csc(-\theta) = -\csc\theta$

$\cos(-\theta) = \cos\theta$ $\sec(-\theta) = \sec\theta$

$\tan(-\theta) = -\tan\theta$ $\cot(-\theta) = -\cot\theta$

Periodic Formulas
If n is an integer.

$\sin(\theta + 2\pi n) = \sin\theta$ $\csc(\theta + 2\pi n) = \csc\theta$

$\cos(\theta + 2\pi n) = \cos\theta$ $\sec(\theta + 2\pi n) = \sec\theta$

$\tan(\theta + \pi n) = \tan\theta$ $\cot(\theta + \pi n) = \cot\theta$

Double Angle Formulas

$$\sin(2\theta) = 2\sin\theta\cos\theta$$

$$\begin{aligned}\cos(2\theta) &= \cos^2\theta - \sin^2\theta \\ &= 2\cos^2\theta - 1 \\ &= 1 - 2\sin^2\theta\end{aligned}$$

$$\tan(2\theta) = \frac{2\tan\theta}{1-\tan^2\theta}$$

Degrees to Radians Formulas
If x is an angle in degrees and t is an angle in radians then

$$\frac{\pi}{180} = \frac{t}{x} \quad\Rightarrow\quad t = \frac{\pi x}{180} \quad\text{and}\quad x = \frac{180t}{\pi}$$

Half Angle Formulas (alternate form)

$\sin\frac{\theta}{2} = \pm\sqrt{\frac{1-\cos\theta}{2}}$ $\sin^2\theta = \frac{1}{2}\left(1-\cos(2\theta)\right)$

$\cos\frac{\theta}{2} = \pm\sqrt{\frac{1+\cos\theta}{2}}$ $\cos^2\theta = \frac{1}{2}\left(1+\cos(2\theta)\right)$

$\tan\frac{\theta}{2} = \pm\sqrt{\frac{1-\cos\theta}{1+\cos\theta}}$ $\tan^2\theta = \frac{1-\cos(2\theta)}{1+\cos(2\theta)}$

Sum and Difference Formulas

$$\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$$

$$\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$$

$$\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$$

Product to Sum Formulas

$$\sin\alpha\sin\beta = \frac{1}{2}\left[\cos(\alpha-\beta) - \cos(\alpha+\beta)\right]$$

$$\cos\alpha\cos\beta = \frac{1}{2}\left[\cos(\alpha-\beta) + \cos(\alpha+\beta)\right]$$

$$\sin\alpha\cos\beta = \frac{1}{2}\left[\sin(\alpha+\beta) + \sin(\alpha-\beta)\right]$$

$$\cos\alpha\sin\beta = \frac{1}{2}\left[\sin(\alpha+\beta) - \sin(\alpha-\beta)\right]$$

Sum to Product Formulas

$$\sin\alpha + \sin\beta = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right)$$

$$\sin\alpha - \sin\beta = 2\cos\left(\frac{\alpha+\beta}{2}\right)\sin\left(\frac{\alpha-\beta}{2}\right)$$

$$\cos\alpha + \cos\beta = 2\cos\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right)$$

$$\cos\alpha - \cos\beta = -2\sin\left(\frac{\alpha+\beta}{2}\right)\sin\left(\frac{\alpha-\beta}{2}\right)$$

Cofunction Formulas

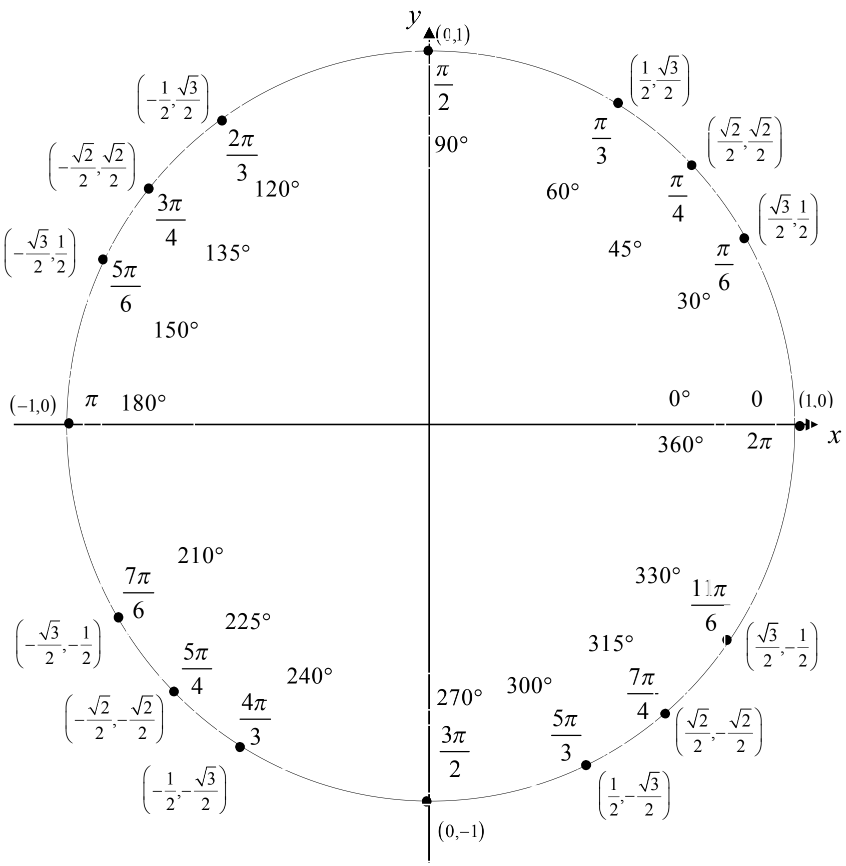
$\sin\left(\frac{\pi}{2}-\theta\right) = \cos\theta$ $\cos\left(\frac{\pi}{2}-\theta\right) = \sin\theta$

$\csc\left(\frac{\pi}{2}-\theta\right) = \sec\theta$ $\sec\left(\frac{\pi}{2}-\theta\right) = \csc\theta$

$\tan\left(\frac{\pi}{2}-\theta\right) = \cot\theta$ $\cot\left(\frac{\pi}{2}-\theta\right) = \tan\theta$

© 2005 Paul Dawkins

Unit Circle



For any ordered pair on the unit circle (x,y) : $\cos\theta = x$ and $\sin\theta = y$

Example

$$\cos\left(\frac{5\pi}{3}\right)=\frac{1}{2} \qquad \sin\left(\frac{5\pi}{3}\right)=-\frac{\sqrt{3}}{2}$$

© 2005 Paul Dawkins

Inverse Trig Functions

Definition

$y=\sin^{-1}x$ is equivalent to $x=\sin y$

$y=\cos^{-1}x$ is equivalent to $x=\cos y$

$y=\tan^{-1}x$ is equivalent to $x=\tan y$

Domain and Range

Function	Domain	Range
$y=\sin^{-1}x$	$-1\le x\le 1$	$-\frac{\pi}{2}\le y\le\frac{\pi}{2}$
$y=\cos^{-1}x$	$-1\le x\le 1$	$0\le y\le\pi$
$y=\tan^{-1}x$	$-\infty<x<\infty$	$-\frac{\pi}{2}< y<\frac{\pi}{2}$

Inverse Properties

$\cos\left(\cos^{-1}(x)\right)=x \qquad \cos^{-1}\left(\cos(\theta)\right)=\theta$

$\sin\left(\sin^{-1}(x)\right)=x \qquad \sin^{-1}\left(\sin(\theta)\right)=\theta$

$\tan\left(\tan^{-1}(x)\right)=x \qquad \tan^{-1}\left(\tan(\theta)\right)=\theta$

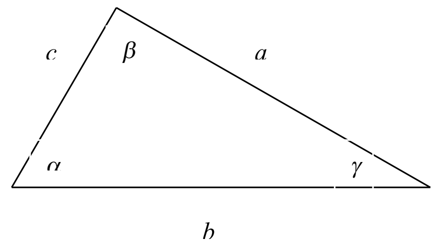
Alternate Notation

$\sin^{-1}x=\arcsin x$

$\cos^{-1}x=\arccos x$

$\tan^{-1}x=\arctan x$

Law of Sines, Cosines and Tangents



Law of Sines

$$\frac{\sin\alpha}{a}=\frac{\sin\beta}{b}=\frac{\sin\gamma}{c}$$

Law of Cosines

$$a^2=b^2+c^2-2bc\cos\alpha$$

$$b^2=a^2+c^2-2ac\cos\beta$$

$$c^2=a^2+b^2-2ab\cos\gamma$$

Mollweide's Formula

$$\frac{a+b}{c}=\frac{\cos\frac{1}{2}(\alpha-\beta)}{\sin\frac{1}{2}\gamma}$$

Law of Tangents

$$\frac{a-b}{a+b}=\frac{\tan\frac{1}{2}(\alpha-\beta)}{\tan\frac{1}{2}(\alpha+\beta)}$$

$$\frac{b-c}{b+c}=\frac{\tan\frac{1}{2}(\beta-\gamma)}{\tan\frac{1}{2}(\beta+\gamma)}$$

$$\frac{a-c}{a+c}=\frac{\tan\frac{1}{2}(\alpha-\gamma)}{\tan\frac{1}{2}(\alpha+\gamma)}$$

© 2005 Paul Dawkins