Contents

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)

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

```
//inclusive segment tree [L,R]
int T[4 * N];
void init(int node = 1,int l = 0,int r = n - 1){
  if(l == r)T[node] = v[l];
  else{
    int mid = (l + r) >> 1;
    init(2 * node,l,mid);
    init(2 * node + 1,mid + 1,r);
    T[node] = op(T[2 * node],T[2 * node + 1]);
  }
}
void update(int pos,int val,int node = 1,int l = 0,int r = n - 1){
  if(r < pos || l > pos)return;
  if(l == r)T[node] = val;
  else{
    int mid = (l + r) >> 1;
    update(pos,val,2 * node,l,mid);
    update(pos,val,2 * node + 1,mid + 1,r);
    T[node] = op(T[2 * node],T[2 * node + 1]);
```

```
20      }
21  }
22  int query(int x,int y,int node = 1,int l = 0,int r = n - 1){
23      if(r < x || l > y)return NEUTRO;
24      if(x <= l && r <= y)return T[node];
25      else{
26          int mid = (l + r) >> 1;
27          return op(query(x,y,2 * node,l,mid),query(x,y,2 * node + 1,mid + 1,r
                ));
28      }
29  }
```

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

```
1   //Segment tree iterative [l,r)
2   int T[2 * N];
3   void init(){
4       for(int i = n; i < 2 * n;i++)T[i] = val[i];
5       for(int i = n - 1; i >= 1; i--)T[i] = op(T[i << 1],T[i << 1 | 1]);
6   }
7   int op(int a,int b){
8       //an asociative function
9       return a + b;
10  }
11  void update(int pos,int u){
12      pos += n;
13      for(pos >>= 1; pos >= 1; pos >>= 1)T[pos] = op(T[pos << 1],T[pos << 1
            | 1]);
14  }
15
16  int query(int l,int r){
17      l += n, r += n;
18      int ans = NEUTRO;
19      while(l  < r){
20          if(l & 1)ans = op(ans,T[l++]);
21          if(r & 1)ans = op(ans,T[--r]);
22          l >>= 1,r >>= 1;
23      }
24      return ans;
25  }
```

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

```
1   /*Segment Tree modificar un rango, acceder a una posicion
```

```
2       solo sirve cuando la operacion que realizamos es conmutativa
3       por ejemplo la suma, pero no funciona con la asignacion
4   */
5   //adiciona value al rango [l, r)
6   void modify(int l, int r, int value) {// rango [l, r)
7       for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
8           if (l&1) t[l++] += value;
9           if (r&1) t[--r] += value;
10      }
11  }
12  //acceder a la posicion
13  int query(int p) {
14      int res = 0;
15      for (p += n; p > 0; p >>= 1) res += t[p];
16      return res;
17  }
18  //Si necesitamos actualizar todo lo podemos hacer en O(n)
19  //Y luego acceder a las hojas en O(1)
20  void push() {
21      for (int i = 1; i < n; ++i) {
22          t[i<<1] += t[i];
23          t[i<<1|1] += t[i];
24          t[i] = 0;
25      }
26  }
```

2.2.4   Segment Tree con Punteros

```
1   /*La creacion y las queries son [0, n)
2   [cerrado, abierto)
3   Por alguna razon el destructor hace que se borre todo
4   inmediatamente despues de que termina el constructor*/
5   const int maxn = 1000000;
6   tipo v[maxn];
7
8   const tipo NEUTRO = PONGA_AQUI_EL_NEUTRO;
9   tipo oper(tipo a, tipo b) {
10      return min(a, b);
11  }
12  struct segment_tree {
13      segment_tree *L, *R;
14      int l, r;
15      tipo value;
```

```cpp
segment_tree(): L(nullptr), R(nullptr), value(NEUTRO) {}
segment_tree(int _l, int _r) : l(_l), r(_r) {
  if (l + 1 == r) {
    value = v[l];
  } else {
    int mid = (l + r) >> 1;
    L = new segment_tree(l, mid);
    R = new segment_tree(mid, r);
    value = oper(L->value, R->value);
  }
}
//~segment_tree() {delete L; delete R;}//NO PONER!!! ERROR!!!
void update(int pos, int val) {
  if (r <= pos || l > pos) return;
  if (l + 1 == r) {
    v[pos] = value = val;
  } else {
    L->update(pos, val);
    R->update(pos, val);
    value = oper(L->value, R->value);
  }
}
tipo query(int a, int b) {
  if (a <= l && r <= b) return value;
  if (l >= b || r <= a) return NEUTRO;
  return oper(L->query(a, b), R->query(a, b));
}
} tree;
```

2.2.5 Segment Tree 2D

```cpp
typedef long long ll;
struct segmetree{
  int n;
  vector<ll>T;
  segmetree(){n = 0;}
  segmetree(int _){
    n = _;
    T.resize(2 * n + 1);
  }
  void rupdate(int pos,int value){
    pos += n;
    T[pos] = value;
```

```cpp
    for(pos >>= 1; pos >= 1; pos >>= 1)
      T[pos] = T[pos << 1] + T[pos << 1 | 1];
  }
  void update(int pos,int value){
    pos += n;
    T[pos] += value;
    for(pos >>= 1; pos >= 1; pos >>= 1)
      T[pos] = T[pos << 1] + T[pos << 1 | 1];
  }
  int query(int l,int r){
    l += n;r+= n;
    int ans = 0;
    while(l < r){
      if(l & 1)ans += T[l++];
      if(r & 1)ans += T[--r];
      l >>= 1, r >>= 1;
    }
    return ans;
  }
};
struct st{
  int n;
  vector<segmetree>T;
  st(){}
  st(int _){
    n = _;
    for(int i = 0;i < 2 * n;i++){
      T.push_back(segmetree(n));
    }
  }
  void update(int x,int y,int val){
    x += n;
    T[x].update(y,val);
    segmetree ok;
    for(x >>= 1; x >= 1; x >>= 1){
      T[x].rupdate(y,T[x << 1].query(y,y + 1));
      T[x].update(y,T[x << 1 | 1].query(y,y + 1));
    }
  }
  ll query(int l,int b,int r,int t){
    l += n;
    r += n;
    r++,t++;
```

```
56    ll ans = 0LL;
57    while(l < r){
58      if(l & 1)ans += T[l++].query(b,t);
59      if(r & 1)ans += T[--r].query(b,t);
60      l >>= 1, r >>= 1;
61    }
62    return ans;
63  }
64 };
```

2.2.6   Segment Tree Lazy - Suma

```
1  //Todo es [l, r)
2  void build(int id = 1,int l = 0,int r = n){
3    if(r - l < 2){  //  l + 1 == r
4      s[id] = a[l];
5      return ;
6    }
7    int mid = (l+r)/2;
8    build(id * 2, l, mid);
9    build(id * 2 + 1, mid, r);
10    s[id] = s[id * 2] + s[id * 2 + 1];
11 }
12 //aqui poner los arrays lazy and id
13 void upd(int id,int l,int r,int x){// increase all members in this
        interval by x
14    lazy[id] += x;
15    s[id] += (r - l) * x;
16 }
17 //A function to pass the update information to its children :
18 void shift(int id,int l,int r){//pass update information to the children
19    int mid = (l+r)/2;
20    upd(id * 2, l, mid, lazy[id]);
21    upd(id * 2 + 1, mid, r, lazy[id]);
22    lazy[id] = 0;// passing is done
23 }
24 //A function to perform increase queries :
25 void increase(int x,int y,int v,int id = 1,int l = 0,int r = n){
26    if(x >= r or l >= y)  return ;
27    if(x <= l && r <= y){
28      upd(id, l, r, v);
29      return ;
30    }
```

```
31    shift(id, l, r);
32    int mid = (l+r)/2;
33    increase(x, y, v, id * 2, l, mid);
34    increase(x, y, v, id*2+1, mid, r);
35    s[id] = s[id * 2] + s[id * 2 + 1];
36 }
37 //(We should call increase(l r x))
38 int sum(int x,int y,int id = 1,int l = 0,int r = n){
39    if(x >= r or l >= y)  return 0;
40    if(x <= l && r <= y)  return s[id];
41    shift(id, l, r);
42    int mid = (l+r)/2;
43    return sum(x, y, id * 2, l, mid) +
44          sum(x, y, id * 2 + 1, mid, r);
45 }
```

2.2.7   Segment Tree Lazy - Pintar

```
1  void shift(int id){
2    if(lazy[id])
3      lazy[2 * is] = lazy[2 * id + 1] = lazy[id];
4    lazy[id] = 0;
5  }
6  //color > 1, por que se usa el 0 para decir que no hay lazy
7  void upd(int x,int y,int color, int id = 0,int l = 0,int r = n){//
        painting the interval [x,y) whith color "color"
8    if(x >= r or l >= y)  return ;
9    if(x <= l && r <= y){
10      lazy[id] = color;
11      return ;
12    }
13    int mid = (l+r)/2;
14    shift(id);
15    upd(x, y, color, 2 * id, l, mid);
16    upd(x, y, color, 2*id+1, mid, r);
17 }
18
19 set <int> se;
20 void cnt(int id = 1,int l = 0,int r = n){
21    if(lazy[id]){
22      se.insert(lazy[id]);
23      return ; // there is no need to see the children, because all the
            interval is from the same color
```

```
24      }
25      if(r - l < 2) return ;
26      int mid = (l+r)/2;
27      cnt(2 * id, l, mid);
28      cnt(2*id+1, mid, r);
29  }
```

2.2.8   Segment Tree Persistente

```
1   int segcnt = 0;
2   struct segment {
3       int l, r, lid, rid, sum;
4   } segs[2000000];
5   int build(int l, int r) {
6       if (l > r) return -1;
7       int id = segcnt++;
8       segs[id].l = l;
9       segs[id].r = r;
10      if (l == r) segs[id].lid = -1, segs[id].rid = -1;
11      else {
12          int m = (l + r) / 2;
13          segs[id].lid = build(l , m);
14          segs[id].rid = build(m + 1, r); }
15      segs[id].sum = 0;
16      return id; }
17  int update(int idx, int v, int id) {
18      if (id == -1) return -1;
19      if (idx < segs[id].l || idx > segs[id].r) return id;
20      int nid = segcnt++;
21      segs[nid].l = segs[id].l;
22      segs[nid].r = segs[id].r;
23      segs[nid].lid = update(idx, v, segs[id].lid);
24      segs[nid].rid = update(idx, v, segs[id].rid);
25      segs[nid].sum = segs[id].sum + v;
26      return nid; }
27  int query(int id, int l, int r) {
28      if (r < segs[id].l || segs[id].r < l) return 0;
29      if (l <= segs[id].l && segs[id].r <= r) return segs[id].sum;
30      return query(segs[id].lid, l, r) + query(segs[id].rid, l, r); }
```

2.3   Fenwick Tree

```
1   vector<tipo> tree;
2   int maxn;
```

```
3   void init(int n) {
4       tree = vector<tipo>(n, 0);
5       maxn = n;
6   }
7   void add(int i, tipo k) { //i valid [1, n)
8       for(; i < maxn; i += i&-i ) tree[i] += k;
9   }
10
11  tipo get(int i){//returns sum [1, i]
12      tipo s = 0;
13      for(; i > 0; i-=i&-i) s+=tree[i];
14      return s;
15  }
```

2.3.1   Fenwick Tree 2D

```
1
2   ll T[1025][1025];
3   int n;
4
5   ll query(int x, int y)
6   {
7       ll res = 0;
8       for(int i = x; i >= 0; i = (i & (i+1)) - 1)
9           for(int j = y; j >= 0; j = (j & (j+1)) - 1)
10              res += T[i][j];
11      return res;
12  }
13
14  void update(int x, int y, int val)
15  {
16      for(int i = x; i < n; i = (i | (i+1)))
17          for(int j = y; j < n; j = (j | (j+1)))
18              T[i][j] += val;
19  }
```

2.4   Union Find con rank

```
1   /*=============== <Union find rangos> ==================
2   Complexity: O(N)
3   index 0 to n - 1 warning
4   Complexity O(N)
5   */
6   #define MAX INSERTE_VALOR_AQUI
```

```
7   int padre[MAX];
8   int rango[MAX];
9   void MakeSet(int n){
10      for (int i = 0 ; i < n ; ++i) {
11          padre[i] = i; rango[i] = 0; }
12  }
13  int Find(int x) {
14      if(x == padre[x])
15          return x;
16      return padre[x] = Find(padre[x]);
17  }
18  void UnionbyRank(int x , int y){
19      int xRoot = Find(x);
20      int yRoot = Find(y);
21      //el padre de ambas componentes sera el de mayor altura
22      if(rango[xRoot] > rango[yRoot])//X tiene mas altura que Y
23          padre[yRoot] = xRoot;
24      }else{//Y >= X
25          padre[xRoot] = yRoot;
26          if(rango[xRoot] == rango[yRoot])//si poseen la misma altura
27              rango[yRoot]++;//incremento el rango de la nueva raiz
28      }
29  }
```

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   Problem for practice: UVA 494
8   */
9   // base and base_digits must be consistent
10  const int base = 1000000000;
11  const int base_digits = 9;
12
13  struct bigint {
14      vector<int> a;
15      int sign;
16
```

```
17      bigint() :
18          sign(1) {
19      }
20
21      bigint(long long v) {
22          *this = v;
23      }
24
25      bigint(const string &s) {
26          read(s);
27      }
28
29      void operator=(const bigint &v) {
30          sign = v.sign;
31          a = v.a;
32      }
33
34      void operator=(long long v) {
35          sign = 1;
36          if (v < 0)
37              sign = -1, v = -v;
38          for (; v > 0; v = v / base)
39              a.push_back(v % base);
40      }
41
42      bigint operator+(const bigint &v) const {
43          if (sign == v.sign) {
44              bigint res = v;
45
46              for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size
                        ()) || carry; ++i) {
47                  if (i == (int) res.a.size())
48                      res.a.push_back(0);
49                  res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
50                  carry = res.a[i] >= base;
51                  if (carry)
52                      res.a[i] -= base;
53              }
54              return res;
55          }
56          return *this - (-v);
57      }
58
```

```cpp
    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();
```

```cpp
140        }
141
142        bigint operator/(int v) const {
143            bigint res = *this;
144            res /= v;
145            return res;
146        }
147
148        int operator%(int v) const {
149            if (v < 0)
150                v = -v;
151            int m = 0;
152            for (int i = a.size() - 1; i >= 0; --i)
153                m = (a[i] + m * (long long) base) % v;
154            return m * sign;
155        }
156
157        void operator+=(const bigint &v) {
158            *this = *this + v;
159        }
160        void operator-=(const bigint &v) {
161            *this = *this - v;
162        }
163        void operator*=(const bigint &v) {
164            *this = *this * v;
165        }
166        void operator/=(const bigint &v) {
167            *this = *this / v;
168        }
169
170        bool operator<(const bigint &v) const {
171            if (sign != v.sign)
172                return sign < v.sign;
173            if (a.size() != v.a.size())
174                return a.size() * sign < v.a.size() * v.sign;
175            for (int i = a.size() - 1; i >= 0; i--)
176                if (a[i] != v.a[i])
177                    return a[i] * sign < v.a[i] * sign;
178            return false;
179        }
180
181        bool operator>(const bigint &v) const {
182            return v < *this;
183        }
184        bool operator<=(const bigint &v) const {
185            return !(v < *this);
186        }
187        bool operator>=(const bigint &v) const {
188            return !(*this < v);
189        }
190        bool operator==(const bigint &v) const {
191            return !(*this < v) && !(v < *this);
192        }
193        bool operator!=(const bigint &v) const {
194            return *this < v || v < *this;
195        }
196
197        void trim() {
198            while (!a.empty() && !a.back())
199                a.pop_back();
200            if (a.empty())
201                sign = 1;
202        }
203
204        bool isZero() const {
205            return a.empty() || (a.size() == 1 && !a[0]);
206        }
207
208        bigint operator-() const {
209            bigint res = *this;
210            res.sign = -sign;
211            return res;
212        }
213
214        bigint abs() const {
215            bigint res = *this;
216            res.sign *= res.sign;
217            return res;
218        }
219
220        long long longValue() const {
221            long long res = 0;
222            for (int i = a.size() - 1; i >= 0; i--)
223                res = res * base + a[i];
224            return res * sign;
225        }
```

```cpp
        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;
        }
```

```cpp
        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);
```

```
310        vll a2b2 = karatsubaMultiply(a2, b2);
311
312        for (int i = 0; i < k; i++)
313            a2[i] += a1[i];
314        for (int i = 0; i < k; i++)
315            b2[i] += b1[i];
316
317        vll r = karatsubaMultiply(a2, b2);
318        for (int i = 0; i < (int) a1b1.size(); i++)
319            r[i] -= a1b1[i];
320        for (int i = 0; i < (int) a2b2.size(); i++)
321            r[i] -= a2b2[i];
322
323        for (int i = 0; i < (int) r.size(); i++)
324            res[i + k] += r[i];
325        for (int i = 0; i < (int) a1b1.size(); i++)
326            res[i] += a1b1[i];
327        for (int i = 0; i < (int) a2b2.size(); i++)
328            res[i + n] += a2b2[i];
329        return res;
330    }
331
332    bigint operator*(const bigint &v) const {
333        vector<int> a6 = convert_base(this->a, base_digits, 6);
334        vector<int> b6 = convert_base(v.a, base_digits, 6);
335        vll a(a6.begin(), a6.end());
336        vll b(b6.begin(), b6.end());
337        while (a.size() < b.size())
338            a.push_back(0);
339        while (b.size() < a.size())
340            b.push_back(0);
341        while (a.size() & (a.size() - 1))
342            a.push_back(0), b.push_back(0);
343        vll c = karatsubaMultiply(a, b);
344        bigint res;
345        res.sign = sign * v.sign;
346        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
347            long long cur = c[i] + carry;
348            res.a.push_back((int) (cur % 1000000));
349            carry = (int) (cur / 1000000);
350        }
351        res.a = convert_base(res.a, 6, base_digits);
352        res.trim();
```

```
353        return res;
354    }
355 };
356
357 int main() {
358     bigint a=0;
359     //("9999999999999999999999999999999999999999999");
360     bigint b;
361     //("1999999999999999999999999999999999999999998");
362     bigint n;
363     while(cin >> n) {
364         if(n==0){break;}
365         a += n;
366     }
367     cout<<a<<endl;
368 }
```

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 #include <ext/pb_ds/assoc_container.hpp>
7 #include <ext/pb_ds/tree_policy.hpp>
```

```
8   using namespace __gnu_pbds;
9   /*typedef tree<int,null_type,less<int>,rb_tree_tag,
        tree_order_statistics_node_update> ordered_set;
10  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().
11  The first returns an iterator to the k-th largest element (counting from
        zero), the second - the number of items
12  in a set that are strictly smaller than our item. Example of use:*/
13  template <typename T>
14  using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
        tree_order_statistics_node_update>;
15  int main(){
16    ordered_set<int>  s;
17    s.insert(1);
18    s.insert(3);
19    cout << *s.find_by_order(0) << endl; // print the 0-th smallest number
          in s(0-based)
20    cout << s.order_of_key(2) << endl; // the number of elements in the s
        less than 2
21    //find_by_order(i) devuelve iterador al i-esimo elemento
22    //order_of_key(k): devuelve la pos del lower bound de k
23    //Ej: 12, 100, 505, 1000, 10000.
24    //order_of_key(10) == 0, order_of_key(100) == 1,
25    //order_of_key(707) == 3, order_of_key(9999999) == 5
26    return 0;
27  }
```

2.8   Treap Modo Set

```
1   struct treap {
2     typedef struct _node {
3       int x, y, cnt;
4       _node *l, *r;
5       _node(int _x) : x(_x), y((rand() << 16) ^ rand()), cnt(1), l(nullptr
            ), r(nullptr) {}
6       ~_node() {delete l; delete r;}
7       void recalc() {
8         cnt = 1;
9         if (l) cnt += l->cnt;
10        if (r) cnt += r->cnt;
```

```
11      }
12    } *node;
13    treap(): root(nullptr) {}
14    ~treap() {delete root;}
15    node root;
16    /*Divide el arbol que tiene como raiz a "t", guarda en
        L todos los nodos con key menor o igual a "x"
        R todos los nodos con key mayor estricto a x
        t es destruido/modificado
      */
21    void split(node t, int x, node &L, node &R) {
22      if (t == nullptr) {L = R = nullptr; return;}
23      if (t->x <= x) {
24        split(t->r, x, t->r, R);
25        L = t;
26      } else {
27        split(t->l, x, L, t->l);
28        R = t;
29      }
30      t->recalc();
31    }
32    /*Une los dos nodos L y R en un solo arbol y los devuelve.
        L y R son modificados
      */
35    node merge(node L, node R) {
36      if (L == nullptr) return R;
37      if (R == nullptr) return L;
38      if (L->y > R->y) {
39        L->r = merge(L->r, R);
40        L->recalc();
41        return L;
42      } else {
43        R->l = merge(L, R->l);
44        R->recalc();
45        return R;
46      }
47    }
48    /*Inserta un solo nodo con key igual a "x"*/
49    void insert(int x) {
50      //verificar que no se inserten elementos repetidos
51      //pueden ocasionar que ya no cumplan la propiedad de BST
52      //pueden haber keys iguales a izquierda y derecha
53      node L, R;
```

```
54      split(root, x, L, R);
55      root = merge(merge(L, new _node(x)), R);
56    }
57    /*Borra todos los nodos con key igual a "x"
58      No pasa nada si no hay nodos con key igual a "x"*/
59    void erase(int x) {
60      node L, m, R;
61      split(root, x, L, R);
62      split(L, x - 1, L, m);
63      root = merge(L, R);
64    }
65    int count(int x) {
66      node L, m, R;
67      split(root, x, L, R);
68      split(L, x - 1, L, m);
69      int ans = cnt(m);
70      root = merge(merge(L, m), R);
71      return ans;
72    }
73    /*Borra un nodo con key igual a "x"
74      No sucede nada si no hay nodo con ese key*/
75    void erase_one(int x) {
76      node L, m, R;
77      split(root, x, L, R);
78      split(L, x - 1, L, m);
79      if (m)
80        m = merge(m->l, m->r);
81      root = merge(merge(L, m), R);
82    }
83    /*k-esimo indexado desde 0
84       devuelve INT_MAX si no hay k-esimo*/
85    int kthElement(int k) {
86      node cur = root;
87      while (cur != nullptr) {
88        int sizeLeft = cnt(cur->l);
89        if (sizeLeft == k)
90          return cur->x;
91        cur = sizeLeft > k ? cur->l : cur->r;
92        if (sizeLeft < k)
93          k -= (sizeLeft + 1);
94      }
95      return INT_MAX;
96    }
97    /*Numero de elementos con keys menores a "x"*/
98    int less(int x) {
99      node cur = root;
100     int ans = 0;
101     while (cur != nullptr) {
102       if (cur->x >= x) {
103         cur = cur->l;
104       } else {
105         ans += cnt(cur->l) + 1;
106         cur = cur->r;
107       }
108     }
109     return ans;
110   }
111   int cnt(node t) const {
112     return t ? t->cnt : 0;
113   }
114   int size() const {
115     return cnt(root);
116   }
117 /*//from e-maxx.ru son mas rapidos(no x mucho) pero menos entendibles
118 //solo borra un elemento
119   void erase(node &t, int x) {
120     if (t->x == x)
121       t = merge(t->l, t->r);
122     else
123       erase(x < t->x?t->l:t->r, x);
124     if (t)
125       t->recalc();
126   }
127   void erase_one(int x) {erase(root, x);}
128   void insert(node &t, node it) {
129     if (t == nullptr)
130       t = it;
131     else if (it->y > t->y)
132       split(t, it->x, it->l, it->r), t = it;
133     else
134       insert(it->x < t->x? t->l: t->r, it);
135     t->recalc();
136   }
137   void insert(int x) {insert(root, new _node(x));}*/
138 };
```

2.9   Treap Implicito(Rope)

```
1  struct rope {
2    typedef struct _node {
3      int value, y, cnt;
4      _node *l, *r;
5      _node(int _value) : value(_value), y((rand() << 16) ^ rand()), cnt
             (1), l(nullptr), r(nullptr) {}
6      ~_node() {delete l; delete r;}
7      void recalc() {
8        cnt = 1;
9        if (l) cnt += l->cnt;
10       if (r) cnt += r->cnt;
11     }
12   } *node;
13   rope(): root(nullptr) {}
14   ~rope() {delete root;}
15   node root;
16   /*Divide el arbol que tiene como raiz a "t", guarda en
17    L los primeros "x" elementos del array
18    R el primer elemento de R es el elemento en posicion x(indexado desde
            0) del array
19    t es destruido/modificado
20    L = [0, x)
21    R = [x, n) */
22   void split(node t, int x, node &L, node &R) {
23     if (t == nullptr) {L = R = nullptr; return;}
24     int curIndex = cnt(t->l) + 1;
25     if (curIndex <= x) {
26       split(t->r, x - curIndex, t->r, R);
27       L = t;
28     } else {
29       split(t->l, x, L, t->l);
30       R = t;
31     }
32     t->recalc();
33   }
34   /* Une los dos nodos L y R en un solo arbol y los devuelve.
35      L y R son modificados */
36   node merge(node L, node R) {
37     if (L == nullptr) return R;
38     if (R == nullptr) return L;
39     if (L->y > R->y) {
40       L->r = merge(L->r, R);
41       L->recalc();
42       return L;
43     } else {
44       R->l = merge(L, R->l);
45       R->recalc();
46       return R;
47     }
48   }
49   /*Inserta "value" en la posicion "pos"(indexado desde 0) recorre todos
          los elementos a la derecha desde la posicion pos*/
50   void insert(int pos, int value) {
51     node L, R;
52     split(root, pos, L, R);//en R esta pos
53     root = merge(merge(L, new _node(value)), R);
54   }
55   /*Borra el elemento en posicion pos*/
56   void erase(int pos) {
57     node L, m, R;
58     split(root, pos, L, R);
59     split(R, 1, m, R);
60     root = merge(L, R);
61   }
62   int cnt(node t) const {
63     return t ? t->cnt : 0;
64   }
65   int size() const {
66     return cnt(root);
67   }
68 };
69 int main() {srand(time(nullptr));return 0;}
```

2.10   Treap - Toby and Stones

```
1  const int PAINT = 0, FLIP = 1, REVERSE = 2;
2  struct rope {
3    typedef struct _node {
4      int value, y, cnt;
5      int negros;
6      bool rev;
7      bool lazy_flip;
8      int lazy_pintar;
9      _node *l, *r;
```

```
10      _node(int _value) : value(_value), y((rand() << 16) ^ rand()), cnt
            (1), l(nullptr), r(nullptr) {
11        negros = _value;
12        rev = false;
13        lazy_flip = false;
14        lazy_pintar = -1;
15      }
16      ~_node() {delete l; delete r;}
17      void recalc() {
18        cnt = 1;
19        negros = value;
20        if (l) cnt += l->cnt, negros += l->negros;
21        if (r) cnt += r->cnt, negros += r->negros;
22      }
23      void push_lazy(int type, int param = -1) {
24        if (type == PAINT) {
25          rev = false;
26          lazy_flip = false;
27          lazy_pintar = param;
28          negros = param * cnt;
29        }
30        if (type == FLIP) {
31          if (lazy_pintar != -1) {
32            assert(rev == false && lazy_flip == false);
33            lazy_pintar = (1 - lazy_pintar);
34            negros = lazy_pintar * cnt;
35          } else {
36            lazy_flip ^= true;
37            negros = cnt - negros;
38          }
39        }
40        if (type == REVERSE) {
41          if (lazy_pintar == -1) {
42            rev ^= true;
43          }
44        }
45      }
46      void verify() {
47        bool op1 = (lazy_pintar != -1);
48        bool op2 = (rev || lazy_flip);
49        //solo puede pasar uno de los 2 casos, o ninguno esta activado o
              solo uno de los dos
50        assert((!op1 and !op2) or (op1 xor op2));
51      }
52      string to_string() {
53        stringstream ss;
54        ss << "value=" << value << ",_negros=" << negros << ",_cnt="<< cnt
              << "("<<rev<< "," << lazy_pintar<<","<< lazy_flip << ")";
55        return ss.str();
56      }
57    } *node;
58    rope(): root(nullptr) {}
59    ~rope() {delete root;}
60    node root;
61    void push(node &t) {
62      if (t == nullptr) return;
63      bool op1 = ((t->lazy_pintar) != -1);
64      bool op2 = ((t->rev) || (t->lazy_flip));
65      //solo puede pasar uno de los 2 casos, o ninguno esta activado o
            solo uno de los dos
66      assert((!op1 and !op2) or (op1 xor op2));
67      if (op1) {
68        t->value = t->lazy_pintar;
69        if (t->l) t->l->push_lazy(PAINT, t->lazy_pintar);
70        if (t->r) t->r->push_lazy(PAINT, t->lazy_pintar);
71        t->lazy_pintar = -1;
72      }
73      if (op2) {//no importa el orden en que se aplique estos 2
74        //reverse
75        if (t->rev) {
76          swap(t->l, t->r);
77          t->rev = false;
78          if (t->l) t->l->push_lazy(REVERSE);
79          if (t->r) t->r->push_lazy(REVERSE);
80        }
81        //invertir colores
82        if (t->lazy_flip) {
83          t->value = 1 - (t->value);
84          t->lazy_flip = false;
85          if (t->l) t->l->push_lazy(FLIP);
86          if (t->r) t->r->push_lazy(FLIP);
87        }
88      }
89    }
90    void split(node t, int x, node &L, node &R) {
91      push(t);
```

```
 92     if (t == nullptr) {L = R = nullptr; return;}
 93     int curIndex = cnt(t->l) + 1;
 94     if (curIndex <= x) {
 95       split(t->r, x - curIndex, t->r, R);
 96       L = t;
 97     } else {
 98       split(t->l, x, L, t->l);
 99       R = t;
100     }
101     t->recalc();
102   }
103   node merge(node L, node R) {
104     push(L); push(R);
105     if (L == nullptr) return R;
106     if (R == nullptr) return L;
107     if (L->y > R->y) {
108       L->r = merge(L->r, R);
109       L->recalc();
110       return L;
111     } else {
112       R->l = merge(L, R->l);
113       R->recalc();
114       return R;
115     }
116   }
117   void insert(int pos, int value) {
118     node L, R;
119     split(root, pos, L, R);//en R esta pos
120     root = merge(merge(L, new _node(value)), R);
121   }
122   int cnt(node t) {
123     return t ? t->cnt : 0;
124   }
125   int size() {
126     return cnt(root);
127   }
128   void reverse(int i, int j) {
129     node L, m, R;
130     split(root, i, L, R);
131     split(R, j - i + 1, m, R);
132     m->push_lazy(REVERSE);
133     root = merge(merge(L, m), R);
134   }
```

```
135   void flip(int i, int j) {
136     node L, m, R;
137     split(root, i, L, R);
138     split(R, j - i + 1, m, R);
139     m->push_lazy(FLIP);
140     root = merge(merge(L, m), R);
141   }
142   void pintar(int i, int j, int color) {
143     node L, m, R;
144     split(root, i, L, R);
145     split(R, j - i + 1, m, R);
146     m->push_lazy(PAINT, color);
147     root = merge(merge(L, m), R);
148   }
149   void query(int i, int j) {
150     node L, m, R;
151     split(root, i, L, R);
152     split(R, j - i + 1, m, R);
153     int negros = m->negros;
154     int blancos = m->cnt - negros;
155     printf("%d %d\n", negros, blancos);
156     root = merge(merge(L, m), R);
157   }
158   void print(node &t, string spacio, char lado = ' ') {
159     if (t == nullptr) return;
160     cout << spacio << lado <<" -> " << (t->to_string()) << endl;
161     print(t->l, spacio + "\t", 'L');
162     print(t->r, spacio + "\t", 'R');
163   }
164   void print() {
165     print(root, "");
166   }
167 };
```

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
```

```cpp
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);
```

```
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
         antes de usarlos)
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   SQRT Decomposition Basic

```
1    const int maxn = 500010;
2    int n;
3
4    tipo v[maxn];//vector principal
5
6    tipo lazy[maxn];
7    pair<tipo, tipo> t[maxn];//para poder reordenar los elementos
8
9    int SQRT;
10   int N;//nro. de buckets
11
12   //Recalcula y aplica el lazy al bucket con indice idx
13   //guarda la informacion necesaria del bucket en otros vectores
14   //podria ser la suma del bucket, o el min/max del bucket
15   void recalc(int idx) {
16     int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
17     for (int i = a; i < b; i++) {
18       v[i] += lazy[idx];
19       t[i] = make_pair(v[i], i);
20     }
21     lazy[idx] = 0;
22     sort(t + a, t + b);
23   }
24
25   //adiciona delta a todos los elementos
26   //en el intervalo cerrado [a, b]
27   void add(int a, int b, tipo delta) {
28     int idx_a = a / SQRT, idx_b = b / SQRT;
29     if (idx_a == idx_b) {
30       for (int i = a; i <= b; i++)
31         v[i] += delta;
32       recalc(idx_a);
33     } else {
34       //head
35       for (int i = a, lim = min(n, (idx_a + 1) * SQRT); i < lim; i++)
36         v[i] += delta;
37       recalc(idx_a);//OJO puede ser necesario
38       //body
39       for (int i = idx_a + 1; i < idx_b; i++)
40         lazy[i] += delta;
41       //tail
42       for (int i = idx_b * SQRT; i <= b; i++)
43         v[i] += delta;
44       recalc(idx_b);//OJO puede ser necesario
45     }
46   }
47
48   //tambien podria ser en un rango como en el add
49   tipo query(tipo val) {
50     tipo ans = 0;
51     //recorro todos los buckets
52     for (int idx = 0; idx < N; idx++) {
53       int a = idx * SQRT, b = min(n, (idx + 1) * SQRT);
54       //... hacer algo ...
55     }
56     return ans;
57   }
58   int main() {
```

```
59    //leer n, q y los elementos de v
60
61    SQRT = (int)sqrt(n) + 1;
62    N = (n + SQRT - 1) / SQRT;//nro. de buckets
63    //construir cada bucket
64    for (int idx = 0; idx < N; idx++)
65      recalc(idx);
66
67    //resto del programa
68    return 0;
69  }
```

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

```
1   //insersion y consulta de cuantos <= en log n
2   struct leqset {
3     int maxl; vector<int> c;
4     int pref(int n, int l) { return (n>>(maxl-l))|(1<<l); }
5     void ini(int ml) { maxl=ml; c=vector<int>(1<<(maxl+1)); }
6     //inserta c copias de e, si c es negativo saca c copias
7     void insert(int e, int q=1) { forn(l,maxl+1) c[pref(e,l)]+=q; }
8     int leq(int e) {
9       int r=0,a=1;
10      forn(i,maxl) {
11        a<<=1; int b=(e>>maxl-i-1)&1;
12        if (b) r+=c[a]; a|=b;
13      } return r + c[a]; //sin el c[a] da los estrictamente menores
14    }
15    int size() { return c[1]; }
16    int count(int e) { return c[e|(1<<maxl)]; }
17  };
```

3   Algos

3.1   LIS en O(n log n) con Reconstruccion

```
1   //Para non-increasing, cambiar comparaciones y revisar busq binaria
2   //Given an array, paint it in the least number of colors so that each
        color turns to a non-increasing subsequence.
3   //Solution:Min number of colors=Length of the longest increasing
        subsequence
4
5   // Las lineas marcadas con // Camino no son necesarias si no se desea
        reconstruir el camino.
```

```
6   #define MAXN 1000000
7   int v[MAXN]; // INPUT del algoritmo.
8   int mv[MAXN];
9   int mi[MAXN] ,p[MAXN]; // Camino
10  int l[MAXN]; // Aca apareceria la maxima subsecuencia creciente(los
        indices)
11  int lis(int n) {
12    forn(i,n) mv[i] = INF;
13    forn(i,n) mi[i] = -1; // Camino
14    forn(i,n) p [i] = -1; // Camino
15    mv[0] = -INF;
16    int res = 0;
17    forn(i,n) {
18      // Con upper_bound es maxima subsecuencia no decreciente.
19      // Con lower_bound es maxima subsecuencia creciente.
20      int me = upper_bound(mv,mv+n,v[i]) - mv;
21      p[i] = mi[me-1]; // Camino
22      mv[me] = v[i];
23      mi[me] = i; // Camino
24      if (me > res) res = me;
25    }
26    for(int a = mi[res], i = res - 1;a != -1; a = p[a], i--) // Camino
27      l[i] = a; // Indices: poniendo l[i] = v[a] quedan los valores.
28    return res;
29  }
```

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;
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       }
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

```
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

```
1  vector<int> FindOccurrences(const string& pattern, const string& text) {
2    vector<int> result;
3    int minIndex = 0,maxIndex = text.size();
4    while(minIndex < maxIndex){
5      int mid = (minIndex + maxIndex) >> 1;
6      if(cmp(pattern,sa[mid]) > 0)minIndex = mid + 1;
7      else maxIndex = mid;
8    }
9    int start = minIndex;
10   maxIndex = text.size();
11   while(minIndex < maxIndex){
12     int mid = (minIndex + maxIndex) >> 1;
13     if(cmp(pattern,sa[mid]) < 0)maxIndex = mid;
14     else minIndex = mid + 1;
15   }
16   int end = maxIndex;
17   for(int i = start;  i < end;i++){
18     result.push_back(sa[i]);
19   }
20   return result;
21 }
```

4.5.4   Matching con BWT

```
1  map<char,int>fo;//first ocurrence
2  map<char,vector<int> >count;//count the i-th ocurrence of symbol
3  string first;//first colum of bwt
4  string alpha = "ACGT$";//change this
5  void preprocess(const string& bwt) {//recieves a BWT
6    string ans = "";
7    first = bwt;
8    sort(first.begin(),first.end());
9    for(int i = 0;first[i];i++){
10     if(!fo.count(first[i]))fo[first[i]] = i;
11   }
12   for(char u:alpha)count[u].push_back(0);
13   for(int i = 1; i <= bwt.size();i++){
14     for(char u:alpha)
15       count[u].push_back(count[u].back() + (bwt[i - 1] == u));
16   }
17 }
18 //return the number of ocurrences of the pattern
19 int bwtmatch(int bot,string &pattern){
20   int top = 0;
21   while(top <= bot){
22     if(pattern.size()){
23       char letter = pattern.back();
24       pattern.pop_back();
25       if(count[letter][bot + 1]){
26         top = fo[letter] + count[letter][top];
27         bot = fo[letter] + count[letter][bot + 1] - 1;
28       }
29       else return 0;
30     }
31     else return bot - top + 1;
32   }
33   return 0;
34 }
```

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

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

Teorema de Pick: (Area, puntos interiores y puntos en el borde)
$$A = I + \frac{B}{2} - 1$$

6.2   Ec. Caracteristica

$$a_0 T(n) + a_1 T(n-1) + ... + a_k T(n-k) = 0$$
$$p(x) = a_0 x^k + a_1 x^{k-1} + ... + a_k$$
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)

```
1  tipo mod;
2  struct M22 {          //|a b|
3    tipo a, b, c, d; //|c d|
4    M22 operator * (const M22 &p) const {
5      return (M22){(a*p.a+b*p.c) % mod, (a*p.b+b*p.d) % mod, (c*p.a+d*p.c)
             % mod, (c*p.b+d*p.d) % mod};
6    }
7  };
8  M22 operator ^ (const M22 &p, tipo n) {
9    if(!n) return (M22){1, 0, 0, 1};//matriz identidad
10   M22 q = p ^ (n/2); q = q * q;
11   return (n % 2)? p * q : q;
12 }
13 //devuelve el n-esimo fibonacci (index 0)
14 //f0 = fibo(0), f1 = fibo(1);
15 tipo fibo(tipo n, tipo f0, tipo f1) {
16   M22 mat=(M22){0, 1, 1, 1}^n;
17   return (mat.a*f0 + mat.b*f1) % mod;
18 }
```

6.7 Gauss Jordan

```cpp
const int N = 300;
typedef vector<double> col;
typedef vector<double> row;
typedef vector<row>Matrix;
col solution;
int main(){
  Matrix M;
  M.resize(300);
  solution.resize(300);
  for(int i = 0; i < 30;i++)M[i].resize(30);
  int n;
  cin >> n;
  for(int i = 0; i < n;i++)
    for(int j = 0; j <= n;j++)
      cin >> M[i][j];

  for(int j = 0; j < n - 1;j++){
    int l =j;
    for(int i = j + 1; i < n;i++){
      if(fabs(M[i][j]) > fabs(M[l][j]))l = i;
    }
    for(int k = j; k <= n;k++){
      swap(M[j][k],M[l][k]);
    }
    for(int i = j + 1; i < n;i++)
      for(int k = n; k >= j;k--)
        M[i][k] -= M[j][k] * M[i][j] / M[j][j];
  }
  double t = 0;
  for(int j = n - 1; j >= 0; j--){
    t = 0.0;
    for(int k = j + 1; k < n;k++)t += M[j][k] * solution[k];
    solution[j] = (M[j][n] - t) / M[j][j];
  }
  cout.precision(4);
  for(int i = 0; i < n;i++)cout<<fixed << solution[i] << "␣";
  return 0;
}
```

6.8 Simplex

```cpp
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[
        i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n +
        1] = b[i]; }
```

```
42        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
43        N[n] = -1; D[m + 1][n] = 1;
44      }
45
46      void Pivot(int r, int s) {
47        double inv = 1.0 / D[r][s];
48        for (int i = 0; i < m + 2; i++) if (i != r)
49          for (int j = 0; j < n + 2; j++) if (j != s)
50            D[i][j] -= D[r][j] * D[i][s] * inv;
51        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
52        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
53        D[r][s] = inv;
54        swap(B[r], N[s]);
55      }
56
57      bool Simplex(int phase) {
58        int x = phase == 1 ? m + 1 : m;
59        while (true) {
60          int s = -1;
61          for (int j = 0; j <= n; j++) {
62            if (phase == 2 && N[j] == -1) continue;
63            if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
                  N[s]) s = j;
64          }
65          if (D[x][s] > -EPS) return true;
66          int r = -1;
67          for (int i = 0; i < m; i++) {
68            if (D[i][s] < EPS) continue;
69            if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
70                (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] < B
                    [r]) r = i;
71          }
72          if (r == -1) return false;
73          Pivot(r, s);
74        }
75      }
76
77      DOUBLE Solve(VD &x) {
78        int r = 0;
79        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
80        if (D[r][n + 1] < -EPS) {
81          Pivot(r, n);
82          if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -numeric_limits<
                DOUBLE>::infinity();//NO SOLUTION
83          for (int i = 0; i < m; i++) if (B[i] == -1) {
84            int s = -1;
85            for (int j = 0; j <= n; j++)
86              if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j]
                    < N[s]) s = j;
87            Pivot(i, s);
88          }
89        }
90        if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();//
              INFINITY
91        x = VD(n);
92        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
93        return D[m][n + 1];//solution find
94      }
95    };
96
97    int main() {
98
99      const int m = 4;
100     const int n = 3;
101     DOUBLE _A[m][n] = {
102       { 6, -1, 0 },
103       { -1, -5, 0 },
104       { 1, 5, 1 },
105       { -1, -5, -1 }
106     };
107     DOUBLE _b[m] = { 10, -4, 5, -5 };
108     DOUBLE _c[n] = { 1, -1, 0 };
109
110     VVD A(m);
111     VD b(_b, _b + m);
112     VD c(_c, _c + n);
113     for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
114
115     LPSolver solver(A, b, c);
116     VD x;
117     DOUBLE value = solver.Solve(x);
118
119     cerr << "VALUE: " << value << endl; // VALUE: 1.29032
120     cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
121     for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
122     cerr << endl;
```

```
123      return 0;
124  }
```

6.9   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  }
```

6.10   Teorema Chino del Resto

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

6.11   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();
```

6.12   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   }
```

6.13   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    }
```

6.14   GCD

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

6.15   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    }
```

6.16   LCM

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

6.17   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 }
```

6.18   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.;}
```

6.19   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 };
```

6.20   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    poly S; S.c.pb(0);
48    forn(i,sz(x)) { poly Li;
49      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  }
```

6.21   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  }
```

6.22   Karatsuba

```
1  // g++ -std=c++11 "karatsuba.cpp" -o hld
2
3  /***
4  =============== <karatsuba> ==================
5  Complexity: O(N^1.7)
6  Call to karatsuba function paramter two vectors
7  * INPUT: two vectors A,B cointains the coeficients of the polynomail
8  * OUTPUT a vector coitains the coeficients of A * B
9  */
10
11  int p,k;
12  vector<int> b,r;
13
14  void trim(vector<int>& a){
15      while (a.size() > 0 && a.back() == 0) a.pop_back();
16  }
17
18  vector<int> multiply(const vector<int>& a, const vector<int>& b){
19      vector<int> c(a.size() + b.size() + 1, 0);
20      for (int i = 0; i < a.size(); i++) {
21          for (int j = 0; j < b.size(); j++) {
```

```
22              c[i+j] += a[i] * b[j];
23          }
24      }
25      trim(c);
26      return c;
27  }
28  // a = a + b*(10^k)
29  void addTo(vector<int>& a, const vector<int>& b, int k){
30      if (a.size() < b.size() + k) a.resize(b.size() + k);
31      for (int i = 0; i < b.size(); i++) a[i+k] += b[i];
32  }
33  void subFrom(vector<int>& a, const vector<int>& b){
34      for (int i = 0; i < b.size(); i++) a[i] -= b[i];
35  }
36  // a = a + b
37  void addTo(vector<int>& a, const vector<int>& b){
38      addTo(a, b, 0);
39  }
40  vector<int> karatsuba(const vector<int>& a, const vector<int>& b)
41  {
42      int alen = a.size();
43      int blen = b.size();
44      if (alen == 0 || blen == 0) return vector<int>();
45      if (alen < blen) return karatsuba(b, a);
46      if (alen < 50) return multiply(a, b);
47
48      int half = alen / 2;
49      vector<int> a0(a.begin(), a.begin() + half);
50      vector<int> a1(a.begin() + half, a.end());
51      vector<int> b0(b.begin(), b.begin() + min<int>(blen, half));
52      vector<int> b1(b.begin() + min<int>(blen, half), b.end());
53
54      vector<int> z0 = karatsuba(a0, b0);
55      vector<int> z2 = karatsuba(a1, b1);
56      addTo(a0, a1);
57      addTo(b0, b1);
58      vector<int> z1 = karatsuba(a0, b0);
59      subFrom(z1, z0);
60      subFrom(z1, z2);
61
62      vector<int> res;
63      addTo(res, z0);
64      addTo(res, z1, half);
65      addTo(res, z2, half + half);
66
67      trim(res);
68      return res;
69  }
```

6.23   FFT

```
1   #define lowbit(x) (((x) ^ (x-1)) & (x))
2   typedef complex<long double> Complex;
3
4   void FFT(vector<Complex> &A, int s){
5       int n = A.size();
6       int p = __builtin_ctz(n);
7
8       vector<Complex> a = A;
9
10      for(int i = 0;i<n;++i){
11          int rev = 0;
12          for(int j = 0;j<p;++j){
13              rev <<= 1;
14              rev |= ((i >> j) & 1);
15          }
16          A[i] = a[rev];
17      }
18
19      Complex w,wn;
20
21      for(int i = 1;i<=p;++i){
22          int M = (1<<i), K = (M>>1);
23          wn = Complex(cos(s*2.0*M_PI/(double)M), sin(s*2.0*M_PI/(double)M
                 ));
24
25          for(int j = 0;j<n;j += M){
26              w = Complex(1.0, 0.0);
27              for(int l = j;l<K+j;++l){
28                  Complex t = w;
29                  t *= A[l + K];
30                  Complex u = A[l];
31                  A[l] += t;
32                  u -= t;
33                  A[l + K] = u;
34                  w *= wn;
```

```
35              }
36          }
37      }
38
39      if(s==-1){
40          for(int i = 0;i<n;++i)
41              A[i] /= (double)n;
42      }
43  }
44
45  vector<Complex> FFT_Multiply(vector<Complex> &P, vector<Complex> &Q){
46      int n = P.size()+Q.size();
47      while(n!=lowbit(n)) n += lowbit(n);
48
49      P.resize(n,0);
50      Q.resize(n,0);
51
52      FFT(P,1);
53      FFT(Q,1);
54
55      vector<Complex> R;
56      for(int i=0;i<n;i++) R.push_back(P[i]*Q[i]);
57
58      FFT(R,-1);
59
60      return R;
61  }
62
63  // Para multiplicacion de enteros grandes
64  const long long B = 100000;
65  const int D = 5;
```

6.24   Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

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

```cpp
int negative_cycle(vector<vector<int> > &G, vector<vector<int> > &cost)
     {
  //write your code here
  bool nc = false;
  int n = G.size();
  vector<int>dist(n,INT_MAX / 2);
  dist[0] = 0;
  for(int i = 0; i < n - 1; i++)
  for(int u = 0; u < n;u++)
    for(int j = 0; j < G[u].size();j++){
      int v = G[u][j];
      int w = cost[u][j];
      dist[v] = min(dist[v], dist[u] + w);
    }
```

```cpp
    for(int u = 0; u < n;u++){
    for(int j = 0; j < G[u].size();j++){
      int v = G[u][j];
      int w = cost[u][j];
      if(dist[v] > dist[u] + w)nc = true;
    }
    }
    return nc;
}
```

7.2 dijkstra grafos densos

7.3 2 SAT definitivamente no con Tarjan

```cpp
// g++ -std=c++11 "twosat.cpp" -o run
/***
=============== <Two Sat> ==================
Complexity: O(N)
Input: number of variables, then number of clause clauses in format (u
    or v)
if u,v > 0 then is equivalent to u,v
if u,v < 0 then is equivalent to  u ,  v
Output: UNSATISFIABLE can't find a solution
SATISFIABLE if exist a solution then print the assignment of all
    variables (negative for xi = false)

Examples:
Input:
3 3
1 -3
-1 2
-2 -3
Output
SATISFIABLE
1 2 -3
*
Input
1 2
1 1
-1 -1
Output
UNSATISFIABLE
*/
#include <bits/stdc++.h>
```

```cpp
29  using namespace std;
30  vector<int>G[2][2000010],G2[2000010];
31  int n, m;
32  int scc[2000010];
33  bool vis[2000010];
34  vector<int>comp[2000010];
35  int assign[2000010];
36  int cc = 0;
37  stack<int>st;
38  vector<int>sta;
39  void dfs(int u,int type){
40    if(scc[u] != -1)return;
41    scc[u] = cc;
42    for(int v:G[type][u]){
43      dfs(v,type);
44    }
45    if(!type)st.push(u);
46  }
47  void topo(int u){
48    if(vis[u])return;
49    vis[u] = true;
50    for(int v:G2[u])topo(v);
51    sta.push_back(u);
52  }
53  void buildGraphWitouthLoop(){
54    for(int i = 0;i < 2 * n;i++){
55      for(int j = 0;j < G[0][i].size();j++){
56        if(scc[i] != scc[G[0][i][j]])
57          G2[scc[i]].push_back(scc[G[0][i][j]]);
58      }
59    }
60  }
61  int main() {
62      ios::sync_with_stdio(false);cin.tie(0);
63      cin >> n >> m;
64      for(int i = 0,u,v; i < m;i++){
65      cin >> u >> v;
66      int uu = (u > 0?(u - 1) * 2:(-u - 1) * 2 + 1);
67      int vv = (v > 0?(v - 1) * 2:(-v - 1) * 2 + 1);
68    //  cout << uu << " " << (uu ^ 1) << "\n";
69      G[0][uu ^ 1].push_back(vv);
70      G[0][vv ^ 1].push_back(uu);
71      G[1][vv].push_back(uu ^ 1);
72      G[1][uu].push_back(vv ^ 1);
73    }
74    memset(scc,-1,sizeof scc);
75    for(int i = 0; i < 2 * n;i++){
76      if(scc[i] == -1)dfs(i,0);
77    }
78    memset(scc,-1,sizeof scc);
79    while(!st.empty()){
80      int u = st.top();st.pop();
81      if(scc[u] == -1){
82        dfs(u,1);
83        cc++;
84      }
85    }
86    bool unsat = false;
87    for(int i = 0; i < 2 * n;i++){
88      if(scc[i] == scc[i ^ 1])unsat = true;
89      comp[scc[i]].push_back(i);
90    }
91    if(unsat){
92      return cout << "UNSATISFIABLE",0;
93    }
94    cout << "SATISFIABLE\n";
95    buildGraphWitouthLoop();
96    for(int i = 0; i < 2 * n;i++){
97      if(!vis[i])topo(i);
98    }
99    for(int u:sta){//inverse of topological sort
100     for(int v:comp[u]){//transitivite Skew-Symmetry
101       if(!assign[v]){
102         assign[v] = 1;
103         assign[v ^ 1] = -1;
104       }
105     }
106   }
107   for(int i = 0,j = 1; i < 2 * n; i += 2,j++){
108     cout << (j) * (assign[i]) << " ";
109   }
110     return 0;
111 }
```

7.4   Prim

7.5   Articulataion Points (desgraciadamente tarjan)

```cpp
// g++ -std=c++11 "articulationpointsandbridges.cpp" -o run
/***
=============== <Articulation points and bridges c++ version>
    ==================
Given a graph return a vector of paris with the bridges and a bool array
    art[]
true if the node is an articulation point
* false otherwise
Graph nodes: 0 to N - 1
*/
using namespace std;
vector<int>G[10010];
int low[10010],num[10010],parent[10010],cc;
//cc is my timer
int art[10010];//bool for detect art point, int for detect how many
    nodes are connected to my articulation point
int root,rC;
int n;
vector<pair<int,int> >bridges;
void dfs(int u){
  low[u]=num[u]=cc++;
  for(int v:G[u]){
    if(num[v]==-1){
      parent[v]=u;
      if(u==root)rC++;
      dfs(v);
      if(low[v]>=num[u])art[u]++;//is a articulation point
      if(low[v]>num[u])bridges.push_back({u,v});//this is a bridge
      low[u]=min(low[u],low[v]);
    }
    else if(v!=parent[u]){
        low[u]=min(low[u],num[v]);
    }
  }
}
void init(){
  bridges.clear();
  for(int i=0;i<n;i++){
    art[i]=low[i]=0;
    num[i]=parent[i]=-1;
    G[i].clear();
  }
  cc=0;
}
void callARTBRID(){
  for(int i=0;i<n;i++){
    if(num[i]==-1){
      root=i,rC=0;dfs(i);
      art[root]=(rC>1);
    }
  }
}
```

7.6   componentes biconexas y puentes (block cut tree)

```cpp
int V;
vector<int> G[MAXN];
int dfn[MAXN],low[MAXN];
vector< vector<int> > C;
stack< pair<int, int> > stk;
void cache_bc(int x, int y){
    vector<int> com;
    int tx,ty;
    do{
        tx = stk.top().first, ty = stk.top().second;
        stk.pop();
        com.push_back(tx), com.push_back(ty);
    }while(tx!=x || ty!=y);
    C.push_back(com);
}

void DFS(int cur, int prev, int number){
    dfn[cur] = low[cur] = number;
    for(int i = G[cur].size()-1;i>=0;--i){
        int next = G[cur][i];
        if(next==prev) continue;
        if(dfn[next]==-1){
            stk.push(make_pair(cur,next));
            DFS(next,cur,number+1);
            low[cur] = min(low[cur], low[next]);
            if(low[next]>=dfn[cur]) cache_bc(cur,next);
        }else low[cur] = min(low[cur],dfn[next]);
```

```
28        }
29  }
30
31  void biconn_comp(){
32      memset(dfn,-1,sizeof(dfn));
33      C.clear();
34      DFS(0,0,0);
35      int comp = C.size();
36      printf("%d\n",comp);
37      for(int i = 0;i < comp;++i){
38          sort(C[i].begin(),C[i].end());
39          C[i].erase(unique(C[i].begin(),C[i].end()),C[i].end());
40          int m = C[i].size();
41          for(int j = 0;j < m;++j) printf("%d ",1 + C[i][j]);
42          printf("\n");
43      }
44  }
```

7.7   LCA saltitos potencias de 2

7.8   LCA sparse table query O(1)

7.9   HLD

```
1   // g++ -std=c++11 "hld.cpp" -o hld
2
3   /***
4   =============== <HLD> ===================
5   Complexity: O(N*log (N))
6   Given a tree and asociative operation in the paths of this tree ask for
        many querys, and updates
7   in nodes or edges
8   Input of this example:
9   N number of nodes, then N elements values in each node
10  then n - 1 conections
11  Q querys if T == 1 query on the path u,v
12  else update node U with value val.
13
14  Example problems: Spoj QTREE1 to QTREE6, toby and tree UVA
15  */
16
17  #include <bits/stdc++.h>
18  using namespace std;
19  const int maxn = 1e5;
```

```
20  const int NEUTRO = 0; // a null value for my ST
21  int vec[maxn];
22  vector<int>G[maxn]; //the graph
23  //int idx[maxn]; // case with value in the edge
24  int op(int u,int v){// an operation for my path (using ST)
25    //return __gcd(u,v);
26    //return max(u,v);
27    return u + v;
28  }
29  int n;
30  //ask to Branimir for information about this
31  struct SegmentTree{
32    int T[2*maxn];
33    void init(){
34      memset(T,0,sizeof T);
35    }
36    void set(int pos,int val){
37      pos += n;
38      T[pos] = val;
39      for(pos >>= 1; pos > 0; pos >>=1){
40        T[pos] = op( T[pos << 1] , T[(pos << 1)|1] );
41      }
42    }
43    int query(int l,int r){
44      l += n;
45      r += n;
46      int ans = NEUTRO;
47      while( l < r){
48        if ( l & 1 ) ans = op(ans, T[l++] );
49        if (r & 1 ) ans = op( ans, T[--r] );
50        l >>= 1;
51        r >>= 1;
52      }
53      return ans;
54    }
55  };
56  struct hld{
57    int ncad; // store actual number of chain
58    int root; // the root of a tree generally 0 or 1
59    int pos; // pos of node in chain
60
61    int sz[maxn]; // store the subsize of subtrees
62    int depth[maxn]; //depth of the node, useful for LCA via HLD
```

```
63    int parent[maxn]; // useful for LCA
64    int where[maxn]; // where chain is the node?
65    //int edgepos[maxn]; // if the value is on the edge: stored in a node
66    int chainIdx[maxn]; // position in the chain of the node
67    int head[maxn]; // the head of the i-th chain
68    //int val[maxn]; // if the value is on the edge
69    SegmentTree tree; // this ST allow operations in the path
70
71    void init(){//settings value, and process de HLD
72      root = 0;
73      ncad = 0;
74      pos = 0;
75      for(int i = 0; i <=n; i++){
76        where[i] = head[i] = -1;
77      }
78      depth[root] = 0;
79      dfs(root , -1);
80      descompose(root);
81      tree.init();
82      /* case with values in edges
83      for(int i=0;i<n;i++){
84        tree.set(i,val[i]);
85      }
86      */
87    }
88
89
90    ///init descomposition
91    void dfs(int u,int pu){
92      sz[u] = 1; //init the sz of this subtree
93      parent[u] = pu; // assign the parent
94      for(int i = 0; i < G[u].size(); i++){
95        int v = G[u][i];
96        if ( v == pu )continue;
97        //edgepos[idx[u][i]] = v;
98        depth[v] = depth[u] + 1;
99        dfs(v,u);
100       sz[u] += sz[v];
101     }
102   }
103   //descompose graph in HLD descomposition
104   void descompose(int u){
105     if( head[ncad] == -1)head[ncad] = u; // the head of ncad is u
```

```
106     where[u] = ncad; // assign where tu node
107     //val[pos] = cost; cost another parameter in descompose for graphs
           with values in edges
108     chainIdx[u] = pos++; //assing pos to this node
109     int maxi = -1, sc = -1; //finding a special child
110     for(int v:G[u]){
111       if( sz[v] > maxi && where[v] == -1){
112         maxi = sz[v];
113         sc = v;
114       }
115     }
116     if(sc != -1)descompose(sc);
117     //light nodes here:
118     for(int v:G[u]){
119       if(where[v] == -1){
120         ncad++;
121         descompose(v);
122       }
123     }
124   }
125   ///end descomposition
126
127   int lca(int u,int v){
128     while(where[u]!=where[v]){
129       if(depth[ head[ where[u] ] ] > depth[ head[ where[v] ] ])u =
             parent[ head[ where[u] ] ];
130       else v = parent[ head[ where[v] ] ];
131     }
132     return depth[u] < depth[v] ? u:v;
133   }
134
135   void update(int u, int val){
136     tree.set(chainIdx[u],val);
137   }
138
139   int query(int u,int v){
140     // if ( u == v) return NEUTRO; value in edges
141     int vChain = where[v];
142     int ans = NEUTRO;
143     while(true){
144       int uChain = where[u];
145       if(uChain == vChain){
146         // return op(ans, tree.query( chainIdx[v] + 1, chainIdx[u] + 1)
```

```
147          ); value in edges
             return op(ans, tree.query( chainIdx[v], chainIdx[u] + 1) );
148        }
149        int hu = head[uChain];
150        ans = op( ans, tree.query(chainIdx[hu], chainIdx[u] + 1) );
151        u = parent[hu];
152      }
153    }
154
155    int Q(int u,int v){
156      int L = lca(u,v);
157      return op( query(u,L) , query(v,L) );
158    }
159  }HLD;
160  int main(){
161    //ios::sync_with_stdio(false);cin.tie(0);
162    while(cin >> n){
163      for(int i = 0; i < n; i++)G[i].clear();
164      for(int i = 0; i < n; i++){
165        cin >> vec[i];
166      }
167      for(int i = 1, u,v ; i < n; i++){
168        cin >> u >> v;
169        G[u].push_back(v);
170        G[v].push_back(u);
171        /* case with value in edges
172         G[u].push_back(make_pair(v,w));
173        idx[u].push_back(i-1);
174        G[v].push_back(make_pair(u,w));
175        idx[v].push_back(i-1);
176
177         */
178      }
179      HLD.init();
180      for(int i = 0; i < n; i++){
181        HLD.update(i, vec[i]);
182      }
183      int question;
184      cin >> question;
185      for(int i = 0, t, u ,v; i < question; i++){
186        cin >> t >> u >> v;
187        if( t == 1){
188          cout << HLD.Q(u,v) << "\n";
```

```
189          }
190          else HLD.update(u,v);
191      }
192    }
193  }
```

7.10   centroid descomposition

7.11   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  }
```

7.12   diámetro y centro de un árbol

```
/***
=============== <Diameter and center of a tree> ==================
//Problem: Given a tree get the center (or centers)
/* the nodes in the tree that minimize the length of the longest path
    from it to any other node.
* *Finding tree centers:
 * If diameter length is even, then we have one tree center. If odd,
    then we have 2 centers.
 * E.g. 1-2-3-4-5   -> center is 3
 * E.g. 1-2-3-4-5-6 -> center is 3, 4
 * On other side, we can get the worst nodes through the center nodes.
 * A worst node is one that is an end of a diameter, so it has the worst
    tree height
Input:
* No
Output:
* No
dfs: calculate the diameter of the tree
* maxi stores the diameter
findingCenters() return the centers
Nodes in graph 1 to N careful with this
Complexity: O(N)
*/


vector<int>G[5010];
int maxi=-1,far;
int n;
int pre[5010];
int Queue[5010];

void dfs(int path,int u,int parent){
  pre[u]=parent;
  if(path>=maxi){
    maxi=path;
    far=u;
  }
  for(int v:G[u]){
    if(parent!=v){
      dfs(path+1,v,u);//path + w if the graph as weighted
    }
```

```
  }
}
pair<int,int> findingCenters(){
  maxi=-1;
  dfs(0,1,-1);
  dfs(0,far,-1);
  int t=far,L=0;
  while(t!=-1){
    Queue[L]=t;
    t=pre[t];
    ++L;
  }
  int a=-1,b=-1;
  if(L&1){
    a=Queue[L/2];
  }
  else{
    a=min(Queue[L/2-1],Queue[L/2]),b=max(Queue[L/2-1],Queue[L/2]);
  }
  return {a,b};
}
```

7.13   algoritmo hungaro

7.14   union find dinámico

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

struct UnionFind {
    int n, comp;
```

```
19      vector<int> pre,si,c;
20      UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
21          forn(i,n) pre[i] = i; }
22      int find(int u){return u==pre[u]?u:find(pre[u]);}
23      bool merge(int u, int v) {
24          if((u=find(u))==(v=find(v))) return false;
25          if(si[u]<si[v]) swap(u, v);
26          si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
27          return true;
28      }
29      int snap(){return sz(c);}
30      void rollback(int snap){
31          while(sz(c)>snap){
32              int v = c.back(); c.pop_back();
33              si[pre[v]] -= si[v], pre[v] = v, comp++;
34          }
35      }
36  };
37  enum {ADD,DEL,QUERY};
38  struct Query {int type,u,v;};
39  struct DynCon {
40      vector<Query> q;
41      UnionFind dsu;
42      vector<int> match,res;
43      map<ii,int> last;//se puede no usar cuando hay identificador para
                cada arista (mejora poco)
44      DynCon(int n=0):dsu(n){}
45      void add(int u, int v) {
46          if(u>v) swap(u,v);
47          q.pb((Query){ADD, u, v}), match.pb(-1);
48          last[ii(u,v)] = sz(q)-1;
49      }
50      void remove(int u, int v) {
51          if(u>v) swap(u,v);
52          q.pb((Query){DEL, u, v});
53          int prev = last[ii(u,v)];
54          match[prev] = sz(q)-1;
55          match.pb(prev);
56      }
57      void query() {//podria pasarle un puntero donde guardar la respuesta
58          q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
59      void process() {
60          forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] =
                    sz(q);
61          go(0,sz(q));
62      }
63      void go(int l, int r) {
64          if(l+1==r){
65              if (q[l].type == QUERY)//Aqui responder la query usando el
                        dsu!
66                  res.pb(dsu.comp);//aqui query=cantidad de componentes
                            conexas
67              return;
68          }
69          int s=dsu.snap(), m = (l+r) / 2;
70          forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i
                    ].v);
71          go(l,m);
72          dsu.rollback(s);
73          s = dsu.snap();
74          forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[
                    i].v);
75          go(m,r);
76          dsu.rollback(s);
77      }
78  }dc;
79
80  // Problema ejemplo: http://codeforces.com/gym/100551/problem/A
81
82  int n,k;
83
84  int main() {
85      //~ freopen("in", "r", stdin);
86      freopen("connect.in", "r", stdin);
87      freopen("connect.out", "w", stdout);
88      ios::sync_with_stdio(0);
89      while(cin >> n >> k){
90      dc=DynCon(n);
91      forn(_,k) { string ord; cin >> ord;
92        if (ord=="?") {
93          dc.query();
94        } else if (ord=="+") { int a,b; cin>>a>>b; a--;b--;
95          dc.add(a,b);
96        } else if (ord=="-") { int a,b; cin>>a>>b; a--;b--;
97          dc.remove(a,b);
98        } else assert(false);
```

```
 99        }
100            if(!k) continue;//k==0 WTF
101            dc.process();
102            forn(i,sz(dc.res)) cout << dc.res[i] << endl;
103        }
104        return 0;
105 }
```

7.15   truquitos estúpidos por ejemplo second MST es con LCA

7.16   erdos galloi

```
 1  // g++ -std=c++11 "erdosgalloi.cpp" -o run
 2  /***
 3  =============== <Erdosgalloi c++ version> ===================
 4  Given the grades of each node of a graph return if this form a valid
        graph
 5  includes: algorithm, functional, numeric, forn
 6  // Receives a sorted degree sequence (non ascending)
 7  O(NlgN)
 8
 9  */
10  bool isGraphicSequence(const vector<int> &seq)  // O(n lg n)
11  {
12    vector<int> sum;
13    int n = seq.size();
14
15    if (n == 1 && seq[0] != 0) return false;
16
17    sum.reserve(n + 1);
18    sum.push_back(0);
19    for (int i = 0; i < n; ++i) sum.push_back(sum[i] + seq[i]);
20    if ((sum[n] & 1) == 1) return false;
21
22    for (long long k = 1; k <= n - 1 && seq[k - 1] >= k; ++k) {
23      int j = distance(seq.begin(), upper_bound(seq.begin() + k, seq.end()
          , k,
24                                            greater<int>())) +
25          1;
26      long long left = sum[k];
27      long long right = k * (k - 1) + (j - k - 1) * k + (sum[n] - sum[j -
          1]);
28
29      if (left > right) return false;
```

```
30    }
31
32    return true;
33 }
```

7.17   grafo funcional hallar k-esimo partiendo de un nodo

7.18   konig

7.19   min-vertex cover bipartitos

7.20   max-flow (min cost versión)

```
 1  // g++ -std=c++11 "maxflowmincost.cpp" -o run
 2  /***
 3  =============== <Max flow-min cost c++ version> =================
 4  Given a grapth with edges with a capacity C and weight D
 5  * compute the max-flow min cost
 6  Edmond karps idea
 7  * Complexity O( v  *E*log(v))
 8  Problem for practice: Dijkstra Dijkstra uva
 9  */
10  #define REP(i,j,k) for(int (i)=(j);(i)<(k);++(i))
11  #define MP make_pair
12
13  using namespace std;
14
15  #define MAXN 500
16  #define MAXM MAXN * 5
17  typedef vector<int> VI;
18  typedef long long ll;
19  const int INF = 1E9; // $infinity$: be careful to make this big enough
        !!!
20  int S; // source
21  int T; // sink
22  int FN; // number of nodes
23  int FM; // number of edges (initialize this to 0)
24  // ra[a]: edges connected to a (NO MATTER WHICH WAY!!!); clear this in
        the beginning
25  VI ra[MAXN];
26  int kend[MAXM], cap[MAXM], cost[MAXM]; // size: TWICE the number of
        edges
27
28  // Adds an edge from a to b with capacity c and cost d and returns the
```

```
          number of the new edge
29
30  int addedge(int a, int b, int c, int d) {
31    int i = 2*FM;
32    kend[i] = b;
33    cap[i] = c;
34    cost[i] = d;
35    ra[a].push_back(i);
36    kend[i+1] = a;
37    cap[i+1] = 0;
38    cost[i+1] = -d;
39    ra[b].push_back(i+1);
40    FM++;
41    return i;
42  }
43  int n;
44  int dst[MAXM], pre[MAXM], pret[MAXM];
45  //finding the shortest path via fanding duan, also it works with bellman
          ford
46  //or dijkstra (careful of negative cycles)
47  bool spfa(){
48    REP(i,0,FN) dst[i] = INF;
49    dst[S] = 0;
50    queue<int> que; que.push(S);
51    while(!que.empty()){
52      int x = que.front(); que.pop();
53      for (int t : ra[x]){
54        int y = kend[t], nw = dst[x] + cost[t];
55        if(cap[t] > 0 && nw<dst[y]){
56          dst[y] = nw; pre[y] = x; pret[y] = t; que.push(y);
57        }
58      }
59    }
60    return dst[T]!=INF;
61  }
62  // returns the maximum flow and the minimum cost for this flow
63  pair<ll,ll> solve(){
64    ll totw = 0, totf = 0;
65    while(spfa()){
66      int minflow = INF;
67      for (int x = T; x!=S; x = pre[x]){
68        minflow = min(minflow, cap[pret[x]]);
69      }
```

```
70      for (int x = T; x!=S; x = pre[x]){
71        cap[pret[x]] -= minflow;
72        cap[pret[x]^1] += minflow;
73      }
74      totf += minflow;
75      totw += minflow*dst[T];
76    }
77    return make_pair(totf, totw);
78  }
79  void init(){
80    FN=4*n+15;//make this big n=number of nodes of the graph
81    FM=0;
82    S=0,T=n+1;
83    for(int i=0;i<FN;i++)ra[i].clear();//clear the graph be careful
84  }
```

7.21   max-flow corto con matriz

```
1   // g++  "maxflowMVEK.cpp" -o run
2   /***
3   =============== <Max Flow with matriz Edmonds karp c++ version>
          ==================
4   //Given a graph with capacitys find the max-flow
5
6   Nodes indexed 1 to N
7   * Complexity O( N *E)
8   Problem for practice: UVA 820
9   */
10  #define N 500
11  int cap[N][N], pre[N], n;
12  int s;//source
13  int t;//destination
14  bool bfs() {
15    queue<int>q;
16    q.push(s);
17    memset(pre,-1,sizeof pre);
18    pre[s]=s;
19    while(!q.empty()){
20      int u=q.front();q.pop();
21      if(u==t)return true;
22      for(int i=1;i<=n;i++){//nodes 1 to n
23        if(pre[i]==-1&&cap[u][i])pre[i]=u,q.push(i);
24      }
```

```
25        }
26        return false;
27    }
28
29    int maxFlow() {
30        int mf=0,f,v;//max flow, flow for a path, the vertex
31        while(bfs()){//while encountered a path  source to destination
32            v=t;//min
33            f=INT_MAX;//make this big enough
34            while(pre[v]!=v){f=min(f,cap[pre[v]][v]),v=pre[v];}//finding the
                    min capacity
35            v=t;mf+=f;
36            while(pre[v]!=v){cap[pre[v]][v]-=f,cap[v][pre[v]]+=f,v=pre[v];}
                    //update the flow
37        }
38        return mf;
39    }
40    void init(){
41      memset(cap,0,sizeof cap);
42      //cap[u][v]+=capacidad,cap[v][u]+=capacidad
43    }
```

7.22   max-flow sin matriz

```
1    // g++ -std=c++11 "maxflowNMEK.cpp" -o run
2    /***
3    =============== <Max Flow with-out matriz Edmonds karp c++ version>
          ==================
4    //Given a graph with capacitys find the max-flow
5
6    Nodes indexed 1 to N
7    * Complexity O( N *E)
8    Problem for practice: UVA 820
9    * Input N number of nodes,
10   * M edges conections
11   * compute the flow with source 1 and sink N
12   */
13   using namespace std;
14   const int N = 110;
15   const int M = 10010 * 2;
16   vector<int>G[N];
17   int kend[M], cap[M], cost[M];
18   int edge = 0;
19   int s,t;
20   void add(int u,int v,int c){
21      int forward = edge * 2, backward = edge * 2 + 1;
22      kend[forward] = v;
23      cap[forward] = c;
24      G[u].push_back(forward);
25      kend[backward] = u;
26      cap[backward] = 0;
27      G[v].push_back(backward);
28      edge++;
29   }
30   int vis[M],pre[M],pret[M];
31   bool bfs(){
32      for(int i = 0; i <= 100;i++)vis[i] = false;
33      vis[s] = true;
34      queue<int>q;
35      q.push(s);
36      while(!q.empty()){
37         int u = q.front();q.pop();
38         for(int edge:G[u]){
39            int v = kend[edge];
40            if(cap[edge] > 0 && !vis[v]){
41               vis[v] = true;
42               pre[v] = u;
43               pret[v] = edge;//the edge store the information
44               q.push(v);
45            }
46         }
47      }
48      return vis[t];
49   }
50   int max_flow(){
51      int totf = 0LL;
52      while(bfs()){
53         int minflow = INT_MAX;
54         for(int x = t; x != s; x = pre[x]){
55            minflow = min(minflow,cap[pret[x]]);
56         }
57         for(int x = t; x != s; x = pre[x]){
58            cap[pret[x]] -= minflow;
59            cap[pret[x] ^ 1] += minflow;
60         }
61         totf += minflow;
```

```
62        }
63        return totf;
64    }
65    int main(){
66        int n,m;
67        scanf("%d %d",&n,&m);
68        for(int i = 0,u,v,ca; i < m;i++){
69            scanf("%d %d %d",&u,&v,&ca);
70            add(u,v,ca);
71        }
72        s = 1, t = n;
73        printf("%lld\n",max_flow());
74    }
```

7.23   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 Independet 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(
             cap) {}
16   };
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
             forman el min-cut
57       return result; }
```

7.24   máximo emparejamiento bipartito

```cpp
// g++ -std=c "bipartitematching.cpp" -o run
/***
=============== <MCBM max cardinality bipartite matching c++ version>
    ==================
Return the bipartite matching of a Graph
* Format of nodes: 1 to N
*/

const int N = 100010;
vector<int>G[N];
bool v[N];//for the greedy speed up
int match[N];
bool vis[N];
int n,m;
//calling aumenting path
bool aug(int u){
    if(vis[u])return false;
    vis[u]=true;
    for(int i=0;i<(int)G[u].size();++i){
    int r=G[u][i];
        if(match[r]==-1||aug(match[r])){
            match[r]=u;match[u]=r;return true;
        }
    }
    return 0;
}
int mc;
//findging all augmenting path's
int solve(){
    bool check=true;
    while(check){
        check=false;
        memset(vis,0,sizeof vis);
        for(int i=1;i<=n;++i){
        if(!v[i]&&match[i]==-1){
            bool op=aug(i);
            check|=op;
            mc+=op;
        }
        }
    }
    return mc;
}
```

```cpp
void init(){
    memset(v,0,sizeof v);
    memset(vis,false,sizeof vis);
    mc=0;
    memset(match,-1,sizeof match);
    for(int i=0;i<=n;i++)G[i].clear();
}
void greedySpeedUp(){
    //greedy optimization, match with the first not matched
    for(int i=1;i<=n;++i){
        for(int j=0;j<(int)G[i].size();++j){
            if(match[G[i][j]]==-1){
            match[G[i][j]]=i,match[i]=G[i][j],mc++,v[i]=true;break;
        }
        }
    }
}
```

7.25   max-independent set en bipartitos

7.26   min-path cover (ver tópicos raros de halim)

7.27   min-cost arborescence

7.28   lema de diapositivas de nico de grafos funcionales

7.29   minimax y maximini con kruskal y dijkstra

```cpp
// g++ -std=c++11 "maximini.cpp" -o run
/***
=============== <maximini c++ version> ==================
Given a weighted graph return the maximini (the maximun of the minimum)
or the minimax (the minimum of the maximum) in the path a,b
*
Minimax as definded as: finding the minimum of maximum edge weight among
    all possible paths
* between two verrtices a to b, the cost for a path fron a to b is
    determined by maximum edge
* weight along this path. Among all these possible paths from a to b,
    pick the one with the minimum
* ax-edge-weight
* Complexity O(E*log(E) + V + E)
*
Problem for practice: UVA 534,544
```

```
14  */
15  int n;
16  pair<int,pair<int,int> >Edges[20000];
17  int t;
18  map<string,int>mp;
19  int parent[210];
20  pair<int,int>child[210];
21  bool vis[210];
22  vector<pair<int,int> >G[210];
23
24  int find(int u){return u==parent[u]?u:parent[u]=find(parent[u]);}
25  void Union(int u,int v){
26    int pu=find(u),pv=find(v);
27    if(pu!=pv){
28      parent[pv]=pu;
29    }
30  }
31  int mst(int a,int b){
32    sort(Edges,Edges+t);
33    reverse(Edges,Edges+t);//don't reverse for the minimax
34    for(int i=0;i<=200;i++)parent[i] = i;
35    int w,u,v, maximini = 1e8, minimax = 0;
36    for(int i=0;i<t;i++){
37      tie(w,u,v) = make_tuple(Edges[i].first, Edges[i].second.first, Edges
            [i].second.second);
38      if(find(u) != find(v)){
39        Union(u,v);
40        G[u].push_back({v,w});
41        G[v].push_back({u,w});
42      }
43    }
44    queue<int>q;
45    q.push(a);
46    vis[a]=true;
47    while(!q.empty()){
48      int u = q.front();q.pop();
49      //if(u==1)break;
50      for(pair<int,double>node: G[u]){
51        if(!vis[node.first]){
52          vis[node.first] = true;
53          q.push(node.first);
54          //maximini=max(maximini,node.second);
55          child[node.first].first = u;
```

```
56          child[node.first].second = node.second;
57        }
58      }
59    }
60    for(int t = b;t != -1;t = child[t].first){
61      //cout<<t<<" "<<child[t].second<<"\n";
62      //minimax=max(minimax,child[t].second);
63      maximini = min(maximini,child[t].second);
64    }
65    return maximini;
66  }
```

8   Teoria de juegos

8.1   Teorema fundamental de los juegos optimos

```
1  boolean isWinning(position pos) {
2    moves[] = possible positions to which I can move from the position
          pos;
3    for (all x in moves)
4      if (!isWinning(x)) return true;
5
6    return false;
7  }
```

8.2   Como calcular grundy

```
1  int grundyNumber(position pos) {
2    moves[] = possible positions to which I can move from pos
3    set s;
4    for (all x in moves) insert into s grundyNumber(x);
5    //return the smallest non-negative integer not in the set s;
6    int ret=0;
7    while (s.contains(ret)) ret++;
8    return ret;
9  }
```

9   Algunas formulas de probabilidad

Importante
Verificar la independencia de eventos.

9.1   Regla general de la probabilidad

$$P(E) = \frac{Nro.casos favorables}{Nro.casos posibles}$$

9.2   Teorema de bayes (Probabilidad condicional)

$$P(A/B) = \frac{P(A \cap B)}{P(B)}$$

9.3   Regla de la suma

Se suman 2 probabilidades cuando los eventos no pueden ocurrir al mismo tiempo "Pasa el evento X o pasa el evento Y"

$$P(A) \cup P(B) = P(A) + P(B)$$

Regla de la suma con eventos compatibles e incompatibles (que pueden pasar al mismo tiempo o no)

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

9.4   Regla de la multiplicacion

La misma idea de arriba, "Pasa el evento X y pasa el evento Y"

$$P(A \cap B) = P(A) * P(B)$$

9.5   Esperanza matematica

$$E[X] = \sum_{i=0}^{n} X * P(X)$$

En el caso continuo reemplazar sumatoria por integral

9.6   Ley de la esperanza total

Esto se usa cuando quieres calcular el valor en el que pasa el evento X ej. calcular el dia esperado en el que muere el i-esimo pez

$$E(X) = E(E(X|Y))$$

9.7   Esperanza de variables independientes

$$E(X * Y) = E(X) * E(Y)$$

9.8   Varianza

$$\sigma^2 = E(X^2) - (E(X))^2$$

9.9   Distribucion Binomial

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Ejemplo. Supongamos que se lanza un dado (con 6 caras) 51 veces y queremos conocer la probabilidad de que el número 3 salga 20 veces.

$$f(X = 20) = \binom{51}{20} \frac{1}{6}^{20} * (1 - \frac{1}{6})^{51-20}$$

Esperanza:

$$E(x) = n * p$$

Varianza:

$$V(x) = np * (1-p)$$

10   Otros/utilitarios

10.1   josephus

```
int survivor(int n, int m){
    int s = 0;
    for (int i=1;i<=n;++i) s = (s+m)%i;
    return (s+1);
}
```

10.2 josephus k = 2

```java
//////////JAVA
  /**
   *
   * @param n the number of people standing in the circle
   * @return the safe position who will survive the execution
   *    f(N) = 2L + 1 where N =2^M + L and 0 <= L < 2^M
   */
  public int getSafePosition(int n) {
    // find value of L for the equation
    int valueOfL = n - Integer.highestOneBit(n);
    int safePosition = 2 * valueOfL  + 1;
    return safePosition;
  }
```

10.3 poker

10.4 iterar subconjuntos

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

10.5 como reconstruir una DP (normal)

```cpp
/*
You just need to revisit your steps in the DP. In case of 0-1 knapsack,
    lets say the original DP function was solve, and the function
    reconstruct will give you the actual solution (I'm writing the code
    in C++):
*/
int solve(int pos, int capacity){
    if(pos == no_of_objects) return 0;
    if(memo[pos][capacity] != -1) return memo[pos][capacity];
    int r1 = solve(pos + 1, capacity); //dont take
    int r2 = 0;
    if(weight[pos] <= capacity){
        r2 = solve(pos + 1, capacity - weight[pos]) + profit[pos]; //
            take
    }
    return memo[pos][capacity] = max(r1, r2);
}
void reconstruct(int pos, int capacity){
    if(pos == no_of_objects) return; //you have completed reconstruction
    int r1 = memo[pos + 1][capacity]; //dont take
    int r2 = 0;
    if(weight[pos] <= capacity)r2 = memo[pos + 1][capacity - weight[pos
        ]] + profit[pos]; //take
    if(r1 > r2) {reconstruct(pos + 1, capacity);}
    else{
        cout << "Take object " << pos << endl;
        reconstruct(pos + 1, capacity - weight[pos]) + profit[pos];
    }
}
After executing reconstruct, it will print all those objects that give
    you the optimal solution. As you can see, at most no_of_objects
    calls will be made in the reconstruct function.
Similarly, you can reconstruct the solution of any DP greedily.
```

10.6 muajaja con j

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

Expandir pila

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

10.7 comparar doubles for noobs

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

10.8 infix to postfix

```cpp
//infix to postfix with shunting yard, Halim interpretation
//plus eval function given a postfix return the result of the operation
```

```
3   //format: string like  ( x o x  ( x o x ) ) o=operation x=value
4   string s;
5   bool isOperator(string u){
6     return u=="+"||u=="-"||u=="*"||u=="/";
7   }
8   bool precede(string u){
9     if(u=="*"||u=="/")return true;
10    return false;
11  }
12  void solve(){
13    getline(cin,s);
14    stack<string>st;
15    vector<string>v;
16    stringstream ss;
17    ss<<s;
18    while(ss>>s){
19      if(isOperator(s)){
20        while(!st.empty()&&isOperator(st.top())&&precede(st.top())>=
              precede(s)){
21          v.push_back(st.top());st.pop();
22        }
23        st.push(s);
24      }
25      else{
26        if(s=="("){
27          st.push(s);
28        }
29        else{
30          if(s==")"){
31            while(!st.empty()&&st.top()!="("){
32              v.push_back(st.top());st.pop();
33            }
34            if(!st.empty()&&st.top()=="(")st.pop();
35          }
36          else {
37            v.push_back(s);
38          }
39        }
40      }
41    }
42    while(!st.empty()){
43      v.push_back(st.top());st.pop();
44    }
```

```
45    stack<double>stans;
46    double x;
47    for(string eva:v){
48      if(!isOperator(eva)){
49        stringstream nu;
50        nu<<eva;
51        nu>>x;
52        stans.push(x);
53      }
54      else{
55        double a=stans.top();stans.pop();
56        double b=stans.top();stans.pop();
57        if(eva=="*")b*=a;
58        if(eva=="/")b/=a;
59        if(eva=="+")b+=a;
60        if(eva=="-")b-=a;
61        stans.push(b);
62      }
63    }
64    cout<<fixed<<stans.top()<<"\n";
65  }
```

10.9   numeros romanos

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   map<int,string>cvt;
4
5   string aromano(int n){
6     cvt[1000] = "M";cvt[900] = "CM",cvt[500] = "D", cvt[400] = "CD";
7     cvt[100] = "C";cvt[90] = "XC"; cvt[50] = "L";
8     cvt[40] = "XL";cvt[10] = "X";cvt[9] = "IX";cvt[5] = "V"; cvt[4] = "IV"
          ;
9     cvt[1] = "I";
10    string ans = "";
11    for(map<int,string>::reverse_iterator it = cvt.rbegin();it != cvt.rend
          ();it++)
12      while(n >= it->first){
13        ans += it->second;
14        n -= it->first;
15      }
16    return ans;
17  }
```

```
18  map<string,int>crn;
19  int anumero(string R){
20    map<char, int> crn;
21    crn['I'] = 1;   crn['V'] = 5;   crn['X'] = 10;   crn['L'] = 50;
22    crn['C'] = 100; crn['D'] = 500; crn['M'] = 1000;
23    int value = 0;
24    for (int i = 0; R[i]; i++)
25      if (i + 1 < R.size() && crn[R[i]] < crn[R[i+1]]) {
26        value += crn[R[i+1]] - crn[R[i]];
27        i++;
28      }
29      else value += crn[R[i]];
30    return value;
31  }
```

10.10   get k-th permutacion

```
1   vector<int>v;
2   //finding the number of permutation 0.....n-1
3   int main()
4   {
5       string s;
6       while(getline(cin,s)){
7           stringstream ss;
8           ss<<s;
9           int pos=0,u;
10          v.clear();
11          while(ss>>u){
12              v.push_back(u-1);
13          }
14          vector<int>le(v.size(),0);
15          for(int i=0;i<v.size();i++){
16              for(int j=i+1;j<v.size();j++){
17                  if(v[i]>v[j])le[i]++;
18              }
19          }
20          long long ans=0LL,fact=0LL,por=1LL;
21          for(int i=le.size()-1;i>=0;i--){
22              if(fact!=0LL)por*=fact;
23              fact++;
24              ans=ans+por*le[i];
25          }
26          cout<<ans+1<<"\n";
```

```
27      }
28      return 0;
29  }
```

10.11   sliding window

10.12   permutaciones de un dado

```
1   // izquierda, derecha, arriba, al frente, abajo, atras
2
3   int p[][6] = {
4       {0,1,2,3,4,5},
5       {0,1,3,4,5,2},
6       {0,1,4,5,2,3},
7       {0,1,5,2,3,4},
8       {1,0,2,5,4,3},
9       {1,0,3,2,5,4},
10      {1,0,4,3,2,5},
11      {1,0,5,4,3,2},
12      {2,4,5,1,3,0},
13      {2,4,1,3,0,5},
14      {2,4,3,0,5,1},
15      {2,4,0,5,1,3},
16      {3,5,2,1,4,0},
17      {3,5,1,4,0,2},
18      {3,5,4,0,2,1},
19      {3,5,0,2,1,4},
20      {4,2,5,0,3,1},
21      {4,2,0,3,1,5},
22      {4,2,3,1,5,0},
23      {4,2,1,5,0,3},
24      {5,3,2,0,4,1},
25      {5,3,0,4,1,2},
26      {5,3,4,1,2,0},
27      {5,3,1,2,0,4}
28  };
```

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