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"
alias myg++='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

```cpp
#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

```java
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
4. For class sorting, use code `implements Comparable<Class name>`. Or, use code `new Comparator<Interval>() {}` at `Collections.sort()` second argument

2 System Testing

1. Setup vimrc and bashrc
2. Test g++ and Java 8 compiler
3. Look for compilation parameter and code it into bashrc
4. Test if c++ and Java templates work properly on local and judge machine (bits, auto, and other c++11 stuff)
5. Test ”divide by 0” → RE/TLE?
6. Make a complete graph and run Floyd warshall, to test time complexity upper bound
7. Make a linear graph and use DFS to test stack size
8. Test output with extra newline and spaces
9. Go to $Eclipse \rightarrow preference \rightarrow Java \rightarrow Editor \rightarrow ContentAssist$, add $.abcdefghijklmnopqrstuvwxyz$ to auto activation triggers for Java in Eclipse

3 Reminder

1. 隊友的建議，要認真聽！要記得心平氣和的小聲討論喔！通常隊友的建議都會突破你盲點。
2. 每一題都要小心讀，尤其是 IO 的格式和限制都要看清楚。
3. 小心估計時間複查度和 空間複雜度
4. Coding 要兩人一組，要相信你的隊友的實力！
5. 1WA 罰 20 分鐘！放輕鬆，不要急，多產幾組測資後再丟。
6. 範測一定要過！產個幾組極端測資，例如 input 下限、特殊 cases 0, 1, -1、空集合等等
7. 比賽是連續測資，一定要全部讀完再開始 solve 喔！
8. Bus error: 有`scanf, fgets` 但是卻沒東西可以讀取了！可能有 early termination 但是時機不對。

9. 圖論一定要記得檢查連通性。最簡單的做法就是 loop 過所有的點
10. `long long = int * int` 會完蛋
11. `long long int` 的位元運算要記得用 `1LL << 35`
12. 記得清理 Global variable

4 Topic list

1. 列舉、窮舉 enumeration
2. 貪心 greedy
3. 排序 sorting, topological sort
4. 二分搜 binary search (數學算式移項合併後查詢)
5. 爬行法 (右跑左追) Two Pointer
6. 離散化
7. Dynamic programming, 矩陣快速冪
8. 鴿籠原理 Pigeonhole
9. 最近共同祖先 LCA (倍增法, LCA 轉 RMQ)
10. 折半完全列舉 (能用 vector 就用 vector)
11. 離線查詢 Offline (DFS, LCA)
12. 圖的連通性 Directed graph connectivity -> DFS. Undirected graph -> Union Find
13. 因式分解
14. 從答案推回來
15. 寫出數學式，有時就馬上出現答案了！

5 Useful code

5.1 Leap year $O(1)$

```
(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 $O(logn)$

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 $O(log(min(a+b)))$

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 $O(log(min(a+b)))$

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 $O(nloglogn)$

```
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

```
vector/deque
    ::[]: [idx] -> val // O(1)
    ::erase: [it] -> it
    ::erase: [it s, it t] -> it
    ::resize: [sz, val = 0] -> void
    ::insert: [it, val] -> void // insert before it
    ::insert: [it, cnt, val] -> void // insert before it
    ::insert: [it pos, it from_s, it from_t] -> void // insert before it

set/mulitset
    ::insert: [val] -> pair<it, bool> // bool: if val already exist
    ::erase: [val] -> void
    ::erase: [it] -> void
    ::clear: [] -> void
    ::find: [val] -> it
    ::count: [val] -> sz
    ::lower_bound: [val] -> it
    ::upper_bound: [val] -> it
    ::equal_range: [val] -> pair<it, int>
```

```

map/mulitmap
    ::begin/end: [] -> it (*it = pair<key, val>)
    ::[]: [val] -> map_t&
    ::insert: [pair<key, val>] -> pair<it, bool>
    ::erase: [key] -> sz
    ::clear: [] -> void
    ::find: [key] -> it
    ::count: [key] -> sz
    ::lower_bound: [key] -> it
    ::upper_bound: [key] -> it
    ::equal_range: [key] -> it

algorithm
    ::any_of: [it s, it t, unary_func] -> bool // C++11
    ::all_of: [it s, it t, unary_func] -> bool // C++11
    ::none_of: [it s, it t, unary_func] -> bool // C++11
    ::find: [it s, it t, val] -> it
    ::find_if: [it s, it t, unary_func] -> it
    ::count: [it s, it t, val] -> int
    ::count_if: [it s, it t, unary_func] -> int
    ::copy: [it fs, it ft, it ts] -> void // t should be allocated
    ::equal: [it s1, it t1, it s2, it t2] -> bool
    ::remove: [it s, it t, val] -> it (it = new end)
    ::unique: [it s, it t] -> it (it = new end)
    ::random_shuffle: [it s, it t] -> void
    ::lower_bound: [it s, it t, val, binary_func(a, b): a < b] -> it
    ::upper_bound: [it s, it t, val, binary_func(a, b): a < b] -> it
    ::binary_search: [it s, it t, val] -> bool ([s, t) sorted)
    ::merge: [it s1, it t1, it s2, it t2, it o] -> void (o allocated)
    ::includes: [it s1, it t1, it s2, it t2] -> bool (if 2 included in 1)


string::
    ::replace(idx, len, string) -> void
    ::replace(it s1, it t1, it s2, it t2) -> void
string <-> int
    ::stringstream; // remember to clear
    ::sscanf(s.c_str(), "%d", &i);
    ::sprintf(result, "%d", i); string s = result;

numeric
    ::accumulate(it s, it t, val init);

math/cstdlib
    ::atan2(0, -1) -> pi
    ::sqrt(db/ldb) -> db/ldb
    ::fabs(db/ldb) -> db/ldb
    ::abs(int) -> int
    ::ceil(db/ldb) -> db/ldb
    ::floor(db/ldb) -> db/ldb
    ::llabs(ll) -> ll (C++11)
    ::round(db/ldb) -> db/ldb (C99, C++11)
    ::log2(db) -> db (C99)
    ::log2(ldb) -> ldb (C++11)

ctype
    ::toupper(char) -> char (remain same if input is not alpha)
    ::tolower(char) -> char (remain same if input is not alpha)
    ::isupper(char) -> bool
    ::islower(char) -> bool
    ::isalpha(char) -> bool
    ::isdigit(char) -> bool

io printf/scanf
    ::int:                  "%d"       /   "%d"
    ::double:               "%lf","f"  /   "%lf"
    ::string:               "%s"       /   "%s"
    ::long long:            "%lld"     /   "%lld"
    ::long double:          "%Lf"      /   "%Lf"
    ::unsigned int:         "%u"       /   "%u"
    ::unsigned long long:   "%ull"     /   "%ull"
    ::oct:                  "0%o"
    ::hex:                  "0x%x"
    ::scientific:           "%e"
    ::width:                "%05d"
    ::precision:            "%.5f"
    ::adjust left:          "%-5d"

io cin/cout
    ::oct:                  cout << oct << showbase;
    ::hex:                  cout << hex << showbase;
    ::scientific:           cout << scientific;
    ::width:                cout << setw(5);
    ::precision:            cout << fixed << setprecision(5);
    ::adjust left:          cout << setw(5) << left;
```

6 Search

6.1 Ternary Search $O(nlogn)$

```
double l = ..., r = ....; // input
for(int i  = 0; i < 100; i++) {
   double m1 = l + (r - l) / 3, m2 = r - (r - l) / 3;
   if (f (m1) < f (m2)) // f - convex function
      l = m1;
   else
      r = m2;
}
f(r) - maximum of function
```

6.2 N Puzzle

```
const int dr[4] = {0, 0, +1, -1};
const int dc[4] = {+1, -1, 0, 0};
const int dir[4] = {'R', 'L', 'D', 'U'};
const int INF = 0x3f3f3f3f;
const int FOUND = -1;
vector<char> path;
int A[15][15], Er, Ec;

int H() {
    int h = 0;
```

```
    for (int r = 0; r < 4; r++) {
        for (int c = 0; c < 4; c++) {
            if (A[r][c] == 0) continue;
            int expect_r = (A[r][c] - 1) / 4;
            int expect_c = (A[r][c] - 1) % 4;
            h += abs(expect_r - r) + abs(expect_c - c);
        }
    }
    return h;
}

int dfs(int g, int pdir, int bound) {
    int h = H();
    int f = g + h;
    if (f > bound) return f;
    if (h == 0) return FOUND;

    int mn = INF;
    for (int i = 0; i < 4; i++) {
        if (i == (pdir ^ 1)) continue;

        int nr = Er + dr[i];
        int nc = Ec + dc[i];
        if (nr < 0 || nr >= 4) continue;
        if (nc < 0 || nc >= 4) continue;

        path.push_back(dir[i]);
        swap(A[nr][nc], A[Er][Ec]);
        swap(nr, Er); swap(nc, Ec);
        int t = dfs(g + 1, i, bound);
        if (t == FOUND) return FOUND;
        if (t < mn) mn = t;
        swap(nr, Er); swap(nc, Ec);
        swap(A[nr][nc], A[Er][Ec]);
        path.pop_back();
    }

    return mn;
}

bool IDAstar() {
    int bound = H();
    for (;;) {
        int t = dfs(0, -1, bound);
        if (t == FOUND) return true;
        if (t == INF) return false;
        // 下次要搜的 bound >= 50，真的解也一定 >= 50，剪枝
        if (t >= 50) return false;
        bound = t;
    }
    return false;
}

bool solvable() {
    // cnt: 對於每一項 A[r][c] 有多少個小於它且在他之後的數，加總
    // (cnt + Er(1-based) % 2 == 0) <-> 有解
}
```

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];

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) {
^^I// 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

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

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

    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

```
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);
    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

```
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))]);
    }
```

```cpp

    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 Tree

8.1 LCA

```cpp
const int MAX_N = 10000;
const int MAX_LOG_N = 14; // (1 << MAX_LOG_N) > MAX_N

int N;
int root;
int dep[MAX_N];
int par[MAX_LOG_N][MAX_N];

vector<int> child[MAX_N];

void dfs(int u, int p, int d) {
    dep[u] = d;
    for (int i = 0; i < int(child[u].size()); i++) {
        int v = child[u][i];
        if (v != p) {
            dfs(v, u, d + 1);
        }
    }
}

void build() {
    // par[0][u] and dep[u]
    dfs(root, -1, 0);

    // par[i][u]
    for (int i = 0; i + 1 < MAX_LOG_N; i++) {
        for (int u = 0; u < N; u++) {
            if (par[i][u] == -1)
                par[i + 1][u] = -1;
            else
                par[i + 1][u] = par[i][par[i][u]];
        }
    }
}

int lca(int u, int v) {
    if (dep[u] > dep[v]) swap(u, v); // 讓 v 較深
    int diff = dep[v] - dep[u]; // 將 v 上移到與 u 同層
    for (int i = 0; i < MAX_LOG_N; i++) {
        if (diff & (1 << i)) {
            v = par[i][v];
        }
    }

    if (u == v) return u;

    for (int i = MAX_LOG_N - 1; i >= 0; i--) { // 必需倒序
        if (par[i][u] != par[i][v]) {
            u = par[i][u];
            v = par[i][v];
        }
    }
    return par[0][u];
}
```

8.2 Tree Center

```cpp
int diameter = 0, radius[N], deg[N]; // deg = in + out degree
int findRadius()
{
    queue<int> q; // add all leaves in this group
    for (auto i : group)
        if (deg[i] == 1)
            q.push(i);

    int mx = 0;
    while (q.empty() == false) {
        int u = q.front();
        q.pop();

        for (int v : g[u]) {
            deg[v]--;
            if (deg[v] == 1) {
                q.push(v);
                radius[v] = radius[u] + 1;
                mx = max(mx, radius[v]);
            }
        }
    }

    int cnt = 0; // crucial for knowing if there are 2 centers or not
    for (auto j : group)
        if (radius[j] == mx)
            cnt++;

    // add 1 if there are 2 centers (radius, diameter)
    diameter = max(diameter, mx * 2 + (cnt == 2));
    return mx + (cnt == 2);
}
```

8.3 Treap

```cpp
// Remember srand(time(NULL))
struct Treap { // val: bst, pri: heap
    int pri, size, val;
    Treap *lch, *rch;
    Treap() {}
    Treap(int v) {
        pri = rand();
        size = 1;
```

```
        val = v;
        lch = rch = NULL;
    }
};

inline int size(Treap* t) {
    return (t ? t->size : 0);
}
// inline void push(Treap* t) {
//     push lazy flag
// }
inline void pull(Treap* t) {
    t->size = 1 + size(t->lch) + size(t->rch);
}

int NN = 0;
Treap pool[30000];

Treap* merge(Treap* a, Treap* b) { // a < b
    if (!a || !b) return (a ? a : b);
    if (a->pri > b->pri) {
        // push(a);
        a->rch = merge(a->rch, b);
        pull(a);
        return a;
    }
    else {
        // push(b);
        b->lch = merge(a, b->lch);
        pull(b);
        return b;
    }
}

void split(Treap* t, Treap*& a, Treap*& b, int k) {
    if (!t) { a = b = NULL; return; }
    // push(t);
    if (size(t->lch) < k) {
        a = t;
        split(t->rch, a->rch, b, k - size(t->lch) - 1);
        pull(a);
    }
    else {
        b = t;
        split(t->lch, a, b->lch, k);
        pull(b);
    }
}

// get the rank of val
// result is 1-based
int get_rank(Treap* t, int val) {
    if (!t) return 0;
    if (val < t->val)
        return get_rank(t->lch, val);
    else
        return get_rank(t->rch, val) + size(t->lch) + 1;
}

// get kth smallest item
// k is 1-based
Treap* get_kth(Treap*& t, int k) {
    Treap *a, *b, *c, *d;
    split(t, a, b, k - 1);
    split(b, c, d, 1);
    t = merge(a, merge(c, d));
    return c;
}

void insert(Treap*& t, int val) {
    int k = get_rank(t, val);
    Treap *a, *b;
    split(t, a, b, k);
    pool[NN] = Treap(val);
    Treap* n = &pool[NN++];
    t = merge(merge(a, n), b);
}

// Implicit key treap init
void insert() {
    for (int i = 0; i < N; i++) {
        int val; scanf("%d", &val);
        root = merge(root, new_treap(val)); // implicit key(index)
    }
}
```

9 Graph

9.1 Articulation point / Bridge

```
// timer = 1, dfs arrays init to 0, set root carefully!
int timer, dfsTime[N], dfsLow[N], root;
bool articulationPoint[N]; // set<ii> bridge;
void findArticulationPoint(int u, int p)
{
    dfsTime[u] = dfsLow[u] = timer++;

    int child = 0; // root child counter for articulation point
    for(auto v : g[u]) { // vector<int> g[N]; // undirected graph
        if(v == p) // don't go back to parent
            continue;

        if(dfsTime[v] == 0) {
            child++; // root child counter for articulation point
            findArticulationPoint(v, u);
            dfsLow[u] = min(dfsLow[u], dfsLow[v]);

            // <= for articulation point, < for bridge
            if(dfsTime[u] <= dfsLow[v] && root != u)
                articulationPoint[u] = true;
            // special case for articulation point root only
```

```
            if(u == root && child >= 2)
                articulationPoint[u] = true;
        } else { // visited before (back edge)
            dfsLow[u] = min(dfsLow[u], dfsTime[v]);
        }
    }
}
```

9.2 2-SAT

$$
\begin{aligned}
&(x_i \vee x_i)\ \text{建邊}(\neg x_i,\, x_j)\\
&(x_i \vee x_j)\ \text{建邊}(\neg x_i,\, x_j),\, (\neg x_j,\, x_i)\\
&p \vee (q \wedge r)\\
&\ = ((p \wedge q) \vee (p \wedge r))\\
&p \oplus q\\
&\ = \neg((p \wedge q) \vee (\neg p \wedge \neg q))\\
&\ = (\neg p \vee \neg q) \wedge (p \vee q)
\end{aligned}
$$

```
// 建圖
// (x1 or x2) and ... and (xi or xj)
// (xi or xj) 建邊
// ~xi -> xj
// ~xj -> xi

tarjan(); // scc 建立的順序是倒序的拓璞排序
for (int i = 0; i < 2 * N; i += 2) {
    if (belong[i] == belong[i ^ 1]) {
        // 無解
    }
}
for (int i = 0; i < 2 * N; i += 2) { // 迭代所有變數
    if (belong[i] < belong[i ^ 1]) { // i 的拓璞排序比 ~i 的拓璞排序大
        // i = T
    }
    else {
        // i = F
    }
}
```

9.3 CC

9.3.1 BCC

以 Edge 做分界的話，stack 要裝入 (u - v)，並 pop 終止條件為!= (u - v)
以 Articulation point 做為分界 (code below)，注意有無坑人的重邊

```
int cnt, root, dfsTime[N], dfsLow[N], timer, group[N]; // max N nodes
stack<int> s;
bool in[N];
void dfs(int u, int p)
{
    s.push(u);
    in[u] = true;

    dfsTime[u] = dfsLow[u] = timer++;

    for (int i = 0; i < (int)g[u].size(); i++) {
        int v = g[u][i];

        if (v == p)
            continue;

        if (dfsTime[v] == 0) {
            dfs(v, u);
            dfsLow[u] = min(dfsLow[u], dfsLow[v]);
        } else {
            if (in[u]) // gain speed
                dfsLow[u] = min(dfsLow[u], dfsTime[v]);
        }
    }

    if (dfsTime[u] == dfsLow[u]) { //dfsLow[u]== dfsTime[u] -> SCC found
        cnt++;
        while (true) {
            int v = s.top();
            s.pop();
            in[v] = false;

            group[v] = cnt;
            if (v == u)
                break;
        }
    }
}

// get SCC degree
int deg[n + 1];
memset(deg, 0, sizeof(deg));
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < (int)g[i].size(); j++) {
        int v = g[i][j];
        if (group[i] != group[v])
            deg[group[i]]++;
    }
}
```

9.3.2 SCC

First of all we run DFS on the graph and sort the vertices in decreasing of their finishing time (we can use a stack).
Then, we start from the vertex with the greatest finishing time, and for each vertex v that is not yet in any SCC, do : for each u that v is reachable by u and u is not yet in any SCC, put it in the SCC of vertex v. The code is quite simple.

```
const int MAX_V = ...;
const int INF = 0x3f3f3f3f;
int V;
vector<int> g[MAX_V];

int dfn_idx = 0;
int scc_cnt = 0;
int dfn[MAX_V];
int low[MAX_V];
int belong[MAX_V];
bool in_st[MAX_V];
```

```cpp
vector<int> st;

void scc(int v) {
    dfn[v] = low[v] = dfn_idx++;
    st.push_back(v);
    in_st[v] = true;

    for (int i = 0; i < int(g[v].size()); i++) {
        const int u = g[v][i];
        if (dfn[u] == -1) {
            scc(u);
            low[v] = min(low[v], low[u]);
        }
        else if (in_st[u]) {
            low[v] = min(low[v], dfn[u]);
        }
    }

    if (dfn[v] == low[v]) {
        int k;
        do {
            k = st.back(); st.pop_back();
            in_st[k] = false;
            belong[k] = scc_cnt;
        } while (k != v);
        scc_cnt++;
    }
}

void tarjan() { // scc 建立的順序即為反向的拓璞排序
    st.clear();
    fill(dfn, dfn + V, -1);
    fill(low, low + V, INF);
    dfn_idx = 0;
    scc_cnt = 0;
    for (int v = 0; v < V; v++) {
        if (dfn[v] == -1) {
            scc(v);
        }
    }
}
```

9.4 Shortest Path

Time complexity notations: V = vertex, E = edge
Minimax: $dp[u][v] = min(dp[u][v], max(dp[u][k], dp[k][v]))$

9.4.1 Dijkatra (next-to-shortest path)

密集圖別用 priority queue!

```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];
}
```

9.4.2 SPFA

```cpp
typedef pair<int, int> ii;
vector< ii > g[N];

bool SPFA()
{
    vector<ll> d(n, INT_MAX);
    d[0] = 0; // origin

    queue<int> q;
    vector<bool> inqueue(n, false);
    vector<int> cnt(n, 0);
    q.push(0);
    inqueue[0] = true;
    cnt[0]++;
```

```

    while(q.empty() == false) {
        int u = q.front();
        q.pop();
        inqueue[u] = false;

        for(auto i : g[u]) {
            int v = i.first, w = i.second;
            if(d[u] + w < d[v]) {
                d[v] = d[u] + w;
                if(inqueue[v] == false) {
                    q.push(v);
                    inqueue[v] = true;
                    cnt[v]++;

                    if(cnt[v] == n) { // loop!
                        return true;
                    }
                }
            }
        }
    }

    return false;
}
```

9.4.3 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] != INT_MAX && d[u] + w < d[v]) {
                hasChange = true;
                d[v] = d[u] + w;
            }
        }

        if (i == n - 1 && hasChange == true)
            loop = true;
        else if (hasChange == false)
            break;
    }
}
```

9.4.4 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]);
```

9.5 MST

9.5.1 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

9.5.2 Prim

```
int ans = 0;
bool used[n];
memset(used, false, sizeof(used));

priority_queue<ii, vector<ii>, greater<ii>> pq;
pq.push(ii(0, 0)); // push (0, origin)
while (!pq.empty())
{
    ii cur = pq.top();
    pq.pop();

    int u = cur.second;
    if (used[u])
        continue;
    ans += cur.first;
    used[u] = true;

    for (int i = 0; i < (int)g[u].size(); i++) {
        int v = g[u][i].first, w = g[u][i].second;
        if (used[v] == false)
            pq.push(ii(w, v));
    }
}
```

10 Flow

10.1 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;
        }
    }
}
```

10.2 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()});
    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;
}
```

10.3 Bipartite Matching

```
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］每次遞迴都找一個未匹配點 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;
}
```

11 String

11.1 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;
}
```

11.2 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;
}
```

11.3 Z Algorithm

```
int len = strlen(inp), z[len];
z[0] = 0; // initial

int l = 0, r = 0; // z box bound [l, r]
for (int i = 1; i < len; i++)
{
    if (i > r) { // i not in z box
        l = r = i; // z box contains itself only
        while (r < len && inp[r - l] == inp[r])
            r++;
        z[i] = r - l;
        r--;
    } else { // i in z box
        if (z[i - l] + i < r) // over shoot R bound
            z[i] = z[i - l];
        else {
            l = i;
            while (r < len && inp[r - l] == inp[r])
                r++;
            z[i] = r - l;
            r--;
        }
    }
}
```

11.4 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 Matrix

12.1 Gauss Jordan

```
typedef long long ll;
typedef vector<ll> vec;
typedef vector<vec> mat;

vec gauss_jordan(mat A) {
    int n = A.size(), m = A[0].size();
    for (int i = 0; i < n; i++) {
        // float: find j s.t. A[j][i] is max
        // mod: find min j s.t. A[j][i] is not 0
        int pivot = i;
        for (int j = i; j < n; j++) {
            // if (fabs(A[j][i]) > fabs(A[pivot])) {
            //     pivot = j;
            // }
            if (A[pivot][i] != 0) {
                pivot = j;
                break;
            }
        }

        swap(A[i], A[pivot]);
        if (A[i][i] == 0) { // if (fabs(A[i][i]) < eps)
            // 無解或無限多組解
            // 可改成 continue, 全部做完後再判
            return vec();
        }

        ll divi = inv(A[i][i]);
        for (int j = i; j < m; j++) {
            // A[i][j] /= A[i][i];
            A[i][j] = (A[i][j] * divi) % MOD;
        }

        for (int j = 0; j < n; j++) {
            if (j != i) {
                for (int k = i + 1; k < m; k++) {
                    // A[j][k] -= A[j][i] * A[i][k];
                    ll p = (A[j][i] * A[i][k]) % MOD;
                    A[j][k] = (A[j][k] - p + MOD) % MOD;
                }
            }
        }
    }

    vec x(n);
    for (int i = 0; i < n; i++)
        x[i] = A[i][m - 1];
    return x;
}
```

12.2 Determinant

```
typedef long long ll;
typedef vector<ll> vec;
typedef vector<vec> mat;

ll determinant(mat m) { // square matrix
    const int n = m.size();
    ll det = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int a = i, b = j;
            while (m[b][i]) {
                ll q = m[a][i] / m[b][i];
                for (int k = 0; k < n; k++)
                    m[a][k] = m[a][k] - m[b][k] * q;
                swap(a, b);
            }

            if (a != i) {
                swap(m[i], m[j]);
                det = -det;
            }
        }

        if (m[i][i] == 0)
            return 0;
        else
            det *= m[i][i];
    }
    return det;
}
```

13 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

13.1 EPS

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

13.2 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));

        // 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()
```

14 Math

14.1 Euclid's formula (Pythagorean Triples)

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

14.2 Difference between two consecutive numbers' square is odd

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

14.3 Summation

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

14.4 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 {
```

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

    return c;
}
```

14.5 Combination

14.5.1 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
        }
    }
}
```

14.5.2 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;
}
```

14.6 Chinese remainder theorem

```
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
    };
}
```

14.7 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 $d > R - r$

14.8 Fun Facts

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

14.9 2^n table

1 : 2
2 : 4
3 : 8
4 : 16
5 : 32
6 : 64
7 : 128
8 : 256
9 : 512
10 : 1024
11 : 2048
12 : 4096
13 : 8192
14 : 16384
15 : 32768
16 : 65536
17 : 131072
18 : 262144
19 : 524288
20 : 1048576
21 : 2097152
22 : 4194304
23 : 8388608
24 : 16777216
25 : 33554432

15 Dynamic Programming - Problems collection

```
// # 零一背包 (poj 1276)
fill(dp, dp + W + 1, 0);
for (int i = 0; i < N; i++)
    for (int j = W; j >= items[i].w; j--)
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
return dp[W];

// # 多重背包二進位拆解 (poj 1276)
for_each(ll v, w, num) {
    for (ll k = 1; k <= num; k *= 2) {
        items.push_back((Item) {k * v, k * w});
        num -= k;
    }
    if (num > 0)
        items.push_back((Item) {num * v, num * w});
}

// # 完全背包
// dp[i][j] = 前 i + 1 個物品, 在重量 j 下所能組出的最大價值
// 第 i 個物品, 不放或至少放一個
```

```
// dp[i][j] = max(dp[i - 1][j], dp[i][j - w[i]] + v[i])
fill(dp, dp + W + 1, 0);
for (int i = 0; i < N; i++)
    for (int j = w[i]; j <= W; j++)
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
return dp[W];

// # Coin Change (2015 桂冠賽 E)
// dp[i][j] = 前 i + 1 個物品，組出 j 元的方法數
// 第 i 個物品，不用或用至少一個
// dp[i][j] = dp[i - 1][j] + dp[i][j - coin[i]]

// # Cutting Sticks (2015 桂冠賽 F)
// 補上二個切點在最左與最右
// dp[i][j] = 使 (i, j) 區間中的所有切點都被切的最小成本
// dp[i][j] = min(dp[i][c] + dp[c][j] + (p[j] - p[i]) for i < c < j)
// dp[i][i + 1] = 0
// ans = dp[0][N + 1]

// # Throwing a Party (itsa dp 06)
// 給定一棵有根樹，代表公司職位層級圖，每個人有其權重，現從中選一個點集合出來，
// 且一個人不能與其上司一都在集合中，並最大化集合的權重和，輸出該總和。
// dp[u][0/1] = u 在或不在集合中，以 u 為根的子樹最大權重和
// dp[u][0] = max(max(dp[c][0], dp[c][1]) for children c of u) + val[u]
// dp[u][1] = max(dp[c][0] for children c of u)
// bottom up dp

// # LIS (O(N^2))
// dp[i] = 以 i 為結尾的 LIS 的長度
// dp[i] = max(dp[j] for 0 <= j < i) + 1
// ans = max(dp)

// # LIS (O(nlgn)), poj 1631
// dp[i] = 長度為 i + 1 的 LIS 的最後一項的最小值，不存在時為 INF
fill(dp, dp + N, INF);
for (int i = 0; i < N; i++)
    *lower_bound(dp, dp + N, A[i]) = A[i];
ans = lower_bound(dp, dp + N, INF) - dp;

// # Maximum Subarray

// # Not equal on a Segment (cf edu7 C)
// 給定長度為 n 的陣列 a[] 與 m 個詢問。
// 針對每個詢問 l, r, x 請輸出 a[l, r] 中不等於 x 的任一位置。
// 不存在時輸出 -1
// dp[i] = max j such that j < i and a[j] != a[i]
// dp[0] = -1
// dp[i] = dp[i - 1] if a[i] == a[i - 1] else i - 1
// 針對每筆詢問 l, r, x
// 1. a[r] != x               -> 輸出 r
// 2. a[r] = x && dp[r] >= l  -> 輸出 dp[r]
// 3. a[r] = x && dp[r] < l   -> 輸出 -1

// # bitmask dp, poj 2686
// 給定一個無向帶權圖，代表 M 個城市之間的路，與 N 張車票，
// 每張車票有一個數值 t[i]，若欲使用車票 t[i] 從城市 u 經由路徑 d[u][v] 走到城市 v，
// 所花的時間為 d[u][v] / t[i]。請問，從城市 A 走到城市 B 最快要多久?
// dp[S][v] = 從城市 A 到城市 v 的最少時間，其中 S 為用過的車票的集合
// 考慮前一個城市 u 是誰，使用哪個車票 t[i] 而來，可以得到轉移方程式:
// dp[S][v] = min([
//     dp[S - {v}][u] + d[u][v] / t[i]
//     for all city u has edge to v, for all ticket in S
// ])

// # Tug of War
// N 個人參加拔河比賽，每個人有其重量 w[i]，欲使二隊的人數最多只差一，雙方的重量和越接近越好
// 請問二隊的重量和分別是多少?
// dp[i][j][k] = 只考慮前 i + 1 個人，可不可以使左堆的重量為 j，且左堆的人數為 k
// dp[i][j][k] = dp[i - 1][j - w[i][k - 1] or dp[i - 1][j][k]
// dp[i][j] = (dp[i - 1][j - w[i]] << 1) | (dp[i - 1][j])

// # Modulo Sum (cf 319 B)
// 給定長度為 N 的序列 A 與一正整數 M，請問該序列中有無一個子序列，子序列的總合是 M 的倍數
// 若 N > M，則根據鴿籠原理，必有至少兩個前綴和的值 mod M 為相同值，解必定存在
// dp[i][j] = 前 i + 1 個數可否組出 mod m = j 的數
// dp[i][j] = true if
//     dp[i - 1][(j - (a[i] mod m)) mod m] or
//     dp[i - 1][j] or
//     j = a[i] % m

// # POJ 2229
// 給定正整數 N，請問將 N 拆成一堆 2^x 之和的方法數
// dp[i] = 拆解 N 的方法數
// dp[i] = dp[i / 2] if i is odd
//       = dp[i - 1] + dp[i / 2] if i is even

// # POJ 3616
// 給定 N 個區間 [s, t)，每個區間有權重 w[i]，從中選出一些不相交的區間，使權重和最大
// dp[i] = 考慮前 i + 1 個區間，且必選第 i 個區間的最大權重和
// dp[i] = max(dp[j] | 0 <= j < i) + w[i]
// ans = max(dp)

// # POJ 2184
// N 隻牛每隻牛有權重 <s, f>，從中選出一些牛的集合，
// 使得 sum(s) + sum(f) 最大，且 sum(s) > 0, sum(f) > 0。
// 枚舉 sum(s) , 將 sum(s) 視為重量對 f 做零一背包。

// # POJ 3666
// 給定長度為 N 的序列，請問最少要加多少值，使得序列單調遞增
// dp[i][j] = 使序列前 i+1 項變為單調，且將 A[i] 變為「第 j 小的數」的最小成本
// dp[i][j] = min(dp[i - 1][k] | 0 <= k <= j) + abs(S[j] - A[i])
// min(dp[i - 1][k] | 0 <= k <= j) 動態維護
for (int j = 0; j < N; j++)
    dp[0][j] = abs(S[j] - A[0]);
for (int i = 1; i < N; i++) {
    int pre_min_cost = dp[i][0];
    for (int j = 0; j < N; j++) {
        pre_min_cost = min(pre_min_cost, dp[i-1][j]);
```

```
        dp[i][j] = pre_min_cost + abs(S[j] - A[i]);
    }
}
ans = min(dp[N - 1])

// # POJ 3734
// N 個 blocks 上色，R, G, Y, B，上完色後紅色的數量與綠色的數量都要是偶數。請問方法數。
// dp[i][0/1/2/3] = 前 i 個 blocks 上完色，紅色數量為奇數/偶數，綠色數量為數/偶數
// 用遞推，考慮第 i + 1 個 block 的顏色，找出個狀態的轉移，整理可發現
// dp[i + 1][0] = dp[i][2] + dp[i][1] + 2 * dp[i][0]
// dp[i + 1][1] = dp[i][3] + dp[i][0] + 2 * dp[i][1]
// dp[i + 1][2] = dp[i][0] + dp[i][3] + 2 * dp[i][2]
// dp[i + 1][3] = dp[i][1] + dp[i][2] + 2 * dp[i][3]
// 矩陣快速冪加速求 dp[N - 1][0][0]

// # POJ 3171
// 數線上，給定 N 個區間 [s[i], t[i]]，每個區間有其代價，求覆蓋區間 [M, E] 的最小代價。
// dp[i][j] = 最多使用前 i + 1 個區間，使 [M, j] 被覆蓋的最小代價
// 考慮第 i 個區間用或不用，可得:
// dp[i][j] =
//     1. min(dp[i - 1][k] for k in [s[i] - 1, t[i]]) + cost[i] if j = t[i]
//     2. dp[i - 1][j] if j ≠ t[i]
// 壓空間，使用線段樹加速。
// dp[t[i]] = min(dp[t[i]],
//     min(dp[i - 1][k] for k in [s[i] - 1, t[i]]) + cost[i]
// )
fill(dp, dp + E + 1, INF);
seg.init(E + 1, INF);
int idx = 0;
while (idx < N && A[idx].s == 0) {
    dp[A[idx].t] = min(dp[A[idx].t], A[idx].cost);
    seg.update(A[idx].t, A[idx].cost);
    idx++;
}
for (int i = idx; i < N; i++) {
    ll v = min(dp[A[i].t], seg.query(A[i].s - 1, A[i].t + 1) + A[i].cost);
    dp[A[i].t] = v;
    seg.update(A[i].t, v);
}
```

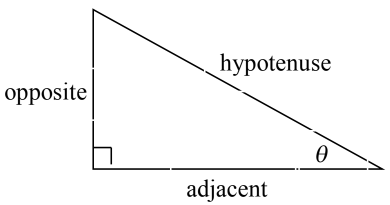
Trig Cheat Sheet

Definition of the Trig Functions

Right triangle definition

For this definition we assume that

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



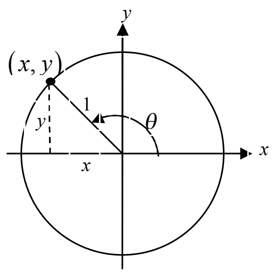
$$\sin\theta = \frac{\text{opposite}}{\text{hypotenuse}} \qquad \csc\theta = \frac{\text{hypotenuse}}{\text{opposite}}$$

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

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

Unit circle definition

For this definition θ is any angle.



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

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

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

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

$\tan\theta$, $\theta \neq \left(n+\frac{1}{2}\right)\pi,\quad n = 0, \pm 1, \pm 2, \ldots$

$\csc\theta$, $\theta \neq n\pi,\quad n = 0, \pm 1, \pm 2, \ldots$

$\sec\theta$, $\theta \neq \left(n+\frac{1}{2}\right)\pi,\quad n = 0, \pm 1, \pm 2, \ldots$

$\cot\theta$, $\theta \neq n\pi,\quad n = 0, \pm 1, \pm 2, \ldots$

Range

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

$$-1 \le \sin\theta \le 1 \qquad \csc\theta \ge 1 \text{ and } \csc\theta \le -1$$

$$-1 \le \cos\theta \le 1 \qquad \sec\theta \ge 1 \text{ and } \sec\theta \le -1$$

$$-\infty < \tan\theta < \infty \qquad -\infty < \cot\theta < \infty$$

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.

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

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

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

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

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

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

Formulas and Identities

Tangent and Cotangent Identities

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

Reciprocal Identities

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

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

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

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

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

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

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

Periodic Formulas

If n is an integer.

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

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

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

Double Angle Formulas

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

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

$$= 2\cos^2\theta - 1$$

$$= 1 - 2\sin^2\theta$$

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

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)**

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

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

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

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

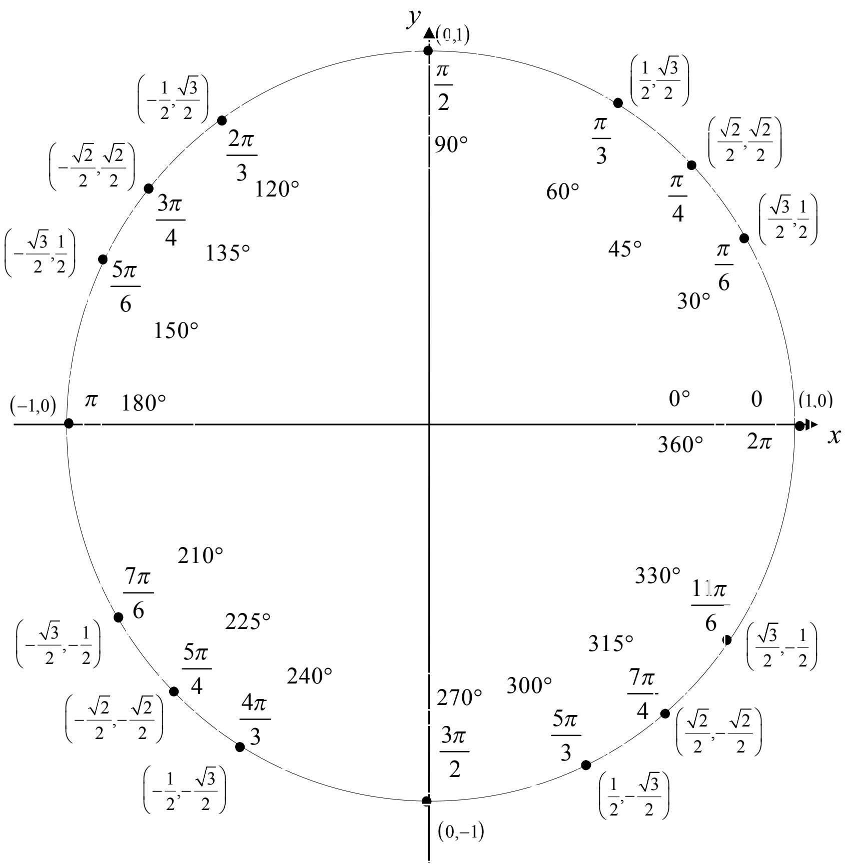
$$\sin\left(\frac{\pi}{2} - \theta\right) = \cos\theta \qquad \cos\left(\frac{\pi}{2} - \theta\right) = \sin\theta$$

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

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

© 2005 Paul Dawkins

Unit Circle



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

Example

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

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

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

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

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

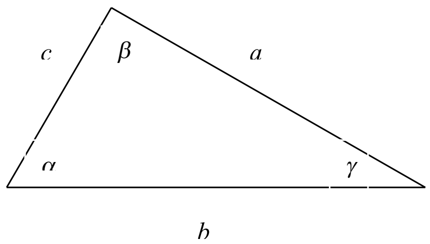
Alternate Notation

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

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

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

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