Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, . . .) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, . . .)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i \in$[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. RMQ (static) - MODIFICAR

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: LVL $\geq$ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
struct RMQ{
  #define LVL 10
  tipo vec[LVL][1<<(LVL+1)];
  tipo &operator[](int p){return vec[0][p];}
  tipo get(int i, int j) {//intervalo [i,j)
    int p = 31-__builtin_clz(j-i);
    return min(vec[p][i],vec[p][j-(1<<p)]);
  }
  void build(int n) {//O(nlogn)
    int mp = 31-__builtin_clz(n);
    forn(p, mp) forn(x, n-(1<<p))
      vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
  }};
```

2.2. Segment Tree

2.2.1. Segment Tree Recursivo

2.2.2. ST Iterativo - (Consulta en rango, modificacion a posicion)

2.2.3. ST Iterativo - (Consulta a posicion, modificacion en rango)

```
/*Segment Tree modificar un rango, acceder a una posicion
  solo sirve cuando la operacion que realizamos es conmutativa
  por ejemplo la suma, pero no funciona con la asignacion
*/
//adiciona value al rango [l, r)
void modify(int l, int r, int value) {// rango [l, r)
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) t[l++] += value;
    if (r&1) t[--r] += value;
  }
}
//acceder a la posicion
int query(int p) {
  int res = 0;
  for (p += n; p > 0; p >>= 1) res += t[p];
  return res;
}
//Si necesitamos actualizar todo lo podemos hacer en O(n)
//Y luego acceder a las hojas en O(1)
void push() {
  for (int i = 1; i < n; ++i) {
    t[i<<1] += t[i];
    t[i<<1|1] += t[i];
    t[i] = 0;
  }
}
```

2.2.4. Segment Tree con Punteros

2.2.5. Segment Tree 2D

2.2.6. Segment Tree Lazy - Suma

2.2.7. Segment Tree Lazy - Pintar

2.2.8. Segment Tree Persistente

2.3. Fenwick Tree

2.3.1. Fenwick Tree 2D

2.4. Union Find con rank

```
/*=============== <Union find rangos> ===================
Complexity: O(N)
index 0 to n - 1 warning
Complexity O(N)
*/
#define MAX INSERTE_VALOR_AQUI
int padre[MAX];
int rango[MAX];
void MakeSet(int n){
    for (int i = 0 ; i < n ; ++i) {
        padre[i] = i; rango[i] = 0; }
}
int Find(int x) {
    if(x == padre[x])
        return x;
    return padre[x] = Find(padre[x]);
}
void UnionbyRank(int x , int y){
    int xRoot = Find(x);
    int yRoot = Find(y);
    //el padre de ambas componentes sera el de mayor altura
    if(rango[xRoot] > rango[yRoot])//X tiene mas altura que Y
        padre[yRoot] = xRoot;
    }else{//Y >= X
        padre[xRoot] = yRoot;
        if(rango[xRoot] == rango[yRoot])//si poseen la misma altura
            rango[yRoot]++;//incremento el rango de la nueva raiz
    }
}
```

2.5. BigInteger C++

```
// g++ -std=c++11 "bigint.cpp" -o run
/***
=============== <Big Int c++ version> ===================
Contain a useful big int, overload all operators, including cin, cout,
comparator, build via string (prefer this metod) or long long, for now this not have a
to_string method
Problem for practice: UVA 494
*/
// base and base_digits must be consistent
const int base = 1000000000;
const int base_digits = 9;

struct bigint {
    vector<int> a;
    int sign;

    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        for (; v > 0; v = v / base)
            a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i) {
                if (i == (int) res.a.size())
                    res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)
                    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
                    res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
                    carry = res.a[i] < 0;
                    if (carry)
                        res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
    }

    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }

    friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());

        for (int i = a.a.size() - 1; i >= 0; i--) {
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2) / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }

    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }

    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }

    void operator/=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long) base;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }

    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }

    int operator%(int v) const {
        if (v < 0)
            v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long) base) % v;
        return m * sign;
    }

    void operator+=(const bigint &v) {
        *this = *this + v;
    }
    void operator-=(const bigint &v) {
        *this = *this - v;
    }
    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }

    bool operator<(const bigint &v) const {
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] * sign;
        return false;
    }

    bool operator>(const bigint &v) const {
        return v < *this;
    }
    bool operator<=(const bigint &v) const {
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const {
        return !(*this < v);
    }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
    }

    void trim() {
        while (!a.empty() && !a.back())
            a.pop_back();
        if (a.empty())
            sign = 1;
    }

    bool isZero() const {
        return a.empty() || (a.size() == 1 && !a[0]);
    }

    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }

    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--)
            res = res * base + a[i];
        return res * sign;
    }

    friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
    }

    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == ’-’ || s[pos] == ’+’)) {
            if (s[pos] == ’-’)
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - ’0’;
            a.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1)
            stream << ’-’;
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill(’0’) << v.a[i];
        return stream;
    }

    static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
            p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++)
            a2[i] += a1[i];
        for (int i = 0; i < k; i++)
            b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)
            r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            r[i] -= a2b2[i];
```

```cpp

        for (int i = 0; i < (int) r.size(); i++)
            res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)
            res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            res[i + n] += a2b2[i];
        return res;
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
};

int main() {
    bigint a=0;
    //("99999999999999999999999999999999999999999999999999");
    bigint b;
    //("19999999999999999999999999999999999999999999998");
    bigint n;
    while(cin >> n) {
        if(n==0){break;}
        a += n;
    }
    cout<<a<<endl;
}
```

2.6. UnorderedSet

```cpp
//Compilar: g++ --std=c++11
struct Hash{
  size_t operator()(const ii &a)const{
    size_t s=hash<int>()(a.fst);
    return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
  }
  size_t operator()(const vector<int> &v)const{
    size_t s=0;
    for(auto &e : v)
      s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
    return s;
  }
};
unordered_set<ii, Hash> s;
unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.7. Ordered Set

```cpp
/*
 A brief explanation about use of a powerful library: orderd_set
 Reference link: http://codeforces.com/blog/entry/11080
 and a hash for the type pair
 */

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
/*
If we want to get map but not the set, as the second argument type must
    be used mapped type. Apparently,
 the tree supports the same operations as the set (at least I haven't
    any problems with them before),
 but also there are two new features - it is find_by_order() and
    order_of_key().
 The first returns an iterator to the k-th largest element (counting from zero), the second - the number of items
in a set that are strictly smaller than our item. Example of use:
*
* */
```

2.8. Treap Modo Set

2.9. Treap Implicito(Rope)

2.10. Treap - Toby and Bones

2.11. Convex Hull Trick Estatico

```
// g++  "convexhulltrick.cpp" -o run
/***
=============== <Convex hull trick normal version> ===================
Contain a sample about convex hull trick optimization this recivie N pairs:
a "value of length" and a cost, we need to minimize the value of grouping
this pairs taken the most large pair as the cost of the group

Problem for practice: aquire
*/
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope
vector<long long> B; //Holds the y-intercepts of the lines in the envelope
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
  /*
  intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
  intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
  set the former greater than the latter, and cross-multiply to
  eliminate division
  */
  return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
  //First, let's add it to the end
  M.push_back(m);
  B.push_back(b);
  //If the penultimate is now made irrelevant between the antepenultimate
  //and the ultimate, remove it. Repeat as many times as necessary
  while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
  {
    M.erase(M.end()-2);
    B.erase(B.end()-2);
  }
}
//Returns the minimum y-coordinate of any intersection between a given vertical
//line and the lower envelope
long long query(long long x)
{
  //If we removed what was the best line for the previous query, then the
  //newly inserted line is now the best for that query
  if (pointer>=M.size())
    pointer=M.size()-1;
  //Any better line must be to the right, since query values are
  //non-decreasing
  while (pointer<M.size()-1&&
    M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
    pointer++;
  return M[pointer]*x+B[pointer];
}
int main()
{
  int M,N,i;
  pair<int,int> a[50000];
  pair<int,int> rect[50000];
  scanf("%d",&M);
  for (i=0; i<M; i++)
    scanf("%d␣%d",&a[i].first,&a[i].second);
  //Sort first by height and then by width (arbitrary labels)
  sort(a,a+M);
```

```
 for (i=0,N=0; i<M; i++)
 {
  /*
  When we add a higher rectangle, any rectangles that are also
  equally thin or thinner become irrelevant, as they are
  completely contained within the higher one; remove as many
  as necessary
  */
  while (N>0&&rect[N-1].second<=a[i].second)
   N--;
  rect[N++]=a[i]; //add the new rectangle
 }
 long long cost;
 add(rect[0].second,0);
 //initially, the best line could be any of the lines in the envelope,
 //that is, any line with index 0 or greater, so set pointer=0
 pointer=0;
 for (i=0; i<N; i++)
 {
  cost=query(rect[i].first);
  if (i<N)
   add(rect[i+1].second,cost);
 }
 printf("%lld\n",cost);
 return 0;
}
```

2.12. Convex Hull Trick Dinamico

```
// g++ -std=c++11 "convexhulltrick_dynamic.cpp" -o run
/***
=============== <Convex hull trick dynamic version version> ===================
warning with the use of this, this is a black box, try to use only in an emergency.
Problem for practice: aquire
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll is_query = -(1LL<<62);
 struct Line {
 ll m, b;
 mutable multiset<Line>::iterator it;
 const Line *succ(multiset<Line>::iterator it) const;
 bool operator<(const Line& rhs) const {
  if (rhs.b != is_query) return m < rhs.m;
  const Line *s=succ(it);
  if(!s) return 0;
  ll x = rhs.m;
  return b - s->b < (s->m - m) * x;
 }
};
struct HullDynamic : public multiset<Line>{ // will maintain upper hull for maximum
 bool bad(iterator y) {
  iterator z = next(y);
  if (y == begin()) {
   if (z == end()) return 0;
   return y->m == z->m && y->b <= z->b;
  }
  iterator x = prev(y);
  if (z == end()) return y->m == x->m && y->b <= x->b;
  return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
 }
 iterator next(iterator y){return ++y;}
 iterator prev(iterator y){return --y;}
  void insert_line(ll m, ll b) {
  iterator y = insert((Line) { m, b });
  y->it=y;
  if (bad(y)) { erase(y); return; }
  while (next(y) != end() && bad(next(y))) erase(next(y));
  while (y != begin() && bad(prev(y))) erase(prev(y));
 }
 ll eval(ll x) {
  Line l = *lower_bound((Line) { x, is_query });
  return l.m * x + l.b;
 }
}h;
const Line *Line::succ(multiset<Line>::iterator it) const{
return (++it==h.end()? NULL : &*it);}
```

2.13. Misof Tree

```
/*
http://codeforces.com/blog/entry/10493#comment-159335
```

```
Sirve para encontrar el i-esimo numero de un conjunto de numeros que vamos insertando en el arbol.
Sirve solo si nuestros numeros son del 0 al n-1 (pero podemos mapearlos antes de usarlos)
La idea es esta:
Funcionamiento:
  - En el fondo sigue siendo un Segment-Tree (hacemos que 'n' sea 2^x)
  - Cada nodo guarda cuantos numeros hay en el intervalo (entonces en tree[1] dice cuantos numeros tenemos en total)
  - Se sigue representando los hijos del nodo 'i' con '2 * i' (izq) y '2 * i + 1' (der);
Query:
  - si kth es mas grande que todos los que tenemos(tree[1]) o es negativo entonces -1
  - siempre nos mantenemos en el nodo de la izquierda y si es necario avanzamos al de la derecha
                      'i <<= 1'
    - si kth es mas grande que el nodo de la izquierda(el actual) quiere decir que podemos quitarle todos esos
    numeros a nuestra busqueda 'kth - tree[i]' y buscar el nuevo kth en el arbol de la derecha
      if (kth > tree [i]) kth -= tree [i++];
    - Ojo en el 'i++' ahi es donde avanzamos al nodo de la derecha
  - luego hace su formula rara que aun no entendi xD:
      'i - leaf + (kth > tree [i])';
*/
const int MaxN = 1e6;

int a [MaxN], s [MaxN];
int leaf, tree [100 + MaxN << 2];

void bld (int n) { leaf = 1 << (32 - __builtin_clz (n)); }
void add (int x) { for (int i = leaf + x; i; i >>= 1) ++tree [i]; }// Podemos insertar mas de una copia la vez tree [i] += xcopies;
void del (int x) { for (int i = leaf + x; i; i >>= 1) --tree [i]; }// Podemos eliminar mas de una copia la vez tree [i] -= xcopies;
// en "leaf + x" esta cuantas copias tenemos de "x"
//Cuidado con intentar hacer del con mas copias de las disponibles, el kth() no funcionaria
long kth  (int kth, int i = -1) {
    if (kth > tree [1] || kth <= 0) return i;
  for (i = 1; i < leaf; i <<= 1) if (kth > tree [i]) kth -= tree [i++];
    return i - leaf + (kth > tree [i]);
}
```

2.14. SQRT Decomposition Basic

```
const int maxn = 500010;
int n;

tipo v[maxn];//vector principal

tipo lazy[maxn];
pair<tipo, tipo> t[maxn];//para poder reordenar los elementos

int SQRT;
int N;//nro. de buckets

//Recalcula y aplica el lazy al bucket con indice idx
//guarda la informacion necesaria del bucket en otros vectores
//podria ser la suma del bucket, o el min/max del bucket
void recalc(int idx) {
  int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
  for (int i = a; i < b; i++) {
    v[i] += lazy[idx];
    t[i] = make_pair(v[i], i);
  }
  lazy[idx] = 0;
  sort(t + a, t + b);
}

//adiciona delta a todos los elementos
//en el intervalo cerrado [a, b]
void add(int a, int b, tipo delta) {
  int idx_a = a / SQRT, idx_b = b / SQRT;
  if (idx_a == idx_b) {
    for (int i = a; i <= b; i++)
      v[i] += delta;
    recalc(idx_a);
  } else {
    //head
    for (int i = a, lim = min(n, (idx_a + 1) * SQRT); i < lim; i++)
      v[i] += delta;
    recalc(idx_a);//OJO puede ser necesario
    //body
    for (int i = idx_a + 1; i < idx_b; i++)
```

```
      lazy[i] += delta;
    //tail
    for (int i = idx_b * SQRT; i <= b; i++)
      v[i] += delta;
    recalc(idx_b);//OJO puede ser necesario
  }
}

//tambien podria ser en un rango como en el add
tipo query(tipo val) {
  tipo ans = 0;
  //recorro todos los buckets
  for (int idx = 0; idx < N; idx++) {
    int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
    //... hacer algo ...
  }
  return ans;
}
int main() {
  //leer n, q y los elementos de v

  SQRT = (int)sqrt(n) + 1;
  N = (n + SQRT - 1) / SQRT;//nro. de buckets
  //construir cada bucket
  for (int idx = 0; idx < N; idx++)
    recalc(idx);

  //resto del programa
  return 0;
}
```

2.15. Nro. Elementos menores o iguales a x en $O(log(n))$

```
//insersion y consulta de cuantos <= en log n
struct leqset {
  int maxl; vector<int> c;
  int pref(int n, int l) { return (n>>(maxl-l))|(1<<l); }
  void ini(int ml) { maxl=ml; c=vector<int>(1<<(maxl+1)); }
  //inserta c copias de e, si c es negativo saca c copias
  void insert(int e, int q=1) { forn(l,maxl+1) c[pref(e,l)]+=q; }
  int leq(int e) {
    int r=0,a=1;
    forn(i,maxl) {
      a<<=1; int b=(e>>maxl-i-1)&1;
      if (b) r+=c[a]; a|=b;
    } return r + c[a]; //sin el c[a] da los estrictamente menores
  }
  int size() { return c[1]; }
  int count(int e) { return c[e|(1<<maxl)]; }
};
```

3. Algos

3.1. LIS en O(n log n) con Reconstruccion

```
//Para non-increasing, cambiar comparaciones y revisar busq binaria
//Given an array, paint it in the least number of colors so that each color turns to a non-increasing subsequence.
//Solution:Min number of colors=Length of the longest increasing subsequence

// Las lineas marcadas con // Camino no son necesarias si no se desea reconstruir el camino.
#define MAXN 1000000
int v[MAXN]; // INPUT del algoritmo.
int mv[MAXN];
int mi[MAXN] ,p[MAXN]; // Camino
int l[MAXN]; // Aca apareceria la maxima subsecuencia creciente(los indices)
int lis(int n) {
  forn(i,n) mv[i] = INF;
  forn(i,n) mi[i] = -1; // Camino
  forn(i,n) p [i] = -1; // Camino
  mv[0] = -INF;
  int res = 0;
  forn(i,n) {
    // Con upper_bound es maxima subsecuencia no decreciente.
    // Con lower_bound es maxima subsecuencia creciente.
    int me = upper_bound(mv,mv+n,v[i]) - mv;
    p[i] = mi[me-1]; // Camino
    mv[me] = v[i];
    mi[me] = i; // Camino
    if (me > res) res = me;
  }
  for(int a = mi[res], i = res - 1;a != -1; a = p[a], i--) // Camino
    l[i] = a; // Indices: poniendo l[i] = v[a] quedan los valores.
  return res;
}
```

3.2. Mo

```
// g++ -std=c++11 "mo.cpp" -o run
/***
=============== <Mo> ===================
Contain a sample about Mo algorithm
Brief explanation when use Mo:
Explain where and when we can use above algorithm

As mentioned, this algorithm is offline, that means we cannot use it when we are forced to stick to given order of queries.
 That also means we cannot use this when there are update operations. Not just that, there is one important possible limitation:
We should be able to write the functions add and remove. There will be many cases where add is trivial but remove is not.
One such example is where we want maximum in a range. As we add elements, we can keep track of maximum. But when we remove elements
it is not trivial. Anyways in that case we can use a set to add elements, remove elements and report minimum.
In that case the add and delete operations are O(log N) (Resulting in O(N * Sqrt(N) * log N) algorithm).

Suggestion first use the add operation, then the erase operation
Problem for practice: DQUERY spoj
Input: N, then N elements of array M querys with a range L,R
*/
const int MAXV = 1e6 + 10;
const int N = 30010;
const int M = 200010;
int cnt[MAXV];
int v[N];

struct query{
  int l,r,pos;
  query(){}
};
int n;
query qu[M];
int ans[M];

int ret = 0;
void add(int pos){
  pos = v[pos];
  cnt[pos]++;
  if(cnt[pos] == 1){
    ret++;
  }
}
void erase(int pos){
  pos = v[pos];
  cnt[pos]--;
  if(!cnt[pos])ret--;
}
int main(){
  n = in();
  for(int i = 0; i < n;i++){
    v[i] = in();
  }
  int block = ceil(sqrt(n));
  int q = in();
  for(int i = 0; i < q;i++){
    qu[i].l = in() - 1,qu[i].r = in() - 1,qu[i].pos = i;
  }
  sort(qu,qu + q,[&](const query &a,const query &b){
    if(a.l / block != b.l / block)
      return a.l / block < b.l / block;
    return a.r < b.r;
  });
  int l = 0, r = 0;
  for(int i = 0; i < q;i++){
    int nl = qu[i].l,nr = qu[i].r;
    while(l > nl){
      add(--l);
    }
    while(r <= nr){
      add(r++);
    }
    while(l < nl){
      erase(l++);
    }
    while(r > nr + 1){
      erase(--r);
    }

    ans[qu[i].pos] = ret;
  }
  for(int i = 0; i < q;i++)printf("%d\n",ans[i]);
}
```

4. Strings

4.1. Manacher

```
vector<int> manacher(const string &_s) {
  int n = _s.size();
  string s(2 * n + 3, '#');
  s[0] = '%', s[s.size() - 1] = '$';//no deben estar en la cadena
  for (int i = 0; i < n; i++)
    s[(i + 1) * 2] = _s[i];

  n = s.size();
  vector<int> P(n, 0);
  int C = 0, R = 0;
  for (int i = 1; i < n - 1; i++) {
    int j = C - (i - C);
    if (R > i)
      P[i] = min(R - i, P[j]);
    while (s[i + 1 + P[i]] == s[i - 1 - P[i]])
      P[i]++;
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }
  return P;
}
bool is_pal(const vector<int> &mnch_vec, int i, int j) {//[i, j] - i<=j
  int len = j - i + 1;
  i = (i + 1) * 2;//idx to manacher vec idx
  j = (j + 1) * 2;
  int mid = (i + j) / 2;
  return mnch_vec[mid] >= len;
}
int main() {
  string s;
  cin >> s;
  vector<int> mnch_vec= manacher(s);
  if (is_pal(mnch_vec, 2, 7)) {
    //la subcadena desde la posicion 2 a la 7 es palindrome
  }
  return 0;
}
```

4.2. Trie - Punteros y bfs

4.3. Suffix Array O(n log n) con LCP (Kasai) O(n)

4.4. Minima rotacion lexicografica

```
/*
Rotacion Lexicografica minima MinRotLex(cadena,tamanio)
para cambiar inicio de la cadena char s[300]; int h; s+h;
retorna inicio de la rotacion minima :D
*/
int MinRotLex(const char *s, const int slen) {
   int i = 0, j = 1, k = 0, x, y, tmp;
   while(i < slen && j < slen && k < slen) {
      x = i + k;
      y = j + k;
      if(x >= slen)  x -= slen;
      if(y >= slen)  y -= slen;
      if(s[x] == s[y]) {
         k++;
      } else if(s[x] > s[y]) {
         i = j+1 > i+k+1 ? j+1 : i+k+1;
         k = 0;
         tmp = i, i = j, j = tmp;
      } else {
         j = i+1 > j+k+1 ? i+1 : j+k+1;
         k = 0;
      }
   }
   return i;
}
int main(){
  int n;
  scanf("%d",&n);getchar();
  while(n--){
    char str[1000009];
    gets(str);
    printf("%d\n",MinRotLex(str,strlen(str))+1);
  }
}
```

4.5. Matching

4.5.1. KMP

```
string T;//cadena donde buscar(where)
string P;//cadena a buscar(what)
int b[MAXLEN];//back table b[i] maximo borde de [0..i)
void kmppre(){//by gabina with love
    int i =0, j=-1; b[0]=-1;
    while(i<sz(P)){
        while(j>=0 && P[i] != P[j]) j=b[j];
        i++, j++, b[i] = j;
    }
}
void kmp(){
    int i=0, j=0;
    while(i<sz(T)){
        while(j>=0 && T[i]!=P[j]) j=b[j];
        i++, j++;
        if(j==sz(P)) printf("P␣is␣found␣at␣index␣%d␣in␣T\n", i-j), j=b[j];
    }
}

int main(){
    cout << "T=";
    cin >> T;
    cout << "P=";
    cin.ignore();
    cin >> P;
    kmppre();
    kmp();
    return 0;
}
```

4.5.2. Z - Por aprender

4.5.3. Matching con suffix array

4.5.4. Matching con BWT

4.5.5. Matching con Aho-Corasick

```

struct trie{
  map<char, trie> next;
  trie* tran[256];//transiciones del automata
  int idhoja, szhoja;//id de la hoja o 0 si no lo es
  //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es hoja
  trie *padre, *link, *nxthoja;
  char pch;//caracter que conecta con padre
  trie(): tran(),  idhoja(), padre(), link() {}
  void insert(const string &s, int id=1, int p=0){//id>0!!!
    if(p<sz(s)){
      trie &ch=next[s[p]];
      tran[(int)s[p]]=&ch;
      ch.padre=this, ch.pch=s[p];
      ch.insert(s, id, p+1);
    }
    else idhoja=id, szhoja=sz(s);
  }
  trie* get_link() {
    if(!link){
      if(!padre) link=this;//es la raiz
      else if(!padre->padre) link=padre;//hijo de la raiz
      else link=padre->get_link()->get_tran(pch);
    }
    return link; }
  trie* get_tran(int c) {
    if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
    return tran[c]; }
  trie *get_nxthoja(){
    if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
    return nxthoja; }
  void print(int p){
    if(idhoja) cout << "found␣" << idhoja << "␣␣at␣position␣" << p-szhoja << endl;
    if(get_nxthoja()) get_nxthoja()->print(p); }
  void matching(const string &s, int p=0){
    print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
}tri;


int main(){
  tri=trie();//clear
  tri.insert("ho", 1);
  tri.insert("hoho", 2);
```

4.6. Suffix Automaton

```
/*#################### Suffix Automata ####################*/
const int N = INSERTE_VALOR;//maxima longitud de la cadena
struct State {  //OJO!!! tamanio del alfabeto, si MLE -> map
    State *pre,*go[26];//se puede usar un map<char, State*> go
    int step;
    void clear() {
        pre=0;
        step=0;
        memset(go,0,sizeof(go));//go.clear();
    }
} *root,*last;
State statePool[N * 2],*cur;
void init() {
    cur=statePool;
    root=last=cur++;
    root->clear();
}
void Insert(int w) {
    State *p=last;
    State *np=cur++;
    np->clear();
    np->step=p->step+1;
    while(p&&!p->go[w])
        p->go[w]=np,p=p->pre;
    if(p==0)
        np->pre=root;
    else {
        State *q=p->go[w];
        if(p->step+1==q->step)
            np->pre=q;
        else {
            State *nq=cur++;
            nq->clear();
            memcpy(nq->go,q->go,sizeof(q->go));//nq->go = q->go; para mapa
            nq->step=p->step+1;
            nq->pre=q->pre;
            q->pre=nq;
            np->pre=nq;
            while(p&&p->go[w]==q)
                p->go[w]=nq, p=p->pre;
        }
    }
    last=np;
}
/*#################### Suffix Automata ####################*/

/*#################### Algunas aplicaciones ####################*/
//Obtiene el LCSubstring de 2 cadenas en O(|A| + |B|)
string lcs(char A[N], char B[N]) {
    int n,m;
    n = strlen(A); m = strlen(B);
    //Construccion: O(|A|)
    //solo hacerlo una vez si A no cambia
    init();
    for(int i=0; i<n; i++)
        Insert(A[i]-'a'); //Fin construccion
    //LCS: O(|B|)
    int ans = 0, len = 0, bestpos = 0;
    State *p = root;
    for(int i = 0; i < m; i++) {
        int x = B[i]-'a';
        if(p->go[x]) {
            len++;
            p = p->go[x];
        } else {
            while (p && !p->go[x]) p = p->pre;
            if(!p) p = root, len = 0;
            else   len = p->step+1, p = p->go[x];
        }
        if (len > ans)
            ans = len, bestpos = i;
    }
    //return ans; //solo el tamanio del lcs
    return string(B + bestpos - ans + 1, B + bestpos + 1);
}

/*Numero de subcadenas distintas + 1(subcadena vacia) en O(|A|)
OJO: Por alguna razon Suffix Array es mas rapido
Se reduce a contar el numero de paths que inician en q0 y terminan
en cualquier nodo. dp[u] = # de paths que inician en u
- Se debe construir el automata en el main(init y Insert's)
- Setear dp en -1
*/
number dp[N * 2];
number num_dist_substr(State *u = root) {
    if (dp[u - statePool] != -1) return dp[u - statePool];
    number ans = 1;//el path vacio que representa este nodo
    for (int v = 0; v < 26; v++)//usar for (auto) para mapa
        if (u->go[v])
            ans += num_dist_substr(u->go[v]);
    return (dp[u - statePool] = ans);
}

/*Suma la longitud de todos los substrings en O(|A|)
- Construir el automata(init y insert's)
- Necesita el metodo num_dist_substr (el de arriba)
- setear dp's en -1
*/
number dp1[N * 2];
number sum_length_dist_substr(State *u = root) {
    if (dp1[u - statePool] != -1) return dp1[u - statePool];
    number ans = 0;//el path vacio que representa este nodo
    for (int v = 0; v < 26; v++)//usar for (auto) para mapa
        if (u->go[v])
            ans += (num_dist_substr(u->go[v]) + sum_length_dist_substr(u->go[v]));
    return (dp1[u - statePool] = ans);
}

/*
Pregunta si p es subcadena de la cadena con la cual esta construida
el automata.
Complejidad: - Construir O(|Texto|) - solo una vez (init e insert's)
             - Por Consulta O(|patron a buscar|)
*/
bool is_substring(char p[N]) {
    State *u = root;
    for (int i = 0; p[i]; i++) {
        if (!u->go.count(p[i]))//esta con map!!!
            return false;
        u = u->go[p[i]];//esta con map!!!
    }
    return true;
}
```

4.7. K-esima permutacion de una cadena

```
//Entrada: Una cadena cad(std::string), un long th
//Salida : La th-esima permutacion lexicografica de cad
string ipermutacion(string cad, long long int th){
  sort(cad.begin(), cad.end());
  string sol = "";
  int pos;
  for(int c = cad.size() - 1; c >= 0; c--){
    pos = th / fact[c];
    th %= fact[c];
    sol += cad[pos];
    cad.erase(cad.begin() + pos);
  }
  return sol;
}
```

5. Geometria

5.1. Interseccion de circunferencias - Sacar de Agustin

5.2. Graham Scan

5.3. Cortar Poligono

```
//cuts polygon Q along the line ab
//stores the left side (swap a, b for the right one) in P
void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
  P.clear();
  forn(i, sz(Q)){
    double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
    if(left1>=0) P.pb(Q[i]);
    if(left1*left2<0)
      P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
  }
}
```

5.4. Interseccion de rectangulos

```
#define MAXC 2501
struct Rect{
  int x1,y1, x2,y2;
  int color;
  int area;
  Rect(int _x1, int _y1, int _x2, int _y2){
    x1 = _x1;
    y1 = _y1;
    x2 = _x2;
    y2 = _y2;
    getArea();
  }
  int getArea(){
    if(x1>=x2 || y1>=y2)return area = 0;
    return area = (x2-x1)*(y2-y1);
  }
};
Rect interseccion(Rect t, Rect r){
  int x1,y1,x2,y2;
  x1 = max(t.x1,r.x1);
  y1 = max(t.y1,r.y1);
  x2 = min(t.x2,r.x2);
  y2 = min(t.y2,r.y2);
  Rect res(x1,y1,x2,y2);
  return res;
}
```

5.5. Distancia punto-recta

```
double distance_point_to_line(const point &a, const point &b, const point &pnt){
  double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr(a, b);
  point intersection;
  intersection.x = a.x + u*(b.x - a.x);
  intersection.y = a.y + u*(b.y - a.y);
  return dist(pnt, intersection);
}
```

5.6. Distancia punto-segmento

```
struct point{
  double x,y;
};
inline double dist(const point &a, const point &b){
  return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
inline double distsqr(const point &a, const point &b){
  return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}
double distance_point_to_segment(const point &a, const point &b, const point &pnt){
  double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr(a, b);
  point intersection;
  intersection.x = a.x + u*(b.x - a.x);
  intersection.y = a.y + u*(b.y - a.y);

  if (u < 0.0 || u > 1.0)
    return min(dist(a, pnt), dist(b, pnt));

  return dist(pnt, intersection);
}
```

5.7. Parametrizacion de rectas - Sacar de codeforces

6. Math

6.1. Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

$$\sum_{i=0}^{n} i\binom{n}{i} = n * 2^{n-1}$$

$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) → Sino ver caso impar y par

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$$

$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$$

$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

6.2. Ec. Caracteristica

$$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$$

$$p(x) = a_0x^k + a_1x^{k-1} + ... + a_k$$

Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$

$$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$

Las constantes c_{ij} se determinan por los casos base.

6.3. Identidades de agustin y mario

6.4. Combinatorio

```
forn(i, MAXN+1){//comb[i][k]=i tomados de a k
  comb[i][0]=comb[i][i]=1;
  forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
}
ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p] precalculado.
  ll aux = 1;
  while (n + k) aux = (aux * comb[n%p][k%p]) %p, n/=p, k/=p;
  return aux;
}
```

6.5. Exp. de Numeros Mod.

```
ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=(q*q)%m;
  return e%2? (b * q)%m : q;
}
```

6.6. Exp. de Matrices y Fibonacci en log(n) - Sacar de Agustin

6.7. Matrices y determinante $O(n^3)$

```
struct Mat {
    vector<vector<double> > vec;
    Mat(int n): vec(n, vector<double>(n) ) {}
    Mat(int n, int m): vec(n, vector<double>(m) ) {}
    vector<double> &operator[](int f){return vec[f];}
    const vector<double> &operator[](int f) const {return vec[f];}
    int size() const {return sz(vec);}
    Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
        Mat m(sz(b),sz(b[0]));
        forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
        return m;    }
    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
        int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
        Mat mat(n,t);
        forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
        return mat;    }
    double determinant(){//sacado de e maxx ru
        double det = 1;
        int n = sz(vec);
        Mat m(*this);
        forn(i, n){//para cada columna
            int k = i;
            forr(j, i+1, n)//busco la fila con mayor val abs
                if(abs(m[j][i])>abs(m[k][i])) k = j;
            if(abs(m[k][i])<1e-9) return 0;
            m[i].swap(m[k]);//la swapeo
            if(i!=k) det = -det;
            det *= m[i][i];
            forr(j, i+1, n) m[i][j] /= m[i][i];
            //hago 0 todas las otras filas
            forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
                forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
        }
        return det;
    }
};

int n;
int main() {
//DETERMINANTE:
//https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=625
  freopen("input.in", "r", stdin);
    ios::sync_with_stdio(0);
    while(cin >> n && n){
        Mat m(n);
        forn(i, n) forn(j, n) cin >> m[i][j];
        cout << (ll)round(m.determinant()) << endl;
    }
    cout  << "*" << endl;
  return 0;
}
```

6.8. Teorema Chino del Resto

$$y = \sum_{j=1}^{n}(x_j * (\prod_{i=1,i\neq j}^{n} m_i)_{m_j}^{-1} * \prod_{i=1,i\neq j}^{n} m_i)$$

6.9. Criba

```
#define MAXP 100000 //no necesariamente primo
int criba[MAXP+1];
void crearcriba(){
  int w[] = {4,2,4,2,4,6,2,6};
  for(int p=25;p<=MAXP;p+=10) criba[p]=5;
  for(int p=9;p<=MAXP;p+=6) criba[p]=3;
  for(int p=4;p<=MAXP;p+=2) criba[p]=2;
  for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
    for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}
vector<int> primos;
void buscarprimos(){
  crearcriba();
  forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}
//~ Useful for bit trick: #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) ), #define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 ), unsigned int criba[MAXP/32+1];


int main() {
  freopen("primos", "w", stdout);
  buscarprimos();
```

6.10. Funciones de primos

Sea $n = \prod p_i^{k_i}$, fact(n) genera un map donde a cada p_i le asocia su k_i

```
//factoriza bien numeros hasta MAXP^2
map<ll,ll> fact(ll n){ //O (cant primos)
  map<ll,ll> ret;
  forall(p, primos){
    while(!(n%*p)){
      ret[*p]++;//divisor found
      n/=*p;
    }
  }
  if(n>1) ret[n]++;
  return ret;
}
//factoriza bien numeros hasta MAXP
map<ll,ll> fact2(ll n){ //O (lg n)
  map<ll,ll> ret;
  while (criba[n]){
    ret[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) ret[n]++;
  return ret;
}
//Usar asi:  divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::iterator it, ll n=1){
    if(it==f.begin()) divs.clear();
    if(it==f.end()) { divs.pb(n);  return; }
    ll p=it->fst, k=it->snd; ++it;
    forn(_, k+1) divisores(f, divs, it, n), n*=p;
}
ll sumDiv (ll n){
  ll rta = 1;
  map<ll,ll> f=fact(n);
  forall(it, f) {
  ll pot = 1, aux = 0;
  forn(i, it->snd+1) aux += pot, pot *= it->fst;
  rta*=aux;
  }
  return rta;
}
ll eulerPhi (ll n){ // con criba: O(lg n)
  ll rta = n;
  map<ll,ll> f=fact(n);
  forall(it, f) rta -= rta / it->first;
  return rta;
}
ll eulerPhi2 (ll n){ // O (sqrt n)
  ll r = n;
  forr (i,2,n+1){
      if ((ll)i*i > n) break;
      if (n % i == 0){
        while (n%i == 0) n/=i;
        r -= r/i; }
    }
    if (n != 1) r-= r/n;
    return r;
}

int main() {
    buscarprimos();
    forr (x,1, 500000){
      cout << "x␣=␣" << x << endl;
      cout << "Numero␣de␣factores␣primos:␣" << numPrimeFactors(x) << endl;
      cout << "Numero␣de␣distintos␣factores␣primos:␣" <<
          numDiffPrimeFactors(x) << endl;
      cout << "Suma␣de␣factores␣primos:␣" << sumPrimeFactors(x) << endl;
      cout << "Numero␣de␣divisores:␣" << numDiv(x) << endl;
      cout << "Suma␣de␣divisores:␣" << sumDiv(x) << endl;
      cout << "Phi␣de␣Euler:␣" << eulerPhi(x) << endl;
    }
    return 0;
}
```

6.11. Phollard's Rho (rolando)

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

ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overfloor
  ll x = 0, y = a%c;
  while (b > 0){
    if (b % 2 == 1) x = (x+y) % c;
    y = (y*2) % c;
    b /= 2;
  }
  return x % c;
}

ll expmod (ll b, ll e, ll m){//O(log b)
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
  return e%2? mulmod(b,q,m) : q;
}

bool es_primo_prob (ll n, int a)
{
  if (n == a) return true;
  ll s = 0,d = n-1;
  while (d % 2 == 0) s++,d/=2;

  ll x = expmod(a,d,n);
  if ((x == 1) || (x+1 == n)) return true;

  forn (i, s-1){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (ll n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  forn (j,9)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = gcd( x - y , n );
        else d = gcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<ll,ll> prim;
```

```
void factRho (ll n){ //O (lg n)^3. un solo numero
  if (n == 1) return;
  if (rabin(n)){
    prim[n]++;
    return;
  }
  ll factor = rho(n);
  factRho(factor);
  factRho(n/factor);
}
```

6.12. GCD

```
tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}
```

6.13. Extended Euclid

```
void extendedEuclid (ll a, ll b){ //a * x + b * y = d
  if (!b) { x = 1; y = 0; d = a; return;}
  extendedEuclid (b, a%b);
  ll x1 = y;
  ll y1 = x - (a/b) * y;
  x = x1; y = y1;
}
```

6.14. LCM

```
tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}
```

6.15. Inversos

```
#define MAXMOD 15485867
ll inv[MAXMOD];//inv[i]*i=1 mod MOD
void calc(int p){//O(p)
  inv[1]=1;
  forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
}
int inverso(int x){//O(log x)
  return expmod(x, eulerphi(MOD)-2);//si mod no es primo(sacar a mano)
  return expmod(x, MOD-2);//si mod es primo
}
```

6.16. Simpson

```
double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
  double area=0, h=(b-a)/n, fa=f(a), fb;
  forn(i, n){
    fb=f(a+h*(i+1));
    area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
  }
  return area*h/6.;}
```

6.17. Fraction

```
tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
struct frac{
  tipo p,q;
  frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
  void norm(){
    tipo a = mcd(p,q);
    if(a) p/=a, q/=a;
    else q=1;
    if (q<0) q=-q, p=-p;}
  frac operator+(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
  frac operator-(const frac& o){
    tipo a = mcd(q,o.q);
    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
  frac operator*(frac o){
    tipo a = mcd(q,o.p), b = mcd(o.q,p);
    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
  frac operator/(frac o){
    tipo a = mcd(q,o.q), b = mcd(o.p,p);
    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
  bool operator==(frac o){return p==o.p&&q==o.q;}
};
```

6.18. Polinomio

```
        int m = sz(c), n = sz(o.c);
        vector<tipo> res(max(m,n));
        forn(i, m) res[i] += c[i];
        forn(i, n) res[i] += o.c[i];
        return poly(res);    }
    poly operator*(const tipo cons) const {
    vector<tipo> res(sz(c));
    forn(i, sz(c)) res[i]=c[i]*cons;
    return poly(res);    }
  poly operator*(const poly &o) const {
    int m = sz(c), n = sz(o.c);
    vector<tipo> res(m+n-1);
    forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
    return poly(res);    }
 tipo eval(tipo v) {
  tipo sum = 0;
  dforn(i, sz(c)) sum=sum*v + c[i];
  return sum; }
  //poly contains only a vector<int> c (the coeficients)
 //the following function generates the roots of the polynomial
//it can be easily modified to return float roots
 set<tipo> roots(){
  set<tipo> roots;
  tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
  vector<tipo> ps,qs;
  forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
  forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
  forall(pt,ps)
   forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
    tipo root = abs((*pt) / (*qt));
    if (eval(root)==0) roots.insert(root);
   }
  return roots; }
};
pair<poly,tipo> ruffini(const poly p, tipo r) {
 int n = sz(p.c) - 1 ;
 vector<tipo> b(n);
 b[n-1] = p.c[n];
 dforn(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
 tipo resto = p.c[0] + r*b[0];
 poly result(b);
 return make_pair(result,resto);
}
poly interpolate(const vector<tipo>& x,const vector<tipo>& y) {
  poly A; A.c.pb(1);
  forn(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux; }
 poly S; S.c.pb(0);
 forn(i,sz(x)) { poly Li;
  Li = ruffini(A,x[i]).fst;
  Li = Li * (1.0 / Li.eval(x[i])); // here put a multiple of the coefficients instead of 1.0 to avoid using double
  S = S + Li * y[i];  }
 return S;
}

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

6.19. Ec. Lineales

```
bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
 int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
 vector<int> p; forn(i,m) p.push_back(i);
 forn(i, rw) {
  int uc=i, uf=i;
  forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
  if (feq(a[uf][uc], 0)) { rw = i; break; }
  forn(j, n) swap(a[j][i], a[j][uc]);
  swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
  tipo inv = 1 / a[i][i]; //aca divide
  forr(j, i+1, n) {
   tipo v = a[j][i] * inv;
   forr(k, i, m) a[j][k]-=v * a[i][k];
   y[j] -= v*y[i];
  }
 } // rw = rango(a), aca la matriz esta triangulada
 forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de compatibilidad
 x = vector<tipo>(m, 0);
 dforn(i, rw){
  tipo s = y[i];
  forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
  x[p[i]] = s / a[i][i]; //aca divide
 }
 ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
 forn(k, m-rw) {
  ev[k][p[k+rw]] = 1;
  dforn(i, rw){
   tipo s = -a[i][k+rw];
   forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
   ev[k][p[i]] = s / a[i][i]; //aca divide
  }
 }
 return true;
}
```

6.20. FFT

```
//~ typedef complex<double> base; //menos codigo, pero mas lento
//elegir si usar complejos de c (lento) o estos
struct base{
  double r,i;
  base(double r=0, double i=0):r(r), i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
base operator*(const base &a, const base &b){
  return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
base operator+(const base &a, const base &b){
  return base(a.r+b.r, a.i+b.i);}
base operator-(const base &a, const base &b){
  return base(a.r-b.r, a.i-b.i);}
vector<int> rev; vector<base> wlen_pw;
inline static void fft(base a[], int n, bool invert) {
  forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
 for (int len=2; len<=n; len<<=1) {
  double ang = 2*M_PI/len * (invert?-1:+1);
  int len2 = len>>1;
  base wlen (cos(ang), sin(ang));
  wlen_pw[0] = base (1, 0);
    forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
  for (int i=0; i<n; i+=len) {
   base t, *pu = a+i, *pv = a+i+len2,  *pu_end = a+i+len2, *pw = &wlen_pw[0];
   for (; pu!=pu_end; ++pu, ++pv, ++pw)
    t = *pv * *pw, *pv = *pu - t,*pu = *pu + t;
  }
 }
 if (invert) forn(i, n) a[i]/= n;}
inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
  wlen_pw.resize(n), rev.resize(n);
  int lg=31-__builtin_clz(n);
  forn(i, n){
  rev[i] = 0;
    forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg-1-k);
  }}
inline static void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res) {
 vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
  int n=1; while(n < max(sz(a), sz(b))) n <<= 1; n <<= 1;
  calc_rev(n);
 fa.resize (n),  fb.resize (n);
 fft (&fa[0], n, false),  fft (&fb[0], n, false);
 forn(i, n) fa[i] = fa[i] * fb[i];
 fft (&fa[0], n, true);
 res.resize(n);
  forn(i, n) res[i] = int (fa[i].real() + 0.5); }
void toPoly(const string &s, vector<int> &P){//convierte un numero a polinomio
  P.clear();
  dforn(i, sz(s)) P.pb(s[i]-'0');}
```

6.21. Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 (∈ int)
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 (∈ tint)
10! = 3.628.800	21! = 51.090.942.171.709.400.000

max signed tint = 9.223.372.036.854.775.807

max unsigned tint = 18.446.744.073.709.551.615

Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859

863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

Divisores

Cantidad de divisores (σ_0) para *algunos* $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$
Suma de divisores (σ_1) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

7. Grafos

7.1. Dijkstra

```
#define INF 1e9
int N;
#define MAX_V 250001
vector<ii> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a].pb(make_pair(w, b))
ll dijkstra(int s, int t){//O(|E| log |V|)
  priority_queue<ii, vector<ii>, greater<ii> > Q;
  vector<ll> dist(N, INF); vector<int> dad(N, -1);
  Q.push(make_pair(0, s)); dist[s] = 0;
  while(sz(Q)){
    ii p = Q.top(); Q.pop();
    if(p.snd == t) break;
    forall(it, G[p.snd])
      if(dist[p.snd]+it->first < dist[it->snd]){
        dist[it->snd] = dist[p.snd] + it->fst;
        dad[it->snd] = p.snd;
        Q.push(make_pair(dist[it->snd], it->snd));  }
  }
  return dist[t];
  if(dist[t]<INF)//path generator
    for(int i=t; i!=-1; i=dad[i])
      printf("%d%c", i, (i==s?'\n':'␣'));}
```

7.2. Bellman-Ford

```
vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
int dist[MAX_N];
void bford(int src){//O(VE)
  dist[src]=0;
  forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
}

bool hasNegCycle(){
  forn(j, N) if(dist[j]!=INF) forall(it, G[j])
    if(dist[it->snd]>dist[j]+it->fst) return true;
  //inside if: all points reachable from it->snd will have -INF distance (do bfs)
  return false;
}
```

7.3. Floyd-Warshall

```
//G[i][j] contains weight of edge (i, j) or INF
//G[i][i]=0
int G[MAX_N][MAX_N];
void floyd(){//O(N^3)
forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
  G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
}
bool inNegCycle(int v){
  return G[v][v]<0;}
//checks if there's a neg. cycle in path from a to b
bool hasNegCycle(int a, int b){
  forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
    return true;
  return false;
}
```

7.4. Kruskal

```
struct Ar{int a,b,w;};
bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
vector<Ar> E;
ll kruskal(){
    ll cost=0;
    sort(E.begin(), E.end());//ordenar aristas de menor a mayor
    uf.init(n);
    forall(it, E){
        if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
            uf.unir(it->a, it->b);//conectar
            cost+=it->w;
        }
    }
    return cost;
}
```

7.5. Prim

```
bool taken[MAXN];
priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
void process(int v){
    taken[v]=true;
    forall(e, G[v])
        if(!taken[e->second]) pq.push(*e);
}

ll prim(){
    zero(taken);
    process(0);
    ll cost=0;
    while(sz(pq)){
        ii e=pq.top(); pq.pop();
        if(!taken[e.second]) cost+=e.first, process(e.second);
    }
    return cost;
}
```

7.6. 2-SAT + Tarjan SCC

```
//We have a vertex representing a var and other for his negation.
//Every edge stored in G represents an implication. To add an equation of the form a||b, use addor(a, b)
//MAX=max cant var, n=cant var
#define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
vector<int> G[MAX*2];
//idx[i]=index assigned in the dfs
//lw[i]=lowest index(closer from the root) reachable from i
int lw[MAX*2], idx[MAX*2], qidx;
stack<int> q;
int qcmp, cmp[MAX*2];
//verdad[cmp[i]]=valor de la variable i
bool verdad[MAX*2+1];

int neg(int x) { return x>=n? x-n : x+n;}
void tjn(int v){
  lw[v]=idx[v]=++qidx;
  q.push(v), cmp[v]=-2;
  forall(it, G[v]){
    if(!idx[*it] || cmp[*it]==-2){
      if(!idx[*it]) tjn(*it);
      lw[v]=min(lw[v], lw[*it]);
    }
  }
  if(lw[v]==idx[v]){
    int x;
    do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
    verdad[qcmp]=(cmp[neg(v)]<0);
    qcmp++;
  }
}
//remember to CLEAR G!!!
bool satisf(){//O(n)
  memset(idx, 0, sizeof(idx)), qidx=0;
  memset(cmp, -1, sizeof(cmp)), qcmp=0;
  forn(i, n){
    if(!idx[i]) tjn(i);
    if(!idx[neg(i)]) tjn(neg(i));
  }
  forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
  return true;
}
```

7.7. Articulation Points

```
int N;
vector<int> G[1000000];
//V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
int qV, V[1000000], L[1000000], P[1000000];
void dfs(int v, int f){
  L[v]=V[v]=++qV;
  forall(it, G[v])
    if(!V[*it]){
      dfs(*it, v);
      L[v] = min(L[v], L[*it]);
      P[v]+= L[*it]>=V[v];
    }
    else if(*it!=f)
      L[v]=min(L[v], V[*it]);
}
int cantart(){ //O(n)
  qV=0;
  zero(V), zero(P);
  dfs(1, 0); P[1]--;
  int q=0;
  forn(i, N) if(P[i]) q++;
return q;
}
```

7.8. Comp. Biconexas y Puentes

```
struct edge {
  int u,v, comp;
  bool bridge;
};
vector<edge> e;
void addEdge(int u, int v) {
  G[u].pb(sz(e)), G[v].pb(sz(e));
  e.pb((edge){u,v,-1,false});
}
//d[i]=id de la dfs
//b[i]=lowest id reachable from i
int d[MAXN], b[MAXN], t;
int nbc;//cant componentes
int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
void initDfs(int n) {
  zero(G), zero(comp);
  e.clear();
  forn(i,n) d[i]=-1;
  nbc = t = 0;
}
stack<int> st;
void dfs(int u, int pe) {//O(n + m)
  b[u] = d[u] = t++;
  comp[u] = (pe != -1);
  forall(ne, G[u]) if (*ne != pe){
    int v = e[*ne].u ^ e[*ne].v ^ u;
    if (d[v] == -1) {
      st.push(*ne);
      dfs(v,*ne);
      if (b[v] > d[u]){
        e[*ne].bridge = true; // bridge
      }
      if (b[v] >= d[u]){ // art
        int last;
        do {
          last = st.top(); st.pop();
          e[last].comp = nbc;
        } while (last != *ne);
        nbc++;
        comp[u]++;
      }
      b[u] = min(b[u], b[v]);
    }
    else if (d[v] < d[u]) { // back edge
      st.push(*ne);
      b[u] = min(b[u], d[v]);
    }
  }
}
```

7.9. LCA + Climb

```
const int MAXN=100001;
const int LOGN=20;
//f[v][k] holds the 2^k father of v
//L[v] holds the level of v
int N, f[MAXN][LOGN], L[MAXN];
//call before build:
void dfs(int v, int fa=-1, int lvl=0){//generate required data
  f[v][0]=fa, L[v]=lvl;
  forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
void build(){//f[i][0] must be filled previously, O(nlgn)
  forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
#define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
int climb(int a, int d){//O(lgn)
  if(!d) return a;
  dforn(i, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
    return a;}
int lca(int a, int b){//O(lgn)
  if(L[a]<L[b]) swap(a, b);
  a=climb(a, L[a]-L[b]);
  if(a==b) return a;
  dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
  return f[a][0]; }
int dist(int a, int b) {//returns distance between nodes
  return L[a]+L[b]-2*L[lca(a, b)];}
```

7.10. Heavy Light Decomposition

```
int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
int dad[MAXN];//dad[v]=padre del nodo v
void dfs1(int v, int p=-1){//pre-dfs
  dad[v]=p;
  treesz[v]=1;
  forall(it, G[v]) if(*it!=p){
    dfs1(*it, v);
    treesz[v]+=treesz[*it];
  }
}
//PONER Q EN 0  !!!!!
int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
//Las cadenas aparecen continuas en el recorrido!
int cantcad;
int homecad[MAXN];//dada una cadena devuelve su nodo inicial
int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
void heavylight(int v, int cur=-1){
  if(cur==-1) homecad[cur=cantcad++]=v;
  pos[v]=q++;
  cad[v]=cur;
  int mx=-1;
  forn(i, sz(G[v])) if(G[v][i]!=dad[v])
    if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
  if(mx!=-1) heavylight(G[v][mx], cur);
  forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
    heavylight(G[v][i], -1);
}
//ejemplo de obtener el maximo numero en el camino entre dos nodos
//RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
//esta funcion va trepando por las cadenas
int query(int an, int v){//O(logn)
  //si estan en la misma cadena:
  if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
  return max(query(an, dad[homecad[cad[v]]]),
         rmq.get(pos[homecad[cad[v]]], pos[v]+1));
}
```

7.11. Centroid Decomposition

```
int n;
vector<int> G[MAXN];
bool taken[MAXN];//poner todos en FALSE al principio!!
int padre[MAXN];//padre de cada nodo en el centroid tree

int szt[MAXN];
void calcsz(int v, int p) {
  szt[v] = 1;
  forall(it,G[v]) if (*it!=p && !taken[*it])
    calcsz(*it,v), szt[v]+=szt[*it];
}
void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
  if(tam==-1) calcsz(v, -1), tam=szt[v];
  forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
    {szt[v]=0; centroid(*it, f, lvl, tam); return;}
  taken[v]=true;
  padre[v]=f;
  forall(it, G[v]) if(!taken[*it])
    centroid(*it, v, lvl+1, -1);
}
```

7.12. Euler Cycle

```
int n,m,ars[MAXE], eq;
vector<int> G[MAXN];//fill G,n,m,ars,eq
list<int> path;
int used[MAXN];
bool usede[MAXE];
queue<list<int>::iterator> q;
int get(int v){
  while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
  return used[v];
}
void explore(int v, int r, list<int>::iterator it){
  int ar=G[v][get(v)]; int u=v^ars[ar];
  usede[ar]=true;
  list<int>::iterator it2=path.insert(it, u);
  if(u!=r) explore(u, r, it2);
  if(get(v)<sz(G[v])) q.push(it);
}
void euler(){
  zero(used), zero(usede);
  path.clear();
  q=queue<list<int>::iterator>();
  path.push_back(0); q.push(path.begin());
  while(sz(q)){
    list<int>::iterator it=q.front(); q.pop();
    if(used[*it]<sz(G[*it])) explore(*it, *it, it);
  }
  reverse(path.begin(), path.end());
}
void addEdge(int u, int v){
  G[u].pb(eq), G[v].pb(eq);
  ars[eq++]=u^v;
}
```

7.13. Diametro árbol

```
vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
int bfs(int r, int *d) {
  queue<int> q;
  d[r]=0; q.push(r);
  int v;
  while(sz(q)) { v=q.front(); q.pop();
    forall(it,G[v]) if (d[*it]==-1)
      d[*it]=d[v]+1, p[*it]=v, q.push(*it);
  }
  return v;//ultimo nodo visitado
}
vector<int> diams; vector<ii> centros;
void diametros(){
  memset(d,-1,sizeof(d));
  memset(d2,-1,sizeof(d2));
  diams.clear(), centros.clear();
  forn(i, n) if(d[i]==-1){
    int v,c;
    c=v=bfs(bfs(i, d2), d);
    forn(_,d[v]/2) c=p[c];
    diams.pb(d[v]);
    if(d[v]&1) centros.pb(ii(c, p[c]));
    else centros.pb(ii(c, c));
  }
}

int main() {
  freopen("in", "r", stdin);
  while(cin >> n >> m){
    forn(i,m) { int a,b; cin >> a >> b; a--, b--;
      G[a].pb(b);
      G[b].pb(a);
```

7.14. Chu-liu

```
void visit(graph &h, int v, int s, int r,
 vector<int> &no, vector< vector<int> > &comp,
 vector<int> &prev, vector< vector<int> > &next, vector<weight> &mcost,
 vector<int> &mark, weight &cost, bool &found) {
 if (mark[v]) {
  vector<int> temp = no;
  found = true;
  do {
   cost += mcost[v];
   v = prev[v];
   if (v != s) {
    while (comp[v].size() > 0) {
     no[comp[v].back()] = s;
     comp[s].push_back(comp[v].back());
     comp[v].pop_back();
    }
   }
  } while (v != s);
  forall(j,comp[s]) if (*j != r) forall(e,h[*j])
   if (no[e->src] != s) e->w -= mcost[ temp[*j] ];
 }
 mark[v] = true;
 forall(i,next[v]) if (no[*i] != no[v] && prev[no[*i]] == v)
  if (!mark[no[*i]] || *i == s)
   visit(h, *i, s, r, no, comp, prev, next, mcost, mark, cost, found);
}
weight minimumSpanningArborescence(const graph &g, int r) {
  const int n=sz(g);
 graph h(n);
 forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
 vector<int> no(n);
 vector<vector<int> > comp(n);
 forn(u, n) comp[u].pb(no[u] = u);
 for (weight cost = 0; ;) {
  vector<int> prev(n, -1);
  vector<weight> mcost(n, INF);
  forn(j,n) if (j != r) forall(e,h[j])
   if (no[e->src] != no[j])
    if (e->w < mcost[ no[j] ])
     mcost[ no[j] ] = e->w, prev[ no[j] ] = no[e->src];
  vector< vector<int> > next(n);
  forn(u,n) if (prev[u] >= 0)
   next[ prev[u] ].push_back(u);
  bool stop = true;
  vector<int> mark(n);
  forn(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
   bool found = false;
   visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
   if (found) stop = false;
  }
  if (stop) {
   forn(u,n) if (prev[u] >= 0) cost += mcost[u];
   return cost;
  }
 }
}
```

7.15. Hungarian

```
//Dado un grafo bipartito completo con costos no negativos, encuentra el matching perfecto de minimo costo.
tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de adyacencia
int n, max_match, xy[N], yx[N], slackx[N],prev2[N];//n=cantidad de nodos
bool S[N], T[N]; //sets S and T in algorithm
void add_to_tree(int x, int prevx) {
 S[x] = true, prev2[x] = prevx;
 forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
  slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
}
void update_labels(){
 tipo delta = INF;
 forn (y, n) if (!T[y]) delta = min(delta, slack[y]);
 forn (x, n) if (S[x]) lx[x] -= delta;
 forn (y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
}
void init_labels(){
 zero(lx), zero(ly);
 forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
}
void augment() {
 if (max_match == n) return;
 int x, y, root, q[N], wr = 0, rd = 0;
 memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
 memset(prev2, -1, sizeof(prev2));
 forn (x, n) if (xy[x] == -1){
    q[wr++] = root = x, prev2[x] = -2;
    S[x] = true; break; }
  forn (y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] = root;
  while (true){
    while (rd < wr){
      x = q[rd++];
      for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
        if (yx[y] == -1) break; T[y] = true;
        q[wr++] = yx[y], add_to_tree(yx[y], x); }
      if (y < n) break; }
    if (y < n) break;
    update_labels(), wr = rd = 0;
    for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
      if (yx[y] == -1){x = slackx[y]; break;}
      else{
        T[y] = true;
        if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
      }}
    if (y < n) break; }
  if (y < n){
    max_match++;
    for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
      ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
    augment(); }
}
tipo hungarian(){
  tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
  forn (x,n) ret += cost[x][xy[x]]; return ret;
}
```

7.16. Dynamic Conectivity

```
struct UnionFind {
    int n, comp;
    vector<int> pre,si,c;
    UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
        forn(i,n) pre[i] = i; }
    int find(int u){return u==pre[u]?u:find(pre[u]);}
    bool merge(int u, int v) {
        if((u=find(u))==(v=find(v))) return false;
        if(si[u]<si[v]) swap(u, v);
        si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
        return true;
    }
    int snap(){return sz(c);}
    void rollback(int snap){
        while(sz(c)>snap){
            int v = c.back(); c.pop_back();
            si[pre[v]] -= si[v], pre[v] = v, comp++;
        }
    }
};
enum {ADD,DEL,QUERY};
struct Query {int type,u,v;};
struct DynCon {
    vector<Query> q;
    UnionFind dsu;
    vector<int> match,res;
    map<ii,int> last;//se puede no usar cuando hay identificador para cada arista (mejora poco)
    DynCon(int n=0):dsu(n){}
    void add(int u, int v) {
        if(u>v) swap(u,v);
        q.pb((Query){ADD, u, v}), match.pb(-1);
        last[ii(u,v)] = sz(q)-1;
    }
    void remove(int u, int v) {
        if(u>v) swap(u,v);
        q.pb((Query){DEL, u, v});
        int prev = last[ii(u,v)];
        match[prev] = sz(q)-1;
        match.pb(prev);
    }
    void query() {//podria pasarle un puntero donde guardar la respuesta
        q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
    void process() {
        forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
        go(0,sz(q));
    }
    void go(int l, int r) {
        if(l+1==r){
            if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
```

```
                res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
            return;
        }
        int s=dsu.snap(), m = (l+r) / 2;
        forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
        go(l,m);
        dsu.rollback(s);
        s = dsu.snap();
        forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
        go(m,r);
        dsu.rollback(s);
    }
}dc;
```

8. Network Flow

8.1. Dinic

```

const int MAX = 300;
// Corte minimo: vertices con dist[v]>=0 (del lado de src) VS.  dist[v]==-1 (del lado del dst)
// Para el caso de la red de Bipartite Matching (Sean V1 y V2 los conjuntos mas proximos a src y dst respectivamente):
// Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de V2 con it->f>0, es arista del Matching
// Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con dist[v]>0
// Max Independent Set: tomar los vertices NO tomados por el Min Vertex Cover
// Max Clique: construir la red de G complemento (debe ser bipartito!) y encontrar un Max Independet Set
// Min Edge Cover: tomar las aristas del matching + para todo vertices no cubierto hasta el momento, tomar cualquier arista de el
int nodes, src, dst;
int dist[MAX], q[MAX], work[MAX];
struct Edge {
    int to, rev;
    ll f, cap;
    Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(cap) {}
};
vector<Edge> G[MAX];
void addEdge(int s, int t, ll cap){
    G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(Edge(s, sz(G[s])-1, 0, 0));}
bool dinic_bfs(){
    fill(dist, dist+nodes, -1), dist[src]=0;
    int qt=0; q[qt++]=src;
    for(int qh=0; qh<qt; qh++){
        int u =q[qh];
        forall(e, G[u]){
            int v=e->to;
            if(dist[v]<0 && e->f < e->cap)
                dist[v]=dist[u]+1, q[qt++]=v;
        }
    }
    return dist[dst]>=0;
}
ll dinic_dfs(int u, ll f){
    if(u==dst) return f;
    for(int &i=work[u]; i<sz(G[u]); i++){
        Edge &e = G[u][i];
        if(e.cap<=e.f) continue;
        int v=e.to;
        if(dist[v]==dist[u]+1){
                ll df=dinic_dfs(v, min(f, e.cap-e.f));
                if(df>0){
                        e.f+=df, G[v][e.rev].f-= df;
                        return df;  }
        }
    }
    return 0;
}
ll maxFlow(int _src, int _dst){
    src=_src, dst=_dst;
    ll result=0;
    while(dinic_bfs()){
        fill(work, work+nodes, 0);
        while(ll delta=dinic_dfs(src,INF))
            result+=delta;
    }
    // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1
    forman el min-cut
  return result; }
```

8.2. Konig

```
// asume que el dinic YA ESTA tirado
// asume que nodes-1 y nodes-2 son la fuente y destino
int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v no esta matcheado
int s[maxnodes]; // numero de la bfs del koning
queue<int> kq;
// s[e] %2==1  o si e esta en V1 y s[e]==-1-> lo agarras
void koning() {//O(n)
 forn(v,nodes-2) s[v] = match[v] = -1;
 forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
  { match[v]=it->to; match[it->to]=v;}
 forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
 while(!kq.empty()) {
  int e = kq.front(); kq.pop();
  if (s[e] %2==1) {
   s[match[e]] = s[e]+1;
   kq.push(match[e]);
  } else {

   forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
    s[it->to] = s[e]+1;
    kq.push(it->to);
   }
  }
 }
}
```

8.3. Edmonds Karp's

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE){
 if(v==SRC) f=minE;
 else if(p[v]!=-1){
  augment(p[v], min(minE, G[p[v]][v]));
  G[p[v]][v]-=f, G[v][p[v]]+=f;
 }
}
ll maxflow(){//O(VE^2)
 ll Mf=0;
 do{
  f=0;
  char used[MAX_V]; queue<int> q; q.push(SRC);
  zero(used), memset(p, -1, sizeof(p));
  while(sz(q)){
   int u=q.front(); q.pop();
   if(u==SNK) break;
   forall(it, G[u])
    if(it->snd>0 && !used[it->fst])
     used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
  }
  augment(SNK, INF);
  Mf+=f;
 }while(f);
 return Mf;
}
```

8.4. Push-Relabel O(N3)

```
#define MAX_V 1000
int N;//valid nodes are [0...N-1]
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];
//To add an edge use
#define add(a, b, w) G[a][b]=w
ll excess[MAX_V];
int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
queue<int> Q;
void enqueue(int v) {
 if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
void push(int a, int b) {
 int amt = min(excess[a], ll(G[a][b]));
```

```
  if(height[a] <= height[b] || amt == 0) return;
  G[a][b]-=amt, G[b][a]+=amt;
  excess[b] += amt, excess[a] -= amt;
  enqueue(b);
}
void gap(int k) {
  forn(v, N){
    if (height[v] < k) continue;
    count[height[v]]--;
    height[v] = max(height[v], N+1);
    count[height[v]]++;
    enqueue(v);
  }
}
void relabel(int v) {
  count[height[v]]--;
  height[v] = 2*N;
  forall(it, G[v])
    if(it->snd)
      height[v] = min(height[v], height[it->fst] + 1);
  count[height[v]]++;
  enqueue(v);
}
ll maxflow() {//O(V^3)
  zero(height), zero(active), zero(count), zero(excess);
  count[0] = N-1;
  count[N] = 1;
  height[SRC] = N;
  active[SRC] = active[SNK] = true;
  forall(it, G[SRC]){
    excess[SRC] += it->snd;
    push(SRC, it->fst);
  }
  while(sz(Q)) {
    int v = Q.front(); Q.pop();
    active[v]=false;
  forall(it, G[v]) push(v, it->fst);
  if(excess[v] > 0)
    count[height[v]] == 1? gap(height[v]):relabel(v);
  }
  ll mf=0;
  forall(it, G[SRC]) mf+=G[it->fst][SRC];
  return mf;
}
```

8.5. Min-cost Max-flow

```
const int MAXN=10000;
typedef ll tf;
typedef ll tc;
const tf INFFLUJO = 1e14;
const tc INFCOSTO = 1e14;
struct edge {
  int u, v;
  tf cap, flow;
  tc cost;
  tf rem() { return cap - flow; }
};
int nodes; //numero de nodos
vector<int> G[MAXN]; // limpiar!
vector<edge> e;  // limpiar!
void addEdge(int u, int v, tf cap, tc cost) {
  G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
  G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
}
tc dist[MAXN], mnCost;
int pre[MAXN];
tf cap[MAXN], mxFlow;
bool in_queue[MAXN];
void flow(int s, int t) {
  zero(in_queue);
  mxFlow=mnCost=0;
  while(1){
    fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
    memset(pre, -1, sizeof(pre)); pre[s]=0;
    zero(cap); cap[s] = INFFLUJO;
    queue<int> q; q.push(s); in_queue[s]=1;
    while(sz(q)){
      int u=q.front(); q.pop(); in_queue[u]=0;
      for(auto it:G[u]) {
        edge &E = e[it];
        if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9){ // ojo EPS
          dist[E.v]=dist[u]+E.cost;
          pre[E.v] = it;
          cap[E.v] = min(cap[u], E.rem());
          if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
        }
      }
    }
    if (pre[t] == -1) break;
    mxFlow +=cap[t];
    mnCost +=cap[t]*dist[t];
    for (int v = t; v != s; v = e[pre[v]].u) {
      e[pre[v]].flow += cap[t];
      e[pre[v]^1].flow -= cap[t];
    }
  }
}
```

9. Template

```
//touch {a..m}.in; tee {a..m}.cpp < template.cpp
#include <bits/stdc++.h>
using namespace std;
#define forr(i,a,b) for(int i=(a); i<(b); i++)
#define forn(i,n) forr(i,0,n)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
#define forall(it,v) for(auto it=v.begin();it!=v.end();++it)
#define pb push_back
#define fst first
#define snd second
typedef long long ll;
typedef pair<int,int> ii;
#define dforn(i,n) for(int i=n-1; i>=0; i--)
#define dprint(v) cout << #v"=" << v << endl //;)

const int MAXN=100100;
int n;

int main() {
    freopen("input.in", "r", stdin);
    ios::sync_with_stdio(0);
    while(cin >> n){

    }
    return 0;
}
```

10. Ayudamemoria

Cant. decimales

```
#include <iomanip>
cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```
#include <iomanip>
cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de linea

```
#include <sstream>
//hacer cin.ignore() antes de getline()
while(getline(cin, line)){
     istringstream is(line);
     while(is >> X)
       cout << X << "␣";
     cout << endl;
}
```

Aleatorios

```
#define RAND(a, b) (rand()%(b-a+1)+a)
srand(time(NULL));
```

Doubles Comp.

```
const double EPS = 1e-9;
x == y  <=> fabs(x-y) < EPS
x >  y  <=> x > y + EPS
x >= y  <=> x > y - EPS
```

Limites

```
#include <limits>
numeric_limits<T>
  ::max()
  ::min()
  ::epsilon()
```

Muahaha

```
#include <signal.h>
void divzero(int p){
  while(true);}
void segm(int p){
  exit(0);}
//in main
signal(SIGFPE, divzero);
signal(SIGSEGV, segm);
```

Mejorar velocidad

```
ios::sync_with_stdio(false);
```

Mejorar velocidad 2

```
//Solo para enteros positivos
inline void Scanf(int& a){
  char c = 0;
  while(c<33) c = getc(stdin);
  a = 0;
  while(c>33) a = a*10 + c - ’0’, c = getc(stdin);
}
```

Expandir pila

```
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
```

C++11

```
g++ --std=c++1
```

Leer del teclado

```
freopen("/dev/tty", "a", stdin);
```

Iterar subconjunto

```
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

File setup

```
//tambien se pueden usar comas: {a, x, m, l}
touch {a..l}.in; tee {a..l}.cpp < template.cpp
```