

# Dossier grupo C++

Editar - Preferencias - Herramientas - Terminal:

```
gnome-terminal -e "/bin/bash%c"
```

```
#include <bits/stdc++.h>
using namespace std;

// a % b (positive)
int mod(int a, int b) {
    return ((a % b) + b) % b;
}

// greatest common divisor
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

// least common multiple (requires gcd)
int lcm(int a, int b) {
    return a / gcd(a, b) * b ;
}

// Bezout :  $d = ax + by$  (requires gcd)
// Todas las soluciones son de la forma
//  $x = x_0 + M*B/d$ ,  $y = y_0 - M*A/d$ 
int bezout(int a, int b, int *x, int *y) {
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = bezout(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}

//  $ax = b \pmod n$ 
vector<int> mod_equation(int a, int b, int n) {
    int x, y;
    vector<int> solutions;
    int d = bezout(a, n, x, y);
    if (b % d == 0) {
        x = mod(x * (b / d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n / d), n));
    }
    return solutions;
}
```

```

// modular inverse (-1 on failure)
int mod_inverse(int a, int n) {
    int x, y;
    int d = bezout(a, n, x, y);
    if (d > 1) return -1;
    return mod(x, n);
}

// ax + by = c (x = y = -1 on failure)
void linear_equation(int a, int b, int c, int & x, int & y) {
    int d = gcd(a, b);
    if (c % d) x = y = -1;
    else {
        x = c / d * mod_inverse(a / d, b / d);
        y = (c - a*x) / b;
    }
}

// Eratóstenes
vector<int> primes(int n) {
    vector<bool> v(n, true);
    for (int i = 0; i < n; i++) {
        if (v[i])
            for (int cont = 2; i*cont ≤ n; ++cont)
                v[i*cont] = false;
    }
    vector<int> prim;
    for (int i = 2; i < n; i++)
        if (v[i]) prim.push_back(i);
    return prim;
}

// Divisores
vector<int> divisors(int n, vector<int> const& prim) {
    vector<int> divis;
    int i = 0;
    while (prim[i] ≤ n + 1) {
        if (n % prim[i] == 0) {
            n = n / prim[i];
            divis.push_back(prim[i]);
        }
        else ++i;
    }
    return divis;
}

```

```

//Busqueda en profundidad
void dfs(int u, mat &adjList, vector<int> &dfs_num, vector<int> &topo) {
    dfs_num[u] = 1;
    for (int v : adjList[u]) {
        if (dfs_num[v] == 0)
            dfs(v, adjList, dfs_num, topo);
    }
    topo.push_back(u); // Leer al revés para orden top.
}

//Busqueda en anchura
void bfs(mat &adjList, int u, vector<int> &dist) {
    dist[u] = 0;
    queue<int> q;
    q.push(u);
    while (!q.empty()) {
        u = q.front(); q.pop();
        for (auto v : adjList[u]) {
            if (dist[v] == INT_MAX) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}

//Dijkstra (vii : vector<pair>)
void dijkstra(int s, vector<vii> const& grafo, vector<int> &dist) {
    dist.assign(adjList.size(), numeric_limits<int>::max());
    dist[s] = 0;
    priority_queue<ii, vii, greater<ii>> pq; // de minimos
    pq.push({0,s});
    while (!pq.empty()) {
        ii front = pq.top(); pq.pop();
        int d = front.first, u = front.second;
        if (d > dist[u]) continue;
        for (auto v : grafo[u]) {
            if (dist[u] + v.first < dist[v.second]) {
                dist[v.second] = dist[u] + v.first;
                pq.push({dist[v.second], v.second});
            }
        }
    }
}

//Kruskal
typedef pair<int, int> iPair;
struct Graph{
    int V, E;
    vector< pair<int, iPair> > edges;
    Graph(int V, int E){
        this->V = V;
        this->E = E;
    }
    void addEdge(int u, int v, int w){
        edges.push_back({w, {u, v}});
    }
}

```

```

    }
    int kruskalMST();
};

struct DisjointSets{
    int *parent, *rnk;
    int n;
    DisjointSets(int n){
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];
        for (int i = 0; i ≤ n; i++){
            rnk[i] = 0;
            parent[i] = i;
        }
    }

    int find(int u){
        if (u ≠ parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    void merge(int x, int y){
        x = find(x), y = find(y);
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};

int Graph::kruskalMST(){
    int mst_wt = 0;
    sort(edges.begin(), edges.end());

    DisjointSets ds(V);

    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it≠edges.end(); it++){
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
        int set_v = ds.find(v);
        if (set_u ≠ set_v){
            mst_wt += it->first;
            ds.merge(set_u, set_v);
        }
    }
    return mst_wt;
}

```

```

// Floyd Warshall (para grafos dirigidos)
void floydWarshall (int graph[][V]){
    int dist[V][V], i, j, k;
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];
    for (k = 0; k < V; k++){
        for (i = 0; i < V; i++){
            for (j = 0; j < V; j++){
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}

//Monedas finitas.
bool minM(vector<int> const& mon, vector<int> const& cant,
    int n,int P,int & sol){
    vector<int> minMon(P+1,INT_MAX);
    minMon[0] = 0;
    for(int i = 1;i ≤ n;++i){
        for(int j = P;j ≥ mon[i];--j){
            for(int k = 0; k ≤ cant[i] && k*mon[i] ≤ j;++k){
                if(minMon[j-k*mon[i]] ≠ INT_MAX)
                    minMon[j] = min(minMon[j],k + minMon[j-k*mon[i]]);
            }
        }
    }
    sol = minMon[P];
    return sol ≠ INT_MAX;
}

// Longest common subsequence
string lcs(string const& x, string const& y){
    mat m(1+x.size(),vector<int>(1+y.size()));
    for(size_t i = 0; i < x.size()+1; ++i) m[i][0] = 0;
    for(size_t j = 1; j < y.size()+1; ++j) m[0][j] = 0;
    for(size_t i = 0; i < x.size(); ++i) {
        for(size_t j = 0; j < y.size(); ++j) {
            if (x[i]==y[j]) m[i+1][j+1] = 1+m[i][j];
            else m[i+1][j+1] = max(m[i][j+1],m[i+1][j]);
        }
    }
    string s;
    int i = x.size()-1, j = y.size()-1;
    while(i ≥ 0 && j ≥ 0) {
        if(m[i+1][j+1] == m[i][j+1]) i--;
        else if(m[i+1][j+1] == m[i+1][j]) j--;
        else {
            s.push_back(x[i]); i--; j--;
        }
    }
    reverse(s.begin(),s.end());
    return s;
}

```

```

//Quitar minimo numero de letras para formar palindromo
string minP(string const& a){
    int n = a.size();
    vector<vector<int> > minPal(n,vector<int> (n,0));
    for(int i = n-2; i ≥ 0;--i){
        for(int j = i+1;j < n;++j){
            if(a[i] == a[j])
                minPal[i][j] = minPal[i+1][j-1];
            else
                minPal[i][j] = 1 + min(minPal[i][j-1],minPal[i+1][j]);
        }
    }
    stack<char> s;
    int i = 0, j = n-1;
    string st;
    while(i< n && j ≥ 0 && i≤j){
        if(i == j){ // cuando queda una sola letra solo tiene que aparecer
                    //una vez en la solucion, por lo que no se añade a la pila
            st.push_back(a[i]);
            ++i;
        }else if(a[i] == a[j]){
            s.push(a[i]);
            st.push_back(a[i]);
            ++i;--j;
        }else if(minPal[i][j] == 1 + minPal[i+1][j])
            ++i;
        else
            --j;
    }
    while(!s.empty()){
        st.push_back(s.top()); s.pop();
    }
    return st;
}

//Vuelta atras con estimacion y marcado
void res(int k, mat & mt, vector<int> & marc,
        long long int & mejor, long long int act){
    for(int i = 0 ; i < mt.size(); ++i){
        if(marc[i] < 3){
            marc[i]++;
            act += mt[i][k];
            if(k == mt[0].size() - 1){
                if(mejor > act)
                    mejor = act;
            } else if (act < mejor) {
                int est = act + estim(k,mt,marc);
                if (est < mejor)
                    res(k+1,mt,marc,mejor,act);
            }
            marc[i]--;
            act -= mt[i][k];
        }
    }
}

```

**stack:** empty, size, top, push, emplace, pop, swap  
**queue:** empty, size, front, back, push, emplace, pop, swap  
**priority\_queue:** empty, size, top, push, emplace, pop, swap

memset(char[] str, '-', 6);  
cstdlib : atoi (char[] to int), qsort, bsearch  
cstdio : printf, scanf, getc, putc, eof  
ctype : tolower, toupper, islower, isupper, ispunct  
iomanip : setfill, setw, setprecision  
cmath : exp, log, pow, sqrt, cbrt, ceil, floor, round, abs  
algorithm : sort, merge, copy, move, min, max, reverse, rotate  
utility : pair, swap, less  
climits : INT\_MIN, INT\_MAX, LONG\_, LLONG\_

## Non-modifying sequence ops:

### **all\_of**

Test condition on all elements

### **any\_of**

Test if any element fulfills condition

### **none\_of**

Test if no elements fulfill condition

### **for\_each**

Apply function to range

### **find**

Find value

### **find\_if**

Find element

### **find\_if\_not**

Find element (negative condition)

### **find\_end**

Find last subsequence

### **find\_first\_of**

Find element from set

### **adjacent\_find**

Find equal adjacent elements

### **count**

Count appearances of value

### **count\_if**

Return number of elements satisfying condition

### **mismatch**

Return first position where two ranges differ

### **equal**

Test whether the elements in two ranges are equal

### **is\_permutation**

Test whether range is permutation of another

### **search**

Search range for subsequence

### **search\_n**

Search range for elements

## Modifying sequence ops:

### **copy**

Copy range of elements

### **copy\_if**

Copy certain elements of range

### **copy\_backward**

Copy range of elements backward

### **move**

Move range of elements

### **move\_backward**

Move range of elements backward

### **swap**

Exchange values of two objects

### **swap\_ranges**

Exchange values of two ranges

### **iter\_swap**

Exchange values of objects pointed to by two iterators

### **transform**

Transform range

### **replace**

Replace value in range

### **replace\_if**

Replace values in range

### **fill**

Fill range with value

### **remove**

Remove value from range

### **remove\_if**

Remove elements from range

### **unique**

Remove consecutive duplicates

### **reverse**

Reverse range

### **rotate**

Rotate left the elements in range



## Partitions:

### **is\_partitioned**

Test whether range is partitioned

### **partition**

Partition range in two

### **stable\_partition**

Partition range in two (stable)

### **partition\_point**

Get partition point

## Sorting:

### **sort**

Sort elements

### **stable\_sort**

Sort elements preserving order of equivalents

### **partial\_sort**

Partially sort elements in range

### **is\_sorted**

Check whether range is sorted

### **is\_sorted\_until**

Find first unsorted element

### **nth\_element**

Sort element in range

## Binary search:

### **lower\_bound**

Return iterator to lower bound

### **upper\_bound**

Return iterator to upper bound

### **equal\_range**

Get subrange of equal elements

### **binary\_search**

Test if value exists in sorted sequence

## Merge:

### **merge**

Merge sorted ranges

### **inplace\_merge**

Merge consecutive sorted ranges

### **includes**

Test whether sorted range includes another sorted range

### **set\_union**

Union of two sorted ranges

### **set\_intersection**

Intersection of two sorted ranges

### **set\_difference**

Difference of two sorted ranges

### **set\_symmetric\_difference**

Symmetric difference of two sorted ranges

## Min/max:

### **min**

Return the smallest

### **max**

Return the largest

### **minmax**

Return smallest and largest elements

### **min\_element**

Return smallest element in range

### **max\_element**

Return largest element in range

### **minmax\_element**

Return smallest and largest elements in range

## Other:

### **next\_permutation**

Transform range to next permutation

### **prev\_permutation**

Transform range to previous permutation