Essa biblioteca é uma cilada, Bino!

TEAM **C DE CILADA**

#1: Francisco Neto
#2: Leonardo Rezende
#3: Herinson Rodrigues
Coach: Tanilson Dias

"In algorithms, as in life, persistence usually pays off."
- Steven Skiena

Templates

.vimrc

```
filetype indent on
syntax enable
autocmd bufnewfile *.java :0r ~/.vim/java
autocmd bufnewfile *.java :w!
autocmd bufnewfile *.cpp :0r ~/.vim/cpp
autocmd bufnewfile *.cpp :w!
map <F3> :!g++ -g -std=c++11 -lm -O2 % && ./a.out < in > myout <CR>
map <F2> :w <CR>
map <F12> :!gdb ./a.out <CR>
set number
set ts=4
set history=500
set backspace=eol,start,indent
set magic
set expandtab
set shiftwidth=4
set tabstop=4
set ai
set si
set wrap
set autoread
```

.vim/java

```
import java.io.*;
import java.math.*;
import java.util.*;

public class Template {
    static BufferedReader input;
    static StringTokenizer _stk;

    static String readln() throws IOException {
        String l = input.readLine();
        if (l != null)
            _stk = new StringTokenizer(l, " ");
        return l;
    }

    static String next() {
        return _stk.nextToken();
    }

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

    static PrintWriter output = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(System.out)));

    public static void main(String[] args) throws IOException {
        Locale.setDefault(Locale.US);
        input = new BufferedReader(new InputStreamReader(System.in));

    }
}
```

.vim/cpp

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

#define forr(i, a, b) for(int i = a; i < b; ++i)
#define rfor(i, a, b) for(int i = a; (i) >= b; i--)
#define forn(i, n) for(int i = 0 ; i < n ; ++i)

#define range(v, a, b) v.begin() + a, v.begin() + b
#define all(v) v.begin(), v.end()
#define rall(v) v.rbegin(), v.rend()

#define pb push_back
#define eb emplace_back
#define unmap unordered_map
#define unset unordered_set
#define pqueue priority_queue

#define debug(x) cout << x << endl
#define fastio ios::sync_with_stdio(false)

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef long double llf;

typedef vector<int> vi;
typedef vector<char> vc;
typedef vector<double> vd;
typedef vector<string> vs;
typedef pair<int, int> pii;

#define inf(T) numeric_limits<T>::max()
#define countd(x) ceil(log10(fabs(x)+1))

const double PI = 2*acos(0.0);

const double EPS = 1e-10;
int cmp(double x, double y = 0, double tol = EPS) {
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

// memset(memo, -1, sizeof(memo));

int main() {

    return 0;
}
```

Antes da prova

- Revisar os algoritmos disponíveis na biblioteca.
- Revisar a referência STL
- Reler o roteiro

Antes de implementar um problema

- K.I.S.S!!!
- Quem for implementar deve relê-lo antes.
- Peça todas as clarifications que forem necessárias.
- Marque as restrições e faça contas com os limites da entrada.
- Teste o algoritmo no papel e convença outra pessoa de que ele funciona.
- Planeje a resolução para os problemas grandes: a equipe se junta para definir as estruturas de dados, mas cada pessoa escreve uma função.

Debugar uma solução

- Ao encontrar um bug, escreva um caso de teste que o dispare
- Reimplementar trechos de programas entendidos errados.
- Em caso de RE, procure todos os [, / e %.

Roteiro da prova

300 minutos: Início da prova

- Leonardo e Francisco começam lendo os problemas.
- Herinson começa digitando ".vimrc", "compile"e "draft.cpp".
- Herinson gera todos os sources, copiando "draft.cpp"com -i.
- Herinson apaga "draft.cpp"com -i.
- Quando surgir um problema fácil, todos discutem se ele deve ser o primeiro a ser resolvido.
- Quando o primeiro problema for escolhido, Leonardo o implementa, possivelmente tirando Herinson do computador e interrompendo as digitações.
- Se surgir um problema ainda mais fácil que o primeiro, Leonardo passa a implementar esse novo problema.
- Enquanto Leonardo resolve o primeiro problema, Francisco e Herinson leem os demais.
- À medida que Herinson for lendo os problemas, ele os explica para Francisco.
- Francisco preenche a tabela com os problemas até então lidos:
  - AC: Accepted
  - Ordem: ordem de resolução dos problemas (pode ser infinito).
  - Escrito: se já há código escrito neste problema, mesmo que no papel.
  - Leitores: pessoas que já leram o problema.
  - Complexidade: complexidade da solução implementada.
  - Resumo: resumo sobre o problema
- Assim que o primeiro problema começar a ser compilado, Leonardo avisa Francisco e Herinson para escolherem o segundo problema mais fácil.
- Assim que o primeiro problema for submetido, Leonardo sai do computador.
- Herinson assume o computador, e termina as digitações pendentes.
- Herinson implementa o segundo problema mais fácil.
- Fora do computador, Francisco e Leonardo escolhem a ordem e os resolvedores dos problemas, com base no tempo de implementação.
- Se ninguém tiver alguma ideia para resolver um problema, empurre-o para o final (ou seja, a ordem desse problema será $+\infty$).
- Quando Herinson submete o segundo problema e sai do computador, ele revê a ordenação dos problemas com quem ficou fora do computador.

200 minutos: Metade da prova

- A equipe deve resolver no máximo três problemas ao mesmo tempo.
- Escreva o máximo possível de código no papel. - Depure com o código do problema e com a saída do TRACE impressos.
  - Explique seu código para outra pessoa da equipe.
  - Acompanhe o código linha por linha, anotando os valores das variáveis e redesenhando as estruturas de dados à medida que forem alteradas.
- Momentos nos quais quem estiver no computador deve avisar os outros membros da equipe:
  - Quando estiver pensando ou depurando
  - Quando estiver prestes a submeter, para que os outros membros possam fazer testes extras e verificar o formato da saída.
- Submeta sempre em C++, com extensão .cpp.
- Logo após submeter, imprima o código.
- Jogue fora as versões mais antigas do código impresso de um programa.
- Jogue fora todos os papéis de um problema quando receber Accepted.
- Mantenha todos os papéis de um problema grampeados.

100 minutos: Final da prova

- A equipe deve resolver apenas um problema no final da prova.
- Use os balões das outras equipes para escolher o último problema
  - Os problemas mais resolvidos por outras equipes provavelmente são mais fáceis que os outros problemas.
  - Uma equipe mais bem colocada só é informativa quando as demais não o forem.
  - Como Herinson digita mais rápido, ele fica o tempo todo no computador.
  - Francisco e Leonardo sentam ao lado de Herinson e dão sugestões para o problema.

60 minutos: Placar congelado

- Preste atenção nas comemorações das outras equipes: os balões continuam vindo!
- Quando terminar um problema, teste com o exemplo de entrada, submeta e só depois pense em mais casos de teste.
- Nos últimos cinco minutos, faça alterações pequenas no código, remova o TRACE e submeta.

Os 10 mandamentos

1. Não dividirás por zero.
2. Não alocarás dinamicamente.
3. Compararás números de ponto flutuante usando cmp().
4. Verificarás se o grafo pode ser desconexo.
5. Verificarás se as arestas do grafo podem ter peso negativo.
6. Verificarás se pode haver mais de uma aresta ligando dois vértices.
7. Conferirás todos os índices de uma programação dinâmica.
8. Reduzirás o branching factor da DFS.
9. Farás todos os cortes possíveis em uma DFS.
10. Tomarás cuidado com pontos coincidentes e com pontos colineares.

Limites dos tipos primitivos de dados

Tipo	Bits	min .. max	Precisão decimal
char	8	0 .. 127	2
signed char	8	-128 .. 127	2
unsigned char	8	-128 .. 127	2
short	16	-32.768 .. 32.767	4
unsigned short	16	0 .. 65.535	4
int	32	$-2 * 10^9$ .. $2 * 10^9$	9
unsigned int	32	0 .. $4 * 10^9$	9
int64_t	64	$-9 * 10^{18}$ .. $9 * 10^{18}$	18
uint64_t	64	0 .. $18 * 10^{18}$	18

Quantidade de números primos de 1 até 10^n

$\text{pi}(10^1) = 4$
$\text{pi}(10^2) = 25$
$\text{pi}(10^3) = 168$
$\text{pi}(10^4) = 1.229$
$\text{pi}(10^5) = 9.592$
$\text{pi}(10^6) = 78.498$
$\text{pi}(10^7) = 664.579$
$\text{pi}(10^8) = 5.761.455$
$\text{pi}(10^9) = 50.847.534$

OBS: É sempre verdade que $n/ln(n) < pi(n) < 1.26 * n/ln(n)$.

1 Notações

2 Containers

2.1

```
template <class T1, class T2> struct ;
struct  {
    T1 first; T2 second;
    () {}
    (const T1& a, const T2& b):
    first(a), second(b) {}
};
```

2.1.1 Tipos

```
::first_type
::second_type
```

2.1.2 Funções e Operadores

Ver também 2.2.3

```
<T1, T2>;
make_(const T1&, const T2&);
```

2.2 Containers - Geral

C é qualquer container {vector, deque, list, set, multiset, map, multimap}

2.2.1 Tipos

```
C::value_type
C::reference
C::const_reference
C::iterator
C::const_iterator
C::reverse_iterator
C::const_reverse_iterator
C::difference_type
C::size_type
```

2.2.2 Membros e Operadores

```
C::C();
C::C( const C&);
C::~C();
C& C::operator = (const C&);
C::iterator                     C::begin();
C::const_iterator               C::begin() const;
C::iterator                     C::end();
C::const_iterator               C::end() const;
C::reverse_iterator             C::rbegin();
C::const_reverse_iterator       C::rbegin() const;
C::reverse_iterator             C::rend() const;
C::const_reverse_iterator       C::rend() const;

C::size_type C::size() const;
C::size_type C::max_size() const;
bool C::empty() const;
void C::swap(C& c);
```

2.2.3 Operadores de Comparação

Seja, Ca, b. C também pode ser um (2.1).

```
a == b      a != b
a < b       a > b
a <= b      a >= b
```

Também feito lexicograficamente e retorna bool.

2.3 Containers sequenciais

S é qualquer um dos vector, deque, list

2.3.1 Construtores

```
S::S(S::size type n, const S::value type& t);
S::S(S::const iterator first, S::const iterator
↪  last);
```

Ver 7.2, 7.3

2.3.2 Membros

```
S::iterator // cópia inserida
S::insert (S::iterator before, const S::value_type&
↪  val);
S::iterator // cópia inserida
S::insert (S::iterator before, S::size_type nVal,
↪  const S::value type& val);
S::iterator // cópia inserida
S::insert (S::iterator before, S::const_iterator
↪  first, S::const_iterator last);
S:iterator S::erase (S::iterator position);
S::iterator S::erase(S::const_iterator first,
↪  S::const S::value_type& last);
void S::push_back(const S::value_type& x);
void S::pop back();
S::reference S::front();
S::const_reference S::front() const;
S::reference S::back();
S::const_reference S::back() const;
```

2.4 Vector

```
template<class T, class A=allocator>
class vector;

size_type vector::capacity() const;
void vector::reserve(size type n);
vector::reference vector::operator[](size_type i);
vector::const reference vector::operator[](size_type
↪  i) const;
```

2.5 Deque

```
template<class T, class A=allocator>
class deque;

void deque::push_front(const T& x);
void deque::pop_front();
```

Possui também todas as funcionalidades do vector (ver 2.4)

2.6 List

```
template<class T, class A=allocator>
class list;

void deque::pop_front();
void deque::push_front(const T&);
void // move todos x (&x £\nef antes de pos
list::splice(iterator pos, list(T)& x);
void // move xElemPos de x antes de pos
list::splice(iterator pos, list(T)& x, iterator
↪  xElemPos);
void // move [xFirst, xLast) de x antes de pos
list::splice(iterator pos,list(T)& x, iterator
↪  xFirst, iterator xLast);
void list::remove(const T& value);
void list::remove_if(F pred); // para todo this
↪  iterator p, *p != *(p+1)
void list::unique(); // remove repetidos
void // como o anterior, mas ~binPred(*p, *(p+1))
list::unique(B binPred);
// assumindo ambos this e x ordenados
void list::merge(list(T)& x);
```

```
// mescla e ordena por cmp
void list::merge(list(T)& x, C cmp);
void list::reverse();
void list::sort();
void list::sort(C cmp);
```

2.7 Associativos ordenados

A é qualquer container seja {set, multiset, map, multimap}.

2.7.1 Tipos

```
A::key_type      A::value_type
A::key_compare   A::value_compare
```

2.7.2 Construtores

```
A::A(C c=C());
A::A(A::const_iterator first, A::const_iterator
 ↪  last, C c=C());
```

2.7.3 Membros

```
A::key_compare      A::key_comp() const;
A::value_compare    A::value_comp() const;

A::iterator
A::insert(A::iterator hint,const A::value_type&
 ↪  val);
void A::insert(A::iterator first, A::iterator last);
A::size_type // elemento excluído
A::erase(const A::key_type& k);
void A::erase(A::iterator p);
void A::erase(A::iterator first, A::iterator last);
A::size_type A::count(const A::key_type& k) const;
A::iterator A::find(const A::key_type& k) const;
A::iterator A::lower_bound(const A::key_type& k)
 ↪  const;
A::iterator A::upper_bound(const A::key_type& k)
 ↪  const;
<A::iterator, A::iterator> // ver 4.3.1
A::equal_range(const A::key_type& k) const;
```

2.8 Set

```
template<class K, class  C=less(K), class
 ↪  A=allocator>
class set;

set::set(const C& cmp=C());
<set::iterator, bool> // bool = if new
set::insert(const set::value_type& x);
```

2.9 Multiset

```
template<class K, class  C=less(K), class
 ↪  A=allocator>
class multiset;

multiset::multiset(const C& cmp=C());
multiset::multiset(I first, I last,const C&
 ↪  cmp=C()).
multiset::iterator // cópia inserida
multiset::insert(const multiset::value_type& x);
```

2.10 Map

```
template<class K, class T, class C=less(K), class
 ↪  A=allocator>
class map;
```

Ver também 2.2 e 2.7.

2.10.1 Tipos

```
map::value_type // <const K, T>
```

2.10.2 Membros

```
map::map(const C& cmp=C());
<map::iterator, bool> // bool = if new
map::insert(const map::value_type& x);
T& map::operator[](const map::key_type&);
map::const_iterator map::lower_bound(const
 ↪  map::key_type& k) const;
map::const_iterator map::upper_bound(const
 ↪  map::key_type& k) const;
<map::const_iterator, map::const_iterator>
 ↪  map::equal_range(const map::key_type& k) const;
```

Exemplo

```
typedef map<string, int> MSI;
MSI nam2num;

nam2num.insert(MSI::value_type("one", 1));
nam2num.insert(MSI::value_type("two", 2));
nam2num.insert(MSI::value_type("three", 3));
int n3 = nam2num["one"] + nam2num["two"];
cout << n3 << " called ";

for (MSI::const_iterator i = nam2num.begin(); i !=
 ↪  nam2num.end(); ++i)
    if ((*i).second == n3) {
        cout << (*i).first << endl;
    }
```

Saída; 3 called three

2.10.3 Multimap

```
template<class K, class T, class C=less(K), class
 ↪  A=allocator>
class multimap
```

Ver também 2.2 e 2.7

2.10.4 Tipos

```
multimap::value_type // pair<const K, T>
```

2.10.5 Membros

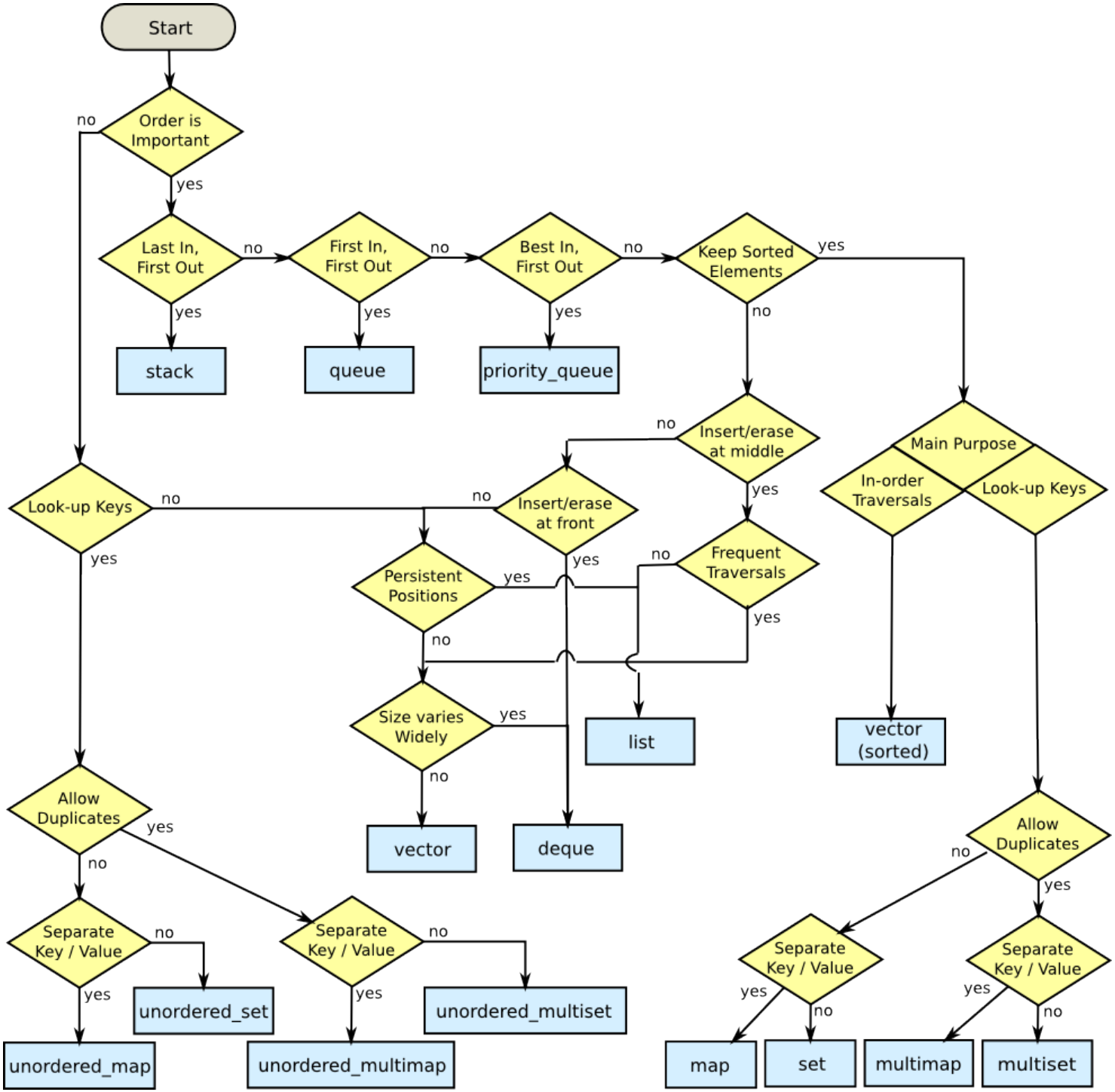
```
template<class K, class  C=less(K), class
 ↪  A=allocator>
class multimap;

multimap::multimap(const C& cmp=C());
multimap::multimap(I first, I last,const C&
 ↪  cmp=C());
multimap::const_iterator multimap::lower_bound(const
 ↪  multimap::key_type& k) const;
multimap::const_iterator multimap::upper_bound(const
 ↪  multimap::key_type& k) const;
<map::const_iterator, multimap::const_iterator>
 ↪  multimap::equal_range(const multimap::key_type&
 ↪  k) const;
```

3 Container Adaptors

3.1 Stack Adaptor

```
template< class T, class C=deque<T> >
class stack;
```


Start
Order is Important
no
yes
Last In, First Out
no
yes
stack
First In, First Out
no
yes
queue
Best In, First Out
no
yes
priority_queue
Keep Sorted Elements
yes
no
Look-up Keys
no
yes
Insert/erase at middle
no
yes
Insert/erase at front
no
yes
Persistent Positions
yes
no
Frequent Traversals
no
yes
Size varies Widely
yes
no
list
vector
deque
Main Purpose
In-order Traversals
Look-up Keys
vector (sorted)
Allow Duplicates
yes
no
Separate Key / Value
no
yes
unordered_set
unordered_map
Separate Key / Value
no
yes
unordered_multiset
unordered_multimap
Allow Duplicates
no
yes
Separate Key / Value
yes
no
map
set
Separate Key / Value
yes
no
multimap
multiset

Longest Increase Subsequence - LIS O(n log n)

```
void lis(const vector<int> &v, vector<int> &asw) {
    vector<int> pd(v.sz, 0), pd_index(v.sz), pred(v.sz);
    int maxi = 0, x, j, ind;

    fori(i, v.sz) {
        x = v[i];
        j = lower_bound(pd.begin(), pd.begin() + maxi, x) - pd.begin();
        pd[j] = x; pd_index[j] = i;

        if( j == maxi ) { maxi++; ind = i; }
        pred[i] = j ? pd_index[j-1] : -1;
    } // return maxi;

    int pos = maxi-1, k = v[ind];
    asw.resize( maxi );

    while ( pos >= 0 ) {
        asw[pos--] = k;
        ind = pred[ind];
        k = v[ind];
    }
}
```