# Contents



# 1 Init (Linux)

開場流程：

```
vim ~/.vimrc
mkdir contest && cd contest

vim template.cpp
for c in {A..P}; do
    cp template.cpp $c.cpp
done

vim run.sh && chmod 777 run.sh
```

## 1.1 vimrc

```
syn on
set nu rnu ru cul mouse=a
set cin et ts=4 sw=4 sts=4
set autochdir
set clipboard=unnamedplus

colo koehler

no <C-h> ^
no <C-l> $
no ; :

inoremap {<CR> {<CR>}<Esc>ko
```

## 1.2 template.cpp

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

void solve() {
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(0);
    int TEST = 1;
    //cin >> TEST;
    while (TEST--) solve();
    return 0;
}
```

## 1.3 run.sh

```
#!/bin/bash

g++ -std=c++17 -O2 -g -fsanitize=undefined,address $1
    && echo DONE COMPILE || exit 1
./a.out
```

# 2 Reminder

## 2.1 Observations and Tricks

- Contribution Technique
- 二分圖/Spanning Tree/DFS Tree
- 行、列操作互相獨立

- 奇偶性
- 當 $s,t$ 遞增並且 $t=f(s)$，對 $s$ 二分搜不好做，可以改成對 $t$ 二分搜，再算 $f(t)$
- 啟發式合併
- Permutation Normalization（做一些平移對齊兩個 permutation）
- 枚舉 $a_1 \sim a_n$ 再枚舉 $a_n \sim a_1$ 可以包在一個迴圈
- 兩個凸型函數相加還是凸型函數，相減不一定

## 2.2 Bug List

- 沒開 long long
- 陣列戳出界／陣列開不夠大
- 寫好的函式忘記呼叫
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- <= 打成 <+
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case
- 線段樹改值懶標初始值不能設為 0
- DFS 的時候不小心覆寫到全域變數
- 浮點數誤差
- unsigned int128
- 多筆測資不能沒讀完直接 return
- 記得刪 cerr
- vector 超級肥，小 vector 請用 array，例如矩陣快速冪

# 3 Basic

## 3.1 template (optional)

```cpp
#define F first
#define S second
#define ep emplace
#define eb emplace_back
#define endl '\n'

template<class T> using V=vector<T>;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef pair<int, ll> pil;
typedef pair<ll, int> pli;

/* ========================================== */
// STL and I/O
// pair
template<typename T1, typename T2>
ostream& operator<<(ostream& os, pair<T1, T2> p) {
    return os << "(" << p.first << ", " << p.second << ")";
}
template<typename T1, typename T2>
istream& operator>>(istream& is, pair<T1, T2>& p) { return is >> p.first >> p.second; }

// vector
template<typename T>
istream& operator>>(istream& is, vector<T>& v) {
    for (auto& x : v) is >> x;
    return is;
}
template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
    for (const auto& x : v) os << x << ' ';
    return os;
}
/* ========================================== */
// debug(), output()
#define RED     "\x1b[31m"
#define GREEN   "\x1b[32m"
#define YELLOW  "\x1b[33m"
#define GRAY    "\x1b[90m"
#define COLOREND "\x1b[0m"

void _debug() {}
template<typename A,typename... B> void _debug(A a,B... b) { cerr << a << ' ', _debug(b...); }
#define debug(...) cerr<<GRAY<<#__VA_ARGS__<<": "<<COLOREND,_debug(__VA_ARGS__),cerr<<endl

void _output() {}
template<typename A,typename... B> void _output(A a,B... b) { cout << a << ' ', _output(b...); }
#define output(...) _output(__VA_ARGS__),cout<<endl
/* ========================================== */
// BASIC ALGORITHM
string binary(ll x, int b = -1) {
    if (b == -1) b = __lg(x) + 1;
    string s = "";
    for (int k = b - 1; k >= 0; k--) {
        s.push_back((x & (1LL<<k)) ? '1' : '0');
    }
    return s;
}
/* ========================================== */
// CONSTANT
const int INF = 1.05e9;
const ll LINF = 4e18;
const int MOD = 1e9 + 7;
//const int MOD = 998244353;
const int maxn = 2e5 + 3;
```

## 3.2 Stress

```sh
g++ gen.cpp -o gen.out
g++ ac.cpp -o ac.out
g++ wa.cpp -o wa.out
for ((i=0;;i++))
do
    echo "$i"
    ./gen.out > in.txt
    ./ac.out < in.txt > ac.txt
    ./wa.out < in.txt > wa.txt
    diff ac.txt wa.txt || break
done
```

## 3.3 PBDS

```cpp
#include <bits/extc++.h>
using namespace __gnu_pbds;

// map
tree<int, int, less<>, rb_tree_tag,
    tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// set
tree<int, null_type, less<>, rb_tree_tag,
    tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// priority queue
__gnu_pbds::priority_queue<int, less<int> > big_q;  // Big First
__gnu_pbds::priority_queue<int, greater<int> > small_q;  // Small First
q1.join(q2);  // join
```

## 3.4 Random

```cpp
mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
#define RANDINT(a, b) uniform_int_distribution<int> (a, b)(rng) // inclusive
#define RANDLL(a, b) uniform_int_distribution<long long>(a, b)(rng)  // inclusive
#define RANDFLOAT(a, b) uniform_real_distribution<float>(a, b)(rng)  // exclusive
```

```cpp
#define RANDDOUBLE(a, b) uniform_real_distribution<
    double>(a, b)(rng)  // exclusive
shuffle(v.begin(), v.end(), gen);
```

# 4 Python

## 4.1 I/O

```python
import sys
input = sys.stdin.readline

# Input
def readInt():
    return int(input())
def readIntList():
    return list(map(int,input().split()))
def readStr():
    s = input()
    return list(s[:len(s) - 1])
def readInts():
    return map(int,input().split())
```

## 4.2 Decimal

```python
from decimal import *
getcontext().prec = 500  # precision
getcontext().Emax = 500  # 科學記號指數最大值
# 將東西轉成 Decimal
Decimal(x)
Decimal(y)
Decimal(0.0)
# 運算子跟一般浮點數一樣
x *= y
# 輸出
print(x.quantize(Decimal("0.000001"), rounding=
    ROUND_HALF_EVEN))
# ROUND_CEILING     (2.9=>3, -2.1=>-2)
# ROUND_FLOOR       (2.1=>2, -2.9=>-3)
# ROUND_HALF_EVEN   (2.5=>2, 3.5=>4)
# ROUND_HALF_UP     (2.5=>3, -2.5=>-3)
# ROUND_HALF_DOWN   (2.5=>2, -2.5=>-2)
# ROUND_UP          (2.1=>3, -2.1=>-3)
# ROUND_DOWN        (2.9=>2, -2.9=>-2)
```

# 5 Data Structure

## 5.1 Mo's Algorithm

```cpp
// segments are 0-based
ll cur = 0;  // current answer
int pl = 0, pr = -1;
for (auto& qi : Q) {
    // get (l, r, qid) from qi
    while (pl < l) del(pl++);
    while (pl > l) add(--pl);
    while (pr < r) add(++pr);
    while (pr > r) del(pr--);
    ans[qid] = cur;
}
```

## 5.2 Segment Tree

```cpp
// Author: Gino
struct node {
    ll sum, add, mod; int ln;
    node(): sum(0), add(0), mod(0), ln(0) {}
};

struct segT {
    int n;
    vector<ll> ar;
    vector<node> st;

    void init(int _n) {
        n = _n;
        reset(ar, n, 0LL);
        reset(st, n*4);
    }
    void pull(int cl, int cr, int i) {
        st[i].sum = st[cl].sum + st[cr].sum;
    }
    void push(int cl, int cr, int i) {
        ll md = st[i].mod, ad = st[i].add;
        if (md) {
            st[cl].sum = md * st[cl].ln, st[cr].sum =
                md * st[cr].ln;
            st[cl].mod = md, st[cr].mod = md;
            st[i].mod = 0;
        }
        if (ad) {
            st[cl].sum += ad * st[cl].ln, st[cr].sum +=
                ad * st[cr].ln;
            st[cl].add += ad, st[cr].add += ad;
            st[i].add = 0;
        }
    }
    void build(int l, int r, int i) {
        if (l == r) {
            st[i].sum = ar[l];
            st[i].ln = 1;
            return;
        }
        int mid = (l+r)>>1, cl = i<<1, cr = i<<1|1;
        build(l, mid, cl);
        build(mid + 1, r, cr);
        pull(cl, cr, i);
        // DONT FORGET THIS
        st[i].ln = st[cl].ln + st[cr].ln;
    }
    void addval(int ql, int qr, ll val, int l, int r,
        int i) {
        if (qr < l || r < ql) return;
        if (ql <= l && r <= qr) {
            st[i].sum += val * st[i].ln;
            st[i].add += val;
            return;
        }
        int mid = (l+r)>>1, cl = i<<1, cr = i<<1|1;
        push(cl, cr, i);
        addval(ql, qr, val, l, mid, cl);
        addval(ql, qr, val, mid + 1, r, cr);
        pull(cl, cr, i);
    }
    void modify(int ql, int qr, ll val, int l, int r,
        int i) {
        if (qr < l || r < ql) return;
        if (ql <= l && r <= qr) {
            st[i].sum = val * st[i].ln;
            st[i].add = 0;
            st[i].mod = val;
            return;
        }
        int mid = (l+r)>>1, cl = i<<1, cr = i<<1|1;
        push(cl, cr, i);
        modify(ql, qr, val, l, mid, cl);
        modify(ql, qr, val, mid+1, r, cr);
        pull(cl, cr, i);
    }
    ll query(int ql, int qr, int l, int r, int i) {
        if (qr < l || r < ql) return 0;
        if (ql <= l && r <= qr) return st[i].sum;
        int mid = (l+r)>>1, cl = i<<1, cr = i<<1|1;
        push(cl, cr, i);
        return (query(ql, qr, l, mid, cl) +
                query(ql, qr, mid+1, r, cr));
    }
};
```

## 5.3 Heavy Light Decomposition

```cpp
// Author: Ian
void build(V<int>&v);
void modify(int p, int k);
int query(int ql, int qr);
// Insert [ql, qr) segment tree here
inline void solve(){
```

```
  int n, q; cin >> n >> q;
  V<int> v(n);
  for (auto& i: v) cin >> i;
  V<V<int>> e(n);
  for(int i = 1; i < n; i++){
    int a, b; cin >> a >> b, a--, b--;
    e[a].emplace_back(b);
    e[b].emplace_back(a);
  }
  V<int> d(n, 0), f(n, 0), sz(n, 1), son(n, -1);
  F<void(int, int)> dfs1 = [&](int x, int pre) {
    for (auto i: e[x]) if (i != pre) {
      d[i] = d[x]+1, f[i] = x;
      dfs1(i, x), sz[x] += sz[i];
      if (son[x] == -1 || sz[son[x]] < sz[i])
        son[x] = i;
    }
  }; dfs1(0,0);
  V<int> top(n, 0), dfn(n, -1);
  F<void(int,int)> dfs2 = [&](int x, int t) {
    static int cnt = 0;
    dfn[x] = cnt++, top[x] = t;
    if (son[x] == -1) return;
    dfs2(son[x], t);
    for (auto i: e[x]) if (!~dfn[i])
      dfs2(i,i);
  }; dfs2(0,0);
  V<int> dfnv(n);
  for (int i = 0; i < n; i++)
    dfnv[dfn[i]] = v[i];
  build(dfnv);
  while(q--){
    int op, a, b, ans; cin >> op >> a >> b;
    switch(op){
      case 1:
        modify(dfn[a-1], b);
        break;
      case 2:
        a--, b--, ans = 0;
        while (top[a] != top[b]) {
          if (d[top[a]] > d[top[b]]) swap(a,b);
          ans = max(ans, query(dfn[top[b]], dfn[b]+1));
          b = f[top[b]];
        }
        if (dfn[a] > dfn[b]) swap(a,b);
        ans = max(ans, query(dfn[a], dfn[b]+1));
        cout << ans << endl;
        break;
    }
  }
}
```

## 5.4 Skew Heap

```
// Author: Ian
// Function: min-heap, with amortized O(lg n) merge
struct node {
  node *l, *r; int v;
  node(int x): v(x) { l = r = nullptr; }
};
node* merge(node* a,node* b) {
  if (!a || !b) return a ?: b;
  if (a->v > b->v) swap(a, b);
  return a->r = merge(a->r, b), swap(a->l, a->r), a;
}
```

## 5.5 Leftist Heap

```
// Author: Ian
// Function: min-heap, with worst-time O(lg n) merge
struct node {
  node *l, *r; int d, v;
  node(int x): d(1), v(x) { l = r = nullptr; }
};
static inline int d(node* x) { return x ? x->d : 0; }
node* merge(node* a, node* b) {
  if (!a || !b) return a ?: b;
  if (a->v>b->v) swap(a,b);
  a->r = merge(a->r, b);
  if (d(a->l) < d(a->r))
    swap(a->l, a->r);
  a->d = d(a->r) + 1;
  return a;
}
```

## 5.6 Persistent Treap

```
// Author: Ian
struct node {
  node *l, *r;
  char c; int v, sz;
  node(char x = '$'): c(x), v(mt()), sz(1) {
    l = r = nullptr;
  }
  node(node* p) {*this = *p;}
  void pull() {
    sz = 1;
    for (auto i : {l, r})
      if (i) sz += i->sz;
  }
} arr[maxn], *ptr = arr;
inline int size(node* p) {return p ? p->sz : 0;}
node* merge(node* a, node* b) {
  if (!a || !b) return a ? : b;
  if (a->v < b->v) {
    node* ret = new(ptr++) node(a);
    ret->r = merge(ret->r, b), ret->pull();
    return ret;
  }
  else {
    node* ret = new(ptr++) node(b);
    ret->l = merge(a, ret->l), ret->pull();
    return ret;
  }
}
P<node*> split(node* p, int k) {
  if (!p) return {nullptr, nullptr};
  if (k >= size(p->l) + 1) {
    auto [a, b] = split(p->r, k - size(p->l) - 1);
    node* ret = new(ptr++) node(p);
    ret->r = a, ret->pull();
    return {ret, b};
  }
  else {
    auto [a, b] = split(p->l, k);
    node* ret = new(ptr++) node(p);
    ret->l = b, ret->pull();
    return {a, ret};
  }
}
```

## 5.7 Li Chao Tree

```
// Author: Ian
// Function: For a set of lines L, find the maximum L_i
    (x) in L in O(lg n).
typedef long double ld;
constexpr int maxn = 5e4 + 5;
struct line {
  ld a, b;
  ld operator()(ld x) {return a * x + b;}
} arr[(maxn + 1) << 2];
bool operator<(line a, line b) {return a.a < b.a;}
#define m ((l+r)>>1)
void insert(line x, int i = 1, int l = 0, int r = maxn)
    {
  if (r - l == 1) {
    if (x(l) > arr[i](l))
      arr[i] = x;
    return;
  }
  line a = max(arr[i], x), b = min(arr[i], x);
  if (a(m) > b(m))
    arr[i] = a, insert(b, i << 1, l, m);
  else
    arr[i] = b, insert(a, i << 1 | 1, m, r);
}
ld query(int x, int i = 1, int l = 0, int r = maxn) {
  if (x < l || r <= x) return -numeric_limits<ld>::max
      ();
```

```
  if (r - l == 1) return arr[i](x);
  return max({arr[i](x), query(x, i << 1, l, m), query(
      x, i << 1 | 1, m, r)});
}
#undef m
```

## 5.8 Time Segment Tree

```
// Author: Ian
constexpr int maxn = 1e5 + 5;
V<P<int>> arr[(maxn + 1) << 2];
V<int> dsu, sz;
V<tuple<int, int, int>> his;
int cnt, q;
int find(int x) {
    return x == dsu[x] ? x : find(dsu[x]);
};
inline bool merge(int x, int y) {
    int a = find(x), b = find(y);
    if (a == b) return false;
    if (sz[a] > sz[b]) swap(a, b);
    his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
         sz[a];
    return true;
};
inline void undo() {
    auto [a, b, s] = his.back(); his.pop_back();
    dsu[a] = a, sz[b] = s;
}
#define m ((l + r) >> 1)
void insert(int ql, int qr, P<int> x, int i = 1, int l
    = 0, int r = q) {
    // debug(ql, qr, x); return;
    if (qr <= l || r <= ql) return;
    if (ql <= l && r <= qr) {arr[i].push_back(x);
        return;}
    if (qr <= m)
        insert(ql, qr, x, i << 1, l, m);
    else if (m <= ql)
        insert(ql, qr, x, i << 1 | 1, m, r);
    else {
        insert(ql, qr, x, i << 1, l, m);
        insert(ql, qr, x, i << 1 | 1, m, r);
    }
}
void traversal(V<int>& ans, int i = 1, int l = 0, int r
     = q) {
    int opcnt = 0;
    // debug(i, l, r);
    for (auto [a, b] : arr[i])
        if (merge(a, b))
            opcnt++, cnt--;
    if (r - l == 1) ans[l] = cnt;
    else {
        traversal(ans, i << 1, l, m);
        traversal(ans, i << 1 | 1, m, r);
    }
    while (opcnt--)
        undo(), cnt++;
    arr[i].clear();
}
#undef m
inline void solve() {
    int n, m; cin>>n>>m>>q,q++;
    dsu.resize(cnt = n), sz.assign(n, 1);
    iota(dsu.begin(), dsu.end(), 0);
    // a, b, time, operation
    unordered_map<ll, V<int>> s;
    for (int i = 0; i < m; i++) {
        int a, b; cin>>a>>b;
        if (a > b) swap(a, b);
        s[((ll)a << 32) | b].emplace_back(0);
    }
    for (int i = 1; i < q; i++) {
        int op,a, b;
        cin>>op>>a>>b;
        if (a > b) swap(a, b);
        switch (op) {
        case 1:
            s[((ll)a << 32) | b].push_back(i);
            break;
        case 2:
            auto tmp = s[((ll)a << 32) | b].back();
            s[((ll)a << 32) | b].pop_back();
            insert(tmp, i, P<int> {a, b});
        }
    }
    for (auto [p, v] : s) {
        int a = p >> 32, b = p & -1;
        while (v.size()) {
            insert(v.back(), q, P<int> {a, b});
            v.pop_back();
        }
    }
    V<int> ans(q);
    traversal(ans);
    for (auto i : ans)
        cout<<i<<' ';
    cout<<endl;
}
```

# 6 DP

- 區間 DP
  - 狀態：$dp[l][r]$ = 區間 $[l, r]$ 的最佳值/方案數
  - 轉移：枚舉劃分點 $k$
  - 思考：是否滿足四邊形不等式、Knuth 優化可加速
- 背包 DP
  - 狀態：$dp[i][w]$ = 前 $i$ 個物品容量 $w$ 的最佳值
  - 判斷是 0/1、多重、分組 → 決定轉移方式
  - 若容量大 → bitset / 數學變形 / meet-in-the-middle
- 樹形 DP
  - 狀態：$dp[u][\text{flag}]$ = 子樹 $u$ 的最佳值
  - 合併子樹資訊 → 小到大合併 / 捲積式轉移
  - 注意 reroot 技巧（dp on tree + dp2 上傳）
- 數位 DP
  - 狀態：$(pos, tight, property)$
  - tight 控制是否貼上界
  - property 常為「餘數、數字和、相鄰限制」
- 狀壓 DP
  - 狀態：$dp[mask][last]$
  - 常見於 TSP / Hamiltonian path / 覆蓋問題
  - $n \leq 20$ 可做，否則要容斥 / FFT
- 期望 / 機率 DP
  - 狀態 $E[s]$ = 從狀態 $s$ 到終點的期望
  - 式子：$E[s] = c + \sum P(s \to s')E[s']$
  - 線性期望：能拆就拆，少算分布
  - 輸出 mod → 分數化 → 模逆元
- 計數 DP / 組合數
  - 狀態表示方案數，常搭配「模數取餘」
  - 若轉移是捲積型 → FFT/NTT 加速
  - 若能公式化（Catalan / Ballot / Stirling）→ 直接套公式
- 優化 DP
  - 判斷轉移方程 $dp[i] = \min_j(dp[j] + C(j, i))$ 的性質
  - 單調性 → 分治優化
  - 凸性 → Convex Hull Trick / 斜率優化
  - 四邊形不等式 → Knuth 優化

## 6.1 Aliens

```
// Author: Gino
// Function: TODO
int n; ll k;
vector<ll> a;
vector<pll> dp[2];
void init() {
  cin >> n >> k;
  for (auto& d : dp) d.clear(), d.resize(n);
  a.clear(); a.resize(n);
```

```
  for (auto& i : a) cin >> i;
}
pll calc(ll p) {
  dp[0][0] = make_pair(0, 0);
  dp[1][0] = make_pair(-a[0], 0);
    for (int i = 1; i < n; i++) {
    if (dp[0][i-1].first > dp[1][i-1].first + a[i] - p)
        {
      dp[0][i] = dp[0][i-1];
    } else if (dp[0][i-1].first < dp[1][i-1].first + a[
        i] - p) {
      dp[0][i] = make_pair(dp[1][i-1].first + a[i] - p,
          dp[1][i-1].second+1);
    } else {
      dp[0][i] = make_pair(dp[0][i-1].first, min(dp[0][
          i-1].second, dp[1][i-1].second+1));
    }
    if (dp[0][i-1].first - a[i] > dp[1][i-1].first) {
      dp[1][i] = make_pair(dp[0][i-1].first - a[i], dp
          [0][i-1].second);
    } else if (dp[0][i-1].first - a[i] < dp[1][i-1].
        first) {
      dp[1][i] = dp[1][i-1];
    } else {
      dp[1][i] = make_pair(dp[1][i-1].first, min(dp[0][
          i-1].second, dp[1][i-1].second));
    }
  }
  return dp[0][n-1];
}
void solve() {
  ll l = 0, r = 1e7;
  pll res = calc(0);
  if (res.second <= k) return cout << res.first << endl
      , void();
  while (l < r) {
    ll mid = (l+r)>>1;
    res = calc(mid);
    if (res.second <= k) r = mid;
    else l = mid+1;
  }
  res = calc(l);
  cout << res.first + k*l << endl;
}
```

## 6.2 SOS DP

```
// Author: Gino
// Function: Solve problems that enumerates subsets of
    subsets (3^n => n*2^n)
for (int msk = 0; msk < (1<<n); msk++) {
    for (int i = 1; i <= n; i++) {
        if (msk & (1<<(i - 1))) {
            // dp[msk][i] = dp[msk][i - 1] + dp[msk ^
                (1<<(i - 1))][i - 1];
        } else {
            // dp[msk][i] = dp[msk][i - 1];
        }
    }
}
```

## 6.3 期望 DP (Expected Value DP)

- 狀態設計：$E[s]$ = 從狀態 $s$ 出發到終點的期望值
- 列式子：

$$E[s] = (\text{當前代價}) + \sum_{s'} P(s \to s') \cdot E[s']$$

- 若存在自環，把 $E[s]$ 移到左邊，整理成

$$(1 - P(s \to s))E[s] = c + \sum_{s' \neq s} P(s \to s') \cdot E[s']$$

- 線性期望技巧：能拆就拆，避免處理整個分布
- 輸出 mod 時，分母要用模逆元：$q^{-1} \equiv q^{M-2} \pmod{M}$ (質數模數)

常見題型

- 擲骰子遊戲（到達終點的期望步數）
- 隨機遊走 hitting time
- 重複試驗直到成功
- 博弈遊戲的期望值
- 機率 DP：計算到某步時在某狀態的機率

範例：擲骰子到 $n$ 格

$$E[i] = 1 + \frac{1}{6}\sum_{d=1}^{6} E[i+d], \quad (i < n), \quad E[n] = 0$$

```
int main(){
    int n;
    cin >> n; // 終點位置

    // E[i] = 從位置 i 走到終點的期望步數
    // 因為每次最多走 6，所以要開 n+6 以避免越界
    vector<double> E(n+7, 0.0);

    // 從終點往前推 (backward DP)
    for(int i=n-1; i>=0; i--){
        double sum=0;
        // 期望公式: E[i] = 1 + (E[i+1]+...+E[i+6]) / 6
        for(int d=1; d<=6; d++) sum += E[i+d];
        E[i] = 1 + sum/6.0;
    }

    // 輸出 E[0]，即從起點到終點的期望擲骰次數
    cout << fixed << setprecision(10) << E[0] << "\n";
}
```

## 6.4 數位 DP (Digit DP)

- 狀態：$(pos, tight, property)$
  - pos = 當前處理到第幾位
  - tight = 是否受限於上界 $N$
  - property = 額外屬性（如數位和、餘數、相鄰限制 …）
- 遞迴：枚舉當前位數字，遞迴下一位
- 終止條件：$pos == $ 長度 $\to$ 回傳屬性是否滿足
- 記憶化：$dp[pos][tight][property]$

常見題型

- 計算 $[0, N]$ 中數位和可被 $k$ 整除的數字個數
- 不含連續相同數字的數字個數
- 含特定數字次數的數字個數
- 位數和 / 餘數 / mod pattern

範例：計算 $[0, N]$ 中數位和 mod$k = 0$ 的數字個數

$$dp[pos][tight][\text{sum mod } k]$$

```
string s; // N 轉成字串，方便逐位處理
int k;    // 除數

// dp[pos][tight][sum_mod]
// pos     = 當前處理到哪一位 (0 = 最高位)
// tight   = 是否仍受限於 N 的數字 (1 = 是, 0 = 否)
// sum_mod = 當前數位和 mod k 的值
long long dp[20][2][105];

// 計算: 從 pos 開始，tight 狀態下，數位和 mod k =
    sum_mod 的方案數
long long dfs(int pos, int tight, int sum_mod){
    // 終止條件：所有位數都處理完
    if(pos == (int)s.size())
        // 若數位和 mod k == 0，算作一個合法數字
        return (sum_mod % k == 0);

```

```
    // 記憶化查詢
    if(dp[pos][tight][sum_mod] != -1)
        return dp[pos][tight][sum_mod];

    long long res = 0;
    // 如果 tight = 1，本位數字上限 = N 的該位數字
    // 如果 tight = 0，本位數字上限 = 9
    int limit = tight ? (s[pos]-'0') : 9;

    // 枚舉當前位可以填的數字
    for(int d=0; d<=limit; d++){
        // 下一位是否仍然 tight？
        int next_tight = (tight && d==limit);
        // 更新數位和 mod k
        int next_mod = (sum_mod + d) % k;
        res += dfs(pos+1, next_tight, next_mod);
    }

    // 存結果
    return dp[pos][tight][sum_mod] = res;
}

int main(){
    long long N;
    cin >> N >> k;
    s = to_string(N); // 把 N 轉成字串，方便取每一位
    memset(dp,-1,sizeof(dp));
    cout << dfs(0,1,0) << "\n"; // 從最高位開始，初始 tight=1，sum=0
}
```

# 7 Graph

## 7.1 Tree Centroid

```
int n;
vector<vector<int>> G;

pii centroid;
vector<int> sz, mxcc;  // mxcc[u]: max component size after removing u

void dfs(int u, int p) {
    sz[u] = 1;
    for (auto& v : G[u]) {
        if (v == p) continue;
        dfs(v, u);
        sz[u] += sz[v];
        mxcc[u] = max(mxcc[u], sz[v]);
    }
    mxcc[u] = max(mxcc[u], n - sz[u]);
}

void find_centroid() {
    centroid = pii{-1, -1};
    reset(sz, n + 1, 0);
    reset(mxcc, n + 1, 0);
    dfs(1, 1);
    for (int u = 1; u <= n; u++) {
        if (mxcc[u] <= n / 2) {
            if (centroid.first != -1) centroid.second = u;
            else centroid.first = u;
        }
    }
}
```

## 7.2 Bellman-Ford + SPFA

```
int n, m;

// Graph
vector<vector<pair<int, ll> > > g;
vector<ll> dis;
vector<bool> negCycle;

// SPFA
vector<int> rlx;
queue<int> q;
vector<bool> inq;
vector<int> pa;
void SPFA(vector<int>& src) {
    dis.assign(n+1, LINF);
    negCycle.assign(n+1, false);
    rlx.assign(n+1, 0);
    while (!q.empty()) q.pop();
    inq.assign(n+1, false);
    pa.assign(n+1, -1);

    for (auto& s : src) {
        dis[s] = 0;
        q.push(s); inq[s] = true;
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop(); inq[u] = false;
        if (rlx[u] >= n) {
            negCycle[u] = true;
        }
        else for (auto& e : g[u]) {
            int v = e.first;
            ll w = e.second;
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                rlx[v] = rlx[u] + 1;
                pa[v] = u;
                if (!inq[v]) {
                    q.push(v);
                    inq[v] = true;
} } } } }


// Bellman-Ford
queue<int> q;
vector<int> pa;
void BellmanFord(vector<int>& src) {
    dis.assign(n+1, LINF);
    negCycle.assign(n+1, false);
    pa.assign(n+1, -1);

    for (auto& s : src) dis[s] = 0;

    for (int rlx = 1; rlx <= n; rlx++) {
        for (int u = 1; u <= n; u++) {
            if (dis[u] == LINF) continue;  // Important !!
            for (auto& e : g[u]) {
                int v = e.first; ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    pa[v] = u;
                    if (rlx == n) negCycle[v] = true;
} } } } }


// Negative Cycle Detection
void NegCycleDetect() {
/* No Neg Cycle: NO
Exist Any Neg Cycle:
YES
v0 v1 v2 ... vk v0 */

    vector<int> src;
    for (int i = 1; i <= n; i++)
        src.emplace_back(i);

    SPFA(src);
    // BellmanFord(src);

    int ptr = -1;
    for (int i = 1; i <= n; i++) if (negCycle[i])
        { ptr = i; break; }

    if (ptr == -1) { return cout << "NO" << endl, void(); }

    cout << "YES\n";
    vector<int> ans;
    vector<bool> vis(n+1, false);
```

```

    while (true) {
        ans.emplace_back(ptr);
        if (vis[ptr]) break;
        vis[ptr] = true;
        ptr = pa[ptr];
    }
    reverse(ans.begin(), ans.end());

    vis.assign(n+1, false);
    for (auto& x : ans) {
        cout << x << ' ';
        if (vis[x]) break;
        vis[x] = true;
    }
    cout << endl;
}

// Distance Calculation
void calcDis(int s) {
    vector<int> src;
    src.emplace_back(s);
    SPFA(src);
    // BellmanFord(src);

    while (!q.empty()) q.pop();
    for (int i = 1; i <= n; i++)
        if (negCycle[i]) q.push(i);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto& e : g[u]) {
            int v = e.first;
            if (!negCycle[v]) {
                q.push(v);
                negCycle[v] = true;
} } } }
```

## 7.3 BCC - AP

```
int n, m;
int low[maxn], dfn[maxn], instp;
vector<int> E, g[maxn];
bitset<maxn> isap;
bitset<maxm> vis;
stack<int> stk;
int bccnt;
vector<int> bcc[maxn];
inline void popout(int u) {
  bccnt++;
  bcc[bccnt].emplace_back(u);
  while (!stk.empty()) {
    int v = stk.top();
    if (u == v) break;
    stk.pop();
    bcc[bccnt].emplace_back(v);
  }
}
void dfs(int u, bool rt = 0) {
  stk.push(u);
  low[u] = dfn[u] = ++instp;
  int kid = 0;
  Each(e, g[u]) {
    if (vis[e]) continue;
    vis[e] = true;
    int v = E[e]^u;
    if (!dfn[v]) {
      // tree edge
      kid++; dfs(v);
      low[u] = min(low[u], low[v]);
      if (!rt && low[v] >= dfn[u]) {
        // bcc found: u is ap
        isap[u] = true;
        popout(u);
      }
    } else {
      // back edge
      low[u] = min(low[u], dfn[v]);
    }
  }
  // special case: root
  if (rt) {
    if (kid > 1) isap[u] = true;
    popout(u);
  }
}
void init() {
  cin >> n >> m;
  fill(low, low+maxn, INF);
  REP(i, m) {
    int u, v;
    cin >> u >> v;
    g[u].emplace_back(i);
    g[v].emplace_back(i);
    E.emplace_back(u^v);
  }
}
void solve() {
  FOR(i, 1, n+1, 1) {
    if (!dfn[i]) dfs(i, true);
  }
  vector<int> ans;
  int cnt = 0;
  FOR(i, 1, n+1, 1) {
    if (isap[i]) cnt++, ans.emplace_back(i);
  }
  cout << cnt << endl;
  Each(i, ans) cout << i << ' ';
  cout << endl;
}
```

## 7.4 BCC - Bridge

```
int n, m;
vector<int> g[maxn], E;
int low[maxn], dfn[maxn], instp;
int bccnt, bccid[maxn];
stack<int> stk;
bitset<maxm> vis, isbrg;
void init() {
  cin >> n >> m;
  REP(i, m) {
    int u, v;
    cin >> u >> v;
    E.emplace_back(u^v);
    g[u].emplace_back(i);
    g[v].emplace_back(i);
  }
  fill(low, low+maxn, INF);
}
void popout(int u) {
  bccnt++;
  while (!stk.empty()) {
    int v = stk.top();
    if (v == u) break;
    stk.pop();
    bccid[v] = bccnt;
  }
}
void dfs(int u) {
  stk.push(u);
  low[u] = dfn[u] = ++instp;

  Each(e, g[u]) {
    if (vis[e]) continue;
    vis[e] = true;

    int v = E[e]^u;
    if (dfn[v]) {
      // back edge
      low[u] = min(low[u], dfn[v]);
    } else {
      // tree edge
      dfs(v);
      low[u] = min(low[u], low[v]);
      if (low[v] == dfn[v]) {
        isbrg[e] = true;
        popout(u);
      }
    }
  }
}
```

```
void solve() {
  FOR(i, 1, n+1, 1) {
    if (!dfn[i]) dfs(i);
  }
  vector<pii> ans;
  vis.reset();
  FOR(u, 1, n+1, 1) {
    Each(e, g[u]) {
      if (!isbrg[e] || vis[e]) continue;
      vis[e] = true;
      int v = E[e]^u;
      ans.emplace_back(mp(u, v));
    }
  }
  cout << (int)ans.size() << endl;
  Each(e, ans) cout << e.F << ' ' << e.S << endl;
}
```

## 7.5 SCC - Tarjan with 2-SAT

```
// Author: Ian
// 2-sat + tarjan SCC
void solve() {
  int n, r, l; cin >> n >> r >> l;
  V<P<int>> v(l);
  for (auto& [a, b] : v)
    cin >> a >> b;
  V<V<int>> e(2 * l);
  for (int i = 0; i < l; i++)
    for (int j = i + 1; j < l; j++) {
      if (v[i].first == v[j].first && abs(v[i].second -
          v[j].second) <= 2 * r) {
        e[i << 1].emplace_back(j << 1 | 1);
        e[j << 1].emplace_back(i << 1 | 1);
      }
      if (v[i].second == v[j].second && abs(v[i].first
          - v[j].first) <= 2 * r) {
        e[i << 1 | 1].emplace_back(j << 1);
        e[j << 1 | 1].emplace_back(i << 1);
      }
    }
  V<bool> ins(2 * l, false);
  V<int> scc(2 * l), dfn(2 * l, -1), low(2 * l, inf);
  stack<int> s;
  function<void(int)> dfs = [&](int x) {
    if (~dfn[x]) return;
    static int t = 0;
    dfn[x] = low[x] = t++;
    s.push(x), ins[x] = true;
    for (auto i : e[x])
      if (dfs(i), ins[i])
        low[x] = min(low[x], low[i]);
    if (dfn[x] == low[x]) {
      static int ncnt = 0;
      int p; do {
        ins[p = s.top()] = false;
        s.pop(), scc[p] = ncnt;
      } while (p != x); ncnt++;
    }
  };
  for (int i = 0; i < 2 * l; i++)
    dfs(i);
  for (int i = 0; i < l; i++)
    if (scc[i << 1] == scc[i << 1 | 1]) {
      cout << "NO" << endl;
      return;
    }
  cout << "YES" << endl;
}
```

## 7.6 Eulerian Path - Undir

```
// Author: Gino
// Usage: build deg, G first, then eulerian()
int n, m;  // number of vertices and edges
vector<int> deg;  // degree
vector<set<pii>> G;  // G[u] := {(v, edge id)}

vector<int> path_u, path_e;
void dfs(int u) {
    while (!G[u].empty()) {
        auto it = G[u].begin();
        auto [v, i] = *it; G[u].erase(it);
        G[v].erase(make_pair(u, i)); dfs(v);
        path_u.emplace_back(v);
        path_e.emplace_back(i);
    }
}
void gogo(int s) {
    path_u.clear(); path_e.clear();
    dfs(s); path_u.emplace_back(s);
    reverse(path_u.begin(), path_u.end());
    reverse(path_e.begin(), path_e.end());
}
bool eulerian() {
    int oddcnt = 0, s = -1;
    for (int u = 1; u <= n; u++)
        if (deg[u] & 1)
            oddcnt++, s = u;

    if (oddcnt != 0 && oddcnt != 2) return false;
    if (s == -1) {
        s = 1; for (int u = 1; u <= n; u++)
            if (deg[u] > 0)
                s = u;
    }
    gogo(s);

    for (int u = 1; u <= n; u++)
        if ((int)G[u].size() > 0)
            return false;
    return true;
}
```

## 7.7 Eulerian Path - Dir

```
// Author: Gino
// Usage: build ind, oud, G first, then eulerian()
int n, m;  // number of vertices, edges
vector<int> ind, oud;  // indegree, outdegree
vector<vector<pii>> G;  // G[u] := {(v, edge id)}

vector<int> path_u, path_e;
void dfs(int u) {
    while (!G[u].empty()) {
        auto [v, i] = G[u].back(); G[u].pop_back();
        dfs(v);
        path_u.emplace_back(v);
        path_e.emplace_back(i);
    }
}
void gogo(int s) {
    path_u.clear(); path_e.clear();
    dfs(s); path_u.emplace_back(s);
    reverse(path_u.begin(), path_u.end());
    reverse(path_e.begin(), path_e.end());
}
bool eulerian() {
    int s = -1;
    for (int u = 1; u <= n; u++) {
        if (abs(oud[u] - ind[u]) > 1) return false;
        if (oud[u] - ind[u] == 1) {
            if (s != -1) return false;
            s = u;
        }
    }
    if (s == -1) {
        s = 1; for (int u = 1; u <= n; u++)
            if (ind[u] > 0)
                s = u;
    }
    gogo(s);
    for (int u = 1; u <= n; u++)
        if ((int)G[u].size() > 0)
            return false;

    return true;
}
```

## 7.8 Kth Shortest Path

```cpp
// time: O(|E| \lg |E|+|V| \lg |V|+K)
// memory: O(|E| \lg |E|+|V|)
struct KSP{ // 1-base
  struct nd{
    int u,v; ll d;
    nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di; }
  };
  struct heap{ nd* edge; int dep; heap* chd[4]; };
  static int cmp(heap* a,heap* b)
  { return a->edge->d > b->edge->d; }
  struct node{
    int v; ll d; heap* H; nd* E;
    node(){}
    node(ll _d,int _v,nd* _E){ d =_d; v=_v; E=_E; }
    node(heap* _H,ll _d){ H=_H; d=_d; }
    friend bool operator<(node a,node b)
    { return a.d>b.d; }
  };
  int n,k,s,t,dst[N]; nd *nxt[N];
  vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
  void init(int _n,int _k,int _s,int _t){
    n=_n; k=_k; s=_s; t=_t;
    for(int i=1;i<=n;i++){
      g[i].clear(); rg[i].clear();
      nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
    }
  }
  void addEdge(int ui,int vi,ll di){
    nd* e=new nd(ui,vi,di);
    g[ui].push_back(e); rg[vi].push_back(e);
  }
  queue<int> dfsQ;
  void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q; Q.push(node(0,t,NULL));
    while (!Q.empty()){
      node p=Q.top(); Q.pop(); if(dst[p.v]!=-1)continue;
      dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
      for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e));
    }
  }
  heap* merge(heap* curNd,heap* newNd){
    if(curNd==nullNd) return newNd;
    heap* root=new heap;memcpy(root,curNd,sizeof(heap));
    if(newNd->edge->d<curNd->edge->d){
      root->edge=newNd->edge;
      root->chd[2]=newNd->chd[2];
      root->chd[3]=newNd->chd[3];
      newNd->edge=curNd->edge;
      newNd->chd[2]=curNd->chd[2];
      newNd->chd[3]=curNd->chd[3];
    }
    if(root->chd[0]->dep<root->chd[1]->dep)
      root->chd[0]=merge(root->chd[0],newNd);
    else root->chd[1]=merge(root->chd[1],newNd);
    root->dep=max(root->chd[0]->dep,
              root->chd[1]->dep)+1;
    return root;
  }
  vector<heap*> V;
  void build(){
    nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd;
    fill(nullNd->chd,nullNd->chd+4,nullNd);
    while(not dfsQ.empty()){
      int u=dfsQ.front(); dfsQ.pop();
      if(!nxt[u]) head[u]=nullNd;
      else head[u]=head[nxt[u]->v];
      V.clear();
      for(auto&& e:g[u]){
        int v=e->v;
        if(dst[v]==-1) continue;
        e->d+=dst[v]-dst[u];
        if(nxt[u]!=e){
          heap* p=new heap;fill(p->chd,p->chd+4,nullNd);
          p->dep=1; p->edge=e; V.push_back(p);
        }
      }
      if(V.empty()) continue;
      make_heap(V.begin(),V.end(),cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for(size_t i=0;i<V.size();i++){
        if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
        else V[i]->chd[2]=nullNd;
        if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
        else V[i]->chd[3]=nullNd;
      }
      head[u]=merge(head[u],V.front());
    }
  }
  vector<ll> ans;
  void first_K(){
    ans.clear(); priority_queue<node> Q;
    if(dst[s]==-1) return;
    ans.push_back(dst[s]);
    if(head[s]!=nullNd)
      Q.push(node(head[s],dst[s]+head[s]->edge->d));
    for(int _=1;_<k and not Q.empty();_++){
      node p=Q.top(),q; Q.pop(); ans.push_back(p.d);
      if(head[p.H->edge->v]!=nullNd){
        q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
        Q.push(q);
      }
      for(int i=0;i<4;i++)
        if(p.H->chd[i]!=nullNd){
          q.H=p.H->chd[i];
          q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
          Q.push(q);
  } }   }
  void solve(){ // ans[i] stores the i-th shortest path
    dijkstra(); build();
    first_K(); // ans.size() might less than k
  }
} solver;
```

## 7.9 System of Difference Constraints

```cpp
vector<vector<pair<int, ll>>> G;
void add(int u, int v, ll w) {
    G[u].emplace_back(make_pair(v, w));
}
```

- $x_u - x_v \le c \Rightarrow$ `add(v, u, c)`
- $x_u - x_v \ge c \Rightarrow$ `add(u, v, -c)`
- $x_u - x_v = c \Rightarrow$ `add(v, u, c)`, `add(u, v -c)`
- $x_u \ge c \Rightarrow$ add super vertex $x_0 = 0$, then $x_u - x_0 \ge c \Rightarrow$ `add(u, 0, -c)`
- Don't for get non-negative constraints for every variable if specified implicitly.
- Interval sum $\Rightarrow$ Use prefix sum to transform into differential constraints. Don't for get $S_{i+1} - S_i \ge 0$ if $x_i$ needs to be non-negative.
- $\frac{x_u}{x_v} \le c \Rightarrow \log x_u - \log x_v \le \log c$

# 8 String

## 8.1 Rolling Hash

```cpp
const ll C = 27;
inline int id(char c) {return c-'a'+1;}
struct RollingHash {
    string s; int n; ll mod;
    vector<ll> Cexp, hs;
    RollingHash(string& _s, ll _mod):
        s(_s), n((int)_s.size()), mod(_mod)
    {
        Cexp.assign(n, 0);
```

```
        hs.assign(n, 0);
        Cexp[0] = 1;
        for (int i = 1; i < n; i++) {
            Cexp[i] = Cexp[i-1] * C;
            if (Cexp[i] >= mod) Cexp[i] %= mod;
        }
        hs[0] = id(s[0]);
        for (int i = 1; i < n; i++) {
            hs[i] = hs[i-1] * C + id(s[i]);
            if (hs[i] >= mod) hs[i] %= mod;
    } }
    inline ll query(int l, int r) {
        ll res = hs[r] - (l ? hs[l-1] * Cexp[r-l+1] :
            0);
        res = (res % mod + mod) % mod;
        return res; }
};
```

## 8.2 Trie

```
struct node {
    int c[26]; ll cnt;
    node(): cnt(0) {memset(c, 0, sizeof(c));}
    node(ll x): cnt(x) {memset(c, 0, sizeof(c));}
};
struct Trie {
    vector<node> t;
    void init() {
        t.clear();
        t.emplace_back(node());
    }
    void insert(string s) { int ptr = 0;
        for (auto& i : s) {
            if (!t[ptr].c[i-'a']) {
                t.emplace_back(node());
                t[ptr].c[i-'a'] = (int)t.size()-1; }
            ptr = t[ptr].c[i-'a']; }
        t[ptr].cnt++; }
} trie;
```

## 8.3 KMP

```
int n, m;
string s, p;
vector<int> f;
void build() {
  f.clear(); f.resize(m, 0);
  int ptr = 0; for (int i = 1; i < m; i++) {
    while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
    if (p[i] == p[ptr]) ptr++;
    f[i] = ptr;
}}
void init() {
  cin >> s >> p;
  n = (int)s.size();
  m = (int)p.size();
  build(); }
void solve() {
  int ans = 0, pi = 0;
  for (int si = 0; si < n; si++) {
    while (pi && s[si] != p[pi]) pi = f[pi-1];
    if (s[si] == p[pi]) pi++;
    if (pi == m) ans++, pi = f[pi-1];
  }
cout << ans << endl; }
```

## 8.4 Z Value

```
string is, it, s;
int n; vector<int> z;
void init() {
    cin >> is >> it;
    s = it+'0'+is;
    n = (int)s.size();
    z.resize(n, 0); }
void solve() {
    int ans = 0; z[0] = n;
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r) z[i] = min(z[i-l], r-i+1);
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i
            ]++;
        if (i+z[i]-1 > r) l = i, r = i+z[i]-1;
        if (z[i] == (int)it.size()) ans++;
    }
    cout << ans << endl; }
```

## 8.5 Manacher

```
int n; string S, s;
vector<int> m;
void manacher() {
s.clear(); s.resize(2*n+1, '.');
for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
m.clear(); m.resize(2*n+1, 0);
// m[i] := max k such that s[i-k, i+k] is palindrome
int mx = 0, mxk = 0;
for (int i = 1; i < 2*n+1; i++) {
  if (mx-(i-mx) >= 0) m[i] = min(m[mx-(i-mx)], mx+mxk-i
      );
  while (0 <= i-m[i]-1 && i+m[i]+1 < 2*n+1 &&
       s[i-m[i]-1] == s[i+m[i]+1]) m[i]++;
  if (i+m[i] > mx+mxk) mx = i, mxk = m[i];
} }
void init() { cin >> S; n = (int)S.size(); }
void solve() {
  manacher();
  int mx = 0, ptr = 0;
  for (int i = 0; i < 2*n+1; i++) if (mx < m[i])
    { mx = m[i]; ptr = i; }
  for (int i = ptr-mx; i <= ptr+mx; i++)
    if (s[i] != '.') cout << s[i];
cout << endl; }
```

## 8.6 Suffix Array

```
#define F first
#define S second
struct SuffixArray {  // don't forget s += "$";
    int n; string s;
    vector<int> suf, lcp, rk;
    vector<int> cnt, pos;
    vector<pair<pii, int> > buc[2];
    void init(string _s) {
        s = _s; n = (int)s.size();
// resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
    }
    void radix_sort() {
        for (int t : {0, 1}) {
            fill(cnt.begin(), cnt.end(), 0);
            for (auto& i : buc[t]) cnt[ (t ? i.F.F : i.
                F.S) ]++;
            for (int i = 0; i < n; i++)
                pos[i] = (!i ? 0 : pos[i-1] + cnt[i-1])
                    ;
            for (auto& i : buc[t])
                buc[t^1][pos[ (t ? i.F.F : i.F.S) ]++]
                    = i;
    }}
    bool fill_suf() {
        bool end = true;
        for (int i = 0; i < n; i++) suf[i] = buc[0][i].
            S;
        rk[suf[0]] = 0;
        for (int i = 1; i < n; i++) {
            int dif = (buc[0][i].F != buc[0][i-1].F);
            end &= dif;
            rk[suf[i]] = rk[suf[i-1]] + dif;
        } return end;
    }
    void sa() {
        for (int i = 0; i < n; i++)
            buc[0][i] = make_pair(make_pair(s[i], s[i])
                , i);
        sort(buc[0].begin(), buc[0].end());
        if (fill_suf()) return;
        for (int k = 0; (1<<k) < n; k++) {
            for (int i = 0; i < n; i++)
                buc[0][i] = make_pair(make_pair(rk[i],
                    rk[(i + (1<<k)) % n]), i);
```

```
            radix_sort();
            if (fill_suf()) return;
    }}
    void LCP() {  int k = 0;
        for (int i = 0; i < n-1; i++) {
            if (rk[i] == 0) continue;
            int pi = rk[i];
            int j = suf[pi-1];
            while (i+k < n && j+k < n && s[i+k] == s[j+k]) k++;
            lcp[pi] = k;
            k = max(k-1, 0);
    }}
};
SuffixArray suffixarray;
```

## 8.7 SA-IS

```
const int N=300010;
struct SA{
#define REP(i,n) for(int i=0;i<int(n);i++)
#define REP1(i,a,b) for(int i=(a);i<=int(b);i++)
  bool _t[N*2]; int _s[N*2],_sa[N*2];
  int _c[N*2],x[N],_p[N],_q[N*2],hei[N],r[N];
  int operator [](int i){ return _sa[i]; }
  void build(int *s,int n,int m){
    memcpy(_s,s,sizeof(int)*n);
    sais(_s,_sa,_p,_q,_t,_c,n,m); mkhei(n);
  }
  void mkhei(int n){
    REP(i,n) r[_sa[i]]=i;
    hei[0]=0;
    REP(i,n) if(r[i]) {
      int ans=i>0?max(hei[r[i-1]]-1,0):0;
      while(_s[i+ans]==_s[_sa[r[i]-1]+ans]) ans++;
      hei[r[i]]=ans;
    }
  }
  void sais(int *s,int *sa,int *p,int *q,bool *t,int *c
      ,int n,int z){
    bool uniq=t[n-1]=true,neq;
    int nn=0,nmxz=-1,*nsa=sa+n,*ns=s+n,lst=-1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa,n);\
memcpy(x,c,sizeof(int)*z); XD;\
memcpy(x+1,c,sizeof(int)*(z-1));\
REP(i,n) if(sa[i]&&!t[sa[i]-1]) sa[x[s[sa[i]-1]]++]=sa[
    i]-1;\
memcpy(x,c,sizeof(int)*z);\
for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[
    sa[i]-1]]]=sa[i]-1;
    MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
    REP(i,z-1) c[i+1]+=c[i];
    if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
    for(int i=n-2;i>=0;i--)
      t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
    MAGIC(REP1(i,1,n-1) if(t[i]&&!t[i-1]) sa[--x[s[i
        ]]]=p[q[i]=nn++]=i);
    REP(i,n) if(sa[i]&&t[sa[i]]&&!t[sa[i]-1]){
      neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
          [i])*sizeof(int));
      ns[q[lst=sa[i]]]=nmxz+=neq;
    }
    sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
    MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i
        ]]]]]=p[nsa[i]]);
  }
}sa;
int H[N],SA[N],RA[N];
void suffix_array(int* ip,int len){
  // should padding a zero in the back
  // ip is int array, len is array length
  // ip[0..n-1] != 0, and ip[len]=0
  ip[len++]=0; sa.build(ip,len,128);
  memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2)
      ;
  for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
  // resulting height, sa array \in [0,len)
}
```

## 8.8 Minimum Rotation

```
// lexicographically minimal rotation
// rotate(begin(s), begin(s)+minRotation(s), end(s))
int minRotation(string s) {
int a = 0, n = s.size(); s += s;
for(int b = 0; b < n; b++) for(int k = 0; k < n; k++) {
    if(a + k == b || s[a + k] < s[b + k]) {
        b += max(0, k - 1);
        break; }
    if(s[a + k] > s[b + k]) {
        a = b;
        break;
    } }
return a; }
```

## 8.9 Aho Corasick

```
struct ACautomata{
  struct Node{
    int cnt;
    Node *go[26], *fail, *dic;
    Node (){
      cnt = 0; fail = 0; dic=0;
      memset(go,0,sizeof(go));
    }
  }pool[1048576],*root;
  int nMem;
  Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
  }
  void init() { nMem = 0; root = new_Node(); }
  void add(const string &str) { insert(root,str,0); }
  void insert(Node *cur, const string &str, int pos){
    for(int i=pos;i<str.size();i++){
      if(!cur->go[str[i]-'a'])
        cur->go[str[i]-'a'] = new_Node();
      cur=cur->go[str[i]-'a'];
    }
    cur->cnt++;
  }
  void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
      Node* fr=que.front(); que.pop();
      for (int i=0; i<26; i++){
        if (fr->go[i]){
          Node *ptr = fr->fail;
          while (ptr && !ptr->go[i]) ptr = ptr->fail;
          fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
          fr->go[i]->dic=(ptr->cnt?ptr:ptr->dic);
          que.push(fr->go[i]);
  } } } }
}AC;
```

# 9 Geometry

## 9.1 Basic Operations

```
// Author: Gino
typedef long long T;
// typedef long double T;
const long double eps = 1e-8;

short sgn(T x) {
    if (abs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}

struct Pt {
T x, y;
Pt(T _x=0, T _y=0):x(_x), y(_y) {}
Pt operator+(Pt a) { return Pt(x+a.x, y+a.y); }
Pt operator-(Pt a) { return Pt(x-a.x, y-a.y); }
Pt operator*(T a)  { return Pt(x*a, y*a); }
Pt operator/(T a)  { return Pt(x/a, y/a); }
T operator*(Pt a)  { return x*a.x + y*a.y; }
```

```
T operator^(Pt a)  { return x*a.y - y*a.x; }  // 不要打反
bool operator<(Pt a)
    { return x < a.x || (x == a.x && y < a.y); }
//return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.y) < 0); }
bool operator==(Pt a)
    { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
};

Pt mv(Pt a, Pt b) { return b-a; }
T len2(Pt a) { return a*a; }
T dis2(Pt a, Pt b) { return len2(b-a); }

short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
bool onseg(Pt p, Pt l1, Pt l2) {
    Pt a = mv(p, l1), b = mv(p, l2);
    return ((a^b) == 0) && ((a*b) <= 0);
}
```

## 9.2 InPoly

```
// Author: Gino
// Function: Check if a point P sits in a polygon (doesn't have to be convex hull)
// 0 = Bound, 1 = In, -1 = Out
short inPoly(Pt p) {
for (int i = 0; i < n; i++)
    if (onseg(p, E[i], E[(i+1)%n])) return 0;
int cnt = 0;
for (int i = 0; i < n; i++)
    if (banana(p, Pt(p.x+1, p.y+2e9), E[i], E[(i+1)%n]))
        cnt ^= 1;
return (cnt ? 1 : -1);
}
```

## 9.3 Sort by Angle

```
// Author: Gino
int ud(Pt a) {  // up or down half plane
    if (a.y > 0) return 0;
    if (a.y < 0) return 1;
    return (a.x >= 0 ? 0 : 1);
}
sort(ALL(E), [&](const Pt& a, const Pt& b){
    if (ud(a) != ud(b)) return ud(a) < ud(b);
    return (a^b) > 0;
});
```

## 9.4 Line Intersect Check

```
// Author: Gino
// Function: check if (p1---p2) (q1---q2) banana
inline bool banana(Pt p1, Pt p2, Pt q1, Pt q2) {
if (onseg(p1, q1, q2) || onseg(p2, q1, q2) ||
    onseg(q1, p1, p2) || onseg(q2, p1, p2)) {
    return true;
}
Pt p = mv(p1, p2), q = mv(q1, q2);
return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 &&
        ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
}
```

## 9.5 Line Intersection

```
// Author: Gino
// T: long double
Pt bananaPoint(Pt p1, Pt p2, Pt q1, Pt q2) {
if (onseg(q1, p1, p2)) return q1;
if (onseg(q2, p1, p2)) return q2;
if (onseg(p1, q1, q2)) return p1;
if (onseg(p2, q1, q2)) return p2;
double s = abs(mv(p1, p2) ^ mv(p1, q1));
double t = abs(mv(p1, p2) ^ mv(p1, q2));
return q2 * (s/(s+t)) + q1 * (t/(s+t));
}
```

## 9.6 Convex Hull

```
// Author: Gino
vector<Pt> hull;
void convexHull() {
hull.clear(); sort(E.begin(), E.end());
for (int t : {0, 1}) {
    int b = (int)hull.size();
    for (auto& ei : E) {
        while ((int)hull.size() - b >= 2 &&
               ori(mv(hull[(int)hull.size()-2], hull.back()),
                   mv(hull[(int)hull.size()-2], ei)) == -1) {
            hull.pop_back();
        }
        hull.emplace_back(ei);
    }
    hull.pop_back();
    reverse(E.begin(), E.end());
} }
```

## 9.7 Lower Concave Hull

```
// Author: Unknown
struct Line {
  mutable ll m, b, p;
  bool operator<(const Line& o) const { return m < o.m; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
    else x->p = div(y->b - x->b, x->m - y->m);
    return x->p >= y->p;
  }
  void add(ll m, ll b) {
    auto z = insert({m, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.m * x + l.b;
  }
};
```

## 9.8 Polygon Area

```
// Author: Gino
// Function: Return doubled area of a polygon
T dbarea(vector<Pt>& e) {
ll res = 0;
for (int i = 0; i < (int)e.size(); i++)
    res += e[i]^e[(i+1)%SZ(e)];
return abs(res);
}
```

## 9.9 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let $i$ = number of points inside the polygon.

Let $b$ = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

## 9.10 Minimum Enclosing Circle

```cpp
// Author: Gino
// Function: Find Min Enclosing Circle using Randomized O(n) Algorithm
Pt circumcenter(Pt A, Pt B, Pt C) {
// a1(x-A.x) + b1(y-A.y) = c1
// a2(x-A.x) + b2(y-A.y) = c2
// solve using Cramer's rule
T a1 = B.x-A.x, b1 = B.y-A.y, c1 = dis2(A, B)/2.0;
T a2 = C.x-A.x, b2 = C.y-A.y, c2 = dis2(A, C)/2.0;
T D  = Pt(a1, b1) ^ Pt(a2, b2);
T Dx = Pt(c1, b1) ^ Pt(c2, b2);
T Dy = Pt(a1, c1) ^ Pt(a2, c2);
if (D == 0) return Pt(-INF, -INF);
return A + Pt(Dx/D, Dy/D);
}
Pt center; T r2;

void minEncloseCircle() {
mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
shuffle(ALL(E), gen);
center = E[0], r2 = 0;

for (int i = 0; i < n; i++) {
    if (dis2(center, E[i]) <= r2) continue;
    center = E[i], r2 = 0;
    for (int j = 0; j < i; j++) {
        if (dis2(center, E[j]) <= r2) continue;
        center = (E[i] + E[j]) / 2.0;
        r2 = dis2(center, E[i]);
        for (int k = 0; k < j; k++) {
            if (dis2(center, E[k]) <= r2) continue;
            center = circumcenter(E[i], E[j], E[k]);
            r2 = dis2(center, E[i]);
        }
    }
} }
```

## 9.11 PolyUnion

```cpp
// Author: Unknown
struct PY{
  int n; Pt pt[5]; double area;
  Pt& operator[](const int x){ return pt[x]; }
  void init(){ //n,pt[0~n-1] must be filled
    area=pt[n-1]^pt[0];
    for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
    if((area/=2)<0)reverse(pt,pt+n),area=-area;
  }
};
PY py[500]; pair<double,int> c[5000];
inline double segP(Pt &p,Pt &p1,Pt &p2){
  if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
  return (p.x-p1.x)/(p2.x-p1.x);
}
double polyUnion(int n){ //py[0~n-1] must be filled
  int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
  for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
  for(i=0;i<n;i++){
    for(ii=0;ii<py[i].n;ii++){
      r=0;
      c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
      for(j=0;j<n;j++){
        if(i==j) continue;
        for(jj=0;jj<py[j].n;jj++){
          ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]));
          tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj+1]));
          if(ta==0 && tb==0){
            if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[i][ii])>0&&j<i){
              c[r++]=make_pair(segP(py[j][jj],py[i][ii],py[i][ii+1]),1);
              c[r++]=make_pair(segP(py[j][jj+1],py[i][ii],py[i][ii+1]),-1);
            }
          }else if(ta>=0 && tb<0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),1);
          }else if(ta<0 && tb>=0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),-1);
      } } }
      sort(c,c+r);
      z=min(max(c[0].first,0.0),1.0); d=c[0].second; s=0;
      for(j=1;j<r;j++){
        w=min(max(c[j].first,0.0),1.0);
        if(!d) s+=w-z;
        d+=c[j].second; z=w;
      }
      sum+=(py[i][ii]^py[i][ii+1])*s;
    }
  }
  return sum/2;
}
```

## 9.12 Minkowski Sum

```cpp
// Author: Unknown
/* convex hull Minkowski Sum*/
#define INF 100000000000000LL
int pos( const Pt& tp ){
  if( tp.Y == 0 ) return tp.X > 0 ? 0 : 1;
  return tp.Y > 0 ? 0 : 1;
}
#define N 300030
Pt pt[ N ], qt[ N ], rt[ N ];
LL Lx,Rx;
int dn,un;
inline bool cmp( Pt a, Pt b ){
  int pa=pos( a ),pb=pos( b );
  if(pa==pb) return (a^b)>0;
  return pa<pb;
}
int minkowskiSum(int n,int m){
  int i,j,r,p,q,fi,fj;
  for(i=1,p=0;i<n;i++){
    if( pt[i].Y<pt[p].Y ||
        (pt[i].Y==pt[p].Y && pt[i].X<pt[p].X) ) p=i; }
  for(i=1,q=0;i<m;i++){
    if( qt[i].Y<qt[q].Y ||
        (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i; }
  rt[0]=pt[p]+qt[q];
  r=1; i=p; j=q; fi=fj=0;
  while(1){
    if((fj&&j==q) ||
       ( (!fi||i!=p) &&
         cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%m]-qt[q]) ) ){
      rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
      p=(p+1)%n;
      fi=1;
    }else{
      rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
      q=(q+1)%m;
      fj=1;
    }
    if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
      r++;
    else rt[r-1]=rt[r];
    if(i==p && j==q) break;
  }
  return r-1;
}
void initInConvex(int n){
  int i,p,q;
  LL Ly,Ry;
  Lx=INF; Rx=-INF;
  for(i=0;i<n;i++){
    if(pt[i].X<Lx) Lx=pt[i].X;
    if(pt[i].X>Rx) Rx=pt[i].X;
  }
  Ly=Ry=INF;
  for(i=0;i<n;i++){
    if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i; }
    if(pt[i].X==Rx && pt[i].Y<Ry){ Ry=pt[i].Y; q=i; }
  }
```

```
  for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
  qt[dn]=pt[q]; Ly=Ry=-INF;
  for(i=0;i<n;i++){
    if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i; }
    if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i; }
  }
  for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
  rt[un]=pt[q];
}
inline int inConvex(Pt p){
  int L,R,M;
  if(p.X<Lx || p.X>Rx) return 0;
  L=0;R=dn;
  while(L<R-1){ M=(L+R)/2;
    if(p.X<qt[M].X) R=M; else L=M; }
    if(tri(qt[L],qt[R],p)<0) return 0;
    L=0;R=un;
    while(L<R-1){ M=(L+R)/2;
      if(p.X<rt[M].X) R=M; else L=M; }
      if(tri(rt[L],rt[R],p)>0) return 0;
      return 1;
}
int main(){
  int n,m,i;
  Pt p;
  scanf("%d",&n);
  for(i=0;i<n;i++) scanf("%lld%lld",&pt[i].X,&pt[i].Y);
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  scanf("%d",&m);
  for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y);
  n=minkowskiSum(n,m);
  for(i=0;i<n;i++) pt[i]=rt[i];
  initInConvex(n);
  scanf("%d",&m);
  for(i=0;i<m;i++){
    scanf("%lld %lld",&p.X,&p.Y);
    p.X*=3; p.Y*=3;
    puts(inConvex(p)?"YES":"NO");
  }
}
```

# 10 Number Theory

## 10.1 Basic

```
// Author: Gino
const int maxc = 5e5;
ll pw(ll a, ll n) {
    ll res = 1;
    while (n) {
        if (n & 1) res = res * a % MOD;
        a = a * a % MOD;
        n >>= 1;
    }
    return res;
}

vector<ll> fac, ifac;
void build_fac() {
    reset(fac, maxc + 1, 1LL);
    reset(ifac, maxc + 1, 1LL);
    for (int x = 2; x <= maxc; x++) {
        fac[x] = x * fac[x - 1] % MOD;
        ifac[x] = pw(fac[x], MOD - 2);
    }
}

ll C(ll n, ll k) {
    if (n < k) return 0LL;
    return fac[n] * ifac[n - k] % MOD * ifac[k] % MOD;
}
```

## 10.2 Prime Sieve and Defactor

```
// Author: Gino
const int maxc = 1e6 + 1;
vector<int> lpf;
vector<int> prime;

void seive() {
    prime.clear();
    lpf.resize(maxc, 1);
    for (int i = 2; i < maxc; i++) {
        if (lpf[i] == 1) {
            lpf[i] = i;
            prime.emplace_back(i);
        }
        for (auto& j : prime) {
            if (i * j >= maxc) break;
            lpf[i * j] = j;
            if (j == lpf[i]) break;
} } }
vector<pii> fac;
void defactor(int u) {
    fac.clear();
    while (u > 1) {
        int d = lpf[u];
        fac.emplace_back(make_pair(d, 0));
        while (u % d == 0) {
            u /= d;
            fac.back().second++;
} } }
```

## 10.3 Harmonic Series

```
// Author: Gino
// O(n log n)
for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j += i) {
        // O(1) code
    }
}

// PIE
// given array a[0], a[1], ..., a[n - 1]
// calculate dp[x] = number of pairs (a[i], a[j]) such
    that
//                   gcd(a[i], a[j]) = x   // (i < j)
//
// idea: let mc(x) = # of y s.t. x|y
//            f(x) = # of pairs s.t. gcd(a[i], a[j]) >=
     x
//            f(x) = C(mc(x), 2)
//           dp[x] = f(x) - sum(dp[y], x < y and x|y)
const int maxc = 1e6;
vector<int> cnt(maxc + 1, 0), dp(maxc + 1, 0);
for (int i = 0; i < n; i++)
    cnt[a[i]]++;

for (int x = maxc; x >= 1; x--) {
    ll cnt_mul = 0;  // number of multiples of x
    for (int y = x; y <= maxc; y += x)
        cnt_mul += cnt[y];

    dp[x] = cnt_mul * (cnt_mul - 1) / 2;  // number of
        pairs that are divisible by x
    for (int y = x + x; y <= maxc; y += x)
        dp[x] -= dp[y];  // PIE: subtract all dp[y] for
             y > x and x|y
}
```

## 10.4 Count Number of Divisors

```
// Author: Gino
// Function: Count the number of divisors for all x <=
    10^6 using harmonic series
const int maxc = 1e6;
vector<int> facs;

void find_all_divisors() {
    facs.clear(); facs.resize(maxc + 1, 0);
    for (int x = 1; x <= maxc; x++) {
        for (int y = x; y <= maxc; y += x) {
            facs[y]++;
        }
    }
```

```
}
```

## 10.5 數論分塊

```
// Author: Gino
/*
n = 17
  i:  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
n/i: 17  8  5  4  3  2  2  2  1  1  1  1  1  1  1  1  1
                     ^     ^
                   L(2)   R(2)

L(x) :=  left bound for n/i = x
R(x) := right bound for n/i = x

====== FORMULA ======
>>> R = n / (n/L) <<<
====================

Example: L(2) = 6
         R(2) = 17 / (17 / 6)
              = 17 / 2
              = 8
*/
// ======= CODE ========

for (ll l = 1, r = 1, q = n; l <= n; l = r + 1) {
    q = n/l;
    r = n/q;
    // Process your code here
}
// q, l, r: 17 1 1
// q, l, r: 8 2 2
// q, l, r: 5 3 3
// q, l, r: 4 4 4
// q, l, r: 3 5 5
// q, l, r: 2 6 8
// q, l, r: 1 9 17
```

## 10.6 Pollard's rho

```
// Author: Unknown
// Function: Find a non-trivial factor of a big number
    in O(n^(1/4) log^2(n))

ll find_factor(ll number) {
    __int128 x = 2;
    for (__int128 cycle = 1; ; cycle++) {
        __int128 y = x;
        for (int i = 0; i < (1<<cycle); i++) {
            x = (x * x + 1) % number;
            __int128 factor = __gcd(x - y, number);
            if (factor > 1)
                return factor;
        }
    }
}
```

```
# Author: Unknown
# Function: Find a non-trivial factor of a big number
    in O(n^(1/4) log^2(n))
from itertools import count
from math import gcd
from sys import stdin

for s in stdin:
    number, x = int(s), 2
    brk = False
    for cycle in count(1):
        y = x
        if brk:
            break
        for i in range(1 << cycle):
            x = (x * x + 1) % number
            factor = gcd(x - y, number)
            if factor > 1:
                print(factor)
                brk = True
                break
```

## 10.7 Miller Rabin

```
// Author: Unknown, Modified by Gino
// Function: Check if a number is a prime in O(100 *
    log^2(n))
//       miller_rabin(): return 1 if prime, 0 otherwise
inline ll mul(ll x, ll y, ll mod) {
    return (__int128)(x) * y % mod;
}
ll mypow(ll a, ll b, ll mod) {
    ll r = 1;
    while (b > 0) {
        if (b & 1) r = mul(r, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return r;
}
bool witness(ll a, ll n, ll u, int t){
  ll x = mypow(a, u, n);
  for(int i = 0; i < t; i++) {
    ll nx = mul(x, x, n);
    if (nx == 1 && x != 1 && x != n-1) return true;
    x = nx;
  }
  return x != 1;
}
bool miller_rabin(ll n) {
    // if n >= 3,474,749,660,383
    // change {2, 3, 5, 7, 11, 13} to
    // {2, 325, 9375, 28178, 450775, 9780504,
        1795265022}
  if (n < 2) return false;
  if(!(n & 1)) return n == 2;
  ll u = n - 1; int t = 0;
  while(!(u & 1)) u >>= 1, t++;
    for (ll a : {2, 3, 5, 7, 11, 13}) {
        if (a % n == 0) continue;
        if (witness(a, n, u, t)) return false;
    }
  return true;
}
// bases that make sure no pseudoprimes flee from test
// if WA, replace randll(n - 1) with these bases:
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
```

## 10.8 Discrete Log

```
// exbsgs — discrete log without coprimality (extended
    BSGS)
// Solve smallest x ≥ 0 s.t. a^x ≡ b (mod m) for m>1 (
    gcd(a,m) may ≠1).
// Returns true and sets x if a solution exists;
    otherwise false.
// Requires: norm_mod(a,m), pow_mod_ll(a,e,m),
    inv_mod_any(a,m,inv)

using ll = long long;

static inline bool exbsgs(ll a, ll b, ll m, ll &x){
  if (m == 1){ x = 0; return (b % 1) == 0; }

  a = norm_mod(a, m);
  b = norm_mod(b, m);

  // a ≡ 0 (mod m): a^0 ≡ 1, a^k ≡ 0 for k≥1
  if (a == 0){
    if (b == 1 % m){ x = 0; return true; }
    if (b == 0){ x = 1; return true; }
    return false;
  }
  if (b == 1 % m){ x = 0; return true; }

  ll cnt = 0;
  ll mult = 1 % m;
  while (true){
    ll g = std::gcd(a, m);
```

```
    if (g == 1) break;
    if (b % g != 0) return false;
    m /= g;
    b /= g;
    mult = (ll)((__int128)mult * (a / g) % m); // mult *= a/g
    ++cnt;
    if (mult == b){ x = cnt; return true; }
  }

  // Now gcd(a,m)==1: solve a^y ≡ b * inv(mult) (mod m) via BSGS, then x = y + cnt.
  ll inv_mult;
  if (!inv_mod_any(mult, m, inv_mult)) return false;
  ll target = (ll)((__int128)b * inv_mult % m);

  ll n = (ll)std::sqrt((long double)m) + 1;

  std::unordered_map<ll, int> baby;
  baby.reserve((size_t)(n * 1.3)); baby.max_load_factor(0.7f);

  ll aj = 1 % m;
  for (int j = 0; j < n; ++j){
    if (!baby.count(aj)) baby.emplace(aj, j);
    aj = (ll)((__int128)aj * a % m);
  }

  ll an = pow_mod_ll(a, n, m);
  ll inv_an;
  if (!inv_mod_any(an, m, inv_an)) return false;

  ll cur = target;
  for (ll i = 0; i <= n; ++i){
    auto it = baby.find(cur);
    if (it != baby.end()){
      x = cnt + i * n + it->second;
      return true;
    }
    cur = (ll)((__int128)cur * inv_an % m);
  }
  return false;
}
```

## 10.9 Discrete Sqrt

```
// tonelli_shanks — modular square root x^2 ≡ a (mod p), p an odd prime
//
// ---------------------------------------------------------

// Returns true and sets x in [0, p-1] if a is a quadratic residue mod p;
// otherwise returns false.  The other root (if x != 0) is p - x.
// Complexity: O(log p) modular multiplications.
//
// Requires: pow_mod_ll(ll a, ll e, ll m)

using ll = long long;
using u64 = unsigned long long;
using u128 = __uint128_t;

static inline bool tonelli_shanks(u64 a, u64 p, u64 &x){
  a %= p;
  if (p == 2) { x = a; return true; }
  if (a == 0) { x = 0; return true; }

  // Euler criterion: a^{(p-1)/2} ≡ 1 (mod p) iff quadratic residue
  if (pow_mod_ll((ll)a, (ll)((p - 1) >> 1), (ll)p) != 1) return false;

  // Shortcut p ≡ 3 (mod 4): x = a^{(p+1)/4} mod p
  if ((p & 3ULL) == 3ULL) {
    x = (u64)pow_mod_ll((ll)a, (ll)((p + 1) >> 2), (ll)p);
    return true;
  }

  // Write p-1 = q * 2^s with q odd
  u64 q = p - 1, s = 0;
  while ((q & 1) == 0) { q >>= 1; ++s; }

  // Find a quadratic non-residue z
  u64 z = 2;
  while (pow_mod_ll((ll)z, (ll)((p - 1) >> 1), (ll)p) != p - 1) ++z;

  // Initialize
  u64 c = (u64)pow_mod_ll((ll)z, (ll)q, (ll)p);
  u64 t = (u64)pow_mod_ll((ll)a, (ll)q, (ll)p);
  u64 r = (u64)pow_mod_ll((ll)a, (ll)((q + 1) >> 1), (ll)p);
  u64 m = s;

  // Loop until t == 1
  while (t != 1) {
    // Find least i in [1..m-1] s.t. t^(2^i) == 1
    u64 t2i = t, i = 0;
    for (i = 1; i < m; ++i) {
      t2i = (u64)((u128)t2i * t2i % p);
      if (t2i == 1) break;
    }

    // b = c^{2^{m-i-1}}
    u64 e = m - i - 1;
    u64 b = 1;
    u64 c_pow = c;
    while (e--) c_pow = (u64)((u128)c_pow * c_pow % p); // c^{2^{m-i-1}}
    b = c_pow;

    // Update r, t, c, m
    r = (u64)((u128)r * b % p);
    u64 bb = (u64)((u128)b * b % p);
    t = (u64)((u128)t * bb % p);
    c = bb;
    m = i;
  }

  x = r;
  return true;
}
```

## 10.10 Fast Power

Note: $a^n \equiv a^{(n \bmod (p-1))} (mod\ p)$

## 10.11 Extend GCD

```
// Author: Gino
// [Usage]
// bezout(a, b, c):
//      find solution to ax + by = c
//      return {-LINF, -LINF} if no solution
// inv(a, p):
//      find modulo inverse of a under p
//      return -1 if not exist
// CRT(vector<ll>& a, vector<ll>& m)
//      find a solution pair (x, mod) satisfies all x = a[i] (mod m[i])
//      return {-LINF, -LINF} if no solution

const ll LINF = 4e18;
typedef pair<ll, ll> pll;
template<typename T1, typename T2>
T1 chmod(T1 a, T2 m) {
    return (a % m + m) % m;
}

ll GCD;
pll extgcd(ll a, ll b) {
    if (b == 0) {
        GCD = a;
        return pll{1, 0};
    }
    pll ans = extgcd(b, a % b);
    return pll{ans.second, ans.first - a/b * ans.second};
}
```

```
pll bezout(ll a, ll b, ll c) {
    bool negx = (a < 0), negy = (b < 0);
    pll ans = extgcd(abs(a), abs(b));
    if (c % GCD != 0) return pll{-LINF, -LINF};
    return pll{ans.first * c/GCD * (negx ? -1 : 1),
               ans.second * c/GCD * (negy ? -1 : 1)};
}
ll inv(ll a, ll p) {
    if (p == 1) return -1;
    pll ans = bezout(a % p, -p, 1);
    if (ans == pll{-LINF, -LINF}) return -1;
    return chmod(ans.first, p);
}
pll CRT(vector<ll>& a, vector<ll>& m) {
    for (int i = 0; i < (int)a.size(); i++)
        a[i] = chmod(a[i], m[i]);

    ll x = a[0], mod = m[0];
    for (int i = 1; i < (int)a.size(); i++) {
        pll sol = bezout(mod, m[i], a[i] - x);
        if (sol.first == -LINF) return pll{-LINF, -LINF
            };

        // prevent long long overflow
        ll p = chmod(sol.first, m[i] / GCD);
        ll lcm = mod / GCD * m[i];
        x = chmod((__int128)p * mod + x, lcm);
        mod = lcm;
    }
    return pll{x, mod};
}
```

## 10.12 Mu + Phi

```
// Author: Gino
const int maxn = 1e6 + 5;
ll f[maxn];
vector<int> lpf, prime;
void build() {
lpf.clear(); lpf.resize(maxn, 1);
prime.clear();
f[1] = ...;  /* mu[1] = 1, phi[1] = 1 */
for (int i = 2; i < maxn; i++) {
    if (lpf[i] == 1) {
        lpf[i] = i; prime.emplace_back(i);
        f[i] = ...;  /* mu[i] = 1, phi[i] = i-1 */
    }
    for (auto& j : prime) {
        if (i*j >= maxn) break;
        lpf[i*j] = j;
        if (i % j == 0) f[i*j] = ...;  /* 0, phi[i]*j
            */
        else f[i*j] = ...;  /* -mu[i], phi[i]*phi[j] */
        if (j >= lpf[i]) break;
} } }
```

## 10.13 Other Formulas

- Pisano Period: 任何線性遞迴（比如費氏數列）模任何一個數字 $M$ 都會循環，找循環節 $\pi(M)$ 先質因數分解 $M = \Pi p_i^{e_i}$，然後 $\pi(M) = lcm(\pi(p_i^{e_i}))$，
- Inversion:
  $aa^{-1} \equiv 1 \pmod m$. $a^{-1}$ exists iff $\gcd(a, m) = 1$.
- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod m$
- Fermat's little theorem:
  $a^p \equiv a \pmod p$ if $p$ is prime.
- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod n$ if $\gcd(a, n) = 1$. If a, n are not coprime: 質因數分解 $n = \Pi p_i^{e_i}$，對每個 $p_i^{e_i}$ 分開看他們跟 $a$ 是否互質（互質：Fermat／不互質：夠大的指數會直接削成 0），最後用 CRT 合併。
- Extended Euclidean algorithm:
  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^{r} p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x}-1}{p_i^x-1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^{r}(a_i + 1)$.
- Chinese remainder theorem (Coprime Moduli):
  $x \equiv a_i \pmod{m_i}$.
  $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
  $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.
- Chinese remainder theorem:
  $x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$
  Solve for $(p, q)$ using ExtGCD.
  $x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{lcm(m_1, m_2)}$
- Avoiding Overflow: $ca \mod cb = c(a \mod b)$
- Dirichlet Convolution: $(f * g)(n) = \sum_{d|n} f(n)g(n/d)$
- Important Multiplicative Functions + Proterties:
  1. $\epsilon(n) = [n = 1]$
  2. $1(n) = 1$
  3. $id(n) = n$
  4. $\mu(n) = 0$ if $n$ has squared prime factor
  5. $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
  6. $\epsilon = \mu * 1$
  7. $\phi = \mu * id$
  8. $[n = 1] = \sum_{d|n} \mu(d)$
  9. $[gcd = 1] = \sum_{d|gcd} \mu(d)$
- Möbius inversion: $f = g * 1 \Leftrightarrow g = f * \mu$

## 10.14 Polynomial

```
// Author: Gino
// Preparation: first set_mod(mod, g), then init_ntt()
// everytime you change the mod, you have to call
    init_ntt() again
//
// [Usage]
// polynomial: vector<ll> a, b
// negation: -a
// add/subtract: a += b, a -= b
// convolution: a *= b
// in-place modulo: mod(a, b)
// in-place inversion under mod x^N: inv(ia, N)


const int maxk = 20;
const int maxn = 1<<maxk;

using u64 = unsigned long long;
using u128 = __uint128_t;

int g;
u64 MOD;
u64 BARRETT_IM; // 2^64 / MOD

inline void set_mod(u64 m, int _g) {
    g = _g;
    MOD = m;
    BARRETT_IM = (u128(1) << 64) / m;
}
inline u64 chmod(u128 x) {
    u64 q = (u64)((x * BARRETT_IM) >> 64);
    u64 r = (u64)(x - (u128)q * MOD);
```

```
    if (r >= MOD) r -= MOD;
    return r;
}
inline u64 mmul(u64 a, u64 b) {
    return chmod((u128)a * b);
}
ll pw(ll a, ll n) {
    ll ret = 1;
    while (n > 0) {
        if (n & 1) ret = mmul(ret, a);
        a = mmul(a, a);
        n >>= 1;
    }
    return ret;
}

vector<ll> X, iX;
vector<int> rev;
void init_ntt() {
    X.assign(maxn, 1);  // x1 = g^((p-1)/n)
    iX.assign(maxn, 1);

    ll u = pw(g, (MOD-1)/maxn);
    ll iu = pw(u, MOD-2);
    for (int i = 1; i < maxn; i++) {
        X[i] = mmul(X[i - 1], u);
        iX[i] = mmul(iX[i - 1], iu);
    }

    if ((int)rev.size() == maxn) return;
    rev.assign(maxn, 0);
    for (int i = 1, hb = -1; i < maxn; i++) {
        if (!(i & (i-1))) hb++;
        rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
} }
template<typename T>
void NTT(vector<T>& a, bool inv=false) {
    int _n = (int)a.size();
    int k = __lg(_n) + ((1<<__lg(_n)) != _n);
    int n = 1<<k;
    a.resize(n, 0);

    short shift = maxk-k;
    for (int i = 0; i < n; i++)
        if (i > (rev[i]>>shift))
            swap(a[i], a[rev[i]>>shift]);
    for (int len = 2, half = 1, div = maxn>>1; len <= n
        ; len<<=1, half<<=1, div>>=1) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < half; j++) {
                T u = a[i+j];
                T v = mmul(a[i+j+half], (inv ? iX[j*div
                    ] : X[j*div]));
                a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
                a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v)
                    ;
    } } }
    if (inv) {
        T dn = pw(n, MOD-2);
        for (auto& x : a) {
            x = mmul(x, dn);
} } }
template<typename T>
inline void shrink(vector<T>& a) {
    int cnt = (int)a.size();
    for (; cnt > 0; cnt--) if (a[cnt-1]) break;
    a.resize(max(cnt, 1));
}
template<typename T>
vector<T>& operator*=(vector<T>& a, vector<T> b) {
    int na = (int)a.size();
    int nb = (int)b.size();
    a.resize(na + nb - 1, 0);
    b.resize(na + nb - 1, 0);

    NTT(a); NTT(b);
    for (int i = 0; i < (int)a.size(); i++)
        a[i] = mmul(a[i], b[i]);
    NTT(a, true);

    shrink(a);
    return a;
}
inline ll crt(ll a0, ll a1, ll m1, ll m2, ll
    inv_m1_mod_m2){
    // x ≡ a0 (mod m1), x ≡ a1 (mod m2)
    // t = (a1 - a0) * inv(m1) mod m2
    // x = a0 + t * m1  (mod m1*m2)
    ll t = chmod(a1 - a0);
    if (t < 0) t += m2;
    t = (ll)((__int128)t * inv_m1_mod_m2 % m2);
    return a0 + (ll)((__int128)t * m1);
}
void mul_crt() {
    // a copy to a1, a2 | b copy to b1, b2
    ll M1 = 998244353, M2 = 1004535809;
    g = 3; set_mod(M1); init_ntt(); a1 *= b1;
    g = 3, set_mod(M2); init_ntt(); a2 *= b2;

    ll inv_m1_mod_m2 = pw(M1, M2 - 2);
    for (int i = 2; i <= 2 * k; i++)
        cout << crt(a1[i], a2[i], M1, M2, inv_m1_mod_m2
            ) << ' ';
    cout << endl;
}

/* P = r*2^k + 1
P                   r   k   g
998244353           119 23  3
1004535809          479 21  3

P                   r   k   g
3                   1   1   2
5                   1   2   2
17                  1   4   3
97                  3   5   5
193                 3   6   5
257                 1   8   3
7681                15  9   17
12289               3   12  11
40961               5   13  3
65537               1   16  3
786433              3   18  10
5767169             11  19  3
7340033             7   20  3
23068673            11  21  3
104857601           25  22  3
167772161           5   25  3
469762049           7   26  3
1004535809          479 21  3
2013265921          15  27  31
2281701377          17  27  3
3221225473          3   30  5
75161927681         35  31  3
77309411329         9   33  7
206158430209        3   36  22
2061584302081       15  37  7
2748779069441       5   39  3
6597069766657       3   41  5
39582418599937      9   42  5
79164837199873      9   43  5
263882790666241     15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
9097271247288401921 505 54  6 */
```

## 10.15 Counting Primes

```
// prime_count — #primes in [1..n] (O(n^{2/3}) time, O
    (sqrt(n)) memory)

using u64 = unsigned long long;
static inline u64 prime_count(u64 n){
  if(n<=1) return 0;
  int v = (int)floor(sqrt((long double)n));
  int s = (v+1)>>1, pc = 0;
```

```
  vector<int> smalls(s), roughs(s), skip(v+1);
  vector<long long> larges(s);

  for(int i=0;i<s;++i){
    smalls[i]=i;
    roughs[i]=2*i+1;
    larges[i]=(long long)((n/roughs[i]-1)>>1);
  }

  for(int p=3;p<=v;p+=2) if(!skip[p]){
    int q = p*p;
    if(1LL*q*q > (long long)n) break;
    skip[p]=1;
    for(int i=q;i<=v;i+=2*p) skip[i]=1;

    int ns=0;
    for(int k=0;k<s;++k){
      int i = roughs[k];
      if(skip[i]) continue;
      u64 d = (u64)i * (u64)p;
      long long sub = (d <= (u64)v)
        ? larges[smalls[(int)(d>>1)] - pc]
        : smalls[(int)((n/d - 1) >> 1)];
      larges[ns] = larges[k] - sub + pc;
      roughs[ns++] = i;
    }
    s = ns;
    for(int i=(v-1)>>1, j=((v/p)-1)|1; j>=p; j-=2){
      int c = smalls[j>>1] - pc;
      for(int e=(j*p)>>1; i>=e; --i) smalls[i] -= c;
    }
    ++pc;
  }

  larges[0] += 1LL*(s + 2*(pc-1))*(s-1) >> 1;
  for(int k=1;k<s;++k) larges[0] -= larges[k];

  for(int l=1;l<s;++l){
    int q = roughs[l];
    u64 m = n / (u64)q;
    long long t = 0;
    int e = smalls[(int)((m/q - 1) >> 1)] - pc;
    if(e < l+1) break;
    for(int k=l+1;k<=e;++k) t += smalls[(int)((m/ (u64)
        roughs[k] - 1) >> 1)];
    larges[0] += t - 1LL*(e - l)*(pc + l - 1);
  }
  return (u64)(larges[0] + 1);
}
```

## 10.16 Linear Sieve for Other Number Theoretic Functions

```
// linear_sieve(n, primes, lp, phi, mu, d, sigma)
// Outputs over the index range 0..n (n >= 1):
//   primes : all primes in [2..n], increasing.
//   lp     : lowest prime factor; lp[1]=0, lp[x] is the smallest prime dividing x.
//   phi    : Euler totient, phi[x] = |{1<=k<=x : gcd(k,x)=1}|.  Multiplicative.
//   mu     : Möbius; mu[1]=1, mu[x]=0 if x has a squared prime factor, else (-1)^{#distinct primes}.
//   d      : number of divisors; if x=∏ p_i^{e_i}, then d[x]=∏(e_i+1).  Multiplicative.
//   sigma  : sum of divisors; if x=∏ p_i^{e_i}, then sigma[x]=∏(1+p_i+...+p_i^{e_i}). (use ll)
//
// Complexity: O(n) time, O(n) memory.
// Notes: Arrays are resized inside; primes is cleared and reserved. sigma uses ll to avoid 32-bit overflow.

static inline void linear_sieve(
  int n,
  std::vector<int> &primes,
  std::vector<int> &lp,
  std::vector<int> &phi,
  std::vector<int> &mu,
  std::vector<int> &d,
  std::vector<ll>  &sigma
) {
  lp.assign(n + 1, 0); phi.assign(n + 1, 0); mu.assign(n + 1, 0); d.assign(n + 1, 0); sigma.assign(n + 1, 0);
  primes.clear(); primes.reserve(n > 1 ? n / 10 : 0);
  std::vector<int> cnt(n + 1, 0), core(n + 1, 1);
  std::vector<ll>  p_pow(n + 1, 1), sum_p(n + 1, 1);
  phi[1] = mu[1] = d[1] = sigma[1] = 1;

  for (int i = 2; i <= n; ++i) {
    if (!lp[i]) {
      lp[i] = i; primes.push_back(i);
      phi[i] = i - 1; mu[i] = -1; d[i] = 2;
      cnt[i] = 1; p_pow[i] = i; core[i] = 1;
      sum_p[i] = 1 + (ll)i; sigma[i] = sum_p[i];
    }
    for (int p : primes) {
      long long ip = 1LL * i * p;
      if (ip > n) break;
      lp[ip] = p;
      if (p == lp[i]) {
        cnt[ip] = cnt[i] + 1; p_pow[ip] = p_pow[i] * p; core[ip] = core[i];
        sum_p[ip] = sum_p[i] + p_pow[ip];
        phi[ip] = phi[i] * p; mu[ip] = 0;
        d[ip] = d[core[ip]] * (cnt[ip] + 1);
        sigma[ip] = sigma[core[ip]] * sum_p[ip];
        break; // critical for linear complexity
      } else {
        cnt[ip] = 1; p_pow[ip] = p; core[ip] = i;
        sum_p[ip] = 1 + (ll)p;
        phi[ip] = phi[i] * (p - 1); mu[ip] = -mu[i];
        d[ip] = d[i] * 2;
        sigma[ip] = sigma[i] * sum_p[ip];
      }
    }
  }
}

// Optional helper: factorize x in O(log x) using lp (requires x in [2..n])
static inline std::vector<std::pair<int,int>> factorize(int x, const std::vector<int>& lp) {
  std::vector<std::pair<int,int>> res;
  while (x > 1) {
    int p = lp[x], e = 0;
    do { x /= p; ++e; } while (x % p == 0);
    res.push_back({p, e});
  }
  return res;
}
```

## 10.17 GCD Convolution

```
// gcd_convolution (correct)
// -------------------------
// Given f,g on 1..N, compute h where
//   h[n] = sum_{gcd(i,j)=n} f[i] * g[j].
// Steps: multiples zeta on f,g → pointwise multiply → Möbius inversion.
// Complexity: O(N log N). Index 0 unused.
// T must support default T(0), +=, -=, *=.

template<class T>
static inline std::vector<T> gcd_convolution(const std::vector<T>& f,
                                             const std::vector<T>& g){
  int n = (int)std::min(f.size(), g.size()) - 1;
  if (n <= 0) return std::vector<T>(1, T(0));

  std::vector<T> F(f.begin(), f.begin()+n+1),
                 G(g.begin(), g.begin()+n+1);

  // multiples zeta: A[i] = sum_{m: i|m, m<=n} a[m]
  auto mult_zeta = [&](std::vector<T>& a){
    for (int i = 1; i <= n; ++i)
      for (int j = i + i; j <= n; j += i)
        a[i] += a[j];
```

```
  };
  mult_zeta(F); mult_zeta(G);

  // pointwise multiply
  std::vector<T> P(n+1);
  for (int i = 1; i <= n; ++i) P[i] = F[i] * G[i];

  // Möbius μ[1..n] by linear sieve
  std::vector<int> mu(n+1, 0), lp(n+1, 0), primes;
  mu[1] = 1;
  for (int i = 2; i <= n; ++i){
    if (!lp[i]){ lp[i] = i; primes.push_back(i); mu[i] = -1; }
    for (int p : primes){
      long long v = 1LL * i * p;
      if (v > n) break;
      lp[v] = p;
      if (i % p == 0){ mu[v] = 0; break; }  // square factor
      else mu[v] = -mu[i];
    }
  }

  // Möbius inversion over multiples:
  // h[i] = sum_{t>=1, i*t<=n} μ[t] * P[i*t]
  std::vector<T> H(n+1);
  for (int i = 1; i <= n; ++i){
    T s = T(0);
    for (int t = 1, k = i; k <= n; ++t, k += i){
      if (mu[t] == 0) continue;
      if (mu[t] > 0) s += P[k];
      else           s -= P[k];
    }
    H[i] = s;
  }
  return H;
}
```

# 11 Linear Algebra

## 11.1 Gaussian-Jordan Elimination

```
int n; vector<vector<ll> > v;
void gauss(vector<vector<ll>>& v) {
int r = 0;
for (int i = 0; i < n; i++) {
    bool ok = false;
    for (int j = r; j < n; j++) {
        if (v[j][i] == 0) continue;
        swap(v[j], v[r]);
        ok = true; break;
    }
    if (!ok) continue;
    ll div = inv(v[r][i]);
    for (int j = 0; j < n+1; j++) {
        v[r][j] *= div;
        if (v[r][j] >= MOD) v[r][j] %= MOD;
    }
    for (int j = 0; j < n; j++) {
        if (j == r) continue;
        ll t = v[j][i];
        for (int k = 0; k < n+1; k++) {
            v[j][k] -= v[r][k] * t % MOD;
            if (v[j][k] < 0) v[j][k] += MOD;
    } }
    r++;
} }
```

## 11.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = product of diagonal elements.
2. Properties of det:
   - Transpose: Unchanged
   - Row Operation 1 - Swap 2 rows: $-det$
   - Row Operation 2 - $k\overrightarrow{r_i}$: $k \times det$
   - Row Operation 3 - $k\overrightarrow{r_i}$ add to $\overrightarrow{r_j}$: Unchaged

# 12 Flow / Matching

## 12.1 Dinic

```
// Author: Benson (Extensions by Gino)
// Function: Max Flow, O(V^2 E)
// Usage: Call init(n) first, then add(u, v, w) based on your model
// (!) vertices 0-based
// >>> flow() := return max flow
// >>> find_cut() := return min cut + store cut set in dinic.cut
// >>> find_matching() := return |M| + store matching plan in dinic.matching
// >>> flow_decomposition := return max flow + store decomposition in dinic.D
#define int long long
#define eb emplace_back
#define ALL(a) a.begin(), a.end()
struct Dinic {
  struct Edge {
    // t: to | C: original capacity | c: current capacity | r: residual edge | f: current flow
    int t, C, c, r, f;
    bool fw;  // is in forward-edge graph
    Edge() {}
    Edge(int _t, int _C, int _r, bool _fw, int _f=0):
      t(_t), C(_C), c(_C), r(_r), fw(_fw), f(_f) {}
  };
  vector<vector<Edge>> G;
  vector<int> dis, iter;
  int n, s, t;
  void init(int _n) {
    n = _n;
    G.resize(n), dis.resize(n), iter.resize(n);
    for(int i = 0; i < n; ++i)
      G[i].clear();
  }
  void add(int a, int b, int c) {
    G[a].eb(b, c, G[b].size(), true);
    G[b].eb(a, 0, G[a].size() - 1, false);
  }
  bool bfs() {
    fill(ALL(dis), -1);
    dis[s] = 0;
    queue<int> que;
    que.push(s);
    while(!que.empty()) {
      int u = que.front(); que.pop();
      for(auto& e : G[u]) {
        if(e.c > 0 && dis[e.t] == -1) {
          dis[e.t] = dis[u] + 1;
          que.push(e.t);
    } } }
    return dis[t] != -1;
  }
  int dfs(int u, int cur) {
    if(u == t) return cur;
    for(int &i = iter[u]; i < (int)G[u].size(); ++i) {
      auto& e = G[u][i];
      if(e.c > 0 && dis[u] + 1 == dis[e.t]) {
        int ans = dfs(e.t, min(cur, e.c));
        if(ans > 0) {
          G[e.t][e.r].c += ans;
          e.c -= ans;
          return ans;
    } } }
    return 0;
  }
// find max flow
  int flow(int a, int b) {
    s = a, t = b;
    int ans = 0;
    while(bfs()) {
      fill(ALL(iter), 0);
      int tmp;
      while((tmp = dfs(s, INF)) > 0)
```

```
        ans += tmp;
    }
    return ans;
  }
// min cut plan
  vector<pair<pair<int, int>, int>> cut;
  int find_cut(int a, int b) {
    int cut_sz = flow(a, b);
    vector<int> vis(n, 0);
    cut.clear();
    function<void(int)> dfs = [&](int u) {
      vis[u] = 1;
      for (auto& e : G[u])
        if (e.c > 0 && !vis[e.t])
          dfs(e.t);
    };
    dfs(a);
    for (int u = 0; u < n; u++)
      if (vis[u])
        for (auto& e : G[u])
          if (!vis[e.t])
            cut.eb(make_pair(make_pair(u, e.t), G[e.t][
                e.r].c));
    return cut_sz;
  }
// bipartite matching plan
  vector<pair<int, int>> matching;
  int find_matching(int Xstart, int Xend, int Ystart,
      int Yend, int a, int b) {
    int msz = flow(a, b);
    matching.clear();
    for (int x = Xstart; x <= Xend; x++)
      for (auto& e : G[x])
        if (e.c == 0 && Ystart <= e.t && e.t <= Yend)
          matching.emplace_back(make_pair(x, e.t));
    return msz;
  }
// flow decomposition
  vector<pair<int, vector<int>>> D; // (flow amount, [
      path p1 ... pk])
  int flow_decomposition(int a, int b) {
    int mxflow = flow(a, b);

    vector<vector<Edge>> fG(n); // graph consists of
        forward edges
    for (int u = 0; u < n; u++) {
      for (auto& e : G[u]) {
        if (e.fw) {
          e.f = e.C - e.c;
          if (e.f > 0) fG[u].eb(e);
    } } }

    vector<int> vis;
    function<int(int, int)> dfs = [&](int u, int cur) {
      if (u == b) {
        D.back().second.eb(u);
        return cur;
      }
      vis[u] = 1;
      for (auto& e : fG[u]) {
        if (e.f > 0 && !vis[e.t]) {
          int ans = dfs(e.t, min(cur, e.f));
          if (ans > 0) {
            e.f -= ans;
            D.back().second.eb(u);
            return ans;
      } } }
      return 0LL;
    };
    D.clear();
    int quota = mxflow;
    while (quota > 0) {
      D.emplace_back(make_pair(0, vector<int>()));
      vis.assign(n, 0);
      int f = dfs(a, INF);
      if (f == 0) break;
      reverse(D.back().second.begin(), D.back().second.
          end());
      D.back().first = f, quota -= f;
    }
    return mxflow;
  }
};
```

## 12.2 ISAP

```
// Author: CRyptoGRapheR
#define SZ(c) ((int)(c).size())
static const int MAXV=50010;
static const int INF =1000000;
struct Maxflow{
  struct Edge{
    int v,c,r;
    Edge(int _v,int _c,int _r):v(_v),c(_c),r(_r){}
  };
  int s,t; vector<Edge> G[MAXV];
  int iter[MAXV],d[MAXV],gap[MAXV],tot;
  void init(int n,int _s,int _t){
    tot=n,s=_s,t=_t;
    for(int i=0;i<=tot;i++){
      G[i].clear(); iter[i]=d[i]=gap[i]=0;
    }
  }
  void addEdge(int u,int v,int c){
    G[u].push_back(Edge(v,c,SZ(G[v])));
    G[v].push_back(Edge(u,0,SZ(G[u])-1));
  }
  int DFS(int p,int flow){
    if(p==t) return flow;
    for(int &i=iter[p];i<SZ(G[p]);i++){
      Edge &e=G[p][i];
      if(e.c>0&&d[p]==d[e.v]+1){
        int f=DFS(e.v,min(flow,e.c));
        if(f){ e.c-=f; G[e.v][e.r].c+=f; return f; }
      }
    }
    if((--gap[d[p]])==0) d[s]=tot;
    else{ d[p]++; iter[p]=0; ++gap[d[p]]; }
    return 0;
  }
  int flow(){
    int res=0;
    for(res=0,gap[0]=tot;d[s]<tot;res+=DFS(s,INF));
    return res;
  } // reset: set iter,d,gap to 0
} flow;
```

## 12.3 Bounded Max Flow

```
// Author: CRyptoGRapheR
// Max flow with lower/upper bound on edges
// use with ISAP, l,r,a,b must be filled
int in[N],out[N],l[M],r[M],a[M],b[M];
int solve(int n, int m, int s, int t){
  flow.init(n+2,n,n+1);
  for(int i=0;i<m;i ++){
    in[r[i]]+=a[i]; out[l[i]]+=a[i];
    flow.addEdge(l[i],r[i],b[i]-a[i]);
    // flow from l[i] to r[i] must in [a[i], b[i]]
  }
  int nd=0;
  for(int i=0;i <= n;i ++){
    if(in[i]<out[i]){
      flow.addEdge(i,flow.t,out[i]-in[i]);
      nd+=out[i]-in[i];
    }
    if(out[i]<in[i])
      flow.addEdge(flow.s,i,in[i]-out[i]);
  }
  // original sink to source
  flow.addEdge(t,s,INF);
  if(flow.flow()!=nd) return -1; // no solution
  int ans=flow.G[s].back().c; // source to sink
  flow.G[s].back().c=flow.G[t].back().c=0;
  // take out super source and super sink
  for(size_t i=0;i<flow.G[flow.s].size();i++){
    Maxflow::Edge &e=flow.G[flow.s][i];
    flow.G[flow.s][i].c=0; flow.G[e.v][e.r].c=0;
  }
  for(size_t i=0;i<flow.G[flow.t].size();i++){
    Maxflow::Edge &e=flow.G[flow.t][i];
    flow.G[flow.t][i].c=0; flow.G[e.v][e.r].c=0;
```

```
  }
  flow.addEdge(flow.s,s,INF);flow.addEdge(t,flow.t,INF)
      ;
  flow.reset(); return ans+flow.flow();
}
```

## 12.4 MCMF

```
// Author: CRyptoGRapheR
// Usage:
// 1. MCMF.init(n, s, t)
// 2. MCMF.add(u, v, cap, cost)
// 3. auto [max_flow, min_cost] = MCMF.flow()
typedef int Tcost;
const int MAXV = 20010;
const int INFf = 1000000;
const Tcost INFc  = 1e9;
struct MCMF {
  struct Edge{
    int v, cap;
    Tcost w;
    int rev;
    bool fw;
    Edge(){}
    Edge(int t2, int t3, Tcost t4, int t5, bool t6)
    : v(t2), cap(t3), w(t4), rev(t5), fw(t6) {}
  };
  int V, s, t;
  vector<Edge> G[MAXV];
  void init(int n){
    V = n;
    for(int i = 0; i <= V; i++) G[i].clear();
  }
  void add(int a, int b, int cap, Tcost w){
    G[a].push_back(Edge(b, cap, w, (int)G[b].size(),
        true));
    G[b].push_back(Edge(a, 0, -w, (int)G[a].size()-1,
        false));
  }
  Tcost d[MAXV];
  int id[MAXV], mom[MAXV];
  bool inqu[MAXV];
  queue<int> q;
  pair<int, Tcost> flow(int _s, int _t){
    s = _s, t = _t;
    int mxf = 0; Tcost mnc = 0;
    while(1){
      fill(d, d+1+V, INFc); // need to use type cast
      fill(inqu, inqu+1+V, 0);
      fill(mom, mom+1+V, -1);
      mom[s] = s;
      d[s] = 0;
      q.push(s); inqu[s] = 1;
      while(q.size()){
        int u = q.front(); q.pop();
        inqu[u] = 0;
        for(int i = 0; i < (int) G[u].size(); i++){
          Edge &e = G[u][i];
          int v = e.v;
          if(e.cap > 0 && d[v] > d[u]+e.w){
            d[v] = d[u]+e.w;
            mom[v] = u;
            id[v] = i;
            if(!inqu[v]) q.push(v), inqu[v] = 1;
          }
        }
      }
      if(mom[t] == -1) break ;
      int df = INFf;
      for(int u = t; u != s; u = mom[u])
        df = min(df, G[mom[u]][id[u]].cap);
      for(int u = t; u != s; u = mom[u]){
        Edge &e = G[mom[u]][id[u]];
        e.cap              -= df;
        G[e.v][e.rev].cap += df;
      }
      mxf += df;
      mnc += df*d[t];
    }
    return make_pair(mxf, mnc);
  }
};
```

## 12.5 Hopcroft-Karp

```
// Author: Gino
// Function: Max Bipartite Matching in O(V sqrt(E))
// Usage:
// >>> init(nx, ny, m) -> add(x, y (+nx))
// >>> hk.max_matching() := the matching plan stores in
     mx, my
// >>> hk.min_vertex_cover() := the vertex cover plan
    stores in vcover
// (!) vertices are 0-based: X = [0, nx), Y = [nx, nx+
    ny)
struct HopcroftKarp {
  int n, nx, ny;
  vector<vector<int> > G;
  vector<int> mx, my;
  void init(int _nx, int _ny) {
    nx = _nx, ny = _ny;
    n = nx + ny;
    G.clear(); G.resize(n);
  }
  void add(int x, int y) {
    G[x].emplace_back(y);
    G[y].emplace_back(x);
  }
  int max_matching() {
    vector<int> dis, vis;
    mx.clear(); mx.resize(n, -1);
    my.clear(); my.resize(n, -1);

    function<bool(int)> dfs = [&](int x) {
      vis[x] = true;
      for (auto& y : G[x]) {
        int px = my[y];
        if (px == -1 ||
            (dis[px] == dis[x]+1 &&
             !vis[px] && dfs(px))) {
          mx[x] = y;
          my[y] = x;
          return true;
      } }
      return false;
    };

    while (true) {
      queue<int> q;
      dis.clear(); dis.resize(n, -1);
      for (int x = 0; x < nx; x++){
        if (mx[x] == -1) {
          dis[x] = 0;
          q.push(x);
      } }
      while (!q.empty()) {
        int x = q.front(); q.pop();
        for (auto& y : G[x]) {
          if (my[y] != -1 && dis[my[y]] == -1) {
            dis[my[y]] = dis[x] + 1;
            q.push(my[y]);
      } } }
      bool brk = true;
      vis.clear(); vis.resize(n, 0);
      for (int x = 0; x < nx; x++)
        if (mx[x] == -1 && dfs(x))
          brk = false;
      if (brk) break;
    }
    int ans = 0;
    for (int x = 0; x < nx; x++) if (mx[x] != -1) ans
        ++;
    return ans;
  }
  vector<int> vcover;
  int min_vertex_cover() {
    int ans = max_matching();
    vcover.clear();

    vector<int> vis(n, 0);
    function<void(int)> dfs = [&](int x) {
      vis[x] = true;
```

```
      for (auto& y : G[x]) {
        if (y == mx[x] || my[y] == -1 || vis[y])
            continue;
        vis[y] = true;
        dfs(my[y]);
      }
    };

    for (int x = 0; x < nx; x++) if (mx[x] == -1) dfs(x
        );
    for (int x = 0; x < nx; x++) if (!vis[x]) vcover.
        emplace_back(x);
    for (int y = nx; y < nx + ny; y++) if (vis[y])
        vcover.emplace_back(y);
    return ans;
  }
} hk;
```

## 12.6 Cover / Independent Set

```
最大邊獨立集 (Ie) 就是最大匹配 (M)
二分圖上，M 和 Cv 對偶
對任何圖都有 |Iv| + |Cv| = |V|
對任何圖都有 |Ie| + |Ce| = |V|
二分圖最小帶權點覆蓋 => 建模 (s, u, w[u]) (u, v, INF) (
    v, t, w[v]) 算最小割
```

## 12.7 Kuhn Munkres

```
// Author: CRyptoGRapheR
static const int MXN=2001;// 1-based
static const ll INF=0x3f3f3f3f;
struct KM{ // max weight,for min negate the weights
  int n,mx[MXN],my[MXN],pa[MXN]; bool vx[MXN],vy[MXN];
  ll g[MXN][MXN],lx[MXN],ly[MXN],sy[MXN];
  void init(int _n){
    n=_n; for(int i=1;i<=n;i++) fill(g[i],g[i]+n+1,0);
  }
  void addEdge(int x,int y,ll w){ g[x][y]=w; }
  void augment(int y){
    for(int x,z;y;y=z) x=pa[y],z=mx[x],my[y]=x,mx[x]=y;
  }
  void bfs(int st){
    for(int i=1;i<=n;++i) sy[i]=INF,vx[i]=vy[i]=0;
    queue<int> q;q.push(st);
    for(;;){
      while(q.size()){
        int x=q.front();q.pop();vx[x]=1;
        for(int y=1;y<=n;++y) if(!vy[y]){
          ll t=lx[x]+ly[y]-g[x][y];
          if(t==0){
            pa[y]=x;
            if(!my[y]){ augment(y); return; }
            vy[y]=1,q.push(my[y]);
          }else if(sy[y]>t) pa[y]=x,sy[y]=t;
        }
      }
      ll cut=INF;
      for(int y=1;y<=n;++y)
        if(!vy[y]&&cut>sy[y]) cut=sy[y];
      for(int j=1;j<=n;++j){
        if(vx[j]) lx[j]-=cut;
        if(vy[j]) ly[j]+=cut;
        else sy[j]-=cut;
      }
      for(int y=1;y<=n;++y) if(!vy[y]&&sy[y]==0){
        if(!my[y]){ augment(y); return; }
        vy[y]=1,q.push(my[y]);
  } } }
  ll solve(){
    fill(mx,mx+n+1,0);fill(my,my+n+1,0);
    fill(ly,ly+n+1,0);fill(lx,lx+n+1,-INF);
    for(int x=1;x<=n;++x) for(int y=1;y<=n;++y)
      lx[x]=max(lx[x],g[x][y]);
    for(int x=1;x<=n;++x) bfs(x);
    ll ans=0;
    for(int y=1;y<=n;++y) ans+=g[my[y]][y];
    return ans;
  }
}graph;
```

# 13 Combinatorics

## 13.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 5 |
| 4 | 14 | 42 | 132 | 429 |
| 8 | 1430 | 4862 | 16796 | 58786 |
| 12 | 208012 | 742900 | 2674440 | 9694845 |

## 13.2 Bertrand's Ballot Theorem

- $A$ always $> B$: $C(p+q,p) - 2C(p+q-1,p)$
- $A$ always $\geq B$: $C(p+q,p) \times \frac{p+1-q}{p+1}$

## 13.3 Burnside's Lemma

Let $X$ be the original set.
Let $G$ be the group of operations acting on $X$.
Let $X^g$ be the set of $x$ not affected by $g$.
Let $X/G$ be the set of orbits.
Then the following equation holds:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

# 14 Special Numbers

## 14.1 Fibonacci Series

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 3 |
| 5 | 5 | 8 | 13 | 21 |
| 9 | 34 | 55 | 89 | 144 |
| 13 | 233 | 377 | 610 | 987 |
| 17 | 1597 | 2584 | 4181 | 6765 |
| 21 | 10946 | 17711 | 28657 | 46368 |
| 25 | 75025 | 121393 | 196418 | 317811 |
| 29 | 514229 | 832040 | 1346269 | 2178309 |
| 33 | 3524578 | 5702887 | 9227465 | 14930352 |

$f(45) \approx 10^9, f(88) \approx 10^{18}$

## 14.2 Prime Numbers

- First 50 prime numbers:

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 7 | 11 |
| 6 | 13 | 17 | 19 | 23 | 29 |
| 11 | 31 | 37 | 41 | 43 | 47 |
| 16 | 53 | 59 | 61 | 67 | 71 |
| 21 | 73 | 79 | 83 | 89 | 97 |
| 26 | 101 | 103 | 107 | 109 | 113 |
| 31 | 127 | 131 | 137 | 139 | 149 |
| 36 | 151 | 157 | 163 | 167 | 173 |
| 41 | 179 | 181 | 191 | 193 | 197 |
| 46 | 199 | 211 | 223 | 227 | 229 |

- Very large prime numbers:
  1000001333 1000500889 2500001909
  2000000659 900004151 850001359
- $\pi(n) \equiv$ Number of primes $\leq n \approx n/((\ln n) - 1)$
  $\pi(100) = 25, \pi(200) = 46$
  $\pi(500) = 95, \pi(1000) = 168$
  $\pi(2000) = 303, \pi(4000) = 550$
  $\pi(10^4) = 1229, \pi(10^5) = 9592$
  $\pi(10^6) = 78498, \pi(10^7) = 664579$