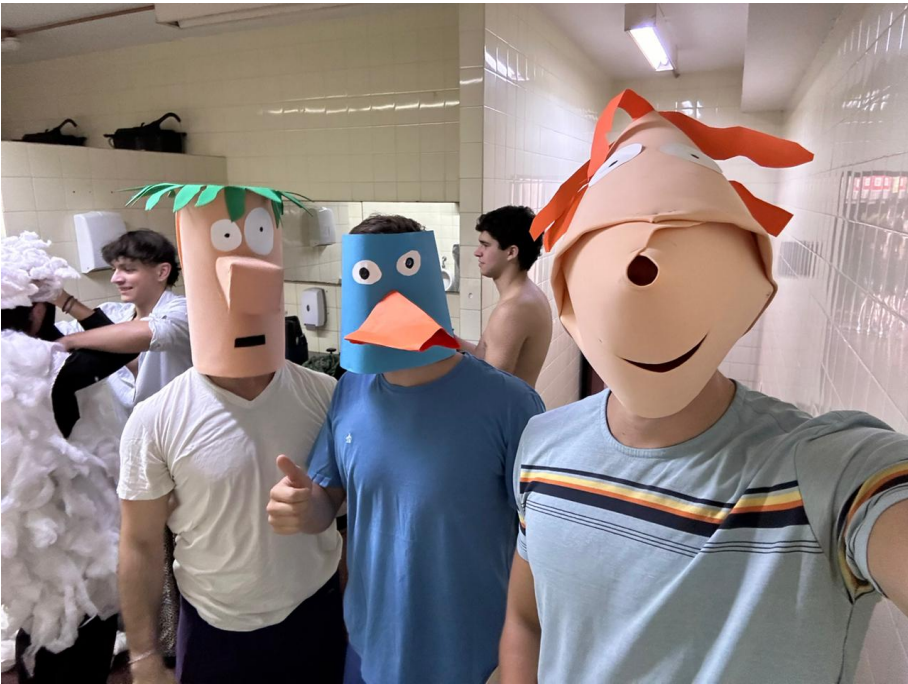


Notebook

UTN FRRO - HOLA

2024



Contents

1	Basics	3
1.1	Template . . . . .	3
1.2	Compilation . . . . .	3
2	Math	3
2.1	Identidades . . . . .	3
2.2	Tablas y cotas (Primos, Divisores, Factoriales, etc) . . . . .	3
2.3	Propiedades de Exponentes y Logaritmos . . . . .	3
2.3.1	1. Exponentes . . . . .	3
2.3.2	2. Logaritmos . . . . .	4
2.4	Reglas de divisibilidad . . . . .	5
2.5	Coprimos . . . . .	6
2.6	Primos . . . . .	6
2.7	Combinatoria . . . . .	7
2.8	Operaciones con modulo . . . . .	7
3	Estructuras	8
3.1	vector . . . . .	8
3.1.1	emplace . . . . .	8
3.1.2	resize . . . . .	8
3.1.3	assign . . . . .	8
3.2	unordered set . . . . .	9
3.3	Iterators . . . . .	9
3.4	Indexed Set . . . . .	9
3.5	Segment Tree . . . . .	10
3.6	Array . . . . .	11
3.7	BitSet . . . . .	11
4	Flujo	12
4.1	Dinic . . . . .	12
4.2	Hungarian . . . . .	13

<b>5</b>	<b>Grafos</b>	<b>14</b>
5.1	Recorrer Grafos . . . . .	14
5.1.1	DFS . . . . .	14
5.1.2	BFS . . . . .	14
5.2	Camino M��nimo . . . . .	15
5.2.1	Bellman-Ford . . . . .	15
5.2.2	Ciclos negativos . . . . .	15
5.2.3	Dijkstra . . . . .	15
5.2.4	Floyd-Warshall . . . . .	16
5.3	Spanning Tree . . . . .	16
5.3.1	UnionFind . . . . .	16
5.3.2	Kruskal . . . . .	17
5.4	Aplicaciones comunes . . . . .	17
5.4.1	Chequear si es conexo . . . . .	17
5.4.2	Bipartito . . . . .	17
5.4.3	Ciclos . . . . .	18
5.4.4	Componentes de un grafo . . . . .	18
5.4.5	Dijkstra modificado . . . . .	18
5.4.6	Max edge in path . . . . .	19
5.4.7	Max path . . . . .	20
5.4.8	Strong Connectivity . . . . .	20
5.4.9	Topological Sort . . . . .	20
5.5	Kosaraju . . . . .	21
<b>6</b>	<b>Strings</b>	<b>22</b>
6.1	Hash . . . . .	22
6.1.1	Examples . . . . .	23
6.1.2	Examples . . . . .	23
<b>7</b>	<b>Geometr��a</b>	<b>25</b>
7.1	Punto . . . . .	25
7.2	Line . . . . .	25
7.3	Segment . . . . .	25
7.4	Polar sort . . . . .	26
7.5	Convex Hull . . . . .	26
7.6	Area poligono . . . . .	27
7.7	Poligono . . . . .	27
7.8	Puntos mas cercanos . . . . .	28
7.9	Puntos mas lejanos . . . . .	30
7.10	Puntos enteros . . . . .	30

7.11	Circle . . . . .	31
<b>8</b>	<b>DP</b>	<b>33</b>
8.1	Game . . . . .	33
8.2	Long common subsequence . . . . .	33
8.3	Long increasing subsequence . . . . .	33
8.4	Matching mask . . . . .	34
8.5	DP rangos . . . . .	34
<b>9</b>	<b>Utils</b>	<b>35</b>
9.1	Binary Search . . . . .	35
9.2	Sort . . . . .	35
9.3	Cout para doubles . . . . .	35
9.4	Prev permutation . . . . .	35
9.5	MO . . . . .	36
9.6	Criba . . . . .	36
9.7	Subconjuntos . . . . .	37
9.8	Propiedades del Bitwise XOR ( $\oplus$ ) . . . . .	37

# 1 Basics

## 1.1 Template

```
1 #include <bits/stdc++.h>
2 #define forr(i, a, b) for (int i = (a); i < (b); i++)
3 #define forn(i, n) forr(i, 0, n)
4 #define dforn(i, n) for (int i = (n) - 1; i >= 0; i--)
5 #define all(v) v.begin(), v.end()
6 #define dbg(v) cerr << #v << ": " << v;
7 #define HOLA ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
8
9 using namespace std;
10 typedef long long ll;
11 typedef pair<int, int> ii;
12
13 int main() {
14     HOLA
15     #ifdef EBUG
16         freopen("<ej>.txt", "r", stdin);
17     #endif
18     return 0;
19 }
```

## 1.2 Compilation

```
1 g++ -DEBUG <ej>.cpp -o a && time ./a
```

# 2 Math

## 2.1 Identidades

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

## 2.2 Tablas y cotas (Primos, Divisores, Factoriales, etc)

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

2.3 Propiedades de Exponentes y Logaritmos

2.3.1 1. Exponentes

- Producto de potencias con misma base:

$$a^b \cdot a^c = a^{b+c}$$
- Potencia de una potencia:

$$(a^b)^c = a^{b \cdot c}$$

2.3.2 2. Logaritmos

- Logaritmo de un producto:

$$\log(ab) = \log a + \log b$$
- Logaritmo de una potencia:

$$\log(a^b) = b \log a$$
- Logaritmo de un cociente:

$$\log\left(\frac{a}{b}\right) = \log a - \log b$$

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

π(10¹) = 4 ; π(10²) = 25 ; π(10³) = 168 ; π(10⁴) = 1229 ; π(10⁵) = 9592  
π(10⁶) = 78.498 ; π(10⁷) = 664.579 ; π(10⁸) = 5.761.455 ; π(10⁹) = 50.847.534  
π(10¹⁰) = 455.052,511 ; π(10¹¹) = 4.118.054.813 ; π(10¹²) = 37.607.912.018

2.4 Reglas de divisibilidad

Nro	Regla	Ejemplo
1	Todos los n��meros	5: porque si divides 5:1=5 y ese n��mero es un m��ltiplo o divisor de cualquier n��mero.
2	El n��mero termina en una cifra par.	378: porque la ��ltima cifra (8) es par.
3	La suma de sus cifras es un m��ltiplo de 3.	480: porque 4+8+0=12 es m��ltiplo de 3.
4	Sus ��ltimos dos d��gitos son 0 o un m��ltiplo de 4.	300 y 516 son divisibles entre 4 porque terminan en 00 y en 16, respectivamente, siendo este ��ltimo un m��ltiplo de 4 (16=4*4).
5	La ��ltima cifra es 0 o 5.	485: porque termina en 5.
7	Un n��mero es divisible entre 7 cuando, al separar la ��ltima cifra de la derecha, multiplicarla por 2 y restarla de las cifras restantes la diferencia es igual a 0 o es un m��ltiplo de 7. Otro sistema: Si la suma de la multiplicaci��n de los n��meros por la serie 2,3,1,-2,-3,-1... da 0 o un m��ltiplo de 7.	34349: separamos el 9, y lo duplicamos (18), entonces 3434-18=3416. Repetimos el proceso separando el 6 (341'6) y duplic��ndolo (12), entonces 341-12=329, y de nuevo, 32'9, 9*2=18, entonces 32-18=14; por lo tanto, 34349 es divisible entre 7 porque 14 es m��ltiplo de 7. Ejemplo m��ltodo 2: 34349: [(2*3)+(3*4)+(1*3)-(2*4)-(3*9)]= 6+12+3-8-27 = -14.8
8	Para saber si un n��mero es divisible entre 8 hay que comprobar que sus tres ��ltimas cifras sean divisibles entre 8. Si sus tres ��ltimas cifras son divisibles entre 8 entonces el n��mero tambi��n es divisible entre 8.	Ejemplo: El n��mero 571.328 es divisible por 8 ya que sus ��ltimas tres cifras (328) son divisibles por 8 (32 = 8*4 y 8 = 8*1). Realizando la divisi��n comprobamos que 571.328 : 8 = 71.416

Contin��a		
Nro	Regla	Ejemplo
9	Un n��mero es divisible por 9 cuando al sumar todas sus cifras el resultado es m��ltiplo de 9.	504: sumamos 5+0+4=9 y como 9 es m��ltiplo de 9 504 es divisible por 9 5346: sumamos 5+3+4+6=18 y como 18 es m��ltiplo de 9, 5346 es divisible por 9.
10	La ��ltima cifra es 0.	4680: porque termina en 0
11	Sumando las cifras (del n��mero) en posici��n impar por un lado y las de posici��n par por otro. Luego se resta el resultado de ambas sumas obtenidas. Si el resultado es cero o un m��ltiplo de 11, el n��mero es divisible entre este. Si el n��mero tiene solo dos cifras y estas son iguales ser�� m��ltiplo de 11.	42702: 4+7+2=13 �� 2+0=2 �� 13-2=11 ��S 42702 es m��ltiplo de 11. 66: porque las dos cifras son iguales. Entonces 66 es m��ltiplo de 11.
13	Un n��mero es divisible entre 13 cuando, al separar la ��ltima cifra de la derecha, multiplicarla por 9 y restarla de las cifras restantes la diferencia es igual a 0 o es un m��ltiplo de 13	3822: separamos el ��ltimo dos (382'2) y lo multiplicamos por 9, 2��9=18, entonces 382-18=364. Repetimos el proceso separando el 4 (36'4) y multiplic��ndolo por 9, 4��9=36, entonces 36-36=0; por lo tanto, 3822 es divisible entre 13.
17	Un n��mero es divisible entre 17 cuando, al separar la ��ltima cifra de la derecha, multiplicarla por 5 y restarla de las cifras restantes la diferencia es igual a 0 o es un m��ltiplo de 17	2142: porque 214'2, 2*5=10, entonces 214-10=204, de nuevo, 20'4, 4*5=20, entonces 20-20=0; por lo tanto, 2142 es divisible entre 17.
19	Un n��mero es divisible entre 19 si al separar la cifra de las unidades, multiplicarla por 2 y sumar a las cifras restantes el resultado es m��ltiplo de 19.	3401: separamos el 1, lo doblamos (2) y sumamos 340+2= 342, ahora separamos el 2, lo doblamos (4) y sumamos 34+4=38 que es m��ltiplo de 19, luego 3401 tambi��n lo es.

Continúa		
Nro	Regla	Ejemplo
20	Un número es divisible entre 20 si sus dos últimas cifras son ceros o múltiplos de 20. Cualquier número par que tenga uno o más ceros a la derecha, es múltiplo de 20.	57860: Sus 2 últimas cifras son 60 (Que es divisible entre 20), por lo tanto 57860 es divisible entre 20.
23	Un número es divisible entre 23 si al separar la cifra de las unidades, multiplicar por 7 y sumar las cifras restantes el resultado es múltiplo de 23.	253: separamos el 3, lo multiplicamos por 7 y sumamos 25+21= 46, 46 es múltiplo de 23 así que es divisible entre 23.
25	Un número es divisible entre 25 si sus dos últimas cifras son 00, o en múltiplo de 25 (25,50,75,...)	650: Es múltiplo de 25 por lo cual es divisible. 400 también sería divisible entre 25.
29	Un número es divisible entre 29 cuando, al separar la última cifra de la derecha, multiplicarla por 3 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 29	2436: separamos el 6 (243'6) y lo multiplicamos por 3, 6*3=18, entonces 243-18=225. Repetimos el proceso separando el 5 (22'5) y multiplicándolo por 3, 5*3=15, entonces 22-15=7, que no es divisible entre 29.

2.5 Coprimos

Son aquellos números enteros *a* y *b* cuyo único factor en común que tienen es 1. Equivalentemente son coprimos, si, y solo si, su máximo común divisor (MCD) es igual a 1. Dos números coprimos no tienen por qué ser primos absolutos de forma individual. 14 y 15 son compuestos, sin embargo son coprimos, pues: *gcd*(14,15) = 1

2.6 Primos

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4
5 #define forn(i, n) for (int i = 0; i < n; i++)
6 const int MAXN = 1e5+1;
7
8 // Si la criba[n] == 0 => n es primo
9 // si no criba[n] es el divisor mas grande de n
10 #define MAXP 100000 //no necesariamente primo
11 int criba[MAXN+1];
12 void crearcriba(){
13     int w[] = {4,2,4,2,4,6,2,6};
14     for(int p=25;p<=MAXP;p+=10) criba[p]=5;
15     for(int p=9;p<=MAXP;p+=6) criba[p]=3;
```

```
16     for(int p=4;p<=MAXP;p+=2) criba[p]=2;
17     for(int p=7,cur=0;p<=MAXP;p+=w[cur++&7]) if (!criba[p])
18         for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
19 }
20
21 vector<int> primos;
22 void buscarprimos(){
23     crearcriba();
24     for(int i = 2; i < MAXP+1; i++) if (!criba[i]) primos.push_back(i);
25 }
26
27 // O(sqrt(n))
28 // para buscar los divisores de un unico numero
29 vector<int> divisors(int n) {
30     vector<int> div;
31     for (int d = 1; d * d <= n; d++) {
32         if (n%d == 0) {
33             div.push_back(d);
34             div.push_back(n/d);
35         }
36         if (d*d == n) div.pop_back(); // para cuadrados perfectos
37     }
38     sort(div.begin(), div.end());
39     return div;
40 }
41
42 // precalcular divisores
43 // O(MAXN * lg(MAXN))
44 void pre_div() {
45     vector<int> sum_div;
46     vector<int> divs[MAXN];
47     forn(i, MAXN) for(int j = i; j < MAXN; j+=i) {
48         sum_div[j] += i;
49         divs[j].push_back(i);
50     }
51 }
52
53 // factorizar en O(sqrt(n))
54 map<ll, ll> fact_sqrt(int n) {
55     map<ll, ll> F;
56     for (int p = 2; p * p <= n; p++) {
57         while (n%p == 0) {
58             F[p]++;
59             n /= p;
60         }
61     }
62     if (n > 1) F[n]++;
63     return F;
64 }
65
66 // factoriza bien numeros hasta MAXP^2
67 // O (cant primos)
68 // esta se podria hacer con la otra criba
69 map<ll,ll> fact(ll n) {
70     map<ll,ll> ret; // factor, cantidad de veces
71     for(auto p : primos){
```

```

72     while(!(n % p)){ // mientras no sea divisor?
73         ret[p]++; //divisor found
74         n /= p;
75     }
76 }
77 if(n > 1) ret[n]++;
78 return ret;
79 }
80
81 // factoriza bien numeros hasta MAXP
82 map<ll,ll> fact2(ll n){ //O (lg n)
83     map<ll,ll> ret;
84     while (criba[n]){
85         ret[criba[n]]++;
86         n/=criba[n];
87     }
88     if(n>1) ret[n]++;
89     return ret;
90 }

```

## 2.7 Combinatoria

**\*\* Permutaciones \*\***

el orden importa

¿Cuántas formas hay de ordenar n elementos?

$$P(n) = n!$$

**\*\* Permutaciones tomadas de a r \*\***

El número de permutaciones de n elementos tomados de r en r se calcula con la siguiente fórmula:

$$P(n, r) = \frac{n!}{(n-r)!}$$

Si tienes 3 objetos diferentes (n=3) y quieres formar arreglos de 2 objetos (k=2), las k-permutaciones serán:

Objetos: {A, B, C}

Arreglos de k=2: {AB, BA, AC, CA, BC, CB}

**\*\* Combinaciones \*\***

¿Cuántas formas hay de elegir k elementos entre n posibles?

$$C \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

## 2.8 Operaciones con modulo

```

1  typedef long long ll;
2  const int MAXN = 1e7, M = 1e9+7;
3
4  ll expmod(ll b, ll e){
5      ll ans = 1;
6      while(e){
7          if(e&1) ans = ans*b %M;
8          b = b*b %M; e >>= 1;
9      }
10     return ans;
11 }
12
13 ll invmod(ll a){ return expmod(a, M-2); }
14
15 // Inversos modulares
16 int INV[MAXN];
17 INV[1] = 1;
18 for(ll a = 2; a < MAXN; a++) INV[a] = M - (ll)(M/a)*INV[M%a]%M;
19
20 // Factoriales
21 ll F[MAXN];
22 F[0] = 1;
23 for(ll i = 1; i < MAXN; i++) F[i] = F[i-1]*i %M;
24
25 // Factorial inverso
26 int FI[MAXN]
27 FI[0] = 1;
28 for(int i = 1; i < MAXN; i++) FI[i] = FI[i-1]*INV[i] %M
29
30 // Combinaciones
31 ll Comb(ll n, ll k){
32     if(n < k) return 0;
33     return F[n]*FI[k] %M *FI[n-k] %M;
34 }

```

3 Estructuras

3.1 vector

FunciÃ³n	ExplicaciÃ³n	O
(constructor)	Construct vector	O(n)
(destructor)	Vector destructor	O(n)
operator=	Assign content	O(n)
begin	Return iterator to beginning	O(1)
end	Return iterator to end	O(1)
rbegin	Return reverse iterator to reverse beginning	O(1)
rend	Return reverse iterator to reverse end	O(1)
cbegin	Return const_iterator to beginning	O(1)
cend	Return const_iterator to end	O(1)
crbegin	Return const_reverse_iterator to reverse beginning	O(1)
crend	Return const_reverse_iterator to reverse end	O(1)
size	Return size	O(1)
resize	Change size	O(n)
empty	Test whether vector is empty	O(1)
operator[]	Access element	O(1)
at	Access element	O(1)
front	Access first element	O(1)
back	Access last element	O(1)
assign	Assign vector content	O(n)
push_back	Add element at the end	O(1)
pop_back	Delete last element	O(1)
insert	Insert elements	O(n)
erase	Erase elements	O(n)
swap	Swap content	O(1)
clear	Clear content	O(n)
emplace	Construct and insert element	O(1)
emplace_back	Construct and insert element at the end	O(1)

3.1.1 emplace

```
1 vector<int> myvector = {10,20,30};
2
3 auto it = myvector.emplace ( myvector.begin()+1, 100 );
4 myvector.emplace ( it, 200 );
5 myvector.emplace ( myvector.end(), 300 );
6
7 for (auto& x: myvector)
8     cout << ' ' << x;
9 // 10 200 100 20 30 300
```

3.1.2 resize

Resizes the container so that it contains n elements.

```
void resize(size_type n, const value_type& val);

1 vector<int> myvector;
2 for (int i=1; i<10; i++) myvector.push_back(i);
3
4 myvector.resize(5);
5 myvector.resize(8, 100);
6 myvector.resize(12);
7
8 for (int i=0; i<myvector.size(); i++)
9     cout << ' ' << myvector[i];
10 // 1 2 3 4 5 100 100 100 0 0 0 0
```

3.1.3 assign

Assigns new contents to the vector, replacing its current contents, and modifying its size accordingly.

```
void assign(size_type n, const value_type& val);

1 vector<int> first;
2 vector<int> second;
3 vector<int> third;
4
5 first.assign (7,100); // 7 ints with a value of 100
6
7 vector<int>::iterator it;
8 it=first.begin()+1;
9
10 second.assign (it,first.end()-1); // the 5 central values of first
11
12 int myints[] = {1776,7,4};
13 third.assign (myints,myints+3); // assigning from array.
14
15 first.size() // -> 7
16 second.size() // -> 5
17 third.size() // -> 3
```



3.2 unordered set

FunciÃşn	ExplicaciÃşn	O
(constructor)	Construct unordered_set	-
(destructor)	Destroy unordered_set	-
operator=	Assign content	O(n)
empty	Test whether container is empty	O(1)
size	Return container size	O(1)
max_size	Return maximum size	O(1)
begin	Return iterator to beginning	O(1)
end	Return iterator to end	O(1)
find	Get iterator to element	O(n)
count	Count elements with a specific key Returns 0 or 1	O(n)
equal_range	Get range of elements with a specific key	O(n)
emplace	Construct and insert element	O(n)
emplace_hint	Construct and insert element with hint	O(n)
insert	Insert elements	O(n)
erase	Erase elements	O(n)
clear	Clear content	O(n)
swap	Swap content	O(1)
reserve	Request a capacity change	O(n)
key_eq	Get key equivalence predicate	O(1)

```
1 unordered_set<int> s;  
2 s.insert(3);  
3 s.insert(5);  
4 s.erase(3);  
5 s.count(3); // -> 0  
6 s.count(5); // -> 1
```

3.3 Iterators

```
1 sort(v.begin(), v.end());  
2 reverse(v.begin(), v.end());  
3 random_shuffle(v.begin(), v.end());  
4  
5 sort(v.begin(), v.end(), sortbysec);
```

sort complexity =  $O(N\log_2N)$ .

Ordenar un vector de pair por su segunda componente

```
1 vector<pair<int, int>> v;  
2  
3 bool sortbysec(const pair<int,int> &a, const pair<int,int> &b){  
4     return (a.second < b.second);  
5 }
```

TambiÃşn es posible ordenar un array normal:

```
1 sort(a, a+n);  
2 reverse(a, a+n);  
3 random_shuffle(a, a+n)
```

3.4 Indexed Set

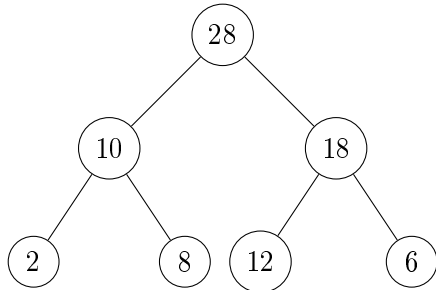
```
1 #include <bits/stdc++.h>  
2 #include <ext/pb_ds/assoc_container.hpp>  
3 #include <ext/pb_ds/tree_policy.hpp>  
4 using namespace __gnu_pbds;  
5 using namespace std;  
6  
7 template<typename T>  
8 using indexed_set = tree<T, null_type, less<T>, rb_tree_tag,  
9     tree_order_statistics_node_update>;  
10 //indexed_set<int> s;  
11 //s.insert(5);  
12 //s.insert(1);  
13 //s.insert(10);  
14 //s.insert(3);  
15  
16 // Elementos ordenados: [1, 3, 5, 10]  
17 //cout << *s.find_by_order(0) << "\n"; // 1
```

```

18 //cout << *s.find_by_order(2) << "\n"; // 5
19
20 //cout << s.order_of_key(5) << "\n"; // 2 (hay dos menores que 5)
21 //cout << s.order_of_key(6) << "\n"; // 3 (1,3,5 < 6)
22 //s.insert(8);
23 //cout << *s.find_by_order(4) << "\n";
24 //s.erase(3);
25 //cout << s.order_of_key(5) << "\n"; // 1 (quedaron 1 y 5 y 10)

```

### 3.5 Segment Tree



```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define dforn(i,n) for(int i=n-1; i>=0; i--)
5 #define fora(p, i, n) for(int i = p; i < n; i++)
6 #define forn(i, n) for(int i = 0; i < n; i++)
7 #define fori(i, n) for(int i = n - 1; i <= 0; i--)
8 #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
9 #define MAXN 200000
10 #define operacion(x, y) make_pair(max(x.first, x.second + y.first), x.
    second + y.second)
11 #define neutro make_pair(0,0)
12 #define tipo pair<ll, ll>
13
14 struct RMQ{
15     int sz;
16     tipo t[4*MAXN];
17     tipo &operator[](int p){return t[sz+p];}
18     void init(int n){//0(nlgn)
19         sz = 1 << (32-__builtin_clz(n));
20         forn(i, 2*sz) t[i] = neutro;
21     }
22     void updall(){
23         dforn(i, sz) t[i] = operacion(t[2*i], t[2*i+1]);
24     }
25     tipo get(int i, int j){return get(i, j, 1, 0, sz);}
26     tipo get(int i, int j, int n, int a, int b){
27         if(j<=a || i>=b) return neutro;
28         if(i<=a && b<=j) return t[n];
29         int c = (a+b)/2;
30         return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));

```

```

31     }
32     void set(int p, tipo val){
33         for(p+=sz; p>0 && t[p] != val;){
34             t[p]=val;
35             p/=2;
36             val = operacion(t[2*p],t[2*p+1]);
37         }
38     }
39 }mx;
40
41 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
    sobre el rango [i, j).
42 typedef int Elem; //Elem de los elementos del arreglo
43 typedef int Alt; //Elem de la alteracion
44 #define operacion(x,y) x+y
45 const Elem eneutro=0; const Alt aneutro=0;
46 #define MAXN 100000
47 struct RMQ{
48     int sz;
49     Elem t[4*MAXN];
50     Alt dirty[4*MAXN]; //las alteraciones pueden ser de distinto Elem
51     Elem &operator[](int p){return t[sz+p];}
52     void init(int n){//0(nlgn)
53         sz = 1 << (32-__builtin_clz(n));
54         forn(i, 2*sz) t[i]=eneutro;
55         forn(i, 2*sz) dirty[i]=aneutro;
56     }
57     void updall(){
58         dforn(i, sz) t[i] = operacion(t[2*i], t[2*i+1]);
59     }
60     void push(int n, int a, int b){//propaga el dirty a sus hijos
61         if(dirty[n]!=aneutro){
62             t[n]+=dirty[n]*(b-a); //altera el nodo
63             if(n<sz){
64                 dirty[2*n]+=dirty[n];
65                 dirty[2*n+1]+=dirty[n];
66             }
67             dirty[n]=aneutro;
68         }
69     }
70     Elem get(int i, int j, int n, int a, int b){//0(lgn)
71         if(j<=a || i>=b) return eneutro;
72         push(n, a, b); //corrige el valor antes de usarlo
73         if(i<=a && b<=j) return t[n];
74         int c=(a+b)/2;
75         return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
76     }
77     Elem get(int i, int j){return get(i,j,1,0,sz);}
78     //altera los valores en [i, j) con una alteracion de val
79     void alterar(Alt val, int i, int j, int n, int a, int b){//0(lgn)
80         push(n, a, b);
81         if(j<=a || i>=b) return;
82         if(i<=a && b<=j){
83             dirty[n]+=val; //ver esto gordo
84             push(n, a, b);
85             return;

```

```
86     }
87     int c=(a+b)/2;
88     alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
89     t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
90 }
91 void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
92 }rmq;
93
94 void inicializate(){
95     vector<Elem> v;
96     mx.init(v.size()); forn(i, v.size()) rmq[i] = v[i]; mx.updall();
97 }
```

3.6 Array

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int dp[50][50];
6     // llenar una lista para DP
7     memset(dp,-1, sizeof(dp));
8
9     for(int i=0; i<50;i++){
10         for(int j=0; j<50;j++){
11             cout << dp[i][j] << " ";
12         }
13         cout << endl;
14     }
15
16 }
```

3.7 BitSet

FunciÃşn	ExplicaciÃşn	O
Template parameters		
N	nÃşmero de bits	O(1)
Member functions		
(constructor)	construye el ‘bitset’	O(N)
Element access		
operator[]	accede a un bit especÃşfico	O(1)
all, any, none	todos, algÃşn o ningÃşn bit estÃş en true	O(N)
count	nÃşmero de bits establecidos en true	O(N)
size	nÃşmero de bits que contiene	O(1)
Modifiers		
operator&=		
operator =		
operator^=		
operator~=		
operator«=		
operator»=		
operator«		
operator»		O(N)
set()		O(N)
set(int pos)	pone todos los bits en el valor dado	O(1)
reset()		O(N)
reset(int pos)	establece bits en ‘false’	O(1)
flip()		O(N)
flip(int pos)	alterna los valores de los bits	O(1)
Conversions		
to_string	representaciÃşn en cadena	

## 4 Flujo

### 4.1 Dinic

```

1 #include <bits/stdc++.h>
2 #define forr(i, a, b) for (int i = (a); i < (b); i++)
3 #define forn(i, n) forr(i, 0, n)
4 #define dforn(i, n) for (int i = (n) - 1; i >= 0; i--)
5 #define forall(it,v) for(auto it=v.begin(); it!=v.end(); ++it)
6 #define pb push_back
7 #define sz(v) ((int)v.size())
8 #define INF 1e18
9
10 using namespace std;
11 typedef long long ll;
12
13 // ojo con el max
14 const int MAX = 501;
15 // Corte minimo: vertices con dist[v]>=0 (del lado de src) VS. dist[v]
16 //   ]=-1 (del lado del dst)
17 // Para el caso de la red de Bipartite Matching (Sean V1 y V2 los
18 //   conjuntos mas proximos a src y dst respectivamente):
19 // Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de
20 //   V2 con it->f>0, es arista del Matching
21 // Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con
22 //   dist[v]>0
23 // Max Independent Set: tomar los vertices NO tomados por el Min Vertex
24 //   Cover
25 // Max Clique: construir la red de G complemento (debe ser bipartito!) y
26 //   encontrar un Max Independent Set
27 // Min Edge Cover: tomar las aristas del matching + para todo vertices no
28 //   cubierto hasta el momento, tomar cualquier arista de el
29
30 int nodes, src, dst;
31 int dist[MAX], q[MAX], work[MAX];
32 struct Edge {
33     int to;
34     int rev; // indice en en la red residual
35     ll f; // flow
36     ll cap; // capacity
37     Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(cap) {}
38 };
39
40 vector<Edge> G[MAX];
41 void addEdge(int s, int t, ll cap){
42     G[s].pb(Edge(t, sz(G[t]), 0, cap));
43     G[t].pb(Edge(s, sz(G[s])-1, 0, 0));
44 }
45
46 bool dinic_bfs(){
47     fill(dist, dist+nodes, -1), dist[src]=0;
48     int qt=0; q[qt++]=src;
49     for(int qh=0; qh<qt; qh++){
50         int u = q[qh];
51         forall(e, G[u]){

```

```

52             int v=e->to;
53             if(dist[v]<0 && e->f < e->cap)
54                 dist[v]=dist[u]+1, q[qt++]=v;
55         }
56     }
57     return dist[dst]>=0;
58 }
59
60 ll dinic_dfs(int u, ll f){
61     if(u == dst) return f;
62     for(int &i=work[u]; i<sz(G[u]); i++){
63         Edge &e = G[u][i];
64         if(e.cap <= e.f) continue;
65         int v = e.to;
66         if(dist[v]==dist[u]+1){
67             ll df = dinic_dfs(v, min(f, e.cap-e.f));
68             if(df>0){
69                 e.f+=df, G[v][e.rev].f-= df;
70                 return df;
71             }
72         }
73     }
74     return 0;
75 }
76
77 ll maxFlow(int _src, int _dst){
78     src=_src, dst=_dst;
79     ll result = 0;
80     while(dinic_bfs()){
81         fill(work, work+nodes, 0);
82         while(ll delta=dinic_dfs(src,INF))
83             result += delta;
84     }
85     // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1
86     //   forman el min-cut
87     return result;
88 }
89
90 bool vis[501][501];
91 void corteMinimo(){
92     forn(i, nodes){
93         if(dist[i] >= 0){
94             for(auto j : G[i]){
95                 if(dist[j.to] == -1 && !vis[i][j.to]){
96                     vis[i][j.to] = true;
97                     cout << i+1 << " " << j.to+1 << endl;
98                 }
99             }
100         }
101     }
102 }
103
104 int capa1, capa2, aristas;
105 void matchingMaximo(){
106     forr(i,1, capa1+1){
107         for(auto j : G[i]){
108             if(j.f > 0){

```

```

99         cout << i << " " << j.to - capai << endl;
100     }
101 }
102 }
103 }
104 vector<int> path;
105 void recrearUnCamino(int i){ //solo te recrea un camino del flujo
106     path.push_back(i);
107     if(i == nodes-1) return;
108     for(auto j: G[i]){
109         if(j.f>0 && !vis[i][j.to]){
110             vis[i][j.to] = true;
111             path.push_back(j.to+1);
112             recrearUnCamino(j.to);
113             break;
114         }
115     }
116 }
117
118 int main() {
119     // Example usage
120     int m;
121     cin >> nodes >> m;
122     forn(i, m){
123         int a, b, c;
124         cin >> a >> b >> c;
125         addEdge(a-1, b-1, (ll)c);
126     }
127     ll flow = maxFlow(0, nodes-1);
128     cout << flow << endl;
129     return 0;
130 }
131 }

```

## 4.2 Hungarian

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define forr(i,a,b) for(int i =(a);i<int(b); ++i)
5 #define forn(i,n) forr(i,0,n)
6 #define dforn(i,a,b) for(int i =(b)-1;i>=int(a);--i)
7 #define dforn(i,n) dforn(i,0,n)
8 #define db(v) cerr << #v << " = " << (v) << "\n";
9 #define sz(v) (int((v).size()))
10 #define all(v) (v).begin(), (v).end()
11 #define pb push_back
12 #define pp pop_back
13 #define fst first
14 #define snd second
15
16 using ll = long long;
17 using ull = unsigned ll;
18 using ld = long double;

```

```

19 using pii = pair<int,int>;
20 using pll = pair<ll,ll>;
21 using vll = vector<ll>;
22
23 const ll MAXN = 110;
24 //const ll INF = 1e18 +100;
25 const ll MOD = 1e9+7;
26 const ld EPS = 1e-9;
27 const ld PI = acos(-1);
28
29 //matching minimo con pesos en las aristas pai
30 // Minima asignacion en O(n^3)
31 // Lado izquierdo [0..n), lado derecho [0..m). OJO: n <= m
32 typedef long double td; typedef vector<int> vi; typedef vector<td> vd;
33 const td INF = 1e100; // Para max asignacion, INF = 0, y negar costos
34 bool zero(td x) {return fabs(x) < 1e-9;} // Para int/ll: return x==0;
35 struct Hungarian{
36     vector<pii> ans; //Guarda las aristas usadas en la asignacion (
37         cada arista es [0..n)x[0..m))
38     int n; vector<vd> cs; vi L, R;
39     Hungarian(int N, int M) : n(max(N,M)), cs(n,vd(n)), L(n), R(n){
40         forn(x, N) forn(y, M) cs[x][y] = INF;
41     }
42     void set(int x, int y, td c) { cs[x][y] = c; }
43     td assign(){
44         int mat = 0; vd ds(n), u(n), v(n); vi dad(n), sn(n);
45         forn(i, n) u[i] = *min_element(all(cs[i]));
46         forn(j, n){
47             v[j] = cs[0][j]-u[0];
48             forr(i, 1, n) v[j] = min(v[j], cs[i][j] - u[i]);
49         }
50         L = R = vi(n, -1);
51         forn(i, n) forn(j, n) if(R[j] == -1 and zero(cs[i][j] - u[i] - v[j]))
52             {
53                 L[i] = j; R[j] = i; mat++; break;
54             }
55         for(; mat < n; mat++){
56             int s = 0, j = 0, i;
57             while(L[s] != -1) s++;
58             fill(all(dad), -1); fill(all(sn), 0);
59             forn(k, n) ds[k] = cs[s][k]-u[s]-v[k];
60             while(true){
61                 j = -1;
62                 forn(k, n) if(!sn[k] and (j == -1 or ds[k] < ds[j])) j = k;
63                 sn[j] = 1; i = R[j];
64                 if(i == -1) break;
65                 forn(k, n) if(!sn[k]){
66                     auto new_ds = ds[j] + cs[i][k] - u[i]-v[k];
67                     if(ds[k] > new_ds) ds[k]=new_ds, dad[k]=j;
68                 }
69             }
70             forn(k, n) if(k!=j and sn[k]) { auto w = ds[k]-ds[j]; v[k] += w, u[
71                 R[k]] -= w; }
72             u[s] += ds[j];

```

```
71     while(dad[j] >= 0) { int d = dad[j]; R[j] = R[d]; L[R[j]] = j; j =  
72         d; }  
73     R[j] = s; L[s] = j;  
74 }  
75 td value = 0; forn(i, n) value += cs[i][L[i]], ans.pb({i, L[i]});  
76 return value;  
77 }  
78  
79 bool f(pii x, pii y){  
80     return x.snd < y.snd;  
81 }
```

## 5 Grafos

### 5.1 Recorrer Grafos

Dado un Grafo como lista de adjacencias

```
1 #include <bits/stdc++.h>  
2 #define MAXN 100000  
3 using namespace std;  
4  
5 vector<int> G[MAXN];  
6 bool visited[MAXN];
```

Podemos recorrerlo con DFS o con BFS.

#### 5.1.1 DFS

```
1 void dfs(int nodo){  
2     if(visited[nodo]) return;  
3     visited[nodo] = true;  
4  
5     for(auto it : G[nodo]){  
6         if(!visited[it]){  
7             // logic  
8         }  
9     }  
10 }
```

#### 5.1.2 BFS

```
1 void bfs(int nodo){  
2     cola.push(nodo);  
3  
4     while(!cola.empty()){  
5         Nodo actual = cola.front();  
6         cola.pop();  
7  
8         for (int vecino : G[actual]) {  
9             if (!visited[vecino]) {  
10                 visited[vecino] = true;  
11                 cola.push(vecino);  
12             }  
13         }  
14     }  
15 }
```

## 5.2 Camino Mximo

### 5.2.1 Bellman-Ford

El algoritmo de Bellman-Ford encuentra el camino desde un nodo de origen a todos los nodos del grafo.

**Complejidad =  $O(nm)$**

```

1 #include <bits/stdc++.h>
2 #define INF 100000000
3 using namespace std;
4
5 vector<tuple<int, int, int>> edges;
6 int n;
7
8 #define op min
9
10 // acepta negativos
11 // se puede cambiar a buscar el camino maximo cambiando
12 // min por max
13 void bellman_ford(int x){
14     vector<int> dist(n, INF);
15     dist[x] = 0;
16
17     for (int i = 1; i <= n-1; i++) {
18         for (auto e : edges) {
19             int a, b, w;
20             tie(a, b, w) = e;
21             dist[b] = min(dist[b], dist[a]+w);
22         }
23     }
24
25     // Buscar ciclos negativos
26     // Un ciclo que se reduce infinitamente
27     unordered_set<int> in_cycle;
28     for (auto [a, b, w] : edges) {
29         if (dist[a]+w < dist[b]) {
30             in_cycle.insert(b);
31         }
32         dist[b] = min(dist[b], dist[a]+w);
33     }
34 }
```

### 5.2.2 Ciclos negativos

El algoritmo es capaz de detectar ciclos negativos. Para eso Se debe correr una vez m'as

### 5.2.3 Dijkstra

El algoritmo necesita que todos los pesos sean 0

**Complejidad =  $O(n + m \log m)$**

```

1 #include <bits/stdc++.h>
2 #define MAXN 100000
3 using namespace std;
4 typedef pair<int, int> ii;
5
6 vector<pair<int, int>> G[MAXN]; //Lista de pares, dest, peso
7 bool visited[MAXN];
8 int n;
9
10 void dijkstra(int x){
11     // La PQ esta ordenada de menor a mayor
12     priority_queue<ii, vector<ii>, greater<ii>> q;
13     vector<int> distance(n, MAXN);
14     distance[x] = 0; // x es el punto de inicio
15     q.push({0, x});
16
17     while (!q.empty()) {
18         int a = q.top().second; q.pop();
19         if (visited[a]) continue;
20         visited[a] = true;
21
22         for (auto [b, w] : G[a]) {
23             if (distance[a]+w < distance[b]) {
24                 distance[b] = distance[a]+w;
25                 q.push({distance[b], b});
26             }
27         }
28     }
29 }
```

### 5.2.4 Floyd-Warshall

```

1  /* minima distancia entre cada par de nodos en un grafo dirigido.
2     O(n^3)
3  */
4  /*
5  int dist[MAX_N][MAX_N]; //Distancia de i a j
6
7  // llenar la matriz de adjayencia
8  // tmb se podria armar a medida que lees la entrada
9  for (int i = 1; i <= n; i++) {
10     for (int j = 1; j <= n; j++) {
11         if (i == j) dist[i][j] = 0;
12         else if (adj[i][j]) dist[i][j] = adj[i][j];
13         else dist[i][j] = INF;
14     }
15 }
16
17 /* After this, the shortest dists can be found as follows: */
18 for (int k = 1; k <= n; k++) {
19     for (int i = 1; i <= n; i++) {
20         for (int j = 1; j <= n; j++) {
21             dist[i][j] = min(dist[i][j], dist[i][k]+dist[k][j]);
22         }
23     }
24 }

```

## 5.3 Spanning Tree

### 5.3.1 UnionFind

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct UnionFind{
5      vector<int> f; //the array contains the parent of each node
6      void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
7      int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));} //O(1)
8      bool join(int i, int j) {
9          bool con=comp(i)==comp(j);
10         if(!con) f[comp(i)] = comp(j);
11         return con;
12     }
13 } uf;

```

Tambien podemos guardar el tamaño de los sets:

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct UnionFind{
5      vector<int> f; //the array contains the parent of each node
6      vector<int> size;
7      int largest;
8      void init(int n){
9          f.clear(); f.insert(f.begin(), n, -1);
10         largest = 1;
11         size.insert(size.begin(), n, 1);
12     }
13     int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));} //O(1)
14     bool join(int i, int j) {
15         int rootI = comp(i);
16         int rootJ = comp(j);
17
18         if (rootI == rootJ) return true; // Ya estan en el mismo
19         componente
20
21         if (size[rootI] < size[rootJ]) {
22             swap(rootI, rootJ);
23         }
24
25         f[rootJ] = rootI;
26         size[rootI] += size[rootJ];
27         largest = max(largest, size[rootI]); // Actualizar la componente
28         mas grande
29         return false;
30     }
31     bool same(int a, int b) {
32         return comp(a) == comp(b);
33     }
34 } uf;

```



### 5.3.2 Kruskal

Obtener el spanning tree de costo minimo dado un grafo ponderado no dirigido

```

1 #include <bits/stdc++.h>
2 #include "union_find.h"
3 using namespace std;
4
5 int n, m;
6 struct Ar{
7     int a, b, w;
8 };
9 vector<Ar> E;
10 bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
11
12 int edges_used = 0;
13 int kruskal(){
14     int cost=0;
15     sort(E.begin(), E.end()); //Ordenar aristas de menor a mayor
16     uf.init(n);
17     for(Ar it : E){
18         if(uf.comp(it.a) != uf.comp(it.b)){ //Si no estan conectados
19             uf.join(it.a, it.b); //Conectar
20             cost += it.w;
21             edges_used++;
22         }
23     }
24     return cost;
25 }

```

## 5.4 Aplicaciones comunes

### 5.4.1 Chequear si es conexo

Tirar un dfs desde un nodo cualquiera v es conexo si alcanzamos todos los otros nodos del grafo

### 5.4.2 Bipartito

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4 #define DBG(x) cerr << #x << " : " << x << endl;
5 #define MAXN 1000000
6
7 /* Comprobar si el grafo es bipartito
8  * BFS pintando intercaladamente
9  */
10
11 int main(){
12     int n, m;
13     // Leer el grafo
14     vector<vector<int>> G(n);
15
16     queue<int> q;
17     vector<int> color(n, -1);
18
19     forn(i, n){
20         if (color[i] != -1) continue;
21
22         q.push(i); color[i] = 0;
23         while(!q.empty()){
24             int n = q.front(); q.pop();
25
26             for (int adj : G[n]) {
27                 if (color[adj] == -1){
28                     color[adj] = color[n] ^ 1; // invertir, si es 1 => 0,
29                     0 => 1
30                     q.push(adj);
31                 }else if (color[adj] == color[n]){
32                     cout << "IMPOSSIBLE\n"; // no es bipartito
33                     return 0;
34                 }
35             }
36         }
37
38         forn(i, n) {
39             cout << color[i] + 1 << " ";
40         }
41         cout << '\n';
42
43         return 0;
44 }

```

### 5.4.3 Ciclos

```

1 // https://cses.fi/problemset/task/1669/
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define forn(i, n) for(int i = 0; i < n; i++)
5 #define DBG(x) cerr << #x << " : " << x << endl;
6 #define MAXN 1000000
7
8 vector<vector<int>> G;
9 vector<bool> visited;
10 vector<int> parent;
11
12 // buscamos algun ciclo
13 int dfs(int nodo, int prev = 0){
14     parent[nodo] = prev;
15     if(visited[nodo]) return nodo;
16     visited[nodo] = true;
17
18     for(int adj : G[nodo]){
19         if (adj == prev) continue; // Para no volver para atras
20         int cycle = dfs(adj, nodo);
21         if (cycle != -1) return cycle; // si hay ciclo
22     }
23     return -1; // si no ahay ciclo devuelve -1
24 }
25
26 int main(){
27     ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
28
29     int n, m;
30
31     // leer grafo
32
33     int cycle_start = -1;
34     forn(i, n){
35         if (visited[i]) continue; // si ya esta visitado, ya forma parte
36             d un ciclo
37         cycle_start = dfs(i);
38         if (cycle_start == -1){
39             cout << "IMPOSSIBLE\n";
40             return 0;
41         }
42     }
43
44     // reconstruir camino
45     vector<int> path;
46     int next = parent[cycle_start];
47     path.push_back(cycle_start+1);
48     while(next != cycle_start){
49         path.push_back(next+1);
50         next = parent[next];
51     }
52     path.push_back(cycle_start+1);
53
54     // mostrar path

```

54 }

### 5.4.4 Componentes de un grafo

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4 #define MAXN 1000000
5
6 /* Buscar todas las componentes del grafo
7  * De cada componente agarrar un elemento cualquiera
8  * */
9
10 vector<int> G[MAXN];
11 int components[MAXN]; //Llenar en -1
12
13 int dfs(int n, int component=1){
14     if (components[n] != -1) return components[n];
15     components[n] = component;
16
17     for (auto adj : G[n]) {
18         if (components[n] == -1) {
19             dfs(adj, component);
20         }
21     }
22     return component;
23 }
24
25 int main(){
26     int n, m; //Leer el grafo
27
28     forn(i, n){
29         dfs(i, i);
30     }
31
32     return 0;
33 }

```

### 5.4.5 Dijkstra modificado

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4
5 typedef unsigned long long ll;
6 typedef tuple<ll, int, bool> e;
7
8 /* Dijkstra modificado
9  * Camino minimo pero tenes un coupon de descuento q usas una vez, q
10   reduce el precio a w/2
11  * Tenemos en cuenta dos estados diferentes para cada ciudad, uno
12   habiendo usado el coupon y otro sin

```

```

11 * Para cada vuelo, si no usamos el cupÃ³n, tenemos dos opciones:
12   - Usar el vuelo al costo original y no usar el cupÃ³n.
13   - Usar el cupÃ³n en ese vuelo, reduciendo su costo a â¬w/2â¬.
14 * */
15
16 int main(){
17     int n, m;
18     vector<vector<pair<int, ll>>> G(n);
19     vector<vector<bool>> visited(n, vector<bool>(2, false));
20
21     // leer el grafo
22     priority_queue<e, vector<e>, greater<e>> q;
23
24     vector<vector<ll>> distance(2, vector<ll>(n, LONG_LONG_MAX));
25     // distance[0][i] => sin usar el cupon, distance[1][i] => usando el
        cupon
26     distance[0][0] = 0;
27     q.push({0, 0, false}); // distance, v, si ya aplique el descuento
28
29     while (!q.empty()) {
30         auto [dist, a, discount] = q.top(); q.pop();
31         if (visited[a][discount]) continue;
32         visited[a][discount] = true;
33
34         for (auto [b, w] : G[a]) {
35             // Si ya aplique el descuento
36             if (distance[discount][a]+w < distance[discount][b]) {
37                 distance[discount][b] = distance[discount][a]+w;
38                 q.push({distance[discount][b], b, discount});
39             }
40             if (!discount){
41                 ll w2 = w/2;
42                 if (distance[0][a]+w2 < distance[1][b]) {
43                     distance[1][b] = distance[0][a]+w2;
44                     q.push({distance[1][b], b, true});
45                 }
46             }
47         }
48     }
49     cout << min(distance[0][n-1], distance[1][n-1]) << '\n';
50 }

```

#### 5.4.6 Max edge in path

Para grafos dirigidos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 1e5 + 1;
4 const int MAXM = 2e5 + 1;
5 const int LOGN = 18;
6
7 tuple<int, int, int> vet[MAXN];
8 int n, m, q, comp[MAXN], anc[MAXN][LOGN], val[MAXN][LOGN], nvl[MAXN];
9 unordered_map<int, int> cost[MAXN];

```

```

10 vector<int> adj[MAXN], custo[MAXN];
11
12 int find(int a){
13     return a == comp[a] ? a : comp[a] = find(comp[a]);
14 }
15
16 void merge(int a, int b){
17     comp[find(a)] = find(b);
18 }
19
20 void dfs(int v, int p){
21     for(int i = 0; i < (int)adj[v].size(); i++){
22         if(adj[v][i] != p){
23             int u = adj[v][i], c = custo[v][i];
24             nvl[u] = nvl[v]+1;
25             anc[u][0] = v;
26             val[u][0] = c;
27             for(int j = 1; j < LOGN; j++){
28                 anc[u][j] = anc[anc[u][j - 1]][j - 1];
29                 val[u][j] = max(val[u][j - 1], val[anc[u][j - 1]][j - 1]);
30             }
31             dfs(u,v);
32         }
33     }
34 }
35
36 int path_max(int u, int v){
37     if(nvl[u] < nvl[v])
38         swap(u,v);
39     int ret = 0;
40     for(int i = LOGN-1; i >= 0; i--){
41         if(nvl[u] - (1<<i) >= nvl[v]){
42             ret = max(ret, val[u][i]);
43             u = anc[u][i];
44         }
45     }
46     if(u != v){
47         for(int i = LOGN-1; i >= 0; i--){
48             if(anc[u][i] != anc[v][i]){
49                 ret = max(ret, val[u][i]);
50                 ret = max(ret, val[v][i]);
51                 u = anc[u][i];
52                 v = anc[v][i];
53             }
54         }
55         ret = max(ret, val[u][0]);
56         ret = max(ret, val[v][0]);
57     }
58     return ret;
59 }
60
61 int main(){
62     iota(comp, comp + MAXN, 0);
63     scanf("%d%d", &n, &m);
64     for(int i = 0; i < m; i++){
65         int u, v, c;

```

```

66     scanf("%d%d%d", &u, &v, &c);
67     u--, v--;
68     cost[u][v] = cost[v][u] = c;
69     vet[i] = make_tuple(c, u, v);
70 }
71 sort(vet, vet + m);
72
73 long long ans = 0;
74 for(int i = 0; i < m; i++){
75     int c, u, v;
76     tie(c, u, v) = vet[i];
77     if(find(u) != find(v)){
78         merge(u, v);
79         adj[u].emplace_back(v);
80         adj[v].emplace_back(u);
81         custo[u].emplace_back(c);
82         custo[v].emplace_back(c);
83         ans += c;
84     }
85 }
86 dfs(0, 0);
87 scanf("%d", &q);
88 while(q--){
89     int u, v;
90     scanf("%d%d", &u, &v);
91     u--, v--;
92     if(anc[u][0] == v || anc[v][0] == u){
93         printf("%lld\n", ans);
94         continue;
95     }
96     printf("%lld\n", ans - path_max(u, v) + cost[u][v]);
97 }
98 }

```

### 5.4.7 Max path

```

1 // Se resuelve con un Bellman-Ford

```

### 5.4.8 Strong Connectivity

Para grafos dirigidos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4
5 typedef long long ll;
6 const int MAXN = 1e5+1;
7
8 /* Checkear si el grafo es fuertemente conexo
9  * DFS para el grafo normal desde una arista cualquiera v (en este caso
10  * 1)

```

```

10  * DFS en el grafo invertido desde la misma artista
11  * Si existe arista sin visitar 'x' en algun dfs => "NO" v x
12  * */
13
14 int n, m;
15
16 vector<int> G[MAXN];
17 vector<int> G_inv[MAXN];
18 vector<bool> visited(MAXN, false);
19
20 void dfs(vector<int> *graph, int x){
21     visited[x] = true;
22     for(int i : graph[x]){
23         if(!visited[i]) dfs(graph, i);
24     }
25 }
26
27 int main(){
28     // - Leer grafo G y el grafo inverso G_inv
29
30     dfs(G, 0);
31     forn(i, n){
32         if (!visited[i]){
33             cout << "NO\n";
34             cout << i << ' ' << i+1 << endl;
35             return 0;
36         }
37     }
38     fill(visited.begin(), visited.end(), false);
39     dfs(G_inv, 0);
40     forn(i, n){
41         if (!visited[i]){
42             cout << "NO\n";
43             cout << i+1 << ' ' << i << endl;
44             return 0;
45         }
46     }
47
48     cout << "YES\n";
49     return 0;
50 }

```

### 5.4.9 Topological Sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4
5 typedef long long ll;
6 const int MAXN = 1e5;
7
8 /* Topological Sort
9  * */
10

```

```

11 vector<int> state; //Tres estados 0, 1, 2
12 //0 -> no visitado, 1 -> visitando, 2 -> totalmente
    visitado
13 vector<int> G[MAXN];
14 vector<int> order;
15
16 bool dfs(int nodo){ //Devolvemos si true si encontramos un ciclo
17     if(state[nodo] == 2) return false;
18     state[nodo] = 1;
19
20     bool cicle = false;
21     for(auto it : G[nodo]){
22         if(state[it] == 0){
23             cicle = dfs(it);
24         }else if(state[it] == 1){
25             return true;
26         }
27     }
28     state[nodo] = 2;
29     order.push_back(nodo);
30     return cicle;
31 }
32
33 int main(){
34     int n, m;
35     // Leer grafo
36
37     state.assign(n+1, 0);
38
39     bool cicle = false;
40     forn(i, n){
41         if (state[i] == 0){
42             cicle = dfs(i);
43         }
44         if (cicle) break;
45     }
46
47     if (cicle){
48         cout << "IMPOSSIBLE\n";
49     }else{
50         reverse(order.begin(), order.end());
51         for(int i : order){
52             cout << i+1 << ' ';
53         }
54         cout << '\n';
55     }
56
57     return 0;
58 }

```

## 5.5 Kosaraju

Encontrar todas las componentes fuertemente conexas en un grafo dirigido

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forr(i, a, b) for(int i = (int) a; i < (int) b; i++)
4 #define forn(i, n) forr(i, 0, n)
5 #define dforn(i, n) for(int i = (int)(n-1); i >= 0; i--)
6 #define pb push_back
7
8 // Sacado del olaf
9 // Not tested, not even a bit
10 struct Kosaraju {
11     static const int default_sz = 1000001;
12     int n;
13     vector< vector<int> > G, revG, C, ady; // ady is the condensed graph
14     vector<int> used, where;
15     Kosaraju(int sz = default_sz){
16         n = sz;
17         G.assign(sz, vector<int>());
18         revG.assign(sz, vector<int>());
19         used.assign(sz, 0);
20         where.assign(sz, -1);
21     }
22     void addEdge(int a, int b){ G[a].pb(b); revG[b].pb(a); }
23     void dfsNormal(vector<int> &F, int v){
24         used[v] = true;
25         forn(i, G[v].size()) if(!used[ G[v][i] ])
26             dfsNormal(F, G[v][i]);
27         F.pb(v);
28     }
29     void dfsRev(vector<int> &F, int v){
30         used[v] = true;
31         forn(i, revG[v].size()) if(!used[ revG[v][i] ])
32             dfsRev(F, revG[v][i]);
33         F.pb(v);
34     }
35     void build(){
36         vector<int> T;
37         fill(used.begin(), used.end(), 0);
38         forn(i, n) if(!used[i]) dfsNormal(T, i);
39         reverse(T.begin(), T.end());
40         fill(used.begin(), used.end(), 0);
41         forn(i, T.size()) if(!used[ T[i] ]){
42             vector<int> F;
43             dfsRev(F, T[i]);
44             forn(i, F.size()) where[ F[i] ] = C.size();
45             C.pb(F);
46         }
47         ady.resize(C.size()); // Create edges between condensed nodes
48         forn(i, n) forn(j, G[i].size()){
49             if(where[i] != where[ G[i][j] ]){
50                 ady[ where[i] ].pb( where[ G[i][j] ] );
51             }
52         }

```

```
53     forn(v, C.size()){
54         sort(ady[v].begin(), ady[v].end());
55         ady[v].erase(unique(ady[v].begin(), ady[v].end()), ady[v].end());
56     }
57 }
58 };
```

## 6 Strings

### 6.1 Hash

```
1 #include <bits/stdc++.h>
2 #define forr(i, a, b) for (int i = (a); i < (b); i++)
3 #define forn(i, n) forr(i, 0, n)
4 #define dforn(i, n) for (int i = (n) - 1; i >= 0; i--)
5 #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
6
7 typedef long long ll;
8
9 using namespace std;
10 struct Hash {
11     ll MOD = 999727999;
12     ll BASE = 325255434;
13     //ll MOD = 97;
14     //ll BASE = 3;
15     vector<ll> h, pot;
16     string str;
17
18     Hash(string& s) {
19         int n = s.size();
20         str = s;
21         h.resize(n + 1);
22         pot.resize(n + 1);
23         h[0] = s[0];
24         pot[0] = 1;
25         for (int i = 0; i < n; i++) {
26             h[i + 1] = (h[i] * BASE + s[i]) % MOD;
27             pot[i + 1] = (pot[i] * BASE) % MOD;
28         }
29     }
30
31     // Devuelve el hash del substring s[s..e] en O(1)
32     ll get(int s, int e) {
33         ll hash_val = (h[e + 1] - h[s] * pot[e - s + 1]) % MOD;
34         if (hash_val < 0) hash_val += MOD;
35         return hash_val;
36     }
37
38     // devuelve el hash del string cambiando en la iesima posicion de c1
39     // a c2.
40     // s: inicio del string, e: fin, i: posicion a cambiar, c1: old char,
41     // c2: new char
42     ll get_change(int s, int e, int i, int c1, int c2) {
43         ll original_hash = get(s, e);
44         int exp = e - i;
45         ll diff = ((c2 - c1) * pot[exp]) % MOD;
46         ll changed_hash = (original_hash + diff) % MOD;
47         if (changed_hash < 0) changed_hash += MOD;
48         return changed_hash;
49     }
50 };
51
52 };
```

### 6.1.1 Examples

```

1  /**
2   * Given a string and a pattern, your task is to count the number of
3   * positions
4   * where the pattern occurs in the string.
5   *
6   * Input:
7   * saippuakauppias
8   * pp
9   * Output:
10  * 2
11  */
12
13 #include <bits/stdc++.h>
14 #include "hashing.h"
15 #define forn(i,n) for(ll i = 0; i < n; i++)
16 #define forp(p, i,n) for(ll i = p; i < n; i++)
17
18 using namespace std;
19
20 #define pb push_back
21 typedef long long ll;
22 #define MAXN 1000000010
23
24 int main() {
25     ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
26
27     string s,t;
28     cin >> s >> t;
29     Hash h(s);
30     Hash ha(t);
31     int n = s.size(), res=0;
32
33     forn(i, n-t.size()+1){
34         if(t.size() > s.size()) break;
35         if(h.get(i,i+t.size()-1) == ha.get(0,t.size()-1)) res++;
36     }
37
38     cout << res << endl;
39     return 0;
40 }

```

### 6.1.2 Examples

```

1
2 #include <bits/stdc++.h>
3 #include <bitset>
4 #include <iostream>
5 #include <string>
6 using namespace std;
7 #define forn(i,n) for(ll i = 0; i < n; i++)
8 #define forp(p, i,n) for(ll i = p; i < n; i++)

```

```

9 #define pb push_back
10 typedef long long ll;
11 #define MAXN 1000000010
12 #ifdef EBUG
13 //local
14 #else
15 //judge
16 #endif
17
18 #define forr(i,s,n) for(int i=s; i<n; i++)
19 struct Hash{
20     int P=1777771, MOD[2], PI[2];
21     vector<int> h[2], pi[2];
22     vector<ll> primos[2];
23     Hash(string& s){
24         MOD[0]=999727999; MOD[1]=1070777777;
25         PI[0]=325255434; PI[1]=10018302;
26         forr(k,0,2){
27             h[k].resize(s.size()+1), pi[k].resize(s.size()+1), primos[k].
                resize(s.size()+1);
28         }
29         forr(k,0,2){
30             h[k][0]=0;
31             pi[k][0]=1;
32             ll p=1;
33             primos[0][0]=p;
34             primos[1][0]=p;
35             forr(i,1,s.size()+1){
36                 h[k][i]=(h[k][i-1]+p*s[i-1])%MOD[k];
37                 pi[k][i]=(1LL* pi[k][i-1]* PI[k])%MOD[k];
38                 p=(p*P)%MOD[k];
39                 primos[k][i]=p;
40             }
41         }
42     }
43     ll get(int s, int e){
44         ll h0=(h[0][e]-h[0][s]+MOD[0])%MOD[0];
45         h0 =(1LL*h0*pi[0][s])%MOD[0];
46         ll h1=(h[1][e]-h[1][s]+MOD[1])%MOD[1];
47         h1 =(1LL*h1*pi[1][s])%MOD[1];
48         return (h0<<32)|h1;
49     }
50
51     //devuelve el hash del string cambiando en la iesima posicion a c1
52     // por c2.
53     // s: inicio del string, e: fin, i: posicion a cambiar, c1: old char,
54     // c2: new char
55     ll get_change(int s, int e, int i, int c1, int c2){
56
57         ll h0=(h[0][e]-h[0][s]+MOD[0])%MOD[0];
58         h0 =(1LL*h0*pi[0][s])%MOD[0];
59         h0 = ((h0 - c1*primos[0][i] % MOD[0]) + MOD[0])%MOD[0];
60         h0 = (h0 + c2*primos[0][i])%MOD[0];
61
62         ll h1=(h[1][e]-h[1][s]+MOD[1])%MOD[1];
63         h1 =(1LL*h1*pi[1][s])%MOD[1];

```

```
62     h1=( h1 - c1*primos[1][i])%MOD[1] + MOD[1])%MOD[1];
63     h1=(h1 + c2*primos[1][i])%MOD[1];
64     return (h0<<32)|h1;
65 }
66
67 void set_change(int s, int e, int i, int c1, int c2) {
68     for (int k = 0; k < 2; ++k) {
69         h[k][e] = (h[k][e] - c1 * primos[k][i] % MOD[k] + MOD[k]) %
70             MOD[k];
71         h[k][e] = (h[k][e] + c2 * primos[k][i]) % MOD[k];
72     }
73 };
74
75 int main(){
76     #ifdef EBUG
77         freopen("input.txt", "r", stdin);
78     #endif
79     ios :: sync_with_stdio(false);
80     cin.tie(NULL);
81     cout.tie(NULL);
82
83
84     int n;
85     cin >> n;
86     string s;
87     cin >> s;
88     int r, m;
89     cin >> r >> m;
90
91     unordered_map<ll,ll> tabla;
92     string abecedario = "abcdefghijklmnopqrstuvwxyz,._-";
93     forn(i,r){
94         string palabra;
95         cin >> palabra;
96         Hash p(palabra);
97         tabla[p.get(0,palabra.size())]++;
98
99         forn(j,m){
100             forn(k, abecedario.size()){
101                 if(palabra[j] == abecedario[k])continue;
102                 tabla[p.get_change(0, m , j, palabra[j], abecedario[k])
103                     ]++;
104             }
105         }
106     }
107
108     Hash secuencia(s);
109     ll ans = 0;
110     for(int i=0 ; i+m <= s.size() ; i++){
111         ll valor = secuencia.get(i,i+m);
112
113         if(tabla.count(valor)>0){
114             ans += tabla[valor];
115         }
```

```
116     }
117     cout << ans << endl;
118
119
120     return 0;
121 }
```



## 7 Geometría

### 7.1 Punto

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 using namespace std;
4
5 // si la entrada es entero cambiar a long long.
6 typedef long long T; // double could be faster but less precise
7 typedef long double ld;
8 const T EPS = 1e-9; // if T is integer, set to 0
9 const T INF = 1e18;
10
11 struct pto{
12     T x, y;
13     //constructor. pto a = (0,0)
14     pto() : x(0), y(0) {}
15     pto(T _x, T _y) : x(_x), y(_y) {}
16
17     //operadores basicos
18     pto operator+(pto b) { return pto(x+b.x, y+b.y); }
19     pto operator-(pto b) { return pto(x-b.x, y-b.y); }
20     pto operator+(T k) { return pto(x+k, y+k); }
21     pto operator*(T k) { return pto(x*k, y*k); }
22     pto operator/(T k) { return pto(x/k, y/k); }
23
24     // dot product
25     /*  $a \cdot b = |a| * |b| * \cos(\hat{a}\hat{b})$ 
26         $\text{proy}_{a \cdot b} = (a \cdot b) / |b|$ 
27         $a \cdot b = 0$  (o  $|a \cdot b| < \text{EPS}$  en flotantes)
28     */
29     T operator*(pto b) { return x*b.x + y*b.y; }
30
31     // cross product
32     T operator^(pto b) { return x*b.y - y*b.x; }
33     /*
34         $|a \wedge b| = |a| * |b| * \sin(\hat{a}\hat{b})$ 
35         $> 0$   $\hat{a}\hat{b}$  b está a la izquierda de a (rotación CCW menor a 180)
36         $< 0$   $\hat{a}\hat{b}$  b está a la derecha de a (rotación CW menor a 180)
37         $= 0$   $\hat{a}\hat{b}$  a y b son colineales.
38        Es el área con signo del paralelogramo formado por a y b.
39        El área del triángulo definido por a y b es  $|a \wedge b| / 2$ .
40     */
41     // vector projection of this above b
42     pto proj(pto b) { return b*((*this)*b) / (b*b); }
43
44     T norm_sq() { return x*x + y*y; }
45     ld norm() { return sqrtl(x*x + y*y); }
46     ld dist(pto b) { return (b - (*this)).norm(); }
47
48     //rotate by theta rads CCW w.r.t. origin (0,0)
49     pto rotate(T ang) {

```

```

50         return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
51     }
52
53     // true if this is at the left side of line ab
54     bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
55     bool operator<(const pto &b) const {
56         return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
57     }
58     bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }
59 };
60
61 pto perp(pto a) {return pto(-a.y, a.x);}
62
63 ld angle(pto a, pto o, pto b) {
64     pto oa = a-o, ob = b-o;
65     return atan2l(oa^ob, oa*ob);
66 }
67 ld angle(pto a, pto b) { // smallest angle between a and b
68     ld cost = (a*b) / a.norm() / b.norm();
69     return acosl(max(ld(-1.), min(ld(1.), cost)));
70 }

```

### 7.2 Line

```

1 #include "pto.cpp"
2
3 int sgn(T x) { return x < 0 ? -1 : !!x; }
4 struct line {
5     T a, b, c; // Ax+By=C
6     line() {}
7     line(T a_, T b_, T c_) : a(a_), b(b_), c(c_) {}
8     // T0 D0: check negative C (multiply everything by -1)
9     line(pto u, pto v) : a(v.y - u.y), b(u.x - v.x), c(a * u.x + b * u.y)
10    {}
11
12     int side(pto v) { return sgn(a * v.x + b * v.y - c); }
13     bool inside(pto v) { return abs(a * v.x + b * v.y - c) <= EPS; }
14     bool parallel(line v) { return abs(a * v.b - v.a * b) <= EPS; }
15     pto inter(line v) {
16         T det = a * v.b - v.a * b;
17         if (abs(det) <= EPS) return pto(INF, INF);
18         return pto(v.b * c - b * v.c, a * v.c - v.a * c) / det;
19     }
20 };

```

### 7.3 Segment

```

1 #include "pto.cpp"
2 #include "line.cpp"
3
4 struct segment {
5     pto s, e;

```

```

6   segment(pto s_, pto e_) : s(s_), e(e_) {}
7
8   //devuelve el punto del segmento más cercano a b
9   pto closest(pto b) {
10      pto bs = b - s, es = e - s;
11      ld l = es * es;
12      if (abs(l) <= EPS) return s;
13      ld t = (bs * es) / l;
14      if (t < 0.) return s;      // comment for lines
15      else if (t > 1.) return e; // comment for lines
16      return s + (es * t);
17   }
18   bool inside(pto b) { //Return true if pto b is inside the segment
19      return abs(s.dist(b) + e.dist(b) - s.dist(e)) < EPS;
20   }
21
22   // si los puntos son muy grandes puede dar overflow. Usar este inside
23   bool inside2(pto b) {
24      return ((s - b) ^ (e - b)) == 0 &&
25         min(s.x, e.x) <= b.x && b.x <= max(s.x, e.x) &&
26         min(s.y, e.y) <= b.y && b.y <= max(s.y, e.y);
27   }
28
29   pto inter(segment b) { // if a and b are collinear, returns one
30      point
31      if ((*this).inside(b.s)) return b.s;
32      if ((*this).inside(b.e)) return b.e;
33      pto in = line(s, e).inter(line(b.s, b.e));
34      if ((*this).inside(in) && b.inside(in)) return in;
35      return pto(INF, INF);
36   }
37
38   // cuando no importa el punto
39   bool intersects(segment b) {
40      pto a1 = s, a2 = e, b1 = b.s, b2 = b.e;
41      auto cross1 = (a2 - a1) ^ (b1 - a1);
42      auto cross2 = (a2 - a1) ^ (b2 - a1);
43      auto cross3 = (b2 - b1) ^ (a1 - b1);
44      auto cross4 = (b2 - b1) ^ (a2 - b1);
45      if ((cross1 > 0 && cross2 < 0 || cross1 < 0 && cross2 > 0) &&
46          (cross3 > 0 && cross4 < 0 || cross3 < 0 && cross4 > 0))
47         return true;
48      return inside(b1) || inside(b2) || b.inside(a1) || b.inside(a2);
49   }
50 }
51 };

```

## 7.4 Polar sort

```

1 #include "pto.cpp"
2
3 /*
4 funcionamiento:

```

```

5   vector<pto> puntos = {pto(1, 2), pto(2, 1), pto(-1, -1), pto(0, 2)};
6   pto referencia(1, 1); // punto de referencia
7   sort(puntos.begin(), puntos.end(), Cmp(referencia));
8   */
9
10  struct Cmp{//orden total de puntos alrededor de un punto r
11     pto r;
12     Cmp(pto r):r(r) {}
13     int cuad(const pto &a) const{
14         if(a.x > 0 && a.y >= 0) return 0;
15         if(a.x <= 0 && a.y > 0) return 1;
16         if(a.x < 0 && a.y <= 0) return 2;
17         if(a.x >= 0 && a.y < 0) return 3;
18         assert(a.x ==0 && a.y==0);
19         return -1;
20     }
21     bool cmp(const pto&p1, const pto&p2) const{
22         int c1 = cuad(p1), c2 = cuad(p2);
23         if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
24         else return c1 < c2;
25     }
26     bool operator()(const pto&p1, const pto&p2) const{
27         return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
28     }
29 };

```

## 7.5 Convex Hull

```

1 #include "pto.cpp"
2 using namespace std;
3 #define pb push_back
4 #define dform(i,n) for(int i=n-1; i>=0; i--)
5 #define forp(p, i, n) for(int i = p; i < n; i++)
6 #define forn(i, n) for(int i = 0; i < n; i++)
7
8 // returns convex hull of p in CCW order
9 // left must return >=0 to delete collinear points
10 vector<pto> CH(vector<pto>& p) {
11     if (p.size() < 3) return p; // edge case, keep line or point
12     vector<pto> ch;
13     sort(p.begin(), p.end());
14     forn(i, p.size()) { // lower hull
15         while (ch.size() >= 2 && ch[ch.size() - 1].left(ch[ch.size() - 2], p[i]))
16             ch.pop_back();
17         ch.pb(p[i]);
18     }
19     ch.pop_back();
20     int k = ch.size();
21     dform(i, p.size()) { // upper hull
22         while (ch.size() >= k + 2 && ch[ch.size() - 1].left(ch[ch.size() - 2], p[i]))
23             ch.pop_back();
24         ch.pb(p[i]);

```

```

25     }
26     ch.pop_back();
27     return ch;
28 }

```

## 7.6 Area poligono

```

1 #include "pto.cpp"
2 #define forn(i, n) for(int i = 0; i < n; i++)
3 double area(vector<pto> &p){//0(p.size())
4     double area=0;
5     forn(i, p.size()) area+=p[i]^p[(i+1)%p.size()];
6     //if points are in clockwise order then area is negative
7     return abs(area)/2;
8     // si los puntos son enteros abs(area) es siempre entero
9 }

```

## 7.7 Poligono

```

1 #include "pto.cpp"
2 #include "line.cpp"
3 #include "circle.cpp"
4 #include "convex-hull.cpp"
5
6 #define sz(v) (int(v.size()))
7 #define forn(i, n) for(int i = 0; i < n; i++)
8 #define forr(i, a, b) for (int i = (a); i < (b); i++)
9
10 #define pb push_back
11
12 struct poly{
13     vector<pto> pt;
14     poly(){}
15     poly(vector<pto> pt_) : pt(pt_) {}
16     void delete_collinears() { // delete collinear points
17         deque<pto> nxt; int len = 0;
18         forn(i, sz(pt)) {
19             if(len>1 && abs((pt[i]-nxt[len-2])^(nxt[len-1]-nxt[len-2])) <= EPS)
20                 nxt.pop_back(), len--;
21             nxt.pb(pt[i]); len++;
22         }
23         if(len>2 && abs((nxt[1]-nxt[len-1])^(nxt[0]-nxt[len-1])) <= EPS)
24             nxt.pop_front(), len--;
25         if(len>2 && abs((nxt[len-1]-nxt[len-2])^(nxt[0]-nxt[len-2])) <= EPS)
26             nxt.pop_back(), len--;
27         pt.clear(); forn(i, sz(nxt)) pt.pb(nxt[i]);
28     }
29
30     // asegura sentido horario, elimina colineales, rota el vector para
31     // que pt[0] sea el primero
32     void normalize() {

```

```

32     delete_collinears();
33     if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end()); //make it
34     CW
35     int n = sz(pt), pi = 0;
36     forn(i, n)
37         if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
38             pi = i;
39     rotate(pt.begin(), pt.begin()+pi, pt.end());
40 }
41
42 bool is_convex() { // delete collinear points first 0(n)
43     int N = sz(pt);
44     if(N < 3) return false;
45     bool isLeft = pt[0].left(pt[1], pt[2]);
46     forr(i, 1, sz(pt))
47         if(pt[i].left(pt[(i+1)%N], pt[(i+2)%N]) != isLeft)
48             return false;
49     return true;
50 }
51
52 // for convex or concave polygons
53 // excludes boundaries, check it manually
54 bool inside(pto p) { // 0(n)
55     bool c = false;
56     forn(i, sz(pt)) {
57         int j = (i+1)%sz(pt);
58         if((pt[j].y>p.y) != (pt[i].y > p.y) &&
59            (p.x < (pt[i].x-pt[j].x)*(p.y-pt[j].y)/(pt[i].y-pt[j].y)+pt[j].x))
60             c = !c;
61     }
62     return c;
63 }
64
65 bool inside_convex(pto p) { // 0(lg(n)) normalize first
66     if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
67     int a = 1, b = sz(pt)-1;
68     while(b-a > 1){
69         int c = (a+b)/2;
70         if(!p.left(pt[0], pt[c])) a = c;
71         else b = c;
72     }
73     return !p.left(pt[a], pt[a+1]);
74 }
75
76 // cuts this along line ab and return the left side
77 // (swap a, b for the right one)
78 poly cut(pto a, pto b) { // 0(n)
79     vector<pto> ret;
80     forn(i, sz(pt)) {
81         ld left1 = (b-a)^(pt[i]-a), left2 = (b-a)^(pt[(i+1)%sz(pt)]-a);
82         if(left1 >= 0) ret.pb(pt[i]);
83         if(left1*left2 < 0)
84             ret.pb(line(pt[i], pt[(i+1)%sz(pt)]).inter(line(a, b)));
85     }
86     return poly(ret);

```

```

87 }
88
89
90 // addition of convex polygons
91 poly minkowski(poly p) { // 0(n+m) n=|this|,m=|p|
92     this->normalize(); p.normalize();
93     vector<pto> a = (*this).pt, b = p.pt;
94     a.pb(a[0]); a.pb(a[1]);
95     b.pb(b[0]); b.pb(b[1]);
96     vector<pto> sum;
97     int i = 0, j = 0;
98     while(i < sz(a)-2 || j < sz(b)-2) {
99         sum.pb(a[i]+b[j]);
100         T cross = (a[i+1]-a[i])^(b[j+1]-b[j]);
101         if(cross <= 0 && i < sz(a)-2) i++;
102         if(cross >= 0 && j < sz(b)-2) j++;
103     }
104     return poly(sum);
105 }
106
107 // busca el punto mas lejano en una direccion dada
108 pto farthest(pto v) { // 0(log(n)) for convex polygons
109     if(sz(pt)<10) {
110         int k=0;
111         forr(i,1,sz(pt)) if(v * (pt[i] - pt[k]) > EPS) k = i;
112         return pt[k];
113     }
114     pt.pb(pt[0]);
115     pto a=pt[1] - pt[0];
116     int s = 0, e = sz(pt)-1, ua = v*a > EPS;
117     if(!ua && v*(pt[sz(pt)-2]-pt[0]) <= EPS){ pt.pop_back(); return pt
118         [0];}
119     while(1) {
120         int m = (s+e)/2; pto c = pt[m+1]-pt[m];
121         int uc = v*c > EPS;
122         if(!uc && v*(pt[m-1]-pt[m]) <= EPS){ pt.pop_back(); return pt[m];}
123         if(ua && (!uc || v*(pt[s]-pt[m]) > EPS)) e = m;
124         else if(ua || uc || v*(pt[s]-pt[m]) >= -EPS) s = m, a = c, ua = uc;
125         else e = m;
126         assert(e > s+1);
127     }
128
129     //hasta aca.
130
131 ld inter_circle(circle c){ // area of intersection with circle
132     ld r = 0.;
133     forn(i,sz(pt)) {
134         int j = (i+1)%sz(pt); ld w = c.inter_triangle(pt[i], pt[j]);
135         if(((pt[j]-c.o)^(pt[i]-c.o)) > 0) r += w;
136         else r -= w;
137     }
138     return abs(r);
139 }
140 // area ellipse = M_PI*a*b where a and b are the semi axis lengths
141 // area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2

```

```

142 ld area(){ // 0(n)
143     ld area = 0;
144     forn(i, sz(pt)) area += pt[i]^pt[(i+1)%sz(pt)];
145     return abs(area)/ld(2);
146 }
147 // returns one pair of most distant points
148 pair<pto,pto> callipers() { // 0(n), for convex poly, normalize first
149     int n = sz(pt);
150     if(n <= 2) return {pt[0], pt[1%n]};
151     pair<pto,pto> ret = {pt[0], pt[1]};
152     T maxi = 0; int j = 1;
153     forn(i,sz(pt)) {
154         while(((pt[(i+1)%n]-pt[i])^(pt[(j+1)%n]-pt[j]))<-EPS)j=(j+1)%sz(pt)
155             ;
156         if(pt[i].dist(pt[j]) > maxi+EPS)
157             ret = {pt[i], pt[j]}, maxi = pt[i].dist(pt[j]);
158     }
159     return ret;
160 }
161 pto centroid(){ // (barycenter, mass center, needs float points)
162     int n = sz(pt);
163     pto r(0,0); ld t=0;
164     forn(i,n) {
165         r = r + (pt[i] + pt[(i+1)%n]) * (pt[i] ^ pt[(i+1)%n]);
166         t += pt[i] ^ pt[(i+1)%n];
167     }
168     return r/t/3;
169 };
170 // Dynamic convex hull trick (based on poly struct)
171 vector<poly> w;
172 void add(pto q) { // add(q), 0(log^2(n))
173     vector<pto> p = {q};
174     while(!w.empty() && sz(w.back().pt) < 2*sz(p)){
175         for(pto v : w.back().pt) p.pb(v);
176         w.pop_back();
177     }
178     w.pb(poly(CH(p))); // CH = convex hull, must delete collinears
179 }
180 T query(pto v) { // max(q*v:q in w), 0(log^2(n))
181     T r = -INF;
182     for(auto& p : w) r = max(r, p.farthest(v)*v);
183     return r;
184 }

```

## 7.8 Puntos mas cercanos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define dfor(n,i,n) for(int i=n-1; i>=0; i--)
5 #define forn(i, n) for(int i = 0; i < n; i++)
6 #define fori(i, n) for(int i = n - 1; i >= 0; i--)
7 #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)

```

```

8 #define forr(i, a, b) for (int i = (a); i < (b); i++)
9 #define sz(v) (int(v.size()))
10 #define pb push_back
11
12 typedef long long T; // double could be faster but less precise
13 typedef long double ld;
14 //const T EPS = 1e-9; // if T is integer, set to 0
15 const T EPS = 0; // if T is integer, set to 0
16 const T INF = 1e18;
17
18 struct pto{
19     T x, y;
20     //constructor. pto a = (0,0)
21     pto() : x(0), y(0) {}
22     pto(T _x, T _y) : x(_x), y(_y) {}
23
24     //operadores basicos
25     pto operator+(pto b) { return pto(x+b.x, y+b.y); }
26     pto operator-(pto b) { return pto(x-b.x, y-b.y); }
27     pto operator+(T k) { return pto(x+k, y+k); }
28     pto operator*(T k) { return pto(x*k, y*k); }
29     pto operator/(T k) { return pto(x/k, y/k); }
30
31     // dot product
32     /*  $a \cdot b = |a| * |b| * \cos(\angle)$ 
33         $proy_{a,b} = (a \cdot b) / |b|^2 * b$ 
34         $a \cdot b \neq 0$   $\Rightarrow$   $a \perp b$   $\Leftrightarrow a \cdot b = 0$  (o  $|a \cdot b| < EPS$  en flotantes)
35     */
36     T operator*(pto b) { return x*b.x + y*b.y; }
37
38     // cross product
39     T operator^(pto b) { return x*b.y - y*b.x; }
40     /*
41         $|a \times b| = |a| * |b| * \sin(\angle)$ 
42         $> 0$   $\Rightarrow$  b está a la izquierda de a (rotación CCW menor a 180°)
43         $< 0$   $\Rightarrow$  b está a la derecha de a (rotación CW menor a 180°)
44         $= 0$   $\Rightarrow$  a y b son colineales.
45        Es el área con signo del paralelogramo formado por a y b.
46        El área del triángulo definido por a y b es  $|a \times b| / 2$ .
47     */
48     // vector projection of this above b
49     pto proj(pto b) { return b*((*this)*b) / (b*b); }
50
51     T norm_sq() { return x*x + y*y; }
52     ld norm() { return sqrtl(x*x + y*y); }
53     ld dist(pto b) { return (b - (*this)).norm(); }
54     ll dist2(pto b) { return (b - (*this)).norm_sq(); }
55
56     //rotate by theta rads CCW w.r.t. origin (0,0)
57     pto rotate(T ang) {
58         return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
59     }
60
61     // true if this is at the left side of line ab

```

```

62     bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
63     bool operator<(const pto &b) const {
64         return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
65     }
66     bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }
67 };
68
69 struct CmpY {
70     bool operator()(const pto& a, const pto& b) const {
71         if (a.y == b.y) return a.x < b.x;
72         return a.y < b.y;
73     }
74 };
75
76 ll closest_pair(vector<pto>& pts) {
77     sort(pts.begin(), pts.end(), [](auto &a, auto &b) {
78         return a.x < b.x;
79     });
80
81     set<pto, CmpY> active;
82     ll d = LLONG_MAX;
83     int j = 0;
84
85     for (int i = 0; i < (int)pts.size(); i++) {
86         pto p = pts[i];
87
88         while (j < i && (p.x - pts[j].x)*(p.x - pts[j].x) > d) {
89             active.erase(pts[j]);
90             j++;
91         }
92
93         // vecinos en  $[p.y - \sqrt{d}, p.y + \sqrt{d}]$ 
94         ll lim = (ll) sqrtl((long double)d) + 1;
95         pto low = {p.x, p.y - lim};
96         pto high = {p.x, p.y + lim};
97
98         auto itlow = active.lower_bound(low);
99         auto ithigh = active.upper_bound(high);
100
101         for (auto it = itlow; it != ithigh; ++it) {
102             d = min(d, p.dist2(*it));
103         }
104
105         active.insert(p);
106     }
107     return d;
108 }
109
110 int main(){
111     #ifdef EBUG
112         freopen("input.txt", "r", stdin);
113     #endif
114     ios :: sync_with_stdio(false);
115     cin.tie(NULL);
116     cout.tie(NULL);
117

```

```

118     int n;
119     cin >> n;
120     vector<pto> a(n);
121     forn(i,n){
122         cin >> a[i].x >> a[i].y;
123     }
124     cout << closest_pair(a) << endl;
125
126     return 0;
127 }

```

## 7.9 Puntos mas lejanos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define dforn(i,n) for(int i=n-1; i>=0; i--)
5 #define forn(i, n) for(int i = 0; i < n; i++)
6 #define fori(i, n) for(int i = n - 1; i <= 0; i--)
7 #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
8 #define forr(i, a, b) for (int i = (a); i < (b); i++)
9 #define sz(v) (int(v.size()))
10 #define pb push_back
11
12
13 int main(){
14     #ifdef EBUG
15         freopen("input.txt", "r", stdin);
16     #endif
17     ios :: sync_with_stdio(false);
18     cin.tie(NULL);
19     cout.tie(NULL);
20
21     int n;
22     cin >> n;
23     vector<ll> mn(4, LLONG_MAX), mx(4, LLONG_MIN);
24
25     for (int i = 0; i < n; i++) {
26         ll x, y; cin >> x >> y;
27
28         ll vals[4] = {x+y, x-y, -x+y, -x-y};
29
30         for (int k = 0; k < 4; k++) {
31             mn[k] = min(mn[k], vals[k]);
32             mx[k] = max(mx[k], vals[k]);
33         }
34
35         ll ans = 0;
36         for (int k = 0; k < 4; k++) {
37             ans = max(ans, mx[k] - mn[k]);
38         }
39         cout << ans << endl;
40     }
41 }

```

```

42     return 0;
43 }

```

## 7.10 Puntos enteros

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long
4 #define dforn(i,n) for(int i=n-1; i>=0; i--)
5 #define forn(i, n) for(int i = 0; i < n; i++)
6 #define fori(i, n) for(int i = n - 1; i <= 0; i--)
7 #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
8 #define forr(i, a, b) for (int i = (a); i < (b); i++)
9 #define sz(v) (int(v.size()))
10 #define pb push_back
11
12
13 // si la entrada es entero cambiar a long long.
14 typedef long long T; // double could be faster but less precise
15 typedef long double ld;
16 //const T EPS = 1e-9; // if T is integer, set to 0
17 const T EPS = 0; // if T is integer, set to 0
18 const T INF = 1e18;
19
20 struct pto{
21     T x, y;
22     //constructor. pto a = (0,0)
23     pto() : x(0), y(0) {}
24     pto(T _x, T _y) : x(_x), y(_y) {}
25
26     //operadores basicos
27     pto operator+(pto b) { return pto(x+b.x, y+b.y); }
28     pto operator-(pto b) { return pto(x-b.x, y-b.y); }
29     pto operator+(T k) { return pto(x+k, y+k); }
30     pto operator*(T k) { return pto(x*k, y*k); }
31     pto operator/(T k) { return pto(x/k, y/k); }
32
33     // dot product
34     /* a · b = |a| * |b| * cos(∠)
35        proy_a_b = ( (a·b) / |b| ) * b
36        a ⊥ b ⇔ a·b = 0 (o |a·b| < EPS en flotantes)
37
38     T operator*(pto b) { return x*b.x + y*b.y; }
39
40     // cross product
41     T operator^(pto b) { return x*b.y - y*b.x; }
42     /*
43        |a × b| = |a| * |b| * sin(∠)
44        > 0 ⇒ b está a la izquierda de a (rotación CCW menor a 180°)
45        < 0 ⇒ b está a la derecha de a (rotación CW menor a 180°)
46        = 0 ⇒ a y b son colineales.
47        Es el área con signo del paralelogramo formado por a y b.

```

```

48     El Área del triángulo definido por a y b es |a × b| / 2.
49     */
50     // vector projection of this above b
51     pto proj(pto b) { return b*((*this)*b) / (b*b); }
52
53     T norm_sq() { return x*x + y*y; }
54     ld norm() { return sqrtl(x*x + y*y); }
55     ld dist(pto b) { return (b - (*this)).norm(); }
56
57     //rotate by theta rads CCW w.r.t. origin (0,0)
58     pto rotate(T ang) {
59         return pto(x*cosl(ang) - y*sinl(ang), x*sinl(ang) + y*cosl(ang));
60     }
61
62     // true if this is at the left side of line ab
63     bool left(pto a, pto b) { return ((a-*this) ^ (b-*this)) > 0; }
64     bool operator<(const pto &b) const {
65         return x < b.x-EPS || (abs(x - b.x) <= EPS && y < b.y-EPS);
66     }
67     bool operator==(pto b){ return abs(x-b.x)<=EPS && abs(y-b.y)<=EPS; }
68 };
69
70 ll area(vector<pto> &p){//0(p.size())
71     ll area=0;
72     forn(i, p.size()) area+=p[i]^p[(i+1)%p.size()];
73     //if points are in clockwise order then area is negative
74     return abs(area)/2;
75     // si los puntos son enteros abs(area) es siempre entero
76 }
77
78
79 int main(){
80     #ifdef EBUG
81         freopen("input.txt", "r", stdin);
82     #endif
83     ios :: sync_with_stdio(false);
84     cin.tie(NULL);
85     cout.tie(NULL);
86
87     int n;
88     cin >> n;
89     vector<pto> a(n);
90     forn(i,n){
91         cin >> a[i].x >> a[i].y ;
92     }
93     ll B=0;
94     forn(i,n){
95         B += __gcd(abs(a[i].x - a[(i+1)%n].x), abs(a[i].y - a[(i+1)%n].y)
96             );
97     }
98     ll A = area(a);
99
100     cout << (A-(B/2)+1) << " " << B << endl;
101
102     return 0;
103 }

```

## 7.11 Circle

```

1  #include <bits/stdc++.h>
2  #include "polarsort.cpp"
3
4  #define ll long long
5  using namespace std;
6  #define dforn(i,n) for(int i=n-1; i>=0; i--)
7  #define forn(i, n) for(int i = 0; i < n; i++)
8  #define fori(i, n) for(int i = n - 1; i <= 0; i--)
9  #define forall(it, v) for (auto it = v.begin(); it != v.end(); it++)
10 #define forr(i, a, b) for (int i = (a); i < (b); i++)
11 #define sz(v) (int(v.size()))
12 #define pb push_back
13
14 #define sqr(a) ((a)*(a))
15 pto perp(pto a){return pto(-a.y, a.x);}
16 line bisector(pto a, pto b){
17     line l = line(a, b); pto m = (a+b)/2;
18     return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
19 }
20
21 struct circle{
22     pto o; T r;
23     circle(){}
24     circle(pto a, pto b, pto c) {
25         o = bisector(a, b).inter(bisector(b, c));
26         r = o.dist(a);
27     }
28     bool inside(pto p) { return (o-p).norm_sq() <= r*r+EPS; }
29     bool inside(circle c) { // this inside of c
30         T d = (o - c.o).norm_sq();
31         return d <= (c.r-r) * (c.r-r) + EPS;
32     }
33     // circle containing p1 and p2 with radius r
34     // swap p1, p2 to get snd solution
35     circle* circle2PtoR(pto a, pto b, T r_) {
36         ld d2 = (a-b).norm_sq(), det = r_*r_/d2 - ld(0.25);
37         if(det < 0) return nullptr;
38         circle *ret = new circle();
39         ret->o = (a+b)/ld(2) + perp(b-a)*sqrt(det);
40         ret->r = r_;
41         return ret;
42     }
43     pair<pto, pto> tang(pto p) {
44         pto m = (p+o)/2;
45         ld d = o.dist(m);
46         ld a = r*r/(2*d);
47         ld h = sqrtl(r*r - a*a);
48         pto m2 = o + (m-o)*a/d;
49         pto per = perp(m-o)/d;
50         return make_pair(m2 - per*h, m2 + per*h);
51     }
52     vector<pto> inter(line l) {
53         ld a = l.a, b = l.b, c = l.c - l.a*o.x - l.b*o.y;
54         pto xy0 = pto(a*c/(a*a + b*b), b*c/(a*a + b*b));

```

```

55     if(c*c > r*r*(a*a + b*b) + EPS) {
56         return {};
57     }else if(abs(c*c - r*r*(a*a + b*b)) < EPS) {
58         return { xy0 + o };
59     }else{
60         ld m = sqrtl((r*r - c*c/(a*a + b*b))/(a*a + b*b));
61         pto p1 = xy0 + (pto(-b,a)*m);
62         pto p2 = xy0 + (pto(b,-a)*m);
63         return { p1 + o, p2 + o };
64     }
65 }
66 vector<pto> inter(circle c) {
67     line l;
68     l.a = o.x - c.o.x;
69     l.b = o.y - c.o.y;
70     l.c = (sqr(c.r)-sqr(r)+sqr(o.x)-sqr(c.o.x)+sqr(o.y)-sqr(c.o.y))
71         /2.0;
72     return (*this).inter(l);
73 }
74 ld inter_triangle(pto a, pto b) { // area of intersection with oab
75     if(abs((o-a)^(o-b)) <= EPS) return 0.;
76     vector<pto> q = {a}, w = inter(line(a,b));
77     if(sz(w) == 2) forn(i,sz(w)) if((a-w[i])*(b-w[i]) < -EPS) q.pb(w[i]);
78     q.pb(b);
79     if(sz(q) == 4 && (q[0]-q[1])*(q[2]-q[1]) > EPS) swap(q[1], q[2]);
80     ld s = 0;
81     forn(i, sz(q)-1){
82         if(!inside(q[i]) || !inside(q[i+1])) {
83             s += r*r*angle((q[i]-o),q[i+1]-o)/T(2);
84         }
85         else s += abs((q[i]-o)^(q[i+1]-o)/2);
86     }
87     return s;
88 };
89 vector<ld> inter_circles(vector<circle> c){
90     vector<ld> r(sz(c)+1); // r[k]: area covered by at least k circles
91     forn(i, sz(c)) { // 0(n^2 log n) (high constant)
92         int k = 1;
93         Cmp s(c[i].o);
94         vector<pair<pto,int>> p = {
95             {c[i].o + pto(1,0)*c[i].r, 0},
96             {c[i].o - pto(1,0)*c[i].r, 0}};
97         forn(j, sz(c)) if(j != i) {
98             bool b0 = c[i].inside(c[j]), b1 = c[j].inside(c[i]);
99             if(b0 && (!b1 || i<j)) k++;
100             else if(!b0 && !b1) {
101                 vector<pto> v = c[i].inter(c[j]);
102                 if(sz(v) == 2) {
103                     p.pb({v[0], 1}); p.pb({v[1], -1});
104                     if(s(v[1], v[0])) k++;
105                 }
106             }
107         }
108         sort(p.begin(), p.end(), [&](pair<pto,int> a, pair<pto,int> b) {

```

```

109             return s(a.first,b.first); });
110         forn(j,sz(p)) {
111             pto p0 = p[j? j-1: sz(p)-1].first, p1 = p[j].first;
112             ld a = angle(p0 - c[i].o, p1 - c[i].o);
113             r[k]+=(p0.x-p1.x)*(p0.y+p1.y)/ld(2)+c[i].r*c[i].r*(a-sinl(a))
114                 /ld(2);
115             k += p[j].second;
116         }
117     }
118     return r;

```



## 8 DP

### 8.1 Game

```

1 #include <bits/stdc++.h>
2 #include <cstdio>
3 #define ll long long
4 using namespace std;
5
6 int N, K, turno;
7 ll A[3000];
8 ll dp[3000][3000];
9
10 ll juegoOptimo(int inicio, int fin)
11 {
12     if (inicio > fin)
13         return 0;
14
15     if (dp[inicio][fin] != -1)
16         return dp[inicio][fin];
17
18     if ((fin - inicio + 1) % 2 == turno)
19         dp[inicio][fin] = max(A[inicio] + juegoOptimo(inicio + 1, fin), A
20 [fin] + juegoOptimo(inicio, fin - 1));
21     else
22         dp[inicio][fin] = min(-A[inicio] + juegoOptimo(inicio + 1, fin),
23 -A[fin] + juegoOptimo(inicio, fin - 1));
24
25     return dp[inicio][fin];
26 }
27
28 int main(){
29     freopen("input.txt", "r", stdin);
30
31     cin >> N;
32     memset(dp, -1, sizeof(dp));
33     for (int i = 0; i < N; i++){
34         cin >> A[i];
35     }
36
37     if(N%2==0) turno=0;
38     else turno=1;
39
40     cout << juegoOptimo(0,N-1) << endl;
41
42     return 0;
43 }

```

### 8.2 Long common subsequence

```

1 /*
2  Print one longest string that is a subsequence of both s and t.
3  axyb

```

```

4     abyxb
5     output: axb
6 */
7
8 #include <bits/stdc++.h>
9 using namespace std;
10 #define ll long long
11
12 string s,t;
13 int dp[3000][3000];
14
15 int subsecuencia(int i, int j){
16     if( i == s.size() || j == t.size() ) return 0;
17
18     if( dp[i][j] != -1 ) return dp[i][j];
19
20     if ( s[i] == t[j] ) {
21         dp[i][j] = 1 + subsecuencia(i+1,j+1);
22         return dp[i][j];
23     }
24     else {
25         dp[i][j] = max(subsecuencia(i+1,j), subsecuencia(i,j+1));
26         return dp[i][j];
27     }
28 }
29
30 string respuesta = "";
31 void sol(int i, int j){
32     if(i == s.size() || j == t.size() ) return;
33
34     if(s[i] == t[j]){
35         respuesta += s[i] , sol(i+1, j+1);
36     }else{
37         if(dp[i+1][j] > dp[i][j+1]) sol(i+1, j);
38         else sol(i, j+1);
39     }
40 }
41
42 int main(){
43     cin >> s >> t;
44     memset(dp, -1, sizeof(dp));
45
46     subsecuencia(0, 0);
47     sol(0, 0);
48     cout << respuesta << endl;
49
50     return 0;
51 }

```

### 8.3 Long increasing subsequence

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define ll long long

```

```

4 ll INF = 1;
5
6 int lengthOfLIS(vector<int>& arr)
7 {
8     int n = arr.size();
9     vector<int> ans;
10    ans.push_back(arr[0]);
11    for (int i = 1; i < n; i++) {
12        if (arr[i] > ans.back()) {
13            ans.push_back(arr[i]);
14        }
15        else {
16            int low = lower_bound(ans.begin(), ans.end(),
17                                arr[i])
18                        - ans.begin();
19            ans[low] = arr[i];
20        }
21    }
22    return ans.size();
23 }

```

## 8.4 Matching mask

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4 #define fori(i, n) for(int i = n - 1; i >= 0; i--)
5 #define mos(v) forn(auto i : v) cout << i << " ";
6 #define pb push_back
7 typedef long long ll;
8 const int mod = 1e9+7;
9 int n;
10 ll dp[22][2097152];
11 ll a[22][22];
12
13 ll solve(int i, ll mask, int sum){
14     if(i==n){
15         if(sum==n){
16             return 1;
17         }
18         else return 0;
19     }
20     if(dp[i][mask]!=-1) return dp[i][mask];
21
22     ll ans = 0;
23     forn(j,n){
24         if(a[i][j]==1){
25             ll aux = 0;
26             aux = (1LL << j);
27             if((aux&mask)==0){
28                 ans = ((ans%mod) + (solve(i+1,mask^aux,sum+1)%mod) %mod);
29             }
30         }
31     }

```

```

32     dp[i][mask] = ans;
33     return dp[i][mask];
34 }
35
36 int main(){
37     cin >> n;
38     int valor;
39     forn(i,n){
40         forn(j,n){
41             cin >> valor;
42             a[i][j]= valor;
43         }
44     }
45     forn(i,22){
46         forn(j,2097152){
47             dp[i][j]=-1;
48         }
49     }
50
51     ll mask = 0;
52     cout << (solve(0,mask,0)%mod) << endl;
53 }
54

```

## 8.5 DP rangos

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4 #define fori(i, n) for(int i = n - 1; i >= 0; i--)
5 #define mos(v) forn(auto i : v) cout << i << " ";
6 #define ll long long
7 #define ld double
8 #define pb push_back
9 #define MAXN 410
10 int n;
11 ll dp[MAXN][MAXN];
12 vector<ll> A;
13 ll suma(int a, int b){
14     if(a==0) return A[b];
15
16     return (A[b]-A[a-1]);
17 }
18 ll sol(int a, int b){
19     if(a==b) return 0;
20     if(a>b) return 0;
21     if(dp[a][b]!=-1) return dp[a][b];
22     ll ans=1e18;
23     for(int k=a;k<b;k++){
24         ans = min(ans, sol(a,k) + sol(k+1,b) + suma(a,b));
25     }
26
27     return dp[a][b]=ans;
28 }

```

```

29
30 int main(){
31     //freopen("input.txt", "r", stdin);
32     cin >> n;
33     forn(i,n){
34         ll valor;
35         cin >> valor;
36         A.pb(valor);
37     }
38     for(ll i=1;i<n;i++) {
39         A[i] += A[i-1];
40     }
41     memset(dp,-1, sizeof(dp));
42
43     cout << sol(0,n-1) << endl;
44
45 }

```

## 9 Utils

### 9.1 Binary Search

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 bool ok(int a){
5     // funcion que hace algo
6     return a>5;
7 }
8 int bs(vector<int> &v, int val){
9     int l = 0; // algo que siempre sea False
10    int r = v.size() - 1; // algo que siempre sea True
11    int mid = (l+r)/2;
12    while((r-l)>1){
13        mid = (l+r)/2;
14        if(ok(v[mid])){
15            r=mid;
16        }else l=mid;
17    }
18    // FFFFFFFF TTTTTT
19    //      l  r
20    // r == l+1
21    // l == ultimo que No cumple. Ultimo FALSE
22    // r == primero que Si cumple. Primer TRUE
23
24    return mid;
25 }

```

### 9.2 Sort

Ordenar un vector de pair por su segunda componente

```

1 vector<pair<int, int>> v;
2
3 bool sortbysec(const pair<int,int> &a, const pair<int,int> &b){
4     return (a.second < b.second);
5 }
6
7 sort(v.begin(), v.end(), sortbysec);

```

### 9.3 Cout para doubles

```

1 cout << fixed << setprecision(20) << ans << endl;

```

### 9.4 Prev permutation

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int main () {
4     vector<int> dias = {1,2,3,4,5,6,7};
5     vector<bool> mask(dias.size(), false);
6     //combinaciones de D elementos.
7     int d = 3;
8     fill(mask.begin(), mask.begin() + d, true); //1110000
9
10    do {
11        for (int i = 0; i < dias.size(); ++i) {
12            if (mask[i]) {
13                // permutacion actual
14                cout << dias[i] << " ";
15            }
16        }
17        cout << endl;
18    } while (prev_permutation(mask.begin(), mask.end()));
19
20    /* Salida
21        1 2 3
22        1 2 4
23        1 2 5
24        1 2 6
25        1 2 7
26        1 3 4
27        1 3 5
28        1 3 6
29        1 3 7
30        ..
31        5 6 7
32    */
33    return 0;
34 }

```

## 9.5 MO

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for(int i = 0; i < n; i++)
4
5 #define MAXN 500010
6
7 int res = 0;
8 vector<int> v, r;
9 void add(int i){ //modificar
10     r[v[i]]++;
11     if(r[v[i]] == 1) res++;
12 }
13 void remove(int i){ //modificar
14     r[v[i]]--;
15     if(r[v[i]] == 0) res--;
16 }

```

```

17
18 int get_ans(){//modificar
19     return res;
20 }
21 int n,sq,nq; // array size, sqrt(array size), #queries
22 struct qu{int l,r,id;}; // 0((n+nq)*sqrt(n)*update)
23 qu qs[MAXN];
24 int ans[MAXN]; // ans[i] = answer to ith query
25 bool qcomp(const qu &a, const qu &b){
26     if(a.l/sq!=b.l/sq) return a.l<b.l;
27     return (a.l/sq)&1?a.r<b.r:a.r>b.r;
28 }
29 void mos(){
30     forn(i,nq)qs[i].id=i;
31     sq=sqrt(n)+.5;
32     sort(qs,qs+nq,qcomp);
33     int l=0,r=0;
34     forn(i,nq){
35         qu q=qs[i];
36         while(l>q.l)add(--l);
37         while(r<q.r)add(r++);
38         while(l<q.l)remove(l++);
39         while(r>q.r)remove(--r);
40         ans[q.id]=get_ans();
41     }
42 }

```

## 9.6 Criba

Nros primos hasta maxp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define dprint(v) cerr << #v"=" << v << endl //;)
4 #define forr(i,a,b) for(int i=(a); i<(b); i++)
5 #define forn(i,n) forr(i,0,n)
6 #define forall(it,v) for(typeof(v.begin()) it=v.begin();it!=v.end();++it)
7 #define sz(c) ((int)c.size())
8 #define zero(v) memset(v, 0, sizeof(v))
9 typedef long long ll;
10 typedef pair<int,int> ii;
11
12 #define MAXP 100000 //no necesariamente primo
13 int criba[MAXP+1];
14 void crearcriba(){
15     int w[] = {4,2,4,2,4,6,2,6};
16     for(int p=25;p<=MAXP;p+=10) criba[p]=5;
17     for(int p=9;p<=MAXP;p+=6) criba[p]=3;
18     for(int p=4;p<=MAXP;p+=2) criba[p]=2;
19     for(int p=7,cur=0;p<=MAXP;p+=w[cur++&7]) if (!criba[p])
20         for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
21 }
22 vector<int> primos;
23 void buscarprimos(){
24     crearcriba();

```

```

25   forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
26 }
27
28
29 int main() {
30     buscarprimos();
31     cout << '{';
32     bool first=true;
33     forall(it, primos){
34         if(first) first=false;
35         else cout << ',';
36         cout << *it;
37     }
38     cout << "};\n";
39     return 0;
40 }

```

## 9.7 Subconjuntos

Subconjuntos distintos de un conjunto

```

1  #include <bits/stdc++.h>
2  #include <bitset>
3  #include <iostream>
4  #include <string>
5  using namespace std;
6  #define forn(i,n) for(ll i = 0; i < n; i++)
7  #define fora(p, i,n) for(ll i = p; i < n; i++)
8  #define pb push_back
9  typedef long long ll;
10 #define MAXN 1000000010
11 #ifdef EBUG
12 //local
13 #else
14 //judge
15 #endif
16
17
18 void subconjuntos(vector<int> &nums, vector<int> &vacio, int pos) {
19     if(pos==nums.size()){
20         for(auto i: vacio){
21             cout << i << " ";
22         }
23         cout << endl;
24         return;
25     }
26     vacio.pb(nums[pos]);
27     subconjuntos(nums, vacio, pos+1);
28     vacio.pop_back();
29     while(pos<nums.size()-1 && nums[pos+1]==nums[pos])pos++;
30     subconjuntos(nums, vacio, pos+1);
31 }
32
33
34 int main(){

```

```

35 #ifdef EBUG
36     freopen("input.txt", "r", stdin);
37 #endif
38 ios :: sync_with_stdio(false);
39 cin.tie(NULL);
40 cout.tie(NULL);
41
42
43
44 vector<int> nums{1,3,6,9};
45 vector<int> vacio;
46 subconjuntos(nums, vacio, 0);
47
48 /*
49     Salida:
50         1 3 6 9
51         1 3 6
52         1 3 9
53         1 3
54         1 6 9
55         1 6
56         1 9
57         1
58         3 6 9
59         3 6
60         3 9
61         3
62         6 9
63         6
64         9
65 */
66 return 0;
67 }

```

## 9.8 Propiedades del Bitwise XOR ( $\oplus$ )

### 1. Identidad:

$$a \oplus 0 = a$$

### 2. Autoinversión:

$$a \oplus a = 0$$

### 3. Conmutativa:

$$a \oplus b = b \oplus a$$

### 4. Asociativa:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

### 5. Autoeliminación:

$$a \oplus a \oplus b = 0 \oplus b = b$$

6. **Uso práctico: Encontrar elemento único**  
Si todos los elementos de un conjunto aparecen dos veces excepto uno:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_n = \text{elemento único}$$