

ACM-ICPC TEAM REFERENCE DOCUMENT

Universidad de Matanzas

Tabla de Contenido

1 C++	1
1.1 Plantilla	1
1.2 Tipos datos	1
1.3 Conversión de tipos	1
1.4 Ordenamiento	2
1.5 Búsqueda	2
2 Java	3
2.1 Plantilla con Scanner	3
2.2 Plantilla con BufferedReader	3
2.3 Tipos datos	4
2.4 Conversión de tipos	4
2.5 BigInteger	4
2.6 Ordenamiento	4
2.7 Búsqueda	5
3 Aritmética y Álgebra	5
3.1 Exponenciación binaria	5
3.2 Máximo común divisor	5
3.3 Mínimo común múltiplo	5
3.4 Aritmética modular	6
3.5 Multiplicación de matrices	6
4 Strings	6
4.1 Knuth Morris Pratt (KMP)	6
4.2 Manacher	6

5 Teoría de números	7
5.1 Autómata de divisibilidad	7
5.2 Reglas de divisibilidad	7
5.3 Números primos	8
5.3.1 Test primalidad	8
5.3.2 Criba de Eratóstenes (Calcular los números pri- mos hasta 10^6)	8
5.3.3 BigInteger (Java)	8
6 Programación dinámica	9
6.1 Coin Change	9
6.2 Edit distance	9
6.3 Longest Common Subsequence	10
6.4 Cantidad de ocurrencias de un patrón como una subse- cuencia de caracteres de no necesariamente consecutivos	10
6.5 Longest Increasing Subsequences	10
6.6 Mochila 0-1	11
7 Estructura de Datos	11
7.1 Árbol binario indexado o Fenwick Tree	11
7.2 Disjoint Set Union	11
7.3 Range Tree	11
7.4 Range Tree + Lazy Propagation	12
8 Geometría	13
8.1 Figuras geométricas	13
8.1.1 Sector circular	13
8.1.2 Triángulo equilátero	13
8.1.3 Círculo	13
8.2 Intersección de figuras	14
8.2.1 Segmento con Segmento	14

8.3	Closest Pair	14
8.4	Convex Hull	14
9	Teoría de grafos	15
9.1	Representación del grafo	15
9.2	BFS	15
9.3	DFS	15
9.4	Dijkstra	16
9.5	BellmanFord	16
9.6	Floyd-Warshall	17
9.7	Kruskal	17
9.8	Prim	18
9.9	Componentes Conexas	18
9.10	Detención Ciclos	19
9.11	Diametro del árbol	19
9.12	Ordenamiento Topológico	19
9.13	LCA usando Range Tree	20
10	Combinatoria	21
10.1	Variación sin repetición	21
10.2	Variación con repetición	21
10.3	Permutación sin repetición	21
10.4	Permutación con repetición	21
10.5	Combinación	21
10.5.1	Algoritmos	21
10.6	Máscara de bits	22
10.7	Números de Catalan	22
10.8	Triángulo de Pascal	22
11	Fórmulas	23

1 C++

1.1 Plantilla

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stdio.h>
#include <math.h>
#include <stack>
#include <set>
#include <queue>
#include <string>
#include <string.h>
#include <map>
#include <complex>
#include <cmath>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define REPl_N(i,n) for(int i=1;i<=(int)n;++i)
#define FOR(i,c) for(__typeof((c).begin()) i=(c).begin();i!=(c).end();++i)
#define ALL(c) (c).begin(), (c).end()
#define MID (left+right)/2
#define MOD 1000000007
#define MAX 100010
#define MARK 1000
#define MAXTREE (MAX << 2)
#define LIMIT 100000
#define ULL unsigned long long
#define LL long long
#define ENDL '\n'
#define LINE_BLANK cout<<ENDL;

using namespace std;

int main() {

    cout.setf (ios::fixed ,ios::floatfield);
    cout.precision (0) ;
    ios_base::sync_with_stdio(0);
    std::cin.tie(0);

    return 0;
}
```

1.2 Tipos datos

Declaración	Rango
bool	true - false
short	$[-2^{15} .. 2^{15}-1]$
int	$[-2^{31} .. 2^{31}-1]$
long long	$[-2^{63} .. 2^{63}-1]$
float	$[\pm 1.18e - 38 .. \pm 3.40e38]$ Precisión científica (7-dígitos)
double	$[\pm 2.23e - 308 .. \pm 1.79e308]$ Precisión científica (15-dígitos)
long double	$[\pm 3.37e - 4932 .. \pm 1.18e4932]$ Precisión científica (18-dígitos)
char	$[-128 ... 127]$
unsigned char	$[0 ... 255]$

En casos de los tipos de datos que soportan datos numéricos de tipo entero si se coloca delante del tipo dato el modificador **unsigned** indica que la variable solo soportará enteros neutros y positivos desplazando el rango de 0 a $2^{16}-1$, $2^{32}-1$ o $2^{64}-1$ según sea el caso.

1.3 Conversión de tipos

```
// from string to int
#include <cstdlib>
int atoi( const char *str );
int i = atoi("512");

string t="512";
int i = atoi(t.c_str());

// from string to double
#include <cstdlib>
double atof( const char *str );
double i = atof("512.00");

string t="512.00";
double i = atof(t.c_str());

// from string to long
#include <cstdlib>
double atol( const char *str );
```

```

long i = atol("51200");

string t="51200";
long i = atoi(t.c_str());

//from int to string
#include <cstdio>
char result[100];
int num = 24;
sprintf( result, "%d", num );
string t=string(result);

//from double to string
#include <cstdio>
char result[100];
double num = 24.23;
sprintf( result, "%f", num );
string t=string(result);
    
```

1.4 Ordenamiento

```

#include <algorithm>
void sort( iterator start, iterator end );
void sort( iterator start, iterator end, StrictWeakOrdering cmp );

void stable_sort( iterator start, iterator end );
void stable_sort( iterator start, iterator end, StrictWeakOrdering cmp );

/*Es una funcion que permite ordenar los primeros N elementos de una
 * coleccion. N elementos es definido por la cantidad de elementos en
 * rango comprendido [start,end)*/
void partial_sort( iterator start, iterator middle, iterator end );
void partial_sort( iterator start, iterator middle, iterator end,
    StrictWeakOrdering cmp );

//Ejemplo #1
vector<int> v;
v.push_back( 23 ); v.push_back( -1 ); v.push_back( 9999 );
v.push_back( 0 ); v.push_back( 4 );
sort( v.begin(), v.end() );

//Ejemplo #2
int array[] = { 23, -1, 9999, 0, 4 };
unsigned int array_size = 5;
sort( array, array + array_size );

//Ejemplo #3
bool cmp( int a, int b )
{
    
```

```

        return a > b;
    }

vector<int> v;
for( int i = 0; i < 10; i++ )
    v.push_back(i);

sort( v.begin(), v.end(), cmp );
    
```

1.5 Búsqueda

```

/*
 * Como parametros se le pasa el inicio y fin de la coleccion, y el
 * valor a buscar como parametro opcional se le puede pasar una
 * funcion que defina como se puede comparar los elementos de la
 * coleccion. Retorna verdadero en caso de encontrarse el elemento y
 * falso en caso contrario. Como precondition para la utilizacion de
 * este metodo la coleccion debe estar ordenada ascendentemente.
 */
#include <algorithm>
bool binary_search( iterator start, iterator end, const TYPE& val );
bool binary_search( iterator start, iterator end, const TYPE& val, Comp f );

/*
 * Devuelve la posicion o iterador mas a la izquierda donde se puede
 * insertar el elemento deseado de manera que que la coleccion sobre la
 * que se opera se mantenga ordenada. Es claro que como precondition
 * para la utilizacion de este metodo debe estar ordenada
 * ascendentemente.*/
#include <algorithm>
iterator lower_bound( iterator first, iterator last, const TYPE& val );
iterator lower_bound( iterator first, iterator last, const TYPE& val, CompFn f );

/*Su funcionamiento es similar a la anterior, se diferencia que este
 * devuelve posicion o iterador mas a la derecha donde se puede insertar
 * el elemento de forma que la coleccion de mantenga ordenada.*/
#include <algorithm>
iterator upper_bound( iterator start, iterator end, const TYPE& val );
iterator upper_bound( iterator start, iterator end, const TYPE& val,
    StrictWeakOrdering cmp );

int nelements;
int niz[MAX_N];

inline int b_search(int left, int right, int x){
    int i = left;
    int j = right;
    while (i < j){
        
```

```

        int mid = (i+j)/2;
        if (niz[mid] == x)
            return mid;
        if (niz[mid] < x)
            i = mid+1;
        else
            j = mid-1;
    }
    if (niz[i] == x)
        return i;
    return -1;
}

// Se invoca asi: b_search(0,nelements-1,valor);

```

2 Java

2.1 Plantilla con Scanner

```

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Scanner;

public class Main
{
    private Scanner in;
    private PrintWriter out;

    public void solve() throws IOException{
        //Desarrollo de la solucion
    }

    public Main() throws IOException {

        in = new Scanner(new InputStreamReader(System.in));
        out = new PrintWriter(System.out);
        solve();
        in.close();
        out.close();
    }

    public static void main(String[] args) throws IOException {
        new Main();
    }
}

```

2.2 Plantilla con BufferedReader

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.StringTokenizer;

public class Main {
    private BufferedReader in;
    private PrintWriter out;
    private StringTokenizer st;

    void solve() throws IOException {
        // aqui se desarrolla la solucion
    }

    Main() throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));
        out = new PrintWriter(System.out);
        eat("");
        solve();
        in.close();
        out.close();
    }

    private void eat(String str) {
        st = new StringTokenizer(str);
    }

    String next() throws IOException {
        while (!st.hasMoreTokens()) {
            String line = in.readLine();
            if (line == null)
                return null;
            eat(line);
        }
        return st.nextToken();
    }

    int nextInt() throws IOException {
        return Integer.parseInt(next());
    }

    long nextLong() throws IOException {
        return Long.parseLong(next());
    }

    double nextDouble() throws IOException {
        return Double.parseDouble(next());
    }
}

```

```

    }

    public static void main(String[] args) throws IOException {
        new Main();
    }
}

```

2.3 Tipos datos

Declaración	Rango
boolean	true - false
byte	[-128 .. 127]
short	[-32,768 .. 32,767]
int	$[-2^{31} .. 2^{31}-1]$
long	$[-2^{63} .. 2^{63}-1]$
float	$[\pm 3.4 * 10^{-38} .. \pm 3.4 * 10^{38}]$
double	$[\pm 1.7 * 10^{-308} .. \pm 1.7 * 10^{308}]$
char	'\u0000' .. '\uffff' o [0 .. 65.535]

2.4 Conversión de tipos

```

// to X from String
String t=String.valueOf(X);

```

```

// to X from Int
int t=Integer.valueOf(X);

```

```

// to X from Int
double t=Double.valueOf(X);

```

2.5 BigInteger

```

import java.math.BigInteger;
import java.util.Scanner;

```

```

public class Main {
    public static void main(String[] args) {

```

```

        Scanner in=new Scanner(System.in);
        BigInteger a=in.nextBigInteger();

        BigInteger b=new BigInteger("495");

        if(a.mod(b).compareTo(BigInteger.ZERO)==0)
            System.out.println("Es divisible");
        else
            System.out.println("No es divisible");
    }
}

```

2.6 Ordenamiento

```

// ordenamiento ascendente lista

```

```

import java.util.List;
import java.util.Collections;

public class Main{

    public static void main( String args[] ){
        String palos[]={ "Corazones", "Diamantes", "Bastos", "Espadas" };
        List< String > lista = Arrays.asList( palos );
        Collections.sort( lista );

    }
}

```

```

// ordenamiento descendente lista

```

```

import java.util.List;
import java.util.Collections;

public class Main{

    public static void main( String args[] ){
        String palos[]={ "Corazones", "Diamantes", "Bastos", "Espadas" };
        List< String > lista = Arrays.asList( palos );
        Collections.sort( lista, Collections.reverseOrder() );

    }
}

```

```

//Ordenamiento mediante un objeto Comparator

```

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

```

```
public class Main {

    public static void main(String[] args) throws IOException {
        List< Tiempo2 > lista = new ArrayList< Tiempo2 >();
        lista.add(new Tiempo2(6, 24, 34 ));
        lista.add(new Tiempo2( 18, 14, 58 ));
        lista.add(new Tiempo2(6, 05, 34 ));
        lista.add(new Tiempo2( 12, 14, 58 ));
        lista.add(new Tiempo2(6, 24, 22 ));
        Collections.sort( lista, new ComparadorTiempo() );
    }

    class Tiempo2{

        public Tiempo2(int i, int j, int k){}

        public int obtenerHora() {...}

        public int obtenerMinuto() {...}

        public int obtenerSegundo() {...}

    }

    class ComparadorTiempo implements Comparator< Tiempo2 > {
        public int compare( Tiempo2 tiempo1, Tiempo2 tiempo2 ){
            int compararHora = tiempo1.obtenerHora() - tiempo2.obtenerHora();
            if ( compararHora != 0 )
                return compararHora;
            int comparaMinuto = tiempo1.obtenerMinuto() - tiempo2.obtenerMinuto();
            if ( comparaMinuto != 0 )
                return comparaMinuto;
            int compararSegundo = tiempo1.obtenerSegundo() - tiempo2.obtenerSegundo();
            return compararSegundo;
        }
    }

    //Ordenamiento de arreglo de ascendente

    import java.io.IOException;
    import java.util.Arrays;

    public class Main {

        public static void main(String[] args) throws IOException {
            double arregloDouble[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
            Arrays.sort( arregloDouble );
        }
    }
}
```

2.7 Búsqueda

*/*las clases Arrays y Collections poseen el metodo busqueda binaria
 * para buscar un elemento bien sea dentro de un arreglo o coleccion,
 * en ambos caso devuelve la posicion donde se encuentra el elemento
 * buscado. En caso de no encontrarse el valor devuelto es -1. El arreglo
 * o coleccion debe esta ordenado de forma ascendente previamente */*

3 Aritmética y Álgebra

3.1 Exponenciación binaria

```
LL binpow (LL a, LL n){ //retorna a^n
    LL res = 1;
    while(n){
        if(n&1)
            res*= a;
        a*=a;
        n>>=1;
    }
    return res;
}
```

3.2 Máximo común divisor

```
int gcd(int a,int b) { // a>= b
    while (b > 0){
        a=a%b;
        a ^= b;
        b ^= a;
        a ^= b;
    }
    return a;
}
```

3.3 Mínimo común múltiplo

```
int gcd(int a,int b) { // a>= b
    while (b > 0){
        a=a%b;
        a ^= b;
        b ^= a;
        a ^= b;
    }
    return a;
}
```

3.4 Aritmética modular

1. $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
2. $(a - b) \bmod n = ((a \bmod n) - (b \bmod n) + n) \bmod n$
3. $(a/b) \bmod n = a * b^{-1} \bmod n$
Donde b^{-1} es el inverso multiplicado entre b y n .
4. $(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$

3.5 Multiplicación de matrices

```
typedef long long lld;

/*
 * complejidad O(n*l*m) donde n son las filas de A, l las columnas y
 * filas de A y B respectivamente, mientras m son las columnas de B.
 */

inline lld** matrixMultiply(lld** a, lld** b, int n, int l, int m){
    lld **c = new lld*[n];
    for (int i=0;i<n;i++) c[i] = new lld[m];

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            c[i][j]=0;
            for(int k=0;k<l;k++){
                c[i][j] += a[i][k]*b[k][j];
            }
        }
    }
    return c;
}
```

4 Strings

4.1 Knuth Morris Pratt (KMP)

```
vector<int> KMP(string _text, string _pattern){
    vector<int> matches;

    int * P=new int[MAX];
    /* MAX debe ser un valor mayor que la maxima
     * longitud de las cadenas */

    int n_pattern=_pattern.size();
    int n_text=_text.size();
    for (int i=0;i<n_pattern;i++){
        P[i] = -1;
    }
    for (int i=0, j=-1;i<n_pattern;){
        while (j > -1 && _pattern[i] != _pattern[j])
            j = P[j];
        i++;j++;
        P[i] = j;
    }

    for (int i=0, j=0;i<n_text;){
        while (j > -1 && _text[i] != _pattern[j])
            j = P[j];
        i++;j++;
        if (j == n_pattern){
            matches.push_back(i-n_pattern);j = P[j];
        }
    }
    return matches;
}
```

4.2 Manacher

```
int manacher(string _text){
    int n=_text.size();
    int rad[2*n];
    int i,j,k;
    for(i=0,j=0;i<2*n;i+=k, j=max(j-k,0)){
        while(i-j>0 && i+j+1<2*n && _text[(i-j)/2]==_text[(i+j+1)/2])
            ++j;
        rad[i]=j;
        for(k=1; i-k>=0 && rad[i]-k>=0 && rad[i-k]!=rad[i]-k ;++k)
            rad[i+k]=min(rad[i-k],rad[i]-k);
    }
```



```

    }
    return *max_element(rad,rad+(2*n));
}

```

5 Teoría de números

5.1 Autómata de divisibilidad

```

int value(char _symbol){
    if('0' <= _symbol && _symbol <='9')
        return (_symbol-'0');
    if('A' <= _symbol && _symbol <='F')
        return (_symbol-'A')+10;
    if('a' <= _symbol && _symbol <='f')
        return (_symbol-'a')+10;
    return 0;
}

bool automatDivisible(string _number,int _base, int _decimal){
    int state=0;
    int size=_number.size();
    REP(i,size)
        state=(state*_base+value(_number[i]))%_decimal;
    if(!state) return true;
    else return false;
}

```

5.2 Reglas de divisibilidad

- **2:** Si termina en 0, 2, 4, 6 ó 8.
- **3:** Si la suma de sus cifras es múltiplo de 3.
- **4:** Si sus dos últimas cifras son 00 ó un múltiplo de 4 (12, 16, 20, 24, 28, 32, 36 y 40).
- **5:** Si termina en 0 y 5.
- **7:** Cuando la diferencia entre el número sin la cifra de las unidades y el doble de la cifra de las unidades es 0 ó múltiplo de 7. 34349: separamos el 9,y lo doblamos (18), entonces $3434-18=3416$. Repetimos el proceso separando el 6 (341'6) y

doblandolo (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 múltiplo de 7.

- **8:** Si sus tres últimas cifras son ceros o múltiplo de 8.
- **9:** Si la suma de sus dígitos nos da múltiplo de 9.
- **10:** Si la cifra de las unidades es 0.
- **11:** Si la suma de las cifras que ocupan un lugar par menos la suma de las otras cifras es 0 ó un múltiplo de 11 (11, 22, 33, 44,...)
- **13:** Al separar la última cifra de la derecha, multiplicarla por 9 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 13.
- **17:** Al separar la última cifra de la derecha, multiplicarla por 5 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 17
- **19:** Si al separar la cifra de las unidades, multiplicarla por 2 y sumar a las cifras restantes el resultado es múltiplo de 19.
- **25:** Si sus dos últimas cifras son ceros o múltiplo de 25.
- **29:** Si al separar la cifra de las unidades, multiplicarla por 3 y sumar a las cifras restantes el resultado es múltiplo de 29.
- **31:** Si al separar la cifra de las unidades, multiplicarla por 3 y restar a las cifras restantes el resultado es múltiplo de 31.

5.3 Números primos

5.3.1 Test primalidad

```
//clasico
bool isPrime(int _n){
    bool prime=true;
    for(int i=2;i*i<=_n;i++)
        if(_n%i==0)
            prime=false;
    if(_n==1) prime=false;
    return prime;
}

// test miller rabin
int powMod (int x, int k, int m) {
    if (k == 0) return 1;
    if (k%2 == 0) return powMod (x*x%m, k/2, m);
    else return x * powMod (x, k-1, m)% m;
}

bool suspect (int a, int s, int d, int n) {
    int x = powMod (a, d, n);
    if (x == 1) return true;
    for (int r = 0; r<s; ++r) {
        if (x == n - 1) return true;
        x = x*x%n;
    }
    return false;
}

// {2, 7, 61, - 1} para n <4759123141 (= 2 ^ 32)
// {2, 3, 5, 7, 11, 13, 17, 19, 23, - 1} para n <10 ^ 16 (hasta el momento)
bool isPrime (int n) {
    if (n<=1 || (n>2 && n%2==0)) return false;
    int test [] = {2, 3, 5, 7, 11, 13, 17, 19, 23, - 1};
    int d = n - 1, s = 0;
    while (d%2==0) ++s, d/= 2;

    for (int i = 0; test[i] <n && test [i]!=-1; ++i)
        if (!suspect (test[i],s,d,n))
            return false;
    return true;
}

//Usando Java y BigInteger
Scanner in = new Scanner(new InputStreamReader(System.in));
PrintWriter out= new PrintWriter(System.out);
BigInteger number = in.nextBigInteger();
if(number.isProbablePrime(10)){
```

```
    out.println("yes");
}
else
{
    out.println("no");
}
```

5.3.2 Criba de Eratóstenes (Calcular los números primos hasta 10^6)

```
#include <math.h>
#include <cstring>
#define MAX_N 1000001

vector<int> primes;
bool mark[MAX_N];

inline void sieve(){
    memset(mark, false, sizeof(mark));
    for(int i=2; i<=sqrt(MAX_N); i++){
        if(!mark[i]){
            primes.push_back(i);
            for(int j=i*i; j<MAX_N; j+=i)
                mark[j]=true;
        }
    }
    for(int e=sqrt(MAX_N)+1; e<MAX_N; e++)
        if(!mark[e])
            primes.push_back(e);
}
```

5.3.3 BigInteger (Java)

```
/*
 * El java.math.BigInteger.isProbablePrime (int certeza)
 * devuelve verdadero si este BigInteger es probablemente primo, falso
 * si es definitivamente compuesto. Si la certeza es <= 0, se devuelve
 * verdadero.
 *
 * certeza: Una medida de la incertidumbre que la persona que llama
 * esta dispuesta a tolerar: si la llamada devuelve verdadera, la
 * probabilidad de que este BigInteger sea primo excede (1 - 1/2 de
 * incertidumbre). El tiempo de ejecucion de este metodo es
 * proporcional al valor de este parametro.
 */
```

```

* */
BigInteger bi1, bi2;

Boolean b1, b2, b3;

bi1 = new BigInteger("7");
bi2 = new BigInteger("9");

b1 = bi1.isProbablePrime(10);
b2 = bi2.isProbablePrime(10);

/*
 * public static BigInteger probablePrime(int bitLength, Random rnd)
 *
 * El metodo probablePrime() devolvera un BigInteger de bits bitLength
 * que es primo. bitLength se proporciona como parametro al metodo
 * probablePrime () y el metodo devolvere un BigInteger principal de
 * bits bitLength. La probabilidad de que un BigInteger devuelto por
 * este metodo sea compuesta y no exceda de 2 ^ -100.
 *
 *
 * bitLength : cantidad de bits usados para generar el numero primo
 * rnd: fuente de bits aleatorios utilizados para seleccionar candidatos
 * para ser probados para primalidad.*/

BigInteger biginteger;

// create a integer value for bitLength
int length = 4;

// create a random object
Random random = new Random();

// call probablePrime method to find next probable prime
// whose bit length is equal to bitLength provided as parameter.
biginteger = BigInteger.probablePrime(length, random);

/*
 * nextProbablePrime (): Otro metodo presente en la clase BigInteger.
 * Esta funcion devuelve el siguiente numero primo mayor que BigInteger
 * actual.
 * */

// create 2 BigInteger objects
BigInteger bi1, bi2;

bi1 = new BigInteger("20");

// assign nextProbablePrime value of bi1 to bi2
bi2 = bi1.nextProbablePrime();

```

6 Programación dinámica

6.1 Coin Change

```

/*Forma de devolver un valor N con k denominaciones de billetes*/
long long nway[MAX+1];

int coin[5] = { 50,25,10,5,1 };
v = 5;
way[0] = 1;
for (i=0; i<v; i++) {
    c=coin[i];
    for (j=c; j<=n; j++)
        way[j]+=way[j-c];
}

/* minima cantidad de billetes necesarios para devolver una cantidad N
 * teniendo infinita cantidad de bittetes de k denominaciones.*/

for (int i=0; i<MAX; i++) {
    changes[i]=i;
}
for (int i=0; i<nmoney; i++)
    changes[money[i]]=1;

for (int i=1; i<MAX; i++) {
    for (int j=0; j<nmoney; j++) {
        if (i+money[j]<MAX && changes[i+money[j]]>changes[i]+1)
            changes[i+money[j]]=changes[i]+1;
    }
}

```

6.2 Edit distance

```

unsigned int edit_distance(string s1, string s2) {
    const size_t len1 = s1.size(), len2 = s2.size();
    vector < vector < unsigned int > > d(len1 + 1, vector < unsigned int > (len2 + 1, 0));
    d[0][0] = 0;

    for (unsigned int i=1; i<=len1; ++i)
        d[i][0]=i;
    for (unsigned int i=1; i<=len2; ++i)
        d[0][i]=i;
}

```

```

for (unsigned int i=1; i<=len1; ++i)
    for (unsigned int j=1; j<=len2; ++j)
        d[i][j] =
            min(
                min(d[i-1][j]+1,d[i][j-1]+1),
                d[i-1][j-1]+(s1[i-1] == s2[j-1] ? 0 : 1));
return d[len1][len2];
}

```

6.3 Longest Common Subsequence

```

#define MAX_N 1001

int n, m;
string A, B;
int dp[MAX_N][MAX_N];

inline int LCS(){
    for (int i=0; i<=n; i++) dp[i][0] = 0;
    for (int j=0; j<=m; j++) dp[0][j] = 0;
    for (int i=1; i<=n; i++){
        for (int j=1; j<=m; j++){
            if (A[i-1] == B[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
        }
    }
    return dp[n][m];
}

inline string getLCS(){
    string ret;
    stack<char> S;
    int ii = n, jj = m;
    while (ii != 0 && jj != 0){
        if (A[ii-1] == B[jj-1]){
            S.push(A[ii-1]);
            ii--; jj--;
        }
        else if (dp[ii-1][jj] > dp[ii][jj-1])
            ii--;
        else
            jj--;
    }
    while (!S.empty()) {
        ret += S.top();
        S.pop();
    }
}

```

```

return ret;
}

```

6.4 Cantidad de ocurrencias de un patrón como una subsecuencia de caracteres de no necesariamente consecutivos

```

unsigned long long count(string X, string Y, int m, int n){
    unsigned long long T[m + 1][n + 1];
    for (int i = 0; i <= m; i++)
        T[i][0] = 1;
    for (int j = 1; j <= n; j++)
        T[0][j] = 0;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            T[i][j] = ((X[i-1] == Y[j-1]) ? T[i-1][j-1] : 0) + T[i-1][j];
    return T[m][n];
}

```

6.5 Longest Increasing Subsequences

```

#include <vector>
#include <algorithm>
using namespace std;

const int inf = 99999999;

int indexOf(vector<int> as, int x){
    return distance(as.begin(), lower_bound(as.begin(), as.end(), x));
}

vector<int> lis_fast (const vector<int> & a) {
    const int n = a.size ();
    vector<int> A (n, inf);
    vector<int> id (n) ;
    for (int i=0; i<n; ++i) {
        id[i]=indexOf(A,a[i]);
        A[id[i]]=a[i];
    }
    int m = * max_element(id.begin(),id.end());
    vector<int> b (m + 1);
    for (int i=n-1; i>=0; --i)
        if (id[i]==m)

```

```

        b[m--]=a[i];
    return b;
}

```

6.6 Mochila 0-1

```

#define MAXELEMENT 250
#define MAXCAPACITYKNAPSACK 1001

int nelements, capacity;
int weight[MAXELEMENT], value[MAXELEMENT], sol[MAXCAPACITYKNAPSACK];

int knapsack01() {
    for(int i=0; i<=capacity; i++) sol[i]=0;
    for(int i=0; i<nelements; i++){
        for(int j=capacity; j>=1; j--){
            if(weight[i]<=j){
                int x=sol[j];
                int y=sol[j-weight[i]]+value[i];
                sol[j] = max(x,y);
            }
        }
    }
    return sol[capacity];
}

```

7 Estructura de Datos

7.1 Árbol binario indexado o Fenwick Tree

```

#include <vector>
using namespace std;
template <class T>
struct fenwick_tree{
    vector <T> x;
    fenwick_tree (int n): x(n, 0) {}
    T sum(int i, int j){
        if(i==0){
            T S = 0;
            for (j; j>= 0; j=(j & (j+1))-1) S+=x[j];
            return S;
        } else return sum (0, j) - sum (0, i-1);
    }
}

```

```

void add (int k, T a){
    for (; k<x.size(); k|=k+1)
        x[k]+=a;
}
};

```

7.2 Disjoint Set Union

```

#include <iostream>
#define MAX_N 1000001
using namespace std;

int numComponents;

struct Node{ int parent; int rank; };
Node DSU[MAX_N];

inline void MakeSet (int x){ DSU[x].parent = x; DSU[x].rank = 0; }

inline int Find(int x){
    if (DSU[x].parent == x) return x;
    DSU[x].parent = Find(DSU[x].parent); return DSU[x].parent; }

inline void Union(int x, int y){
    int xRoot=Find(x); int yRoot=Find(y);
    if (xRoot==yRoot) return;
    numComponents--;
    if (DSU[xRoot].rank<DSU[yRoot].rank){ DSU[xRoot].parent=yRoot; }
    else if (DSU[xRoot].rank>DSU[yRoot].rank){ DSU[yRoot].parent=xRoot; }
    else{ DSU[yRoot].parent = xRoot; DSU[xRoot].rank++; }
}

int main(){ numComponents = 4;
    MakeSet (1);MakeSet (2);MakeSet (3);MakeSet (4);
    Union (1, 2);Union (2, 1);Union (2, 3);Union (1, 3);
    cout<<numComponents<<endl;
    return 0;
}

```

7.3 Range Tree

```

#define MID (left+right)/2
#define MAX_N 1000001
#define MAX_TREE (MAX_N << 2)

```

```

#define INF 987654321
using namespace std;

int n;
int niz[MAX_N];
int RT[MAX_TREE];

void InitTree(int idx, int left, int right){
    if (left == right){
        RT[idx] = niz[left];return;
    }
    InitTree(2*idx, left, MID);
    InitTree(2*idx+1, MID+1, right);
    RT[idx] = min(RT[2*idx], RT[2*idx+1]);
}

void Update(int idx, int x, int val, int left, int right){
    if (left == right){
        RT[idx] = val;return;
    }
    if (x <= MID) Update(2*idx, x, val, left, MID);
    else Update(2*idx+1, x, val, MID+1, right);
    RT[idx] = min(RT[2*idx], RT[2*idx+1]);
}

int Query(int idx, int l, int r, int left, int right){
    if (l <= left && right <= r) return RT[idx];
    int ret = INF;
    if (l <= MID) ret = min(ret, Query(2*idx, l, r, left, MID));
    if (r > MID) ret = min(ret, Query(2*idx+1, l, r, MID+1, right));
    return ret;
}

int main(){
    n = 6;
    niz[1] = 4;niz[2] = 2;
    niz[3] = 5;niz[4] = 1;
    niz[5] = 6;niz[6] = 3;
    InitTree(1, 1, n);
    printf("%d\n",Query(1, 1, 3, 1, n));
    Update(1, 4, 10, 1, n);
    Update(1, 5, 0, 1, n);
    printf("%d\n",Query(1, 4, 6, 1, n));
    return 0;
}

```

7.4 Range Tree + Lazy Propagation

*/*Range Tree con lazy propagation para obtener la suma de intervalos y*

```

    * realizar actualizaciones en el intervalos. inicialmente el valor de*/
#define MID (left+right)/2
#define MAX_N 1000001
#define MAX_TREE (MAX_N << 2)
#define LL long long
using namespace std;

int n;
LL niz[MAX_N];
LL tree[MAX_TREE];
LL lazy[MAX_TREE];

void buildSegTree(int treeIndex, int left, int right){
    if (left == right) {
        tree[treeIndex] = niz[left];lazy[treeIndex]=-1;return;
    }

    buildSegTree( 2 * treeIndex , left, MID);
    buildSegTree( 2 * treeIndex+1, MID + 1, right);

    tree[treeIndex] = tree[2 * treeIndex] + tree[2 * treeIndex + 1];
    lazy[treeIndex]=-1;
}

void updateLazySegTree(int treeIndex, int left, int right, int i,
int j, int val){

    if (left > right || left > j || right < i)
        return;

    if (i <= left && right <= j){
        tree[treeIndex] = (right - left + 1) * val;

        if (left != right){
            lazy[2 * treeIndex ] = val;
            lazy[2 * treeIndex + 1] = val;
        }

        return;
    }

    if (lazy[treeIndex] != -1) {
        tree[treeIndex] = (right - left + 1) * lazy[treeIndex];
        if (left != right) {
            lazy[2 * treeIndex ] = lazy[treeIndex];
            lazy[2 * treeIndex + 1] = lazy[treeIndex];
        }
        lazy[treeIndex] = -1;
    }
}

```

```

updateLazySegTree(2 * treeIndex , left, MID, i, j, val);
updateLazySegTree(2 * treeIndex + 1, MID + 1, right, i, j, val);

tree[treeIndex] = tree[2 * treeIndex ] + tree[2 * treeIndex + 1];
}

LL queryLazySegTree(int treeIndex, int left, int right, int i, int j){

    if (left > j || right < i)
        return 0;

    if (lazy[treeIndex] != -1){
        tree[treeIndex] = (right - left + 1) * lazy[treeIndex];
        if (left != right) {
            lazy[2 * treeIndex ] = lazy[treeIndex];
            lazy[2 * treeIndex + 1] = lazy[treeIndex];
        }

        lazy[treeIndex] = -1;
    }

    if (i <= left && j >= right)
        return tree[treeIndex];

    if (i > MID)
        return queryLazySegTree(2*treeIndex+1,MID+1,right,i,j);
    else if (j <= MID)
        return queryLazySegTree(2*treeIndex,left,MID,i,j);

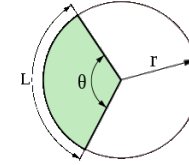
    LL leftQuery =queryLazySegTree(2*treeIndex,left,MID,i,MID);
    LL rightQuery =queryLazySegTree(2*treeIndex+1,MID+1,right,MID+1,j);

    return leftQuery + rightQuery;
}
    
```

8 Geometría

8.1 Figuras geométricas

8.1.1 Sector circular

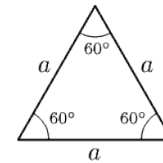


El área de un sector circular depende de dos parámetros, el radio y el ángulo central, y está dada por las siguientes fórmulas equivalentes:

$$A = \frac{rL}{2} = \frac{r^2\theta}{2} = \pi r^2 \frac{\alpha}{360}$$

r es el radio, L es la longitud del arco ($0 < L < 2\pi r$), θ es el ángulo central en radianes ($L = \theta r$ y $0 < \theta < 2\pi$)

8.1.2 Triángulo equilátero



Perímetro es igual a $p = 3a$

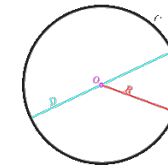
El radio de un círculo circunscrito es $R = a \frac{\sqrt{3}}{3}$

El radio de un círculo inscrito es $r = a \frac{\sqrt{3}}{6}$

La altura es $h = a \frac{\sqrt{3}}{2}$

El valor del área, en función del lado a , es igual a $A = a^2 \frac{\sqrt{3}}{4}$, en función del radio R de la circunferencia circunscrita, es igual a $A = 3R^2 \frac{\sqrt{3}}{4}$, usando el radio r del radio de la circunferencia inscrita, es igual a $A = 3r^2 \sqrt{3}$

8.1.3 Círculo



El perímetro de un círculo es $p = 2r\pi$ o $p = d\pi$

Un círculo de radio r tendrá un área $A = \pi r^2$ en función del radio (r) o $A = \frac{\pi d^2}{4}$ en función del diámetro (d). $A = \frac{C^2}{4\pi}$ en función de la longitud de la circunferencia.

8.2 Intersección de figuras

8.2.1 Segmento con Segmento

```
#include <complex>
#define EPS 1e-8

struct Point : public complex<double> {
    Point(double _x=0, double _y=0):complex<double>(_x,_y){} };

struct Segment{
    Point p1,p2; Segment(Point _P1,Point _P2){p1=_P1;p2=_P2;} Segment(){} };

bool operator <(const Point &a, const Point &b)
{ return real(a)!=real(b) ? real(a)<real(b) : imag(a)<imag(b); }

bool operator ==(const Point &a, const Point &b)
{ return (abs(real(a)-real(b)) <EPS && abs(imag(a)-imag(b)) <EPS); }

double cross(Point a, Point b) { return imag(conj(a)*b); }
double dot(Point a, Point b) { return real(conj(a)*b); }

int ccw(Point a,Point b,Point c){
    b-=a; c-=a;
    if(cross(b,c)>0)return +1; if(cross(b,c)<0)return -1;
    //c - a - b line //a - b - c line
    if(dot(b,c)<0)return +2; if(norm(b)<norm(c))return -2;
    return 0; }

bool intersectSS(Segment s, Segment t){
    if(abs(s.p1-t.p1) < EPS || abs(s.p1-t.p2) < EPS ||
       abs(s.p2-t.p1) < EPS || abs(s.p2-t.p2) < EPS)
        return 1;
    return ccw(s.p1,s.p2,t.p1) * ccw(s.p1,s.p2,t.p2) <= 0 &&
           ccw(t.p1,t.p2,s.p1) * ccw(t.p1,t.p2,s.p2) <= 0; }
```

8.3 Closest Pair

```
struct Point {
    double X,Y;
    Point (double _X=0,double _Y=0){X=_X;Y=_Y;}

    bool operator<(const Point &P) const{
        if(X < P.X)
            return true;
        if(X > P.X)
            return false;
```

```
        return Y < P.Y;
    }

    void readPoint(){cin>>X>>Y;}
};

double distancePointToPoint(Point _a,Point _b){
    double dist=(_a.X-_b.X)*(_a.X-_b.X)+(_a.Y-_b.Y)*(_a.Y-_b.Y);
    return sqrt(dist);
}

double closestPairWithLineSweep(Point _points [], int _npoint){
    sort(_points+1,_points+_npoint+1);
    if(_points == 0)
        return 0;
    double dist = INF; /* INF debe ser un valor que sea mayor que la
        distancia maxima en que se pueda encontrar dos puntos segun el
        problema. Si dicen que las coordenadas de los puntos va estar entre
        -10^9 a 10^9 entonces:
        INF= distancePointToPoint( Point(-10^9,-10^9),Point(10^9,10^9) )+100
        */
    int tBegin = 1, tEnd = 1, tot = 1;

    for(int i=2;i<=_npoint;i++){
        while(tot > 0 && 0.0+_points[i].X-_points[tBegin].X > dist) {
            tBegin++;
            tot--;
        }

        for(int j=tBegin;j<=tEnd;j++){
            dist = min(dist, distancePointToPoint(_points[i],_points[j]));
            tEnd++;
            tot++;
        }

        return dist;
    }
}
```

8.4 Convex Hull

```
struct pt { double x, y;};

bool cmp (pt a, pt b) {
    return a.x<b.x || a.x==b.x && a.y<b.y;
}

bool cw (pt a, pt b, pt c) {
    return a.x * (b.y-c.y) + b.x * (c.y-a.y) + c.x * (a.y-b.y) <0;
}
```



```

bool ccw (pt a, pt b, pt c) {
    return a.x * (b.y-c.y) + b.x * (c.y-a.y) + c.x * (a.y-b.y) > 0;
}

void convexHull (vector<pt> &a) {
    if (a.size () == 1) return;
    sort (a.begin (), a.end (), &cmp);
    pt p1 = a [0], p2 = a.back ();
    vector<pt> up, down;
    up.push_back (p1);
    down.push_back (p1);
    for(size_t i = 1; i <a.size (); ++i) {
        if(i == a.size () -1 || cw (p1, a [i], p2)){
            while(up.size()>=2 && !cw(up[up.size()-2],up[up.size()-1],a[i]))
                up.pop_back ();
            up.push_back (a [i]);
        }
        if(i==a.size()-1 || ccw (p1, a [i], p2)) {
            while(down.size()>=2 &&
                !ccw(down[down.size()-2],down[down.size()-1],a[i]))
                down.pop_back();
            down.push_back (a [i]);
        }
    }
    a.clear ();
    for (size_t i = 0; i <up.size (); ++ i)
        a.push_back (up [i]);
    for (size_t i = down.size () -2; i > 0;--i)
        a.push_back (down [i]);
}

```

9 Teoría de grafos

9.1 Representación del grafo

```

typedef int Weight;
struct Edge {
    int src, dst;
    Weight weight;
    Edge(int src, int dst, Weight weight) :
        src(src), dst(dst), weight(weight) { }
};

bool operator < (const Edge &e, const Edge &f) {
    return e.weight != f.weight ? e.weight > f.weight : // !!INVERSE!!
        e.src != f.src ? e.src < f.src : e.dst < f.dst;
}

```

```

typedef vector<Edge> Edges;
typedef vector<Edges> Graph;

typedef vector<Weight> Array;
typedef vector<Array> Matrix;

```

9.2 BFS

```

#include <queue>
#define MAX 5001
using namespace std;

struct Node{
    vector<int> adj;
};

Node graf[MAX];
bool mark[MAX];

inline void BFS(int start){
    queue<int> bfs_queue;
    bfs_queue.push(start);
    while (!bfs_queue.empty()){
        int xt = bfs_queue.front();
        bfs_queue.pop();
        mark[xt] = true;
        for(int i=0;i<graf[xt].adj.size();i++){
            if(!mark[graf[xt].adj[i]]){
                bfs_queue.push(graf[xt].adj[i]);
                mark[graf[xt].adj[i]] = true;
            }
        }
    }
}

```

9.3 DFS

```

// Variante no recursiva
#include <stack>
#define MAX 5001
using namespace std;

struct Node{
    vector<int> adj;
};

Node graf[MAX];

```

```

bool mark[MAX];

inline void DFS(int start){
    stack<int> dfs_stek;
    dfs_stek.push(start);
    while (!dfs_stek.empty()){
        int xt = dfs_stek.top();
        dfs_stek.pop();
        mark[xt] = true;
        for (int i=0;i<graf[xt].adj.size();i++){
            if (!mark[graf[xt].adj[i]]){
                dfs_stek.push(graf[xt].adj[i]);
                mark[graf[xt].adj[i]] = true;
            }
        }
    }
}

//variante recursiva
inline void DFS(int _node){
    mark[_node]=true;
    for (int i=0;i<graf[_node].adj.size();i++){
        if (!mark[graf[_node].adj[i]]){
            DFS(graf[_node].adj[i]);
        }
    }
}

```

9.4 Dijkstra

```

#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#define MAX_N 100001
#define INF 987654321
using namespace std;

int nnodes;

struct Node{
    int dist;
    vector<int> adj;
    vector<int> weight;
};
Node graf[MAX_N];
bool mark[MAX_N];

```

```

struct pq_entry{
    int node, dist;
    bool operator <(const pq_entry &a) const{
        if (dist != a.dist) return (dist > a.dist);
        return (node > a.node);
    }
};

inline void dijkstra(int source){
    priority_queue<pq_entry> pq;
    pq_entry P;
    for (int i=0;i<nnodes;i++){
        if (i == source){
            graf[i].dist = 0;
            P.node = i;
            P.dist = 0;
            pq.push(P);
        }
        else graf[i].dist = INF;
    }
    while (!pq.empty()){
        pq_entry curr = pq.top();
        pq.pop();
        int nod = curr.node;
        int dis = curr.dist;
        for (int i=0;i<graf[nod].adj.size();i++){
            if (!mark[graf[nod].adj[i]]){
                int nextNode = graf[nod].adj[i];
                if (dis + graf[nod].weight[i] < graf[nextNode].dist){
                    graf[nextNode].dist = dis + graf[nod].weight[i];
                    P.node = nextNode;
                    P.dist = graf[nextNode].dist;
                    pq.push(P);
                }
            }
        }
        mark[nod] = true;
    }
}

```

9.5 BellmanFord

```

#include <stdio.h>
#include <iostream>
#define MAX_N 5001
#define MAX_E 25000001
#define INF 987654321
using namespace std;
typedef long long lld;

```

```

int v, e;

int dist[MAX_N];
struct Edge{
    int x, y, weight;
};
Edge E[MAX_N];

inline int BellmanFord(int source){
    for (int i=0; i<v; i++){
        if (i == source) dist[i]=0;
        else dist[i] = INF;
    }
    bool done = false;
    for (int i=0; !done && i<v; i++){
        done = true;
        for (int j=0; j<e; j++){
            int so = E[j].x;
            int de = E[j].y;
            if (dist[so] + E[j].weight < dist[de]){
                dist[de] = dist[so] + E[j].weight;
                done=false;
            }
        }
    }
    if (!done) return -1; //ciclo negativo detectado
    return 0;
}
    
```

9.6 Floyd-Warshall

```

inline void FloydWarshall(){
    for (int i=1; i<=n; i++){
        for (int j=1; j<=n; j++){
            flojd[i][j] = dist[i][j];
        }
        flojd[i][i] = 0;
    }
    for (int k=1; k<=n; k++){
        for (int i=1; i<=n; i++){
            for (int j=1; j<=n; j++){
                if (flojd[i][k] + flojd[k][j] < flojd[i][j]){
                    flojd[i][j] = flojd[i][k] + flojd[k][j];
                }
            }
        }
    }
}
    
```

9.7 Kruskal

```

#include <algorithm>
#define MAX_NODE 100001
#define MAX_EDGE 50000
using namespace std;
typedef long long lld;

int nnodes, nedges;
int numComponents;

struct Edge{
    int a, b;
    int weight;
    bool operator <(const Edge &e) const{
        return (weight < e.weight);
    }
};
Edge E[MAX_EDGE];

struct Node{
    int parent;
    int rank;
};
Node DSU[MAX_NODE];

inline void MakeSet(int x){
    DSU[x].parent = x;
    DSU[x].rank = 0;
}

inline int Find(int x){
    if (DSU[x].parent == x) return x;
    DSU[x].parent = Find(DSU[x].parent);
    return DSU[x].parent;
}

inline void Union(int x, int y){
    int xRoot = Find(x);
    int yRoot = Find(y);
    if (xRoot == yRoot) return;
    if (DSU[xRoot].rank < DSU[yRoot].rank){
        DSU[xRoot].parent = yRoot;
    }
    else if (DSU[xRoot].rank > DSU[yRoot].rank){
        DSU[yRoot].parent = xRoot;
    }
    else{
        DSU[yRoot].parent = xRoot;
        DSU[xRoot].rank++;
    }
}
    
```

```

inline int kruskal(){
    int ret = 0, numComponents = nnodes;
    for (int i=0;i<nnodes;i++) MakeSet(i);
    sort(E, E+nedges);
    for (int i=0;i < nedges && numComponents > 1;i++){
        if (Find(E[i].a) != Find(E[i].b)){
            Union(E[i].a, E[i].b);
            ret += E[i].weight;
            numComponents--;
        }
    }
    if (numComponents > 1) return -1; // No existe un MST
    return ret;
}

```

9.8 Prim

```

#include <vector>
#include <algorithm>
#include <queue>
#include <stack>
#define MAX_N 101
using namespace std;
typedef long long lld;

int nnodes;
struct Node{
    vector<int> adj;
    vector<int> weight;
};

Node graf[MAX_N];
bool mark[MAX_N];

struct edge{
    int u, v;
    double w;
    bool operator <(const edge &a) const{
        return (w>a.w);
    }
};

priority_queue<edge> pq_prim;

inline int prim(int _source){
    int xt = _source;
    int ret = 0;
    int amt = 0;
    while (amt < nnodes-1){

```

```

        mark[xt]=true;
        for (int i=0;i<graf[xt].adj.size();i++){
            if (!mark[graf[xt].adj[i]]){
                edge E;
                E.u = xt;
                E.v = graf[xt].adj[i];
                E.w = graf[xt].weight[i];
                pq_prim.push(E);
            }
        }
        while (!pq_prim.empty()){
            edge X = pq_prim.top();
            pq_prim.pop();
            if (!mark[X.v]){
                ret += X.w;
                xt = X.v;
                amt++;
                break;
            }
        }
    }
    return ret;
}

```

9.9 Componentes Conexas

```

#include <stack>
#include <vector>
#define MAX_N 5001
using namespace std;
typedef long long lld;

int n;

struct Node{
    vector<int> adj;
};

Node graf[MAX_N];
bool mark[MAX_N];

inline void DFS(int start){
    stack<int> dfs_stek;
    dfs_stek.push(start);
    while (!dfs_stek.empty()){
        int xt = dfs_stek.top();
        dfs_stek.pop();
        mark[xt] = true;
        for (int i=0;i<graf[xt].adj.size();i++){

```

```

        if (!mark[graf[xt].adj[i]]){
            dfs_stek.push(graf[xt].adj[i]);
            mark[graf[xt].adj[i]] = true;
        }
    }
}

inline int numComponents(int n)
{
    int ret = 0;
    for (int i=0;i<n;i++){
        if (!mark[i]){
            DFS(i);
            ret++;
        }
    }
    return ret;
}

```

9.10 Detención Ciclos

```

#include <vector>
#include <stack>
#include <vector>
#include <complex>
#define MAX_N 5001
using namespace std;
typedef long long lld;

int n;

struct Node{
    vector<int> adj;
};

Node graf[MAX_N];
bool mark[MAX_N];

inline bool DFS(int start){
    stack<int> dfs_stek;
    stack<int> previous;
    dfs_stek.push(start);
    previous.push(-1);
    while (!dfs_stek.empty()){
        int xt = dfs_stek.top();
        int pt = previous.top();
        dfs_stek.pop();
        previous.pop();
    }
}

```

```

mark[xt] = true;
for (int i=0;i<graf[xt].adj.size();i++){
    if (!mark[graf[xt].adj[i]]){
        dfs_stek.push(graf[xt].adj[i]);
        previous.push(xt);
        mark[graf[xt].adj[i]] = true;
    }
    else if (graf[xt].adj[i] != pt)
        return true;
}
}

return false;
}

inline bool hasCycle(int n){
    for(int i=0;i<n;i++){
        if(!mark[i]){
            if(DFS(i)) return true;
        }
    }
    return false;
}

```

9.11 Diametro del árbol

```

typedef pair<Weight, int> Result;
Result visit(int p, int v, const Graph &g) {
    Result r(0, v);
    FOR(e, g[v]) if (e->dst != p) {
        Result t = visit(v, e->dst, g);
        t.first += e->weight;
        if (r.first < t.first) r = t;
    }
    return r;
}

Weight diameter(const Graph &g) {
    Result r = visit(-1, 0, g);
    Result t = visit(-1, r.second, g);
    return t.first; // (r.second, t.second) is farthest pair
}

```

9.12 Ordenamiento Topológico

```

#include <queue>
#define MAX_N 5001

```

```

int n;

struct Node{
    vector<int> adj;
};

Node graf[MAX_N];
int indegree[MAX_N];
int toposort[MAX_N];

inline int topSort(){
    queue<int> S;
    for (int i=0;i<n;i++) if (indegree[i] == 0) S.push(i);
    int idx = 0;
    while (!S.empty()){
        int idd = S.front();
        S.pop();
        toposort[idx++] = idd;
        for (int i=0;i<graf[idd].adj.size();i++){
            if (--indegree[graf[idd].adj[i]] == 0) S.push(graf[idd].adj[i]);
        }
    }
    if (idx < n)
        return -1; // Ciclo detectado!
    else
        return 0;
}

```

9.13 LCA usando Range Tree

```

/*
Complejidad:
Construccion : O(N)
Consulta: O (logN)
*/

#define FOR(i,c) for(__typeof((c).begin())i=(c).begin();i!=(c).end();++i)

struct LCA {
    vector<int> height, euler, first, segtree;
    vector<bool> visited;
    int n;

    LCA(vector<vector<int>> &adj, int root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
    }
}

```

```

dfs(adj, root);
int m = euler.size();
segtree.resize(m * 4);
build(1, 0, m - 1);
}

void dfs(vector<vector<int>> &adj, int node, int h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    FOR(i,adj[node]) {
        if (!visited[(*i)]) {
            dfs(adj, (*i), h + 1);
            euler.push_back(node);
        }
    }
}

void build(int node, int b, int e) {
    if (b == e) {
        segtree[node] = euler[b];
    } else {
        int mid = (b + e) / 2;
        build(node << 1, b, mid);
        build(node << 1 | 1, mid + 1, e);
        int l = segtree[node << 1], r = segtree[node << 1 | 1];
        segtree[node] = (height[l] < height[r]) ? l : r;
    }
}

int query(int node, int b, int e, int L, int R) {
    if (b > R || e < L)
        return -1;
    if (b >= L && e <= R)
        return segtree[node];
    int mid = (b + e) >> 1;

    int left = query(node << 1, b, mid, L, R);
    int right = query(node << 1 | 1, mid + 1, e, L, R);
    if (left == -1) return right;
    if (right == -1) return left;
    return height[left] < height[right] ? left : right;
}

int lca(int u, int v) {
    int left = first[u], right = first[v];
    if (left > right)
        swap(left, right);
    return query(1, 0, euler.size() - 1, left, right);
}
};

```

10 Combinatoria

10.1 Variación sin repetición

$$V_{n,p-1} = n * (n-1) * (n-2) * \dots * (n-p+1)$$

10.2 Variación con repetición

$$W_{n,p} = n^k$$

10.3 Permutación sin repetición

$$P_n = n!$$

10.4 Permutación con repetición

$$P_n(n_1, n_2, \dots, n_k) = \frac{n!}{n_1! * n_2! * \dots * n_k!}$$

10.5 Combinación

$$C_{n,p} = \frac{n!}{p! * (n-p)!}$$

10.5.1 Algoritmos

```
long long comb(int n, int m) {
    if (m > n - m) m = n - m;

    long long C = 1;
    // C^{n}_{i} -> C^{n}_{i+1}
    for (int i = 0; i < m; ++i) C = C * (n - i) / (1 + i);
    return C;
}

// Cuando n y m son grandes y se pide comb(n, m) % MOD,
// donde MOD es un numero primo, se puede usar el Teorema de Lucas.

#define MOD 3571
int C[MOD][MOD];

void FillLucasTable() {
    memset(C, 0, sizeof(C));
```

```
    for (int i = 0; i < MOD; ++i) C[i][0] = 1;
    for (int i = 1; i < MOD; ++i) C[i][i] = 1;
    for (int i = 2; i < MOD; ++i)
        for (int j = 1; j < i; ++j) C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
}

int comb(int n, int k) {
    long long ans = 1;

    while (n != 0) {
        int ni = n % MOD, ki = k % MOD;
        n /= MOD;
        k /= MOD;
        ans = (ans * C[ni][ki]) % MOD;
    }

    return (int)ans;
}

/*otra vartiante*/

long gcd(long a, long b) {
    return (a%b==0) ? b : gcd(b, a%b);
}

void Divbygcd(long &a, long &b) {
    long g=gcd(a,b);
    a/=g;b/=g;
}

long C(int n,int k){
    if(n<k) return 0;// Puede ser 1 depende del problema.
    else if(n==k) return 1;
    else{
        long numerator=1,denominator=1,toMul,toDiv,i;
        if (k>n/2) k=n-k;
        for(int i=k;i>0;i--){
            toMul=n-k+i;
            toDiv=i;
            Divbygcd(toMul,toDiv); /* siempre dividir antes de multiplicar */
            Divbygcd(numerator,toDiv);
            Divbygcd(toMul,denominator);
            numerator*=toMul;
            denominator*=toDiv;
        }
        return numerator/denominator;
    }
}
```

10.6 Máscara de bits

```
#include<vector>

vector<char> s('a','b','c');
int n=s.size();
int combinations=pow(2,n);
for (int i=0; i< combinations; i++){
    int number=i;
    vector<char> k;
    for (int e=0; e< n; e++){
        if( number & (1<< e) ){
            k.push_back(s.at[e]);
        }
    }
    /*Ya k tiene los elementos que conforma el subconjunto i-nesimo*/
}
```

10.7 Números de Catalan

Los números de Catalan satisfacen la siguiente relación de recurrencia:

$$C_0 = 1 \quad y \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

Existen múltiples problemas de concurso de programación cuya solución la dan los números de Catalan o son parte ella, algunos ejemplos.

- C_n es el número de palabras de Dyck de longitud $2n$. Una palabra de Dyck es una cadena de caracteres que consiste en n X's y n Y's de forma que no haya ningún segmento inicial que tenga más Y's que X's.
- Reinterpretando el símbolo X como un paréntesis abierto y la Y como un paréntesis cerrado, C_n cuenta el número de expresiones que contienen n pares de paréntesis correctamente colocados.
- C_n es el número de formas distintas de agrupar $n + 1$ factores mediante paréntesis (o el número de formas de asociar n aplicaciones de un operador binario).

- Las aplicaciones sucesivas de un operador binario pueden representarse con un árbol binario. En este caso, C_n es el número de árboles binarios de $n + 1$ hojas, en los que cada nodo tiene cero o dos hijos.
- C_n es el número de caminos monótonos que se pueden trazar a través de las líneas de una malla de $n*n$ celdas cuadradas, de forma que nunca se cruce la diagonal. Un camino monótono es aquél que empieza en la esquina inferior izquierda y termina en la esquina superior derecha, y consiste únicamente en tramos que apuntan hacia arriba o hacia la derecha. El recuento de estos caminos es equivalente a contar palabras de Dyck: X significa "moverse a la derecha" e Y significa "moverse hacia arriba".
- C_n es el número de formas distintas de cortar un polígono convexo de $n+2$ lados en triángulos conectando vértices con líneas rectas sin que ninguna se corte.

10.8 Triángulo de Pascal

```
#define MAX_N 5001

using namespace std;
typedef long long lld;

lld TP[MAX_N][MAX_N];

inline void trianglePascal(){
    TP[0][0] = 1;
    for(int i=1; i<MAX_N; i++){
        for(int j=0; j<=i; j++){
            if(j==0 || j==i) TP[i][j]=1;
            else TP[i][j] = TP[i-1][j-1]+TP[i-1][j];
        }
    }
}
```


11 Fórmulas

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$$

En general:

$$\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$$

Series Geométricas:

$$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad |c| < 1,$$

$$\sum_{i=0}^n i c^i = \frac{n c^{n+2} - (n+1) c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

$$\sum_{i=0}^n i^k c^i = \frac{c^k d^k}{d c^k} \left(\frac{c^{n+1} - 1}{c - 1} \right), \quad c \neq 1, \quad \sum_{i=0}^{\infty} i^k c^i = \frac{c^k d^k}{d c^k} \left(\frac{1}{1 - c} \right), \quad |c| < 1.$$

Series Harmonicas:

$$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n i H_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^n H_i = (n+1) H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right).$$

Distancia entre dos puntos de la cubierta de una esfera de los cuales se conocen su longitud y latitud y el radio de la esfera

$$\begin{aligned} & \text{acos}(\sin(p_{lat}) * \sin(q_{lat}) + \cos(p_{lat}) * \cos(q_{lat}) * \cos(p_{long}) * \cos(q_{long})) + \\ & \cos(p_{lat}) * \cos(q_{lat}) * \sin(p_{long}) * \sin(q_{long})) * r \end{aligned}$$