Setup

Cuando trabajas en la terminal, usa este comando para compilar y ejecutar tu programa con entradas desde un archivo (ej: input.txt)

```
g++ main.cpp -o main && ./main < input.txt
```

Librería estándar

Es un header no estándar que incluye todas las bibliotecas estándar de C++ de una sola vez.

```
#include <bits/stdc++.h>
```

Lectura línea a línea

```
std::string line;
while(std::getline(std::cin, line)){ ... }
```

Lectura número a número

```
int n;
while(std::cin >> n){ ... }
```

Lectura carácter a carácter

No ignora espacios ni saltos de línea

```
char c;
while(std::cin::get(c)){ ... }
```

Optimización de E/S

En problemas con entradas/salidas grandes (ej.: programación competitiva), estas líneas aceleran `cin/cout`:

```
int main() {
    // Desincroniza C++ de C (¡cin/cout más rápidos!)
    std::ios_base::sync_with_stdio(false);
    // Evita que cin vacíe cout antes de cada lectura
    std::cin.tie(nullptr);
    // tu código...
}
```

Operaciones con cadenas

split

```
std::vector<std::string> split(const std::string & str, char delimiter){
    std::vector<std::string> tokens;
    std::stringstream ss(str);
    std::string token;
    while(std::getline(ss, token, delimiter)){
        if(!token.empty()) {tokens.push_back(token); }
    }
    return tokens;
}
```

concatenación

```
std::stringstream msg;
msg << "hola";
std::cout << msg.str() << "\n";
```

Búsqueda subcadena

```
std::string str = "Hello World!";
if(str.find("World") != std::string::npos) { /* Exists*/}
```

Parseo entero

```
int num =  std::stoi(str);
long num = std::stol(str);
long long num = std::stoll(str);
```

Parseo flotante

```
float num = std::stof(str);
double num = std::stod(str);
long double num = std::stold(str)
```

Parseo múltiples números

```
std::istringstream iss("123 456");
int num1, num2;
iss >> num1 >> num2; // num1 = 123, num2=456
```

Estructuras de Datos

Listas

Pilas (Stacks)

Colas (Queues)

Sets

Diccionarios

Segment Tree

Es una estructura empleada para optimizar operaciones sobre rangos (segmentos) de un array.

Algoritmos de Búsqueda y ordenamiento

Búsqueda binaria

Busca en una lista ordenada dividiéndola a la mitad en cada paso. La complejidad en tiempo es de O(log n).

Comprobar si existe elemento

```
bool exists = binary_search(vec.begin(), vec.end(), value);
```

Posición del elemento

```
auto it = lower_bound(vec.begin(), vec.end(), value);
if (it != vec.end() && *it == value) {
  int index = distance(vec.begin(), it); // posición del elemento
  cout << "Posición exacta: " << index << '\n';

}
```

Elemento más pequeño que verifica ser más grande que value

```
it = upper_bound(vec.begin(), vec.end(), value);
if (it != vec.end()) {
```

```
  int index = distance(vec.begin(), it);
  cout << "Más pequeño que es mayor a value: " << index << '\n';
}
```

Elemento más pequeño que verifica ser mayor o igual a “value”

```
it = lower_bound(vec.begin(), vec.end(), value);
  if (it != vec.end()) {
    int index = distance(vec.begin(), it);
     cout << "Más pequeño que es mayor o igual a value: " << index << '\n';
    }
```

Elemento más grande menor que “value”

```
it = lower_bound(vec.begin(), vec.end(), value);
  if (it != vec.begin()) {
    --it; // retrocede al elemento anterior
    int index = distance(vec.begin(), it);
    cout << "Más grande menor que value: " << index << '\n';
}
```

Elemento más grande menor o igual que “value”

```
it = upper_bound(vec.begin(), vec.end(), value);
if (it != vec.begin()) {
  --it; // retrocede al elemento anterior
  if (*it <= value) {
    int index = distance(vec.begin(), it);
    cout << "Más grande menor o igual que value: " << index << '\n';
  }
}
```

Grafos

Un grafo G=(V,E) es sencillamente, un conjunto de vértices V y aristas (E, que almacena la información de conectividad entre los vértices en V).

Lectura de Grafos

Existen diferentes estructuras de datos para almacenar grafos, no obstante, la más empleada es la lista de Adyacencia, que abreviaremos como AL. En caso de ver la nomenclatura AM, nos estamos refiriendo a la matriz de adyacencia.

Matemáticas

Máximo común divisor (GCD)

Python trae una función para calcular el máximo común divisor de forma eficiente.

```
import math

math.gcd(a, b, c)
```

Mínimo común múltiplo (LCM)

Python trae una función para calcular el mínimo común múltiplo de forma eficiente.

```
import math

math.lcm(a, b, c)
```

Criba de Erastótenes

Algoritmo para generar números primos

```
// Proporcionado por noahdris (UCppM)
int N = 30;
vector<bool> es_primo(N+1,true);
vector<int> primos;
for(int i = 2; i <= N; i++){
    if(es_primo[i]){
        primos.push_back(i);
        for(int j = i; j*i <= N; j++) es_primo[j*i] = false;
    }
}
```

Exponenciación rápida

Calcula $a^b \bmod q$ de manera eficiente $O(\log b)$. En Python ya está implementado.

```
pow(a, b, q)
```

Coeficientes binomiales

El coeficiente binomial $\binom{n}{k}$ representa el número de formas de elegir un subconjunto de k elementos de un conjunto de n elementos. Por ejemplo $\binom{4}{2} = 6$ porque el conjunto $\{1, 2, 3, 4\}$ tiene 6 subconjuntos de 2 elementos $\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}$. Para calcular estos coeficientes se utiliza la fórmula $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. En Python ya está implementado.

```
import math

math.comb(n, k)
```

Strings

Trie

```
// Trie construido en base al código de Los BoquerO(n³) UNED
class TrieNode {
public:
    std::unordered_map<char, TrieNode*> children;
    bool isEndOfWord;
    int count;

    TrieNode() : isEndOfWord(false), count(0) {}

    ~TrieNode() {
        for(auto& pair : children) {
            delete pair.second;
        }
    }
};

class Trie {
private:
    TrieNode* root;

public:
    Trie() {
        root = new TrieNode();
```

```
    }

    ~Trie() {
        delete root;
    }

    void insert(std::string word) {
        TrieNode* node = root;
        for (char c : word) {
            if (node->children.find(c) == node->children.end()) {
                node->children[c] = new TrieNode();
            }
            node = node->children[c];
            node->count++;
        }
        node->isEndOfWord = true;
    }

    bool search(std::string word) {
        TrieNode* node = root;
        for (char c : word) {
            if (node->children.find(c) == node->children.end()) {
                return false;
            }
            node = node->children[c];
        }
        return node != nullptr && node->isEndOfWord;
    }

    bool startsWith(std::string prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            if (node->children.find(c) == node->children.end()) {
                return false;
            }
            node = node->children[c];
        }
        return true;
    }

    int countWordsStartingWith(std::string prefix) {
        TrieNode* node = root;
        for (char c : prefix) {
            if (node->children.find(c) == node->children.end()) {
                return 0; // Prefix not found
            }
            node = node->children[c];
        }
        return node->count;
    }
};
```

Geometría

Clase Punto

```
struct pt{
  uds x,y;
  pt(){}
  pt(uds x, uds y): x(x), y(y){}

  pt operator+(const pt &p) const {return {x + p.x, y + p.y};}
  pt operator*(uds k) const {return {k*x, k*y};}
  pt operator-() const { return *this *-1; }
  pt operator-(const pt &p) const {return *this + (-p);}
  pt operator/(uds d) const {
    if (eq(d, 0)) throw runtime_error("operator/ Recuerdos de vietADAMan (p/0)");
    return *this*(1/d);
  }

  pt& operator+=(const pt& p) { return *this = *this+p; };
  pt& operator-=(const pt& p) { return *this = *this-p; };
  pt& operator*=(uds k) { return *this = *this*k; };
  pt& operator/=(uds k) { return *this = *this/k; };

  bool operator==(const pt &a) const { return eq(a.x, x) && eq(a.y,y); }
  bool operator!=(const pt &a) const { return !(a == *this); }
  bool operator>(const pt &a) const { return (eq(a.x, x) ? lt(a.y, y): lt(a.x, x));}
  bool operator<(const pt &a) const { return a > *this; }
  pt rot() const {return pt(-y, x);} // rotate 90 degrees counter-clockwise
};
ostream& operator<<(ostream &out, const pt &p) { return out<<p.x<<" "<<p.y;}
pt operator*(uds k, const pt &p) { return p*k; }
uds dot(const pt &p, const pt &q) { return p.x*q.x + p.y*q.y; }
uds norma(const pt &p) { return sqrt(dot(p,p)); }
uds angle(const pt &p, const pt &q) {return acos(dot(p,q)/(norma(p)*norma(q)));}
uds cross(const pt &p, const pt &q) {return p.x*q.y - p.y*q.x;}

bool collinear(pt a, pt b) { return eq(cross(a, b), 0); } //si son li

uds orient(pt a, pt b, pt c) {return cross(b-a, c-a);} //
int orientS(pt a, pt b, pt c) {return sgn(orient(a, b, c)); }

bool cw(pt a, pt b, pt c, bool in_colli) {
  int o = orientS(a, b, c); return o<0 || (in_colli && o==0); // o < 0
}
bool ccw(pt a, pt b, pt c, bool in_colli) {
  int o = orientS(a, b, c); return o>0 || (in_colli && o==0); // o > 0
}
bool collinear(pt a, pt b, pt c) { return eq(orient(a, b, c),0); } // ver si tres
puntos estan alineados
```