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 ∈[f,l) tq. i=elem, pred(i), i∈[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) ∈ [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) $\forall i$ ∈[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().

```cpp
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 Iterativo

2.2.2.   Segment Tree Recursivo

2.2.3.   Segment Tree con Punteros

2.2.4.   Segment Tree 2D

2.2.5.   Segment Tree Lazy - Suma

2.2.6.   Segment Tree Lazy - Pintar

2.2.7.   Segment Tree Persistente

2.3.   Fenwick Tree

2.3.1.   Fenwick Tree 2D

2.4.   Union Find con rank

```cpp
/*=============== <Union find rangos> ===================
Complexity: O(N)
index 0 to n - 1 warning
Complexity O(N)
*/
#define MAX 10005
int padre[ MAX ];   //Este arreglo contiene el padre del i-esimo nodo
int rango[ MAX ];   //profundidad de cada vertice
//Metodo de inicializacion
void MakeSet( int n ){
    for( int i = 0 ; i < n ; ++i ){
        padre[ i ] = i;        //Inicialmente el padre de cada vertice es
                el mismo vertice
        rango[ i ] = 0;        //Altura o rango de cada vertice es 0
    }
}
//Metodo para encontrar la raiz del vertice actual X
int Find( int x ){
    if( x == padre[ x ] ){          //Si estoy en la raiz
        return x;                   //Retorno la raiz
    }
    //else return Find( padre[ x ] ); //De otro modo busco el padre del
            vertice actual, hasta llegar a la raiz.
    else return padre[ x ] = Find( padre[ x ] ); //Compresion de caminos
}
//Metodo para unir 2 componentes conexas usando sus alturas (rangos)
void UnionbyRank( int x , int y ){
    int xRoot = Find( x );     //Obtengo la raiz de la componente del
            vertice X
    int yRoot = Find( y );     //Obtengo la raiz de la componente del
            vertice Y
    if( rango[ xRoot ] > rango[ yRoot ] ){ //en este caso la altura de
            la componente del vertice X es
                                    //mayor que la altura de la
                                        componente del vertice Y.
        padre[ yRoot ] = xRoot;         //el padre de ambas
            componentes sera el de mayor altura
    }
    else{                      //en este caso la altura de la componente
            del vertice Y es mayor o igual que la de X
        padre[ xRoot ] = yRoot;            //el padre de ambas
            componentes sera el de mayor altura
        if( rango[ xRoot ] == rango[ yRoot ] ){ //si poseen la misma
            altura
            rango[ yRoot ]++;             //incremento el rango de la
                nueva raiz
        }
```

```
37        }
38  }
```

2.5. BigInteger C++

```
1   // g++ -std=c++11 "bigint.cpp" -o run
2   /***
3   =============== <Big Int c++ version> ===================
4   Contain a useful big int, overload all operators, including cin, cout,
5   comparator, build via string (prefer this metod) or long long, for now
        this not have a
6   to_string method
7
8   Problem for practice: UVA 494
9   */
10
11  // base and base_digits must be consistent
12  const int base = 1000000000;
13  const int base_digits = 9;
14
15  struct bigint {
16      vector<int> a;
17      int sign;
18
19      bigint() :
20          sign(1) {
21      }
22
23      bigint(long long v) {
24          *this = v;
25      }
26
27      bigint(const string &s) {
28          read(s);
29      }
30
31      void operator=(const bigint &v) {
32          sign = v.sign;
33          a = v.a;
34      }
35
36      void operator=(long long v) {
37          sign = 1;
38          if (v < 0)
39              sign = -1, v = -v;
40          for (; v > 0; v = v / base)
41              a.push_back(v % base);
42      }
43
44      bigint operator+(const bigint &v) const {
45          if (sign == v.sign) {
46              bigint res = v;
47
48              for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size
                    ()) || carry; ++i) {
49                  if (i == (int) res.a.size())
50                      res.a.push_back(0);
51                  res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
52                  carry = res.a[i] >= base;
53                  if (carry)
54                      res.a[i] -= base;
55              }
56              return res;
57          }
58          return *this - (-v);
59      }
60
61      bigint operator-(const bigint &v) const {
62          if (sign == v.sign) {
63              if (abs() >= v.abs()) {
64                  bigint res = *this;
65                  for (int i = 0, carry = 0; i < (int) v.a.size() || carry
                        ; ++i) {
66                      res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] :
                            0);
67                      carry = res.a[i] < 0;
68                      if (carry)
69                          res.a[i] += base;
70                  }
71                  res.trim();
72                  return res;
73              }
74              return -(v - *this);
75          }
76          return *this + (-v);
77      }
```

```cpp
 78
 79      void operator*=(int v) {
 80          if (v < 0)
 81              sign = -sign, v = -v;
 82          for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
 83              if (i == (int) a.size())
 84                  a.push_back(0);
 85              long long cur = a[i] * (long long) v + carry;
 86              carry = (int) (cur / base);
 87              a[i] = (int) (cur % base);
 88              //asm("divl %%cx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c
 89                  "(base));
 89          }
 90          trim();
 91      }
 92
 93      bigint operator*(int v) const {
 94          bigint res = *this;
 95          res *= v;
 96          return res;
 97      }
 98
 99      friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &
             b1) {
100          int norm = base / (b1.a.back() + 1);
101          bigint a = a1.abs() * norm;
102          bigint b = b1.abs() * norm;
103          bigint q, r;
104          q.a.resize(a.a.size());
105
106          for (int i = a.a.size() - 1; i >= 0; i--) {
107              r *= base;
108              r += a.a[i];
109              int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
110              int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() -
                     1];
111              int d = ((long long) base * s1 + s2) / b.a.back();
112              r -= b * d;
113              while (r < 0)
114                  r += b, --d;
115              q.a[i] = d;
116          }
117
118          q.sign = a1.sign * b1.sign;
119          r.sign = a1.sign;
120          q.trim();
121          r.trim();
122          return make_pair(q, r / norm);
123      }
124
125      bigint operator/(const bigint &v) const {
126          return divmod(*this, v).first;
127      }
128
129      bigint operator%(const bigint &v) const {
130          return divmod(*this, v).second;
131      }
132
133      void operator/=(int v) {
134          if (v < 0)
135              sign = -sign, v = -v;
136          for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
137              long long cur = a[i] + rem * (long long) base;
138              a[i] = (int) (cur / v);
139              rem = (int) (cur % v);
140          }
141          trim();
142      }
143
144      bigint operator/(int v) const {
145          bigint res = *this;
146          res /= v;
147          return res;
148      }
149
150      int operator%(int v) const {
151          if (v < 0)
152              v = -v;
153          int m = 0;
154          for (int i = a.size() - 1; i >= 0; --i)
155              m = (a[i] + m * (long long) base) % v;
156          return m * sign;
157      }
158
159      void operator+=(const bigint &v) {
160          *this = *this + v;
```

```cpp
161        }
162        void operator-=(const bigint &v) {
163            *this = *this - v;
164        }
165        void operator*=(const bigint &v) {
166            *this = *this * v;
167        }
168        void operator/=(const bigint &v) {
169            *this = *this / v;
170        }
171
172        bool operator<(const bigint &v) const {
173            if (sign != v.sign)
174                return sign < v.sign;
175            if (a.size() != v.a.size())
176                return a.size() * sign < v.a.size() * v.sign;
177            for (int i = a.size() - 1; i >= 0; i--)
178                if (a[i] != v.a[i])
179                    return a[i] * sign < v.a[i] * sign;
180            return false;
181        }
182
183        bool operator>(const bigint &v) const {
184            return v < *this;
185        }
186        bool operator<=(const bigint &v) const {
187            return !(v < *this);
188        }
189        bool operator>=(const bigint &v) const {
190            return !(*this < v);
191        }
192        bool operator==(const bigint &v) const {
193            return !(*this < v) && !(v < *this);
194        }
195        bool operator!=(const bigint &v) const {
196            return *this < v || v < *this;
197        }
198
199        void trim() {
200            while (!a.empty() && !a.back())
201                a.pop_back();
202            if (a.empty())
203                sign = 1;
```

```cpp
204        }
205
206        bool isZero() const {
207            return a.empty() || (a.size() == 1 && !a[0]);
208        }
209
210        bigint operator-() const {
211            bigint res = *this;
212            res.sign = -sign;
213            return res;
214        }
215
216        bigint abs() const {
217            bigint res = *this;
218            res.sign *= res.sign;
219            return res;
220        }
221
222        long long longValue() const {
223            long long res = 0;
224            for (int i = a.size() - 1; i >= 0; i--)
225                res = res * base + a[i];
226            return res * sign;
227        }
228
229        friend bigint gcd(const bigint &a, const bigint &b) {
230            return b.isZero() ? a : gcd(b, a % b);
231        }
232        friend bigint lcm(const bigint &a, const bigint &b) {
233            return a / gcd(a, b) * b;
234        }
235
236        void read(const string &s) {
237            sign = 1;
238            a.clear();
239            int pos = 0;
240            while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+'))
241            {
242                if (s[pos] == '-')
243                    sign = -sign;
244                ++pos;
245            }
246            for (int i = s.size() - 1; i >= pos; i -= base_digits) {
```

```
246            int x = 0;
247            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
248                x = x * 10 + s[j] - '0';
249            a.push_back(x);
250        }
251        trim();
252    }
253
254    friend istream& operator>>(istream &stream, bigint &v) {
255        string s;
256        stream >> s;
257        v.read(s);
258        return stream;
259    }
260
261    friend ostream& operator<<(ostream &stream, const bigint &v) {
262        if (v.sign == -1)
263            stream << '-';
264        stream << (v.a.empty() ? 0 : v.a.back());
265        for (int i = (int) v.a.size() - 2; i >= 0; --i)
266            stream << setw(base_digits) << setfill('0') << v.a[i];
267        return stream;
268    }
269
270    static vector<int> convert_base(const vector<int> &a, int old_digits
            , int new_digits) {
271        vector<long long> p(max(old_digits, new_digits) + 1);
272        p[0] = 1;
273        for (int i = 1; i < (int) p.size(); i++)
274            p[i] = p[i - 1] * 10;
275        vector<int> res;
276        long long cur = 0;
277        int cur_digits = 0;
278        for (int i = 0; i < (int) a.size(); i++) {
279            cur += a[i] * p[cur_digits];
280            cur_digits += old_digits;
281            while (cur_digits >= new_digits) {
282                res.push_back(int(cur % p[new_digits]));
283                cur /= p[new_digits];
284                cur_digits -= new_digits;
285            }
286        }
287        res.push_back((int) cur);
```

```
288        while (!res.empty() && !res.back())
289            res.pop_back();
290        return res;
291    }
292
293    typedef vector<long long> vll;
294
295    static vll karatsubaMultiply(const vll &a, const vll &b) {
296        int n = a.size();
297        vll res(n + n);
298        if (n <= 32) {
299            for (int i = 0; i < n; i++)
300                for (int j = 0; j < n; j++)
301                    res[i + j] += a[i] * b[j];
302            return res;
303        }
304
305        int k = n >> 1;
306        vll a1(a.begin(), a.begin() + k);
307        vll a2(a.begin() + k, a.end());
308        vll b1(b.begin(), b.begin() + k);
309        vll b2(b.begin() + k, b.end());
310
311        vll a1b1 = karatsubaMultiply(a1, b1);
312        vll a2b2 = karatsubaMultiply(a2, b2);
313
314        for (int i = 0; i < k; i++)
315            a2[i] += a1[i];
316        for (int i = 0; i < k; i++)
317            b2[i] += b1[i];
318
319        vll r = karatsubaMultiply(a2, b2);
320        for (int i = 0; i < (int) a1b1.size(); i++)
321            r[i] -= a1b1[i];
322        for (int i = 0; i < (int) a2b2.size(); i++)
323            r[i] -= a2b2[i];
324
325        for (int i = 0; i < (int) r.size(); i++)
326            res[i + k] += r[i];
327        for (int i = 0; i < (int) a1b1.size(); i++)
328            res[i] += a1b1[i];
329        for (int i = 0; i < (int) a2b2.size(); i++)
330            res[i + n] += a2b2[i];
```

```
331        return res;
332      }
333
334      bigint operator*(const bigint &v) const {
335        vector<int> a6 = convert_base(this->a, base_digits, 6);
336        vector<int> b6 = convert_base(v.a, base_digits, 6);
337        vll a(a6.begin(), a6.end());
338        vll b(b6.begin(), b6.end());
339        while (a.size() < b.size())
340            a.push_back(0);
341        while (b.size() < a.size())
342            b.push_back(0);
343        while (a.size() & (a.size() - 1))
344            a.push_back(0), b.push_back(0);
345        vll c = karatsubaMultiply(a, b);
346        bigint res;
347        res.sign = sign * v.sign;
348        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
349            long long cur = c[i] + carry;
350            res.a.push_back((int) (cur % 1000000));
351            carry = (int) (cur / 1000000);
352        }
353        res.a = convert_base(res.a, 6, base_digits);
354        res.trim();
355        return res;
356      }
357 };
358
359 int main() {
360      bigint a=0;
361      //
           ("9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999")
           ;
362      bigint b;
363      //
           ("1999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999998")
           ;
364      bigint n;
365      while(cin>>n)
366      {
367          if(n==0){break;}
368          a+=n;
369      }
```

```
370        cout<<a<<endl;
371      }
```

2.6.   UnorderedSet

```
1  //Compilar: g++ --std=c++11
2  struct Hash{
3    size_t operator()(const ii &a)const{
4      size_t s=hash<int>()(a.fst);
5      return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
6    }
7    size_t operator()(const vector<int> &v)const{
8      size_t s=0;
9      for(auto &e : v)
10        s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
11     return s;
12   }
13 };
14 unordered_set<ii, Hash> s;
15 unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.7.   Ordered Set

```
1  /*
2   A brief explanation about use of a powerful library: orderd_set
3   Reference link: http://codeforces.com/blog/entry/11080
4   and a hash for the type pair
5  */
6
7  #include <ext/pb_ds/assoc_container.hpp>
8  #include <ext/pb_ds/tree_policy.hpp>
9  using namespace __gnu_pbds;
10 typedef tree<int,null_type,less<int>,rb_tree_tag,
       tree_order_statistics_node_update> ordered_set;
11 /*
12 If we want to get map but not the set, as the second argument type must
       be used mapped type. Apparently,
13 the tree supports the same operations as the set (at least I haven't
       any problems with them before),
14 but also there are two new features - it is find_by_order() and
       order_of_key().
15 The first returns an iterator to the k-th largest element (counting
       from zero), the second - the number of items
16 in a set that are strictly smaller than our item. Example of use:
```

```
17  *
18  * */
```

2.8.   Treap Modo Set

2.9.   Treap Implicito(Rope)

2.10.   Treap - Toby and Bones

2.11.   Convex Hull Trick Estatico

```
1   // g++  "convexhulltrick.cpp" -o run
2   /***
3   =============== <Convex hull trick normal version> ==================
4   Contain a sample about convex hull trick optimization this recivie N
        pairs:
5   a "value of length" and a cost, we need to minimize the value of
        grouping
6   this pairs taken the most large pair as the cost of the group
7
8   Problem for practice: aquire
9   */
10  #include <iostream>
11  #include <vector>
12  #include <algorithm>
13  using namespace std;
14  int pointer; //Keeps track of the best line from previous query
15  vector<long long> M; //Holds the slopes of the lines in the envelope
16  vector<long long> B; //Holds the y-intercepts of the lines in the
        envelope
17  //Returns true if either line l1 or line l3 is always better than line
        l2
18  bool bad(int l1,int l2,int l3)
19  {
20      /*
21      intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
22      intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
23      set the former greater than the latter, and cross-multiply to
24      eliminate division
25      */
26      return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
27  }
28  //Adds a new line (with lowest slope) to the structure
29  void add(long long m,long long b)
30  {
31      //First, let's add it to the end
32      M.push_back(m);
33      B.push_back(b);
34      //If the penultimate is now made irrelevant between the
            antepenultimate
35      //and the ultimate, remove it. Repeat as many times as necessary
36      while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
37      {
38          M.erase(M.end()-2);
39          B.erase(B.end()-2);
40      }
41  }
42  //Returns the minimum y-coordinate of any intersection between a given
        vertical
43  //line and the lower envelope
44  long long query(long long x)
45  {
46      //If we removed what was the best line for the previous query, then
            the
47      //newly inserted line is now the best for that query
48      if (pointer>=M.size())
49          pointer=M.size()-1;
50      //Any better line must be to the right, since query values are
51      //non-decreasing
52      while (pointer<M.size()-1&&
53      M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
54      pointer++;
55      return M[pointer]*x+B[pointer];
56  }
57  int main()
58  {
59      int M,N,i;
60      pair<int,int> a[50000];
61      pair<int,int> rect[50000];
62      scanf("%d",&M);
63      for (i=0; i<M; i++)
64          scanf("%d %d",&a[i].first,&a[i].second);
65      //Sort first by height and then by width (arbitrary labels)
66      sort(a,a+M);
67      for (i=0,N=0; i<M; i++)
68      {
69          /*
```

```
70       When we add a higher rectangle, any rectangles that are also
71       equally thin or thinner become irrelevant, as they are
72       completely contained within the higher one; remove as many
73       as necessary
74       */
75       while (N>0&&rect[N-1].second<=a[i].second)
76         N--;
77       rect[N++]=a[i]; //add the new rectangle
78     }
79     long long cost;
80     add(rect[0].second,0);
81     //initially, the best line could be any of the lines in the envelope,
82     //that is, any line with index 0 or greater, so set pointer=0
83     pointer=0;
84     for (i=0; i<N; i++)
85     {
86       cost=query(rect[i].first);
87       if (i<N)
88         add(rect[i+1].second,cost);
89     }
90     printf("%lld\n",cost);
91     return 0;
92 }
```

2.12. Convex Hull Trick Dinamico

```
1  // g++ -std=c++11 "convexhulltrick_dynamic.cpp" -o run
2  /***
3  =============== <Convex hull trick dynamic version version>
       ===================
4  warning with the use of this, this is a black box, try to use only in an
       emergency.
5  Problem for practice: aquire
6  */
7  #include <bits/stdc++.h>
8  using namespace std;
9  typedef long long ll;
10 const ll is_query = -(1LL<<62);
11   struct Line {
12   ll m, b;
13   mutable multiset<Line>::iterator it;
14   const Line *succ(multiset<Line>::iterator it) const;
15   bool operator<(const Line& rhs) const {
16       if (rhs.b != is_query) return m < rhs.m;
17       const Line *s=succ(it);
18       if(!s) return 0;
19       ll x = rhs.m;
20       return b - s->b < (s->m - m) * x;
21   }
22 };
23 struct HullDynamic : public multiset<Line>{ // will maintain upper hull
       for maximum
24   bool bad(iterator y) {
25     iterator z = next(y);
26     if (y == begin()) {
27       if (z == end()) return 0;
28       return y->m == z->m && y->b <= z->b;
29     }
30     iterator x = prev(y);
31     if (z == end()) return y->m == x->m && y->b <= x->b;
32     return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
33   }
34   iterator next(iterator y){return ++y;}
35   iterator prev(iterator y){return --y;}
36     void insert_line(ll m, ll b) {
37     iterator y = insert((Line) { m, b });
38     y->it=y;
39     if (bad(y)) { erase(y); return; }
40     while (next(y) != end() && bad(next(y))) erase(next(y));
41     while (y != begin() && bad(prev(y))) erase(prev(y));
42   }
43   ll eval(ll x) {
44     Line l = *lower_bound((Line) { x, is_query });
45     return l.m * x + l.b;
46   }
47 }h;
48 const Line *Line::succ(multiset<Line>::iterator it) const{
49 return (++it==h.end()? NULL : &*it);}
```

2.13. Misof Tree

```
1  /*
2  http://codeforces.com/blog/entry/10493#comment-159335
3  Sirve para encontrar el i-esimo numero de un conjunto de numeros que
       vamos insertando en el arbol.
4  Sirve solo si nuestros numeros son del 0 al n-1 (pero podemos mapearlos
```

```
 5   La idea es esta:
 6   Funcionamiento:
 7     - En el fondo sigue siendo un Segment-Tree (hacemos que 'n' sea 2^x)
 8     - Cada nodo guarda cuantos numeros hay en el intervalo (entonces en
          tree[1] dice cuantos numeros tenemos en total)
 9     - Se sigue representando los hijos del nodo 'i' con '2 * i' (izq) y '2
          * i + 1' (der);
10   Query:
11     - si kth es mas grande que todos los que tenemos(tree[1]) o es
          negativo entonces -1
12     - siempre nos mantenemos en el nodo de la izquierda y si es necario
          avanzamos al de la derecha
13                       'i <<= 1'
14       - si kth es mas grande que el nodo de la izquierda(el actual) quiere
            decir que podemos quitarle todos esos
15       numeros a nuestra busqueda 'kth - tree[i]' y buscar el nuevo kth en
            el arbol de la derecha
16         if (kth > tree [i]) kth -= tree [i++];
17       - Ojo en el 'i++' ahi es donde avanzamos al nodo de la derecha
18     - luego hace su formula rara que aun no entendi xD:
19         'i - leaf + (kth > tree [i])';
20   */
21   const int MaxN = 1e6;
22
23   int a [MaxN], s [MaxN];
24   int leaf, tree [100 + MaxN << 2];
25
26   void bld (int n) { leaf = 1 << (32 - __builtin_clz (n)); }
27   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;
28   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;
29   // en "leaf + x" esta cuantas copias tenemos de "x"
30   //Cuidado con intentar hacer del con mas copias de las disponibles, el
         kth() no funcionaria
31   long kth  (int kth, int i = -1) {
32       if (kth > tree [1] || kth <= 0) return i;
33     for (i = 1; i < leaf; i <<= 1) if (kth > tree [i]) kth -= tree [i++];
34       return i - leaf + (kth > tree [i]);
35   }
```

2.14.   Nro. Elementos menores o iguales a X en log n

3.   Algos

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

3.2.   Mo

```
 1   // g++ -std=c++11 "mo.cpp" -o run
 2   /***
 3   ================ <Mo> ===================
 4   Contain a sample about Mo algorithm
 5   Brief explanation when use Mo:
 6   Explain where and when we can use above algorithm
 7
 8   As mentioned, this algorithm is offline, that means we cannot use it
        when we are forced to stick to given order of queries.
 9    That also means we cannot use this when there are update operations.
           Not just that, there is one important possible limitation:
10   We should be able to write the functions add and remove. There will be
        many cases where add is trivial but remove is not.
11   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
12   it is not trivial. Anyways in that case we can use a set to add elements
        , remove elements and report minimum.
13   In that case the add and delete operations are O(log N) (Resulting in O(
        N * Sqrt(N) * log N) algorithm).
14
15   Suggestion first use the add operation, then the erase operation
16   Problem for practice: DQUERY spoj
17   Input: N, then N elements of array M querys with a range L,R
18   */
19   const int MAXV = 1e6 + 10;
20   const int N = 30010;
21   const int M = 200010;
22   int cnt[MAXV];
23   int v[N];
24
25   struct query{
26       int l,r,pos;
27       query(){}
28   };
29   int n;
```

```cpp
30   query qu[M];
31   int ans[M];
32
33   int ret = 0;
34   void add(int pos){
35     pos = v[pos];
36     cnt[pos]++;
37     if(cnt[pos] == 1){
38       ret++;
39     }
40   }
41   void erase(int pos){
42     pos = v[pos];
43     cnt[pos]--;
44     if(!cnt[pos])ret--;
45   }
46   int main(){
47     n = in();
48     for(int i = 0; i < n;i++){
49       v[i] = in();
50     }
51     int block = ceil(sqrt(n));
52     int q = in();
53     for(int i = 0; i < q;i++){
54       qu[i].l = in() - 1,qu[i].r = in() - 1,qu[i].pos = i;
55     }
56     sort(qu,qu + q,[&](const query &a,const query &b){
57       if(a.l / block != b.l / block)
58         return a.l / block < b.l / block;
59       return a.r < b.r;
60     });
61     int l = 0, r = 0;
62     for(int i = 0; i < q;i++){
63       int nl = qu[i].l,nr = qu[i].r;
64       while(l > nl){
65         add(--l);
66       }
67       while(r <= nr){
68         add(r++);
69       }
70       while(l < nl){
71         erase(l++);
72       }
```

```cpp
73       while(r > nr + 1){
74         erase(--r);
75       }
76
77       ans[qu[i].pos] = ret;
78     }
79     for(int i = 0; i < q;i++)printf("%d\n",ans[i]);
80   }
```

4. Strings

4.1. Manacher

```cpp
1   vector<int> manacher(const string &_s) {
2     int n = _s.size();
3     string s(2 * n + 3, '#');
4     s[0] = '%', s[s.size() - 1] = '$';//no deben estar en la cadena
5     for (int i = 0; i < n; i++)
6       s[(i + 1) * 2] = _s[i];
7
8     n = s.size();
9     vector<int> P(n, 0);
10    int C = 0, R = 0;
11    for (int i = 1; i < n - 1; i++) {
12      int j = C - (i - C);
13      if (R > i)
14        P[i] = min(R - i, P[j]);
15      while (s[i + 1 + P[i]] == s[i - 1 - P[i]])
16        P[i]++;
17      if (i + P[i] > R) {
18        C = i;
19        R = i + P[i];
20      }
21    }
22    return P;
23  }
24  bool is_pal(const vector<int> &mnch_vec, int i, int j) {//[i, j] - i<=j
25    int len = j - i + 1;
26    i = (i + 1) * 2;//idx to manacher vec idx
27    j = (j + 1) * 2;
28    int mid = (i + j) / 2;
29    return mnch_vec[mid] >= len;
30  }
```

```
31   int main() {
32     string s;
33     cin >> s;
34     vector<int> mnch_vec= manacher(s);
35     if (is_pal(mnch_vec, 2, 7)) {
36       //la subcadena desde la posicion 2 a la 7 es palindrome
37     }
38     return 0;
39   }
```

4.2. Trie - Punteros y bfs

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

4.4. Minima rotacion lexicografica

```
1    /*
2    Rotacion Lexicografica minima MinRotLex(cadena,tamanio)
3    para cambiar inicio de la cadena char s[300]; int h; s+h;
4    retorna inicio de la rotacion minima :D
5    */
6    int MinRotLex(const char *s, const int slen) {
7      int i = 0, j = 1, k = 0, x, y, tmp;
8      while(i < slen && j < slen && k < slen) {
9        x = i + k;
10       y = j + k;
11       if(x >= slen)  x -= slen;
12       if(y >= slen)  y -= slen;
13       if(s[x] == s[y]) {
14         k++;
15       } else if(s[x] > s[y]) {
16         i = j+1 > i+k+1 ? j+1 : i+k+1;
17         k = 0;
18         tmp = i, i = j, j = tmp;
19       } else {
20         j = i+1 > j+k+1 ? i+1 : j+k+1;
21         k = 0;
22       }
23     }
24     return i;
25   }
26   int main(){
27     int n;
28     scanf("%d",&n);getchar();
```

```
29     while(n--){
30       char str[1000009];
31       gets(str);
32       printf("%d\n",MinRotLex(str,strlen(str))+1);
33     }
34   }
```

4.5. Matching

4.5.1. KMP

```
1    string T;//cadena donde buscar(where)
2    string P;//cadena a buscar(what)
3    int b[MAXLEN];//back table b[i] maximo borde de [0..i)
4    void kmppre(){//by gabina with love
5      int i =0, j=-1; b[0]=-1;
6      while(i<sz(P)){
7        while(j>=0 && P[i] != P[j]) j=b[j];
8        i++, j++, b[i] = j;
9      }
10   }
11   void kmp(){
12     int i=0, j=0;
13     while(i<sz(T)){
14       while(j>=0 && T[i]!=P[j]) j=b[j];
15       i++, j++;
16       if(j==sz(P)) printf("P is found at index %d in T\n", i-j), j=b[j
           ];
17     }
18   }
19
20   int main(){
21     cout << "T=";
22     cin >> T;
23     cout << "P=";
24     cin.ignore();
25     cin >> P;
26     kmppre();
27     kmp();
28     return 0;
29   }
```

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

```
1
2  struct trie{
3    map<char, trie> next;
4    trie* tran[256];//transiciones del automata
5    int idhoja, szhoja;//id de la hoja o 0 si no lo es
6    //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que
         es hoja
7    trie *padre, *link, *nxthoja;
8    char pch;//caracter que conecta con padre
9    trie(): tran(),  idhoja(), padre(), link() {}
10   void insert(const string &s, int id=1, int p=0){//id>0!!!
11     if(p<sz(s)){
12       trie &ch=next[s[p]];
13       tran[(int)s[p]]=&ch;
14       ch.padre=this, ch.pch=s[p];
15       ch.insert(s, id, p+1);
16     }
17     else idhoja=id, szhoja=sz(s);
18   }
19   trie* get_link() {
20     if(!link){
21       if(!padre) link=this;//es la raiz
22       else if(!padre->padre) link=padre;//hijo de la raiz
23       else link=padre->get_link()->get_tran(pch);
24     }
25     return link; }
26   trie* get_tran(int c) {
27     if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
28     return tran[c]; }
29   trie *get_nxthoja(){
30     if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
31     return nxthoja; }
32   void print(int p){
33     if(idhoja) cout << "found " << idhoja << "  at position " << p-
           szhoja << endl;
34     if(get_nxthoja()) get_nxthoja()->print(p); }
35   void matching(const string &s, int p=0){
36     print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
37  }tri;
38
39
40  int main(){
41    tri=trie();//clear
42    tri.insert("ho", 1);
43    tri.insert("hoho", 2);
```

4.6.   Suffix Automaton

```
1  /*################# Suffix Automata #################*/
2  const int N = INSERTE_VALOR;//maxima longitud de la cadena
3  struct State {  //OJO!!! tamanio del alfabeto, si MLE -> map
4      State *pre,*go[26];//se puede usar un map<char, State*> go
5      int step;
6      void clear() {
7          pre=0;
8          step=0;
9          memset(go,0,sizeof(go));//go.clear();
10     }
11  } *root,*last;
12  State statePool[N * 2],*cur;
13  void init() {
14      cur=statePool;
15      root=last=cur++;
16      root->clear();
17  }
18  void Insert(int w) {
19      State *p=last;
20      State *np=cur++;
21      np->clear();
22      np->step=p->step+1;
23      while(p&&!p->go[w])
24          p->go[w]=np,p=p->pre;
25      if(p==0)
26          np->pre=root;
27      else {
28          State *q=p->go[w];
29          if(p->step+1==q->step)
30              np->pre=q;
31          else {
```

```
32          State *nq=cur++;
33          nq->clear();
34          memcpy(nq->go,q->go,sizeof(q->go));//nq->go = q->go; para
                 mapa
35          nq->step=p->step+1;
36          nq->pre=q->pre;
37          q->pre=nq;
38          np->pre=nq;
39          while(p&&p->go[w]==q)
40              p->go[w]=nq, p=p->pre;
41        }
42      }
43      last=np;
44 }
45 /*################## Suffix Automata ##################*/
46
47 /*################## Algunas aplicaciones ##################*/
48 //Obtiene el LCSubstring de 2 cadenas en O(|A| + |B|)
49 string lcs(char A[N], char B[N]) {
50      int n,m;
51      n = strlen(A); m = strlen(B);
52      //Construccion: O(|A|)
53      //solo hacerlo una vez si A no cambia
54      init();
55      for(int i=0; i<n; i++)
56          Insert(A[i]-'a'); //Fin construccion
57      //LCS: O(|B|)
58      int ans = 0, len = 0, bestpos = 0;
59      State *p = root;
60      for(int i = 0; i < m; i++) {
61          int x = B[i]-'a';
62          if(p->go[x]) {
63              len++;
64              p = p->go[x];
65          } else {
66              while (p && !p->go[x]) p = p->pre;
67              if(!p) p = root, len = 0;
68              else   len = p->step+1, p = p->go[x];
69          }
70          if (len > ans)
71              ans = len, bestpos = i;
72      }
73      //return ans; //solo el tamanio del lcs
```

```
74      return string(B + bestpos - ans + 1, B + bestpos + 1);
75 }
76
77 /*Numero de subcadenas distintas + 1(subcadena vacia) en O(|A|)
78 OJO: Por alguna razon Suffix Array es mas rapido
79 Se reduce a contar el numero de paths que inician en q0 y terminan
80 en cualquier nodo. dp[u] = # de paths que inician en u
81 - Se debe construir el automata en el main(init y Insert's)
82 - Setear dp en -1
83 */
84 number dp[N * 2];
85 number num_dist_substr(State *u = root) {
86      if (dp[u - statePool] != -1) return dp[u - statePool];
87      number ans = 1;//el path vacio que representa este nodo
88      for (int v = 0; v < 26; v++)//usar for (auto) para mapa
89          if (u->go[v])
90              ans += num_dist_substr(u->go[v]);
91      return (dp[u - statePool] = ans);
92 }
93
94 /*Suma la longitud de todos los substrings en O(|A|)
95 - Construir el automata(init y insert's)
96 - Necesita el metodo num_dist_substr (el de arriba)
97 - setear dp's en -1
98 */
99 number dp1[N * 2];
100 number sum_length_dist_substr(State *u = root) {
101      if (dp1[u - statePool] != -1) return dp1[u - statePool];
102      number ans = 0;//el path vacio que representa este nodo
103      for (int v = 0; v < 26; v++)//usar for (auto) para mapa
104          if (u->go[v])
105              ans += (num_dist_substr(u->go[v]) + sum_length_dist_substr(u
                     ->go[v]));
106      return (dp1[u - statePool] = ans);
107 }
108
109 /*
110 Pregunta si p es subcadena de la cadena con la cual esta construida
111 el automata.
112 Complejidad: - Construir O(|Texto|) - solo una vez (init e insert's)
113             - Por Consulta O(|patron a buscar|)
114 */
115 bool is_substring(char p[N]) {
```

```
116      State *u = root;
117      for (int i = 0; p[i]; i++) {
118          if (!u->go.count(p[i]))//esta con map!!!
119              return false;
120          u = u->go[p[i]];//esta con map!!!
121      }
122      return true;
123  }
```

4.7. K-esima permutacion de una cadena

```
1  //Entrada: Una cadena cad(std::string), un long th
2  //Salida : La th-esima permutacion lexicografica de cad
3  string ipermutacion(string cad, long long int th){
4    sort(cad.begin(), cad.end());
5    string sol = "";
6    int pos;
7    for(int c = cad.size() - 1; c >= 0; c--){
8      pos = th / fact[c];
9      th %= fact[c];
10     sol += cad[pos];
11     cad.erase(cad.begin() + pos);
12   }
13   return sol;
14 }
```

5. Geometria

5.1. Interseccion de circunferencias - Sacar de Agustin

5.2. Graham Scan

5.3. Cortar Poligono

```
1  //cuts polygon Q along the line ab
2  //stores the left side (swap a, b for the right one) in P
3  void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
4    P.clear();
5    forn(i, sz(Q)){
6      double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
7      if(left1>=0) P.pb(Q[i]);
8      if(left1*left2<0)
9        P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10   }
```

```
11 }
```

5.4. Interseccion de rectangulos

```
1  #define MAXC 2501
2  struct Rect{
3    int x1,y1, x2,y2;
4    int color;
5    int area;
6    Rect(int _x1, int _y1, int _x2, int _y2){
7      x1 = _x1;
8      y1 = _y1;
9      x2 = _x2;
10     y2 = _y2;
11     getArea();
12   }
13   int getArea(){
14     if(x1>=x2 || y1>=y2)return area = 0;
15     return area = (x2-x1)*(y2-y1);
16   }
17 };
18 Rect interseccion(Rect t, Rect r){
19   int x1,y1,x2,y2;
20   x1 = max(t.x1,r.x1);
21   y1 = max(t.y1,r.y1);
22   x2 = min(t.x2,r.x2);
23   y2 = min(t.y2,r.y2);
24   Rect res(x1,y1,x2,y2);
25   return res;
26 }
```

5.5. Distancia punto-recta

```
1  double distance_point_to_line(const point &a, const point &b, const
       point &pnt){
2    double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) /
         distsqr(a, b);
3    point intersection;
4    intersection.x = a.x + u*(b.x - a.x);
5    intersection.y = a.y + u*(b.y - a.y);
6    return dist(pnt, intersection);
7  }
```

5.6. Distancia punto-segmento

```
1  struct point{
2    double x,y;
3  };
4  inline double dist(const point &a, const point &b){
5    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
6  }
7  inline double distsqr(const point &a, const point &b){
8    return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
9  }
10 double distance_point_to_segment(const point &a, const point &b, const
       point &pnt){
11   double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) /
         distsqr(a, b);
12   point intersection;
13   intersection.x = a.x + u*(b.x - a.x);
14   intersection.y = a.y + u*(b.y - a.y);
15
16   if (u < 0.0 || u > 1.0)
17     return min(dist(a, pnt), dist(b, pnt));
18
19   return dist(pnt, intersection);
20 }
```

5.7.   Parametrizacion de rectas - Sacar de codeforces

6.   Math

6.1.   Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$
$$\sum_{i=0}^{n} i\binom{n}{i} = n*2^{n-1}$$
$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$
$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$
$$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1) \text{ (doubles)} \to \text{Sino ver caso impar y par}$$
$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$$
$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$
$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1}\binom{p}{k}(n+1)^{p-k+1}$$
$$r = e - v + k + 1$$

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

```
1  forn(i, MAXN+1){//comb[i][k]=i tomados de a k
2    comb[i][0]=comb[i][i]=1;
3    forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
4  }
5  ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p]
       precalculado.
6    ll aux = 1;
7    while (n + k) aux = (aux * comb[n%p][k%p]) %p, n/=p, k/=p;
8    return aux;
9  }
```

6.5.   Exp. de Numeros Mod.

```
1  ll expmod (ll b, ll e, ll m){//O(log b)
2    if(!e) return 1;
3    ll q= expmod(b,e/2,m); q=(q*q)%m;
4    return e%2? (b * q)%m : q;
5  }
```

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

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

```
1  struct Mat {
2      vector<vector<double> > vec;
3      Mat(int n): vec(n, vector<double>(n) ) {}
4      Mat(int n, int m): vec(n, vector<double>(m) ) {}
5      vector<double> &operator[](int f){return vec[f];}
6      const vector<double> &operator[](int f) const {return vec[f];}
7      int size() const {return sz(vec);}
8      Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
9          Mat m(sz(b),sz(b[0]));
```

```
10      forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j
            ];
11      return m;     }
12    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
13      int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
14      Mat mat(n,t);
15      forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
16      return mat;     }
17    double determinant(){//sacado de e maxx ru
18      double det = 1;
19      int n = sz(vec);
20      Mat m(*this);
21      forn(i, n){//para cada columna
22          int k = i;
23          forr(j, i+1, n)//busco la fila con mayor val abs
24              if(abs(m[j][i])>abs(m[k][i])) k = j;
25          if(abs(m[k][i])<1e-9) return 0;
26          m[i].swap(m[k]);//la swapeo
27          if(i!=k) det = -det;
28          det *= m[i][i];
29          forr(j, i+1, n) m[i][j] /= m[i][i];
30          //hago 0 todas las otras filas
31          forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
32              forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
33      }
34      return det;
35    }
36 };
37
38 int n;
39 int main() {
40 //DETERMINANTE:
41 //https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&
        page=show_problem&problem=625
42   freopen("input.in", "r", stdin);
43     ios::sync_with_stdio(0);
44     while(cin >> n && n){
45         Mat m(n);
46         forn(i, n) forn(j, n) cin >> m[i][j];
47         cout << (ll)round(m.determinant()) << endl;
48     }
49     cout  << "*" << endl;
50   return 0;
51 }
```

6.8. Teorema Chino del Resto

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

6.9. Criba

```
1 #define MAXP 100000 //no necesariamente primo
2 int criba[MAXP+1];
3 void crearcriba(){
4   int w[] = {4,2,4,2,4,6,2,6};
5   for(int p=25;p<=MAXP;p+=10) criba[p]=5;
6   for(int p=9;p<=MAXP;p+=6) criba[p]=3;
7   for(int p=4;p<=MAXP;p+=2) criba[p]=2;
8   for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
9     for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
10 }
11 vector<int> primos;
12 void buscarprimos(){
13   crearcriba();
14   forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
15 }
16 //~ 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];
17
18
19 int main() {
20   freopen("primos", "w", stdout);
21   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

```
1 //factoriza bien numeros hasta MAXP^2
2 map<ll,ll> fact(ll n){ //O (cant primos)
3   map<ll,ll> ret;
4   forall(p, primos){
5     while(!(n%p)){
6       ret[*p]++;//divisor found
```

```
7        n/=*p;
8      }
9    }
10   if(n>1) ret[n]++;
11   return ret;
12 }
13 //factoriza bien numeros hasta MAXP
14 map<ll,ll> fact2(ll n){ //O (lg n)
15   map<ll,ll> ret;
16   while (criba[n]){
17     ret[criba[n]]++;
18     n/=criba[n];
19   }
20   if(n>1) ret[n]++;
21   return ret;
22 }
23 //Usar asi:  divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
24 void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::
        iterator it, ll n=1){
25     if(it==f.begin()) divs.clear();
26     if(it==f.end()) { divs.pb(n);  return; }
27     ll p=it->fst, k=it->snd; ++it;
28     forn(_, k+1) divisores(f, divs, it, n), n*=p;
29 }
30 ll sumDiv (ll n){
31   ll rta = 1;
32   map<ll,ll> f=fact(n);
33   forall(it, f) {
34   ll pot = 1, aux = 0;
35   forn(i, it->snd+1) aux += pot, pot *= it->fst;
36   rta*=aux;
37   }
38   return rta;
39 }
40 ll eulerPhi (ll n){ // con criba: O(lg n)
41   ll rta = n;
42   map<ll,ll> f=fact(n);
43   forall(it, f) rta -= rta / it->first;
44   return rta;
45 }
46 ll eulerPhi2 (ll n){ // O (sqrt n)
47   ll r = n;
48   forr (i,2,n+1){
```

```
49     if ((ll)i*i > n) break;
50     if (n % i == 0){
51       while (n%i == 0) n/=i;
52       r -= r/i; }
53   }
54   if (n != 1) r-= r/n;
55   return r;
56 }
57
58 int main() {
59   buscarprimos();
60   forr (x,1, 500000){
61     cout << "x = " << x << endl;
62     cout << "Numero de factores primos: " << numPrimeFactors(x) << endl;
63     cout << "Numero de distintos factores primos: " <<
            numDiffPrimeFactors(x) << endl;
64     cout << "Suma de factores primos: " << sumPrimeFactors(x) << endl;
65     cout << "Numero de divisores: " << numDiv(x) << endl;
66     cout << "Suma de divisores: " << sumDiv(x) << endl;
67     cout << "Phi de Euler: " << eulerPhi(x) << endl;
68   }
69   return 0;
70 }
```

6.11.   Phollard's Rho (rolando)

```
1  ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}
2
3  ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overfloor
4    ll x = 0, y = a%c;
5    while (b > 0){
6      if (b % 2 == 1) x = (x+y) % c;
7      y = (y*2) % c;
8      b /= 2;
9    }
10   return x % c;
11 }
12
13 ll expmod (ll b, ll e, ll m){//O(log b)
14   if(!e) return 1;
15   ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
16   return e%2? mulmod(b,q,m) : q;
```

```
17  }
18
19  bool es_primo_prob (ll n, int a)
20  {
21    if (n == a) return true;
22    ll s = 0,d = n-1;
23    while (d % 2 == 0) s++,d/=2;
24
25    ll x = expmod(a,d,n);
26    if ((x == 1) || (x+1 == n)) return true;
27
28    forn (i, s-1){
29      x = mulmod(x, x, n);
30      if (x == 1) return false;
31      if (x+1 == n) return true;
32    }
33    return false;
34  }
35
36  bool rabin (ll n){ //devuelve true si n es primo
37    if (n == 1) return false;
38    const int ar[] = {2,3,5,7,11,13,17,19,23};
39    forn (j,9)
40      if (!es_primo_prob(n,ar[j]))
41        return false;
42    return true;
43  }
44
45  ll rho(ll n){
46      if( (n & 1) == 0 ) return 2;
47      ll x = 2 , y = 2 , d = 1;
48      ll c = rand() % n + 1;
49      while( d == 1 ){
50          x = (mulmod( x , x , n ) + c)%n;
51          y = (mulmod( y , y , n ) + c)%n;
52          y = (mulmod( y , y , n ) + c)%n;
53          if( x - y >= 0 ) d = gcd( x - y , n );
54          else d = gcd( y - x , n );
55      }
56      return d==n? rho(n):d;
57  }
58
59  map<ll,ll> prim;
```

```
60  void factRho (ll n){ //O (lg n)^3. un solo numero
61    if (n == 1) return;
62    if (rabin(n)){
63      prim[n]++;
64      return;
65    }
66    ll factor = rho(n);
67    factRho(factor);
68    factRho(n/factor);
69  }
```

6.12. GCD

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

6.13. Extended Euclid

```
1  void extendedEuclid (ll a, ll b){ //a * x + b * y = d
2    if (!b) { x = 1; y = 0; d = a; return;}
3    extendedEuclid (b, a%b);
4    ll x1 = y;
5    ll y1 = x - (a/b) * y;
6    x = x1; y = y1;
7  }
```

6.14. LCM

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

6.15. Inversos

```
1  #define MAXMOD 15485867
2  ll inv[MAXMOD];//inv[i]*i=1 mod MOD
3  void calc(int p){//O(p)
4    inv[1]=1;
5    forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
6  }
7  int inverso(int x){//O(log x)
8    return expmod(x, eulerphi(MOD)-2);//si mod no es primo(sacar a mano)
9    return expmod(x, MOD-2);//si mod es primo
10 }
```

6.16. Simpson

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

6.17. Fraction

```
1  tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
2  struct frac{
3    tipo p,q;
4    frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
5    void norm(){
6      tipo a = mcd(p,q);
7      if(a) p/=a, q/=a;
8      else q=1;
9      if (q<0) q=-q, p=-p;}
10   frac operator+(const frac& o){
11     tipo a = mcd(q,o.q);
12     return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13   frac operator-(const frac& o){
14     tipo a = mcd(q,o.q);
15     return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16   frac operator*(frac o){
17     tipo a = mcd(q,o.p), b = mcd(o.q,p);
18     return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19   frac operator/(frac o){
20     tipo a = mcd(q,o.q), b = mcd(o.p,p);
21     return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
22   bool operator<(const frac &o) const{return p*o.q < o.p*q;}
23   bool operator==(frac o){return p==o.p&&q==o.q;}
24 };
```

6.18. Polinomio

```
1          int m = sz(c), n = sz(o.c);
2          vector<tipo> res(max(m,n));
3          forn(i, m) res[i] += c[i];
4          forn(i, n) res[i] += o.c[i];
5          return poly(res);     }
6      poly operator*(const tipo cons) const {
7      vector<tipo> res(sz(c));
```

```
8          forn(i, sz(c)) res[i]=c[i]*cons;
9          return poly(res);     }
10     poly operator*(const poly &o) const {
11         int m = sz(c), n = sz(o.c);
12         vector<tipo> res(m+n-1);
13         forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
14         return poly(res);     }
15   tipo eval(tipo v) {
16     tipo sum = 0;
17     dforn(i, sz(c)) sum=sum*v + c[i];
18     return sum; }
19     //poly contains only a vector<int> c (the coeficients)
20   //the following function generates the roots of the polynomial
21 //it can be easily modified to return float roots
22   set<tipo> roots(){
23     set<tipo> roots;
24     tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
25     vector<tipo> ps,qs;
26     forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
27     forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
28     forall(pt,ps)
29       forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
30         tipo root = abs((*pt) / (*qt));
31         if (eval(root)==0) roots.insert(root);
32       }
33     return roots; }
34 };
35 pair<poly,tipo> ruffini(const poly p, tipo r) {
36   int n = sz(p.c) - 1 ;
37   vector<tipo> b(n);
38   b[n-1] = p.c[n];
39   dforn(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
40   tipo resto = p.c[0] + r*b[0];
41   poly result(b);
42   return make_pair(result,resto);
43 }
44 poly interpolate(const vector<tipo>& x,const vector<tipo>& y) {
45     poly A; A.c.pb(1);
46     forn(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux;
47         }
48   poly S; S.c.pb(0);
49   forn(i,sz(x)) { poly Li;
50     Li = ruffini(A,x[i]).fst;
```

```
50      Li = Li * (1.0 / Li.eval(x[i])); // here put a multiple of the
              coefficients instead of 1.0 to avoid using double
51      S = S + Li * y[i];  }
52    return S;
53  }
54
55  int  main(){
56    return 0;
57  }
```

6.19.   Ec. Lineales

```
1  bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
2    int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
3    vector<int> p; forn(i,m) p.push_back(i);
4    forn(i, rw) {
5      int uc=i, uf=i;
6      forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;
              uc=c;}
7      if (feq(a[uf][uc], 0)) { rw = i; break; }
8      forn(j, n) swap(a[j][i], a[j][uc]);
9      swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
10     tipo inv = 1 / a[i][i]; //aca divide
11     forr(j, i+1, n) {
12       tipo v = a[j][i] * inv;
13       forr(k, i, m) a[j][k]-=v * a[i][k];
14       y[j] -= v*y[i];
15     }
16   } // rw = rango(a), aca la matriz esta triangulada
17   forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de
          compatibilidad
18   x = vector<tipo>(m, 0);
19   dforn(i, rw){
20     tipo s = y[i];
21     forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
22     x[p[i]] = s / a[i][i]; //aca divide
23   }
24   ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
25   forn(k, m-rw) {
26     ev[k][p[k+rw]] = 1;
27     dforn(i, rw){
28       tipo s = -a[i][k+rw];
29       forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
```

```
30       ev[k][p[i]] = s / a[i][i]; //aca divide
31     }
32   }
33   return true;
34  }
```

6.20.   FFT

```
1  //~ typedef complex<double> base; //menos codigo, pero mas lento
2  //elegir si usar complejos de c (lento) o estos
3  struct base{
4      double r,i;
5      base(double r=0, double i=0):r(r), i(i){}
6      double real()const{return r;}
7      void operator/=(const int c){r/=c, i/=c;}
8  };
9  base operator*(const base &a, const base &b){
10     return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
11 base operator+(const base &a, const base &b){
12     return base(a.r+b.r, a.i+b.i);}
13 base operator-(const base &a, const base &b){
14     return base(a.r-b.r, a.i-b.i);}
15 vector<int> rev; vector<base> wlen_pw;
16 inline static void fft(base a[], int n, bool invert) {
17     forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
18   for (int len=2; len<=n; len<<=1) {
19     double ang = 2*M_PI/len * (invert?-1:+1);
20     int len2 = len>>1;
21     base wlen (cos(ang), sin(ang));
22     wlen_pw[0] = base (1, 0);
23         forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
24     for (int i=0; i<n; i+=len) {
25       base t, *pu = a+i, *pv = a+i+len2,  *pu_end = a+i+len2, *pw = &
              wlen_pw[0];
26       for (; pu!=pu_end; ++pu, ++pv, ++pw)
27       t = *pv * *pw, *pv = *pu - t,*pu = *pu + t;
28     }
29   }
30   if (invert) forn(i, n) a[i]/= n;}
31 inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
32     wlen_pw.resize(n), rev.resize(n);
33     int lg=31-__builtin_clz(n);
34     forn(i, n){
```

```
35      rev[i] = 0;
36          forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg-1-k);
37      }}
38   inline static void multiply(const vector<int> &a, const vector<int> &b,
         vector<int> &res) {
39     vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
40       int n=1; while(n < max(sz(a), sz(b))) n <<= 1; n <<= 1;
41       calc_rev(n);
42     fa.resize (n),  fb.resize (n);
43     fft (&fa[0], n, false),  fft (&fb[0], n, false);
44     forn(i, n) fa[i] = fa[i] * fb[i];
45     fft (&fa[0], n, true);
46     res.resize(n);
47       forn(i, n) res[i] = int (fa[i].real() + 0.5); }
48   void toPoly(const string &s, vector<int> &P){//convierte un numero a
         polinomio
49       P.clear();
50       dforn(i, sz(s)) P.pb(s[i]-'0');}
```

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

Factoriales

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

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];
```

```
2   priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
3   void process(int v){
4       taken[v]=true;
5       forall(e, G[v])
6           if(!taken[e->second]) pq.push(*e);
7   }
8
9   ll prim(){
10      zero(taken);
11      process(0);
12      ll cost=0;
13      while(sz(pq)){
14          ii e=pq.top(); pq.pop();
15          if(!taken[e.second]) cost+=e.first, process(e.second);
16      }
17      return cost;
18  }
```

7.6.   2-SAT + Tarjan SCC

```
1   //We have a vertex representing a var and other for his negation.
2   //Every edge stored in G represents an implication. To add an equation
        of the form a||b, use addor(a, b)
3   //MAX=max cant var, n=cant var
4   #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5   vector<int> G[MAX*2];
6   //idx[i]=index assigned in the dfs
7   //lw[i]=lowest index(closer from the root) reachable from i
8   int lw[MAX*2], idx[MAX*2], qidx;
9   stack<int> q;
10  int qcmp, cmp[MAX*2];
11  //verdad[cmp[i]]=valor de la variable i
12  bool verdad[MAX*2+1];
13
14  int neg(int x) { return x>=n? x-n : x+n;}
15  void tjn(int v){
16    lw[v]=idx[v]=++qidx;
17    q.push(v), cmp[v]=-2;
18    forall(it, G[v]){
19      if(!idx[*it] || cmp[*it]==-2){
20        if(!idx[*it]) tjn(*it);
21        lw[v]=min(lw[v], lw[*it]);
22      }
23    }
24    if(lw[v]==idx[v]){
25      int x;
26      do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
27      verdad[qcmp]=(cmp[neg(v)]<0);
28      qcmp++;
29    }
30  }
31  //remember to CLEAR G!!!
32  bool satisf(){//O(n)
33    memset(idx, 0, sizeof(idx)), qidx=0;
34    memset(cmp, -1, sizeof(cmp)), qcmp=0;
35    forn(i, n){
36      if(!idx[i]) tjn(i);
37      if(!idx[neg(i)]) tjn(neg(i));
38    }
39    forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
40    return true;
41  }
```

7.7.   Articulation Points

```
1   int N;
2   vector<int> G[1000000];
3   //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4   int qV, V[1000000], L[1000000], P[1000000];
5   void dfs(int v, int f){
6     L[v]=V[v]=++qV;
7     forall(it, G[v])
8       if(!V[*it]){
9         dfs(*it, v);
10        L[v] = min(L[v], L[*it]);
11        P[v]+= L[*it]>=V[v];
12      }
13      else if(*it!=f)
14        L[v]=min(L[v], V[*it]);
15  }
16  int cantart(){ //O(n)
17    qV=0;
18    zero(V), zero(P);
19    dfs(1, 0); P[1]--;
20    int q=0;
21    forn(i, N) if(P[i]) q++;
```

```
22    return q;
23  }
```

7.8.   Comp. Biconexas y Puentes

```
1   struct edge {
2     int u,v, comp;
3     bool bridge;
4   };
5   vector<edge> e;
6   void addEdge(int u, int v) {
7     G[u].pb(sz(e)), G[v].pb(sz(e));
8     e.pb((edge){u,v,-1,false});
9   }
10  //d[i]=id de la dfs
11  //b[i]=lowest id reachable from i
12  int d[MAXN], b[MAXN], t;
13  int nbc;//cant componentes
14  int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
15  void initDfs(int n) {
16    zero(G), zero(comp);
17    e.clear();
18    forn(i,n) d[i]=-1;
19    nbc = t = 0;
20  }
21  stack<int> st;
22  void dfs(int u, int pe) {//O(n + m)
23    b[u] = d[u] = t++;
24    comp[u] = (pe != -1);
25    forall(ne, G[u]) if (*ne != pe){
26      int v = e[*ne].u ^ e[*ne].v ^ u;
27      if (d[v] == -1) {
28        st.push(*ne);
29        dfs(v,*ne);
30        if (b[v] > d[u]){
31          e[*ne].bridge = true; // bridge
32        }
33        if (b[v] >= d[u]){ // art
34          int last;
35          do {
36            last = st.top(); st.pop();
37            e[last].comp = nbc;
38          } while (last != *ne);
```

```
39          nbc++;
40          comp[u]++;
41        }
42        b[u] = min(b[u], b[v]);
43      }
44      else if (d[v] < d[u]) { // back edge
45        st.push(*ne);
46        b[u] = min(b[u], d[v]);
47      }
48    }
49  }
```

7.9.   LCA + Climb

```
1   const int MAXN=100001;
2   const int LOGN=20;
3   //f[v][k] holds the 2^k father of v
4   //L[v] holds the level of v
5   int N, f[MAXN][LOGN], L[MAXN];
6   //call before build:
7   void dfs(int v, int fa=-1, int lvl=0){//generate required data
8     f[v][0]=fa, L[v]=lvl;
9     forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
10  void build(){//f[i][0] must be filled previously, O(nlgn)
11    forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
12  #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
13  int climb(int a, int d){//O(lgn)
14    if(!d) return a;
15    dforn(i, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
16      return a;}
17  int lca(int a, int b){//O(lgn)
18    if(L[a]<L[b]) swap(a, b);
19    a=climb(a, L[a]-L[b]);
20    if(a==b) return a;
21    dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
22    return f[a][0]; }
23  int dist(int a, int b) {//returns distance between nodes
24    return L[a]+L[b]-2*L[lca(a, b)];}
```

7.10.   Heavy Light Decomposition

```
1   int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
2   int dad[MAXN];//dad[v]=padre del nodo v
3   void dfs1(int v, int p=-1){//pre-dfs
```

```
4      dad[v]=p;
5      treesz[v]=1;
6      forall(it, G[v]) if(*it!=p){
7        dfs1(*it, v);
8        treesz[v]+=treesz[*it];
9      }
10  }
11  //PONER Q EN 0  !!!!!
12  int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
13  //Las cadenas aparecen continuas en el recorrido!
14  int cantcad;
15  int homecad[MAXN];//dada una cadena devuelve su nodo inicial
16  int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
17  void heavylight(int v, int cur=-1){
18    if(cur==-1) homecad[cur=cantcad++]=v;
19    pos[v]=q++;
20    cad[v]=cur;
21    int mx=-1;
22    forn(i, sz(G[v])) if(G[v][i]!=dad[v])
23      if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
24    if(mx!=-1) heavylight(G[v][mx], cur);
25    forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
26      heavylight(G[v][i], -1);
27  }
28  //ejemplo de obtener el maximo numero en el camino entre dos nodos
29  //RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
30  //esta funcion va trepando por las cadenas
31  int query(int an, int v){//O(logn)
32    //si estan en la misma cadena:
33    if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
34    return max(query(an, dad[homecad[cad[v]]]),
35          rmq.get(pos[homecad[cad[v]]], pos[v]+1));
36  }
```

7.11.   Centroid Decomposition

```
1  int n;
2  vector<int> G[MAXN];
3  bool taken[MAXN];//poner todos en FALSE al principio!!
4  int padre[MAXN];//padre de cada nodo en el centroid tree
5
6  int szt[MAXN];
7  void calcsz(int v, int p) {
```

```
8    szt[v] = 1;
9    forall(it,G[v]) if (*it!=p && !taken[*it])
10     calcsz(*it,v), szt[v]+=szt[*it];
11  }
12  void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
13    if(tam==-1) calcsz(v, -1), tam=szt[v];
14    forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
15      {szt[v]=0; centroid(*it, f, lvl, tam); return;}
16    taken[v]=true;
17    padre[v]=f;
18    forall(it, G[v]) if(!taken[*it])
19      centroid(*it, v, lvl+1, -1);
20  }
```

7.12.   Euler Cycle

```
1  int n,m,ars[MAXE], eq;
2  vector<int> G[MAXN];//fill G,n,m,ars,eq
3  list<int> path;
4  int used[MAXN];
5  bool usede[MAXE];
6  queue<list<int>::iterator> q;
7  int get(int v){
8    while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9    return used[v];
10  }
11  void explore(int v, int r, list<int>::iterator it){
12    int ar=G[v][get(v)]; int u=v^ars[ar];
13    usede[ar]=true;
14    list<int>::iterator it2=path.insert(it, u);
15    if(u!=r) explore(u, r, it2);
16    if(get(v)<sz(G[v])) q.push(it);
17  }
18  void euler(){
19    zero(used), zero(usede);
20    path.clear();
21    q=queue<list<int>::iterator>();
22    path.push_back(0); q.push(path.begin());
23    while(sz(q)){
24      list<int>::iterator it=q.front(); q.pop();
25      if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26    }
27    reverse(path.begin(), path.end());
```

```
28 }
29 void addEdge(int u, int v){
30   G[u].pb(eq), G[v].pb(eq);
31   ars[eq++]=u^v;
32 }
```

7.13. Diametro árbol

```
1  vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
2  int bfs(int r, int *d) {
3    queue<int> q;
4    d[r]=0; q.push(r);
5    int v;
6    while(sz(q)) { v=q.front(); q.pop();
7      forall(it,G[v]) if (d[*it]==-1)
8        d[*it]=d[v]+1, p[*it]=v, q.push(*it);
9    }
10   return v;//ultimo nodo visitado
11 }
12 vector<int> diams; vector<ii> centros;
13 void diametros(){
14   memset(d,-1,sizeof(d));
15   memset(d2,-1,sizeof(d2));
16   diams.clear(), centros.clear();
17   forn(i, n) if(d[i]==-1){
18     int v,c;
19     c=v=bfs(bfs(i, d2), d);
20     forn(_,d[v]/2) c=p[c];
21     diams.pb(d[v]);
22     if(d[v]&1) centros.pb(ii(c, p[c]));
23     else centros.pb(ii(c, c));
24   }
25 }
26
27 int main() {
28   freopen("in", "r", stdin);
29   while(cin >> n >> m){
30     forn(i,m) { int a,b; cin >> a >> b; a--, b--;
31       G[a].pb(b);
32       G[b].pb(a);
```

7.14. Chu-liu

```
1 void visit(graph &h, int v, int s, int r,
```

```
2    vector<int> &no, vector< vector<int> > &comp,
3    vector<int> &prev, vector< vector<int> > &next, vector<weight> &mcost,
4    vector<int> &mark, weight &cost, bool &found) {
5    if (mark[v]) {
6      vector<int> temp = no;
7      found = true;
8      do {
9        cost += mcost[v];
10       v = prev[v];
11       if (v != s) {
12         while (comp[v].size() > 0) {
13           no[comp[v].back()] = s;
14           comp[s].push_back(comp[v].back());
15           comp[v].pop_back();
16         }
17       }
18     } while (v != s);
19     forall(j,comp[s]) if (*j != r) forall(e,h[*j])
20       if (no[e->src] != s) e->w -= mcost[ temp[*j] ];
21   }
22   mark[v] = true;
23   forall(i,next[v]) if (no[*i] != no[v] && prev[no[*i]] == v)
24     if (!mark[no[*i]] || *i == s)
25       visit(h, *i, s, r, no, comp, prev, next, mcost, mark, cost, found)
           ;
26 }
27 weight minimumSpanningArborescence(const graph &g, int r) {
28     const int n=sz(g);
29   graph h(n);
30   forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
31   vector<int> no(n);
32   vector<vector<int> > comp(n);
33   forn(u, n) comp[u].pb(no[u] = u);
34   for (weight cost = 0; ;) {
35     vector<int> prev(n, -1);
36     vector<weight> mcost(n, INF);
37     forn(j,n) if (j != r) forall(e,h[j])
38       if (no[e->src] != no[j])
39         if (e->w < mcost[ no[j] ])
40           mcost[ no[j] ] = e->w, prev[ no[j] ] = no[e->src];
41     vector< vector<int> > next(n);
42     forn(u,n) if (prev[u] >= 0)
43       next[ prev[u] ].push_back(u);
```

```
44    bool stop = true;
45    vector<int> mark(n);
46    forn(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
47      bool found = false;
48      visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
49      if (found) stop = false;
50    }
51    if (stop) {
52      forn(u,n) if (prev[u] >= 0) cost += mcost[u];
53      return cost;
54    }
55  }
56 }
```

7.15. Hungarian

```
1  //Dado un grafo bipartito completo con costos no negativos, encuentra el
      matching perfecto de minimo costo.
2  tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar: cost=matriz de
      adyacencia
3  int n, max_match, xy[N], yx[N], slackx[N],prev2[N];//n=cantidad de nodos
4  bool S[N], T[N]; //sets S and T in algorithm
5  void add_to_tree(int x, int prevx) {
6    S[x] = true, prev2[x] = prevx;
7    forn(y, n) if (lx[x] + ly[y] - cost[x][y] < slack[y] - EPS)
8      slack[y] = lx[x] + ly[y] - cost[x][y], slackx[y] = x;
9  }
10 void update_labels(){
11   tipo delta = INF;
12   forn (y, n) if (!T[y]) delta = min(delta, slack[y]);
13   forn (x, n) if (S[x]) lx[x] -= delta;
14   forn (y, n) if (T[y]) ly[y] += delta; else slack[y] -= delta;
15 }
16 void init_labels(){
17   zero(lx), zero(ly);
18   forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x][y]);
19 }
20 void augment() {
21   if (max_match == n) return;
22   int x, y, root, q[N], wr = 0, rd = 0;
23   memset(S, false, sizeof(S)), memset(T, false, sizeof(T));
24   memset(prev2, -1, sizeof(prev2));
25   forn (x, n) if (xy[x] == -1){
26     q[wr++] = root = x, prev2[x] = -2;
27     S[x] = true; break; }
28   forn (y, n) slack[y] = lx[root] + ly[y] - cost[root][y], slackx[y] =
        root;
29   while (true){
30     while (rd < wr){
31       x = q[rd++];
32       for (y = 0; y < n; y++) if (cost[x][y] == lx[x] + ly[y] && !T[y]){
33         if (yx[y] == -1) break; T[y] = true;
34         q[wr++] = yx[y], add_to_tree(yx[y], x); }
35       if (y < n) break; }
36     if (y < n) break;
37     update_labels(), wr = rd = 0;
38     for (y = 0; y < n; y++) if (!T[y] && slack[y] == 0){
39       if (yx[y] == -1){x = slackx[y]; break;}
40       else{
41         T[y] = true;
42         if (!S[yx[y]]) q[wr++] = yx[y], add_to_tree(yx[y], slackx[y]);
43       }}
44     if (y < n) break; }
45   if (y < n){
46     max_match++;
47     for (int cx = x, cy = y, ty; cx != -2; cx = prev2[cx], cy = ty)
48       ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
49     augment(); }
50 }
51 tipo hungarian(){
52   tipo ret = 0; max_match = 0, memset(xy, -1, sizeof(xy));
53   memset(yx, -1, sizeof(yx)), init_labels(), augment(); //steps 1-3
54   forn (x,n) ret += cost[x][xy[x]]; return ret;
55 }
```

7.16. Dynamic Conectivity

```
1  struct UnionFind {
2    int n, comp;
3    vector<int> pre,si,c;
4    UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
5      forn(i,n) pre[i] = i; }
6    int find(int u){return u==pre[u]?u:find(pre[u]);}
7    bool merge(int u, int v) {
8      if((u=find(u))==(v=find(v))) return false;
9      if(si[u]<si[v]) swap(u, v);
```

```
10          si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
11          return true;
12      }
13      int snap(){return sz(c);}
14      void rollback(int snap){
15          while(sz(c)>snap){
16              int v = c.back(); c.pop_back();
17              si[pre[v]] -= si[v], pre[v] = v, comp++;
18          }
19      }
20  };
21  enum {ADD,DEL,QUERY};
22  struct Query {int type,u,v;};
23  struct DynCon {
24      vector<Query> q;
25      UnionFind dsu;
26      vector<int> match,res;
27      map<ii,int> last;//se puede no usar cuando hay identificador para
                 cada arista (mejora poco)
28      DynCon(int n=0):dsu(n){}
29      void add(int u, int v) {
30          if(u>v) swap(u,v);
31          q.pb((Query){ADD, u, v}), match.pb(-1);
32          last[ii(u,v)] = sz(q)-1;
33      }
34      void remove(int u, int v) {
35          if(u>v) swap(u,v);
36          q.pb((Query){DEL, u, v});
37          int prev = last[ii(u,v)];
38          match[prev] = sz(q)-1;
39          match.pb(prev);
40      }
41      void query() {//podria pasarle un puntero donde guardar la respuesta
42          q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
43      void process() {
44          forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] =
                 sz(q);
45          go(0,sz(q));
46      }
47      void go(int l, int r) {
48          if(l+1==r){
49              if (q[l].type == QUERY)//Aqui responder la query usando el
                     dsu!
50                  res.pb(dsu.comp);//aqui query=cantidad de componentes
                         conexas
51              return;
52          }
53          int s=dsu.snap(), m = (l+r) / 2;
54          forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i
                 ].v);
55          go(l,m);
56          dsu.rollback(s);
57          s = dsu.snap();
58          forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[
                 i].v);
59          go(m,r);
60          dsu.rollback(s);
61      }
62  }dc;
```

8.   Network Flow

8.1.   Dinic

```
1
2   const int MAX = 300;
3   // Corte minimo: vertices con dist[v]>=0 (del lado de src) VS. dist[v
        ]==-1 (del lado del dst)
4   // Para el caso de la red de Bipartite Matching (Sean V1 y V2 los
        conjuntos mas proximos a src y dst respectivamente):
5   // Reconstruir matching: para todo v1 en V1 ver las aristas a vertices
        de V2 con it->f>0, es arista del Matching
6   // Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con
        dist[v]>0
7   // Max Independent Set: tomar los vertices NO tomados por el Min Vertex
        Cover
8   // Max Clique: construir la red de G complemento (debe ser bipartito!) y
        encontrar un Max Indepedet Set
9   // Min Edge Cover: tomar las aristas del matching + para todo vertices
        no cubierto hasta el momento, tomar cualquier arista de el
10  int nodes, src, dst;
11  int dist[MAX], q[MAX], work[MAX];
12  struct Edge {
13      int to, rev;
14      ll f, cap;
15      Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(
```

```
16   cap) {}
     };
17   vector<Edge> G[MAX];
18   void addEdge(int s, int t, ll cap){
19       G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(Edge(s, sz(G[s])-1, 0,
            0));}
20   bool dinic_bfs(){
21       fill(dist, dist+nodes, -1), dist[src]=0;
22       int qt=0; q[qt++]=src;
23       for(int qh=0; qh<qt; qh++){
24           int u =q[qh];
25           forall(e, G[u]){
26               int v=e->to;
27               if(dist[v]<0 && e->f < e->cap)
28                   dist[v]=dist[u]+1, q[qt++]=v;
29           }
30       }
31       return dist[dst]>=0;
32   }
33   ll dinic_dfs(int u, ll f){
34       if(u==dst) return f;
35       for(int &i=work[u]; i<sz(G[u]); i++){
36           Edge &e = G[u][i];
37           if(e.cap<=e.f) continue;
38           int v=e.to;
39           if(dist[v]==dist[u]+1){
40               ll df=dinic_dfs(v, min(f, e.cap-e.f));
41               if(df>0){
42                   e.f+=df, G[v][e.rev].f-= df;
43                   return df;  }
44           }
45       }
46       return 0;
47   }
48   ll maxFlow(int _src, int _dst){
49       src=_src, dst=_dst;
50       ll result=0;
51       while(dinic_bfs()){
52           fill(work, work+nodes, 0);
53           while(ll delta=dinic_dfs(src,INF))
54               result+=delta;
55       }
56       // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1
57       // forman el min-cut
     return result; }
```

8.2.   Konig

```
1    // asume que el dinic YA ESTA tirado
2    // asume que nodes-1 y nodes-2 son la fuente y destino
3    int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v
        no esta matcheado
4    int s[maxnodes]; // numero de la bfs del koning
5    queue<int> kq;
6    // s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
7    void koning() {//O(n)
8      forn(v,nodes-2) s[v] = match[v] = -1;
9      forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
10        { match[v]=it->to; match[it->to]=v;}
11     forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
12     while(!kq.empty()) {
13       int e = kq.front(); kq.pop();
14       if (s[e]%2==1) {
15         s[match[e]] = s[e]+1;
16         kq.push(match[e]);
17       } else {
18
19         forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
20           s[it->to] = s[e]+1;
21           kq.push(it->to);
22         }
23       }
24     }
25   }
```

8.3.   Edmonds Karp's

```
1    #define MAX_V 1000
2    #define INF 1e9
3    //special nodes
4    #define SRC 0
5    #define SNK 1
6    map<int, int> G[MAX_V];//limpiar esto
7    //To add an edge use
8    #define add(a, b, w) G[a][b]=w
9    int f, p[MAX_V];
10   void augment(int v, int minE){
```

```
11    if(v==SRC) f=minE;
12    else if(p[v]!=-1){
13      augment(p[v], min(minE, G[p[v]][v]));
14      G[p[v]][v]-=f, G[v][p[v]]+=f;
15    }
16  }
17  ll maxflow(){//O(VE^2)
18    ll Mf=0;
19    do{
20      f=0;
21      char used[MAX_V]; queue<int> q; q.push(SRC);
22      zero(used), memset(p, -1, sizeof(p));
23      while(sz(q)){
24        int u=q.front(); q.pop();
25        if(u==SNK) break;
26        forall(it, G[u])
27          if(it->snd>0 && !used[it->fst])
28            used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
29      }
30      augment(SNK, INF);
31      Mf+=f;
32    }while(f);
33    return Mf;
34  }
```

8.4.   Push-Relabel O(N3)

```
1   #define MAX_V 1000
2   int N;//valid nodes are [0...N-1]
3   #define INF 1e9
4   //special nodes
5   #define SRC 0
6   #define SNK 1
7   map<int, int> G[MAX_V];
8   //To add an edge use
9   #define add(a, b, w) G[a][b]=w
10  ll excess[MAX_V];
11  int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
12  queue<int> Q;
13  void enqueue(int v) {
14    if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
15  void push(int a, int b) {
16    int amt = min(excess[a], ll(G[a][b]));
17    if(height[a] <= height[b] || amt == 0) return;
18    G[a][b]-=amt, G[b][a]+=amt;
19    excess[b] += amt, excess[a] -= amt;
20    enqueue(b);
21  }
22  void gap(int k) {
23    forn(v, N){
24      if (height[v] < k) continue;
25      count[height[v]]--;
26      height[v] = max(height[v], N+1);
27      count[height[v]]++;
28      enqueue(v);
29    }
30  }
31  void relabel(int v) {
32    count[height[v]]--;
33    height[v] = 2*N;
34    forall(it, G[v])
35      if(it->snd)
36        height[v] = min(height[v], height[it->fst] + 1);
37    count[height[v]]++;
38    enqueue(v);
39  }
40  ll maxflow() {//O(V^3)
41    zero(height), zero(active), zero(count), zero(excess);
42    count[0] = N-1;
43    count[N] = 1;
44    height[SRC] = N;
45    active[SRC] = active[SNK] = true;
46    forall(it, G[SRC]){
47      excess[SRC] += it->snd;
48      push(SRC, it->fst);
49    }
50    while(sz(Q)) {
51      int v = Q.front(); Q.pop();
52      active[v]=false;
53      forall(it, G[v]) push(v, it->fst);
54      if(excess[v] > 0)
55        count[height[v]] == 1? gap(height[v]):relabel(v);
56    }
57    ll mf=0;
58    forall(it, G[SRC]) mf+=G[it->fst][SRC];
59    return mf;
```

```
60 }
```

8.5.   Min-cost Max-flow

```
1  const int MAXN=10000;
2  typedef ll tf;
3  typedef ll tc;
4  const tf INFFLUJO = 1e14;
5  const tc INFCOSTO = 1e14;
6  struct edge {
7    int u, v;
8    tf cap, flow;
9    tc cost;
10   tf rem() { return cap - flow; }
11 };
12 int nodes; //numero de nodos
13 vector<int> G[MAXN]; // limpiar!
14 vector<edge> e;  // limpiar!
15 void addEdge(int u, int v, tf cap, tc cost) {
16   G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
17   G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
18 }
19 tc dist[MAXN], mnCost;
20 int pre[MAXN];
21 tf cap[MAXN], mxFlow;
22 bool in_queue[MAXN];
23 void flow(int s, int t) {
24   zero(in_queue);
25   mxFlow=mnCost=0;
26   while(1){
27     fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
28     memset(pre, -1, sizeof(pre)); pre[s]=0;
29     zero(cap); cap[s] = INFFLUJO;
30     queue<int> q; q.push(s); in_queue[s]=1;
31     while(sz(q)){
32       int u=q.front(); q.pop(); in_queue[u]=0;
33       for(auto it:G[u]) {
34         edge &E = e[it];
35         if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9){ // ojo EPS
36           dist[E.v]=dist[u]+E.cost;
37           pre[E.v] = it;
38           cap[E.v] = min(cap[u], E.rem());
39           if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
40         }
41       }
42     }
43     if (pre[t] == -1) break;
44     mxFlow +=cap[t];
45     mnCost +=cap[t]*dist[t];
46     for (int v = t; v != s; v = e[pre[v]].u) {
47       e[pre[v]].flow += cap[t];
48       e[pre[v]^1].flow -= cap[t];
49     }
50   }
51 }
```

9.   Template

```
1  //touch {a..m}.in; tee {a..m}.cpp < template.cpp
2  #include <bits/stdc++.h>
3  using namespace std;
4  #define forr(i,a,b) for(int i=(a); i<(b); i++)
5  #define forn(i,n) forr(i,0,n)
6  #define sz(c) ((int)c.size())
7  #define zero(v) memset(v, 0, sizeof(v))
8  #define forall(it,v) for(auto it=v.begin();it!=v.end();++it)
9  #define pb push_back
10 #define fst first
11 #define snd second
12 typedef long long ll;
13 typedef pair<int,int> ii;
14 #define dforn(i,n) for(int i=n-1; i>=0; i--)
15 #define dprint(v) cout << #v"=" << v << endl //;)
16
17 const int MAXN=100100;
18 int n;
19
20 int main() {
21   freopen("input.in", "r", stdin);
22   ios::sync_with_stdio(0);
23   while(cin >> n){
24
25   }
26   return 0;
27 }
```

10. Ayudamemoria

Cant. decimales

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

Rellenar con espacios(para justificar)

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

Leer hasta fin de linea

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

Aleatorios

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

Doubles Comp.

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

Limites

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

Muahaha

```cpp
#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

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

Mejorar velocidad 2

```cpp
//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

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

C++11

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

Leer del teclado

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

Iterar subconjunto

```cpp
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
```