Dossier grupo C+++

Headers		<array>	<vector>	<deque>	<forward_list>	<list>
Members		array	vector	deque	forward_list	list
	constructor	*implicit*	vector	deque	forward_list	list
	destructor	*implicit*	~vector	~deque	~forward_list	~list
	operator=	*implicit*	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin cbefore_begin	cbegin
	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend	crend		crend
capacity	size	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			
element access	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
modifiers	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap
list operations	splice				splice_after	splice
	remove				remove	remove
	remove_if				remove_if	remove_if
	unique				unique	unique
	merge				merge	merge
	sort				sort	sort
	reverse				reverse	reverse
observers	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator
	data	data	data			

Headers		<set>		<map>		<unordered_set>		<unordered_map>	
Members		set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap
	constructor	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap
	destructor	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap
	assignment	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin
	end	end	end	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin	rbegin				
	rend	rend	rend	rend	rend				
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin
	cend	cend	cend	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin	crbegin				
	crend	crend	crend	crend	crend				
capacity	size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty	empty	empty	empty
	reserve					reserve	reserve	reserve	reserve
element access	at			at				at	
	operator[]			operator[]				operator[]	
modifiers	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace
	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint
	insert	insert	insert	insert	insert	insert	insert	insert	insert
	erase	erase	erase	erase	erase	erase	erase	erase	erase
	clear	clear	clear	clear	clear	clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap	swap	swap	swap
operations	count	count	count	count	count	count	count	count	count
	find	find	find	find	find	find	find	find	find
	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range
	lower_bound	lower_bound	lower_bound	lower_bound	lower_bound				
	upper_bound	upper_bound	upper_bound	upper_bound	upper_bound				
	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator

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

cctype : 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 in range

any_of
Test if any element in range fulfills condition

none_of
Test if no elements fulfill condition

for_each
Apply function to range

find
Find value in range

find_if
Find element in range

find_if_not
Find element in range (negative condition)

find_end
Find last subsequence in range

find_first_of
Find element from set in range

adjacent_find
Find equal adjacent elements in range

count
Count appearances of value in range

count_if
Return number of elements in range 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 operations:

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 in range

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 ordering

partition_point
Get partition point

Sorting:

sort
Sort elements in range

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 in range

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

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

//magic
copy(v.begin(), v.end() - 1,
 ostream_iterator<tipoDato>(cout, " "));

// 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
int lcm(int a, int b) {
  return a / gcd(a, b) * b ;
}

// Bezout : d = ax + by
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 (mod 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);
}
```

```cpp
// 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;
  }
}
```

```cpp
//Busqueda en profundidad:
void dfs(int u, vvi &adjList, vi &dfs_num, vi &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); // Read topo in reverse order.
}

//Busqueda en anchura
void bfs(vvi &adjList, int u, vi &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
void dijkstra(int s, vector<vii> const& grafo, vi &dist) {
    dist.assign(adjList.size(), numeric_limits<int>::max());
    dist[s] = 0;
    priority_queue<ii, vii, greater<ii>> pq;
    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);
    }
  }
}

// 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];
            }
        }
    }
}
```