



Maratona**CIn**



SELETIVA AULA 1

Primeiros passos
Complexidade e STL
Prefix Sum

Lua Guimarães <lgf>

rev. 1.1



Objetivos



Maratona**CIn**



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

- Entender por que C++ é a linguagem "padrão" em competições (desempenho, bibliotecas ricas, padrão na indústria de algoritmos).
- Configurar o ambiente de desenvolvimento: um compilador C++ (g++) e um editor de código (VS Code).
- Compilar e executar o primeiro programa.
- Entender e aprender a usar um template.
- Aprender a ler dados da entrada padrão e escrever na saída padrão de forma eficiente. (FastIO)
- Dominar os tipos de dados e, principalmente, seus limites.
- Aprender a usar os contêineres mais importantes da STL.

Por que C++?



Maratona

CIn



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

- Velocidade de execução (código compilado)
- Poderosa biblioteca padrão - Standard Template Library (STL).
- Aceito por todas as competições de programação importantes.
- Vários tutoriais/aulas de algoritmos disponíveis.

Se estiver usando Linux, use o GCC incluído na sua distribuição (build-essential). Se estiver no Windows, utilize o ambiente MSYS2 com o GCC incluído via mingw64. Não se esqueça de instalar também o debugger (gdb).



Primeiros passos para o ACCEPTED

Hello World e templates

Primeiro código



Maratona

Centro de
Informática
UFPE



C++ hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
3
4 int main() {
5     std::string name;
6     std::cin >> name;
7
8     std::cout << "Hello, " << name << "!" << std::endl;
9
10    return 0;
11 }
```

Primeiro código



Maratona

CIn
Centro de
Informática
UFPE



hello.cpp

- □ ×

```
1 #include <iostream>    Bibliotecas
2 #include <string>      para incluir
3
4 int main() {
5     std::string name;
6     std::cin >> name;
7
8     std::cout << "Hello, " << name << "!" << std::endl;
9
10    return 0;
11 }
```

Primeiro código



Maratona

Centro de
Informática
UFPE



C++ hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
```

Bibliotecas
para incluir

Função de entrada

```
3
4 int main() {
5     std::string name;
6     std::cin >> name;
7
8     std::cout << "Hello, " << name << "!" << std::endl;
9
10    return 0;
11 }
```

Primeiro código



Maratona

CIn
Centro de
Informática
UFPE



hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
```

Bibliotecas
para incluir

Função de entrada

```
4 int main() {
5     std::string name;
6     std::cin >> name;
```

Entrada

Saída

```
8     std::cout << "Hello, " << name << "!" << std::endl;
```

```
10    return 0;
11 }
```

Primeiro código



Maratona

CIn



Centro de
Informática
UFPE



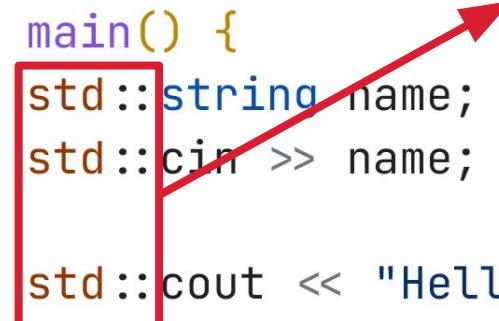
UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
3
4 int main() {
5     std::string name;
6     std::cin >> name;
7
8     std::cout << "Hello, " << name << "!" << std::endl;
9
10    return 0;
11 }
```

Identificador da
biblioteca padrão.



Primeiro código



MaratonaCIn

Centro de
Informática
UFPE



C++ hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
3
4 int main() {
5     std::string name;
6     std::cin >> name;
7
8     std::cout << "Hello, " << name << "!" << std::endl;
9
10    return 0;
11 }
```

Identificador da
biblioteca padrão.
Como remover?



Primeiro código



Maratona

Centro de
Informática
UFPE



C++ hello.cpp

- □ ×

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main() {
7     string name;
8     cin >> name;
9
10    cout << "Hello, " << name << "!" << endl;
11
12    return 0;
13 }
```

Primeiro código



Maratona

CIn

Centro de
Informática
UFPE



C++ hello.cpp

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main() {
7     string name;
8     cin >> name;
9
10    cout << "Hello, " << name << "!" << endl;
11
12    return 0;
13 }
```

Preciso incluir cada coisa que eu for usar da STD? Preciso decorar todos os includes?

Primeiro código



Maratona

Centro de
Informática
UFPE



C++ hello.cpp

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     string name;
7     cin >> name;
8
9     cout << "Hello, " << name << "!" << endl;
10
11    return 0;
12 }
```

Não! Existe um
include “padrão”.

Buffer Sync



Por padrão, os métodos de entrada e saída do C++ (`cin`, `cout`) são sincronizadas com os métodos equivalentes da linguagem C (`scanf`, `printf`). Isso garante que você possa misturar os dois estilos de I/O no mesmo código sem que a ordem das operações seja bagunçada.

Essa compatibilidade, no entanto, tem um custo alto de desempenho. A cada operação de leitura ou escrita, o programa gasta tempo verificando e sincronizando os buffers de ambas as bibliotecas, o que pode causar um veredito de **Time Limit Exceeded (TLE)**.

Buffer Sync

A solução para isso é desligar a sincronização.

Mas atenção: Uma vez desligado, você deverá usar somente printf e scanf ou somente cout e cin.



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

VIRTUS IMPAVIDA

hello.cpp

```
1 #include <bits/stdc++.h>
2
3 #define endl '\n'
4
5 using namespace std;
6
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(0);
10
11     /* code */
12
13     return 0;
14 }
```

Buffer Sync

Evite de usar endl, já que a função além de quebrar linha também força uma sincronização (ela chama flush).

Use '\n' no lugar ou utilize o define ao lado.



MaratonaCIn

Centro de
Informática
UFPE



hello.cpp

```
1 #include <bits/stdc++.h>
2
3 #define endl '\n'
4
5 using namespace std;
6
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(0);
10
11     /* code */
12
13     return 0;
14 }
```

Múltiplos casos

Muitas questões de programação competitiva usam vários casos de teste por execução.

Nesse casos, o template ao lado pode ajudar a abstrair os casos de teste caso eles sejam independentes.



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
VIRTUS IMPAVIDA

hello.cpp

```
1 #include <bits/stdc++.h>
2
3 #define endl '\n'
4
5 using namespace std;
6
7 int solve() {
8
9     /* code */
10
11     return 0;
12 }
13
14 int main() {
15     ios::sync_with_stdio(false);
16     cin.tie(0);
17
18     int tc; cin >> tc; while (tc--) {
19         solve();
20     }
21
22     return 0;
23 }
```

Truques úteis

Existem mais alguns truques que podem ser úteis por vezes, como redirecionar a entrada e saída, definir precisão de pontos flutuantes ou criar atalhos para tipos primitivos longos.

Alguns deles podem ser vistos ao lado.



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

VIRTUS IMPAVIDA

 hello.cpp

```
1 // Permite redirecionar a entrada ou
2 // a saída para um arquivo de texto
3 freopen("input.txt", "r", stdin);
4 freopen("output.txt", "w", stdout);
5
6 // Define uma precisão obrigatória
7 // de saída para pontos flutuantes
8 cout << setprecision(10) << fixed;
9
10 // Definem atalhos para os tipos
11 // de dados mais comuns
12 using ll = long long;
13 using pii = pair<int,int>;
14 using vi = vector<int>;
```



Complexidade Temporal e Espacial

Entendendo o BIG O

Importância

Em competições, não basta ter um código que funciona. Ele precisa ser rápido o suficiente para rodar dentro do tempo limite (geralmente 1 ou 2 segundos). A análise de complexidade nos ajuda a prever o quanto rápido um algoritmo será, sem precisar executá-lo.



A notação Big O descreve como o tempo de execução de um algoritmo cresce conforme o tamanho da entrada (N) aumenta. Focamos no pior caso e ignoramos constantes e termos de menor ordem.

Exemplo: Se um algoritmo faz $3N^2+5N+10$ operações, sua complexidade é $O(N^2)$.

Regra de Ouro:



Considere que um computador de competição executa cerca de 10^8 operações por segundo em C++. Com essa estimativa, você pode prever se sua solução passará:

| | N = 10 | N = 100 | N = 1.000 | N = 10.000 | N = 100.000 | N = 1.000.000 |
|---|-----------|------------------------|--------------|----------------|------------------|-------------------|
| O(logN) $\sim 10^{10}$ | ~ 3 | ~ 7 | ~ 10 | ~ 13 | ~ 17 | ~ 20 |
| O(N) $\sim 10^8$ | 10 | 100 | 1.000 | 10.000 | 100.000 | 1.000.000 |
| O(NlogN) $\sim 10^6$ | ~ 33 | ~ 664 | ~ 9.965 | ~ 132.877 | $\sim 1.660.964$ | $\sim 19.931.568$ |
| O(N ²) $\sim 10^4$ | 100 | 10.000 | 1.000.000 | 100.000.000 | 10^{10} | 10^{12} |
| O(N ³) $\sim 4 \times 10^2$ | 1.000 | 1.000.000 | 10^9 | 10^{12} | 10^{15} | 10^{18} |
| O(2 ^N) ~ 26 | 1.024 | 1.26×10^{30} | Muito Grande | Muito Grande | Muito Grande | Muito Grande |
| O(N!) ~ 11 | 3.628.800 | 9.33×10^{157} | Muito Grande | Muito Grande | Muito Grande | Muito Grande |

E o espaço?

Assim como o tempo, a memória é um recurso limitado. Em programação competitiva, você geralmente tem um limite de memória (ex: 256 MB, 512 MB).

Complexidade de Espaço é a medida de quanta memória seu programa utiliza em função do tamanho da entrada (N).

Escolher o tipo de dado correto é o primeiro passo para controlar o uso de memória.

Cada tipo de dado armazena um tipo diferente de informação e ocupa uma quantidade específica de memória.

Regra de Ouro:



Para saber a memória total usada por uma estrutura de dados, como um array, basta multiplicar o número de elementos pelo tamanho do tipo.

- Um array de 1 milhão de ints usa:
- $10^6 \text{ elementos} \times \text{tamanho do int} = 10^6 \times 4 \text{ bytes} = 4.000.000 \text{ bytes} \approx 4 \text{ MB}$

| | O que armazena | Tamanho Típico | Intervalo de Valores |
|-----------|------------------------------------|----------------|---|
| int | Números inteiros | 4 bytes | aprox. -2×10^9 a 2×10^9 |
| long long | Inteiros muito grandes | 8 bytes | aprox. -9×10^{18} a 9×10^{18} |
| char | Um único caractere | 1 byte | Caracteres ASCII |
| bool | Verdadeiro ou falso | 1 byte | true ou false |
| float | Números com ponto flutuante | 4 bytes | Precisão de ~7 dígitos |
| double | Ponto flutuante com dupla precisão | 8 bytes | Precisão de ~15 dígitos |



Biblioteca Padrão STDLib (STL)

Containers e algoritmos

Importância

A STL é uma coleção de estruturas de dados e algoritmos prontos, testados e eficientes. Usá-la economiza tempo e evita erros.

Você pode importar ela “completamente” usando o <bits/stdc++.h>.



MaratonaCIn



C++ hello.cpp

```
1 #include <bits/stdc++.h>
2
3 #define endl '\n'
4
5 using namespace std;
6
7 int main() {
8     ios::sync_with_stdio(false);
9     cin.tie(0);
10
11     /* code */
12
13     return 0;
14 }
```

Vector<T>

O que é? Um array dinâmico que pode crescer e diminuir de tamanho.

Quando usar? Quase sempre! É a estrutura de dados mais flexível e usada.

Observações: String é um vector<char> com operações extras implementadas.

vector<T>.cpp

```
1 // Cria um vetor de inteiros
2 vector<int> vec; // O(1)
3
4 // Cria um vetor de inteiros com tamanho 10
5 vector<int> vec2(10); // O(n)
6
7 // Cria um vetor de inteiros com tamanho
8 // 10, inicializado com 5
9 vector<int> vec3(10, 5); // O(n)
10
11 // Adiciona o elemento 1 ao final do vetor
12 vec.push_back(1); // O(1) amortizado
13
14 // Remove o último elemento do vetor
15 vec.pop_back(); // O(1)
16
17 // Retorna o tamanho atual do vetor
18 vec.size(); // O(1)
19
20 // Retorna true se o vetor estiver vazio, caso contrário false
21 vec.empty(); // O(1)
22
23 // Acessa o primeiro elemento do vetor e atribui o valor 10
24 vec[0] = 10; // O(1)
```

String

O que é? Um `vector<char>`

Quando usar? Quando precisar lidar com palavras.

Observações: Strings tem operações extras implementadas, que podem ser muito úteis.

strings.cpp

```
1 // Strings são representadas por aspas duplas
2 // Chars são representados por aspas simples
3 string s = "Meu nome é Lua";
4
5 // Retorna uma substring da posição 11 com tamanho 3
6 s.substr(11, 3); // "Lua"
7
8 // Retorna a posição da primeira ocorrência de "nome"
9 s.find("nome"); // 4
10
11 // Se "c++" não for encontrado, find retorna string::npos
12 if (s.find("c++") == string::npos) {
13     cout << "'c++' não foi encontrado";
14 }
15
16 // Converte string para inteiro e vice-versa
17 int n = stoi("123");
18 string n_str = to_string(456);
```

Map<Key, Value>

O que é? Uma coleção de pares chave-valor, onde as chaves são únicas e ordenadas.

Quando usar? Para contagem de frequência ou para associar um valor a outro (como um dict).

Observações: Caso queira permitir chaves duplicadas, pode usar multimap. Mas atenção, essa estrutura não permite acesso por chave, somente find e iterators.



Maratona

CIn

Centro de
Informática
UFPE



map<key, value>.cpp

```
1 // Cria um mapa de inteiros
2 map<int, int> m; // O(1)
3
4 // Insere o elemento 5 no mapa
5 m[5] = 1; // O(log n)
6
7 // Verifica se o elemento 5 está no mapa
8 m.count(5) // O(log n)
9
10 // Encontra o elemento 5 no mapa
11 m.find(5) // O(log n)
12
13 // Acessa o valor associado ao elemento 5 no mapa
14 m[5] // O(log n)
15
16 // Remove o elemento 5 do mapa
17 m.erase(5) // O(log n)
18
19 // Cria um multimap de inteiros
20 multimap<int, int> ms; // O(1)
21
22 // Insere o elemento 5 no multimap
23 ms.insert({5, 1}); // O(log n)
24
25 // Verifica quantas vezes o elemento 5 está no multimap
26 ms.count(5) // O(log n * k),
27 // onde k é o número de ocorrências de 5
28
29 // Encontra o primeiro elemento 5 no multimap
30 ms.find(5) // O(log n)
31
32 // Remove a primeira ocorrência do elemento 5 do multimap
33 ms.erase(ms.find(5)) // O(log n)
```

Set<T>

O que é? Uma coleção que armazena elementos únicos e ordenados.

Quando usar? Para manter uma lista de itens sem repetição e sempre em ordem ou para remover de qualquer lugar.

Observações: Caso queira permitir elementos duplicados, pode usar multimap.

set<T>.cpp

```
1 // Cria um conjunto de inteiros
2 set<int> s; // O(1)
3
4 // Insere o elemento 5 no conjunto
5 s.insert(5) // O(log n)
6
7 // Verifica se o elemento 5 está no conjunto
8 s.count(5) // O(log n)
9
10 // Encontra o elemento 5 no conjunto
11 s.find(5) // O(log n)
12
13 // Remove o elemento 5 do conjunto
14 s.erase(5) // O(log n)
15
16 // Cria um multiconjunto de inteiros
17 multiset<int> ms; // O(1)
18
19 // Insere o elemento 5 no multiconjunto
20 ms.insert(5) // O(log n)
21
22 // Verifica quantas vezes o elemento 5 está no multiconjunto
23 ms.count(5) // O(log n * k),
24 // onde k é o número de ocorrências de 5
25
26 // Encontra o primeiro elemento 5 no multiconjunto
27 ms.find(5) // O(log n)
28
29 // Remove a primeira ocorrência do elemento 5 do multiconjunto
30 ms.erase(ms.find(5)) // O(log n)
```

Stack<T> Queue<T>

O que é são?

(Stack): Estrutura LIFO (Last-In, First-Out). O último a entrar é o primeiro a sair.

(Fila): Estrutura FIFO (First-In, First-Out). O primeiro a entrar é o primeiro a sair.

Quando usar? Problemas que precisem de uma ordem específico de tratamento dos dados.



Maratona

CIn

Centro de
Informática
UFPE



stack<T> & queue<T>.cpp

```
1 // Cria uma pilha de inteiros
2 stack<int> st; // O(1)
3
4 // Adiciona o elemento 5 ao topo da pilha
5 st.push(5); // O(1)
6
7 // Remove o elemento do topo da pilha
8 st.pop(); // O(1)
9
10 // Retorna o elemento do topo da pilha
11 st.top(); // O(1)
12
13 // Cria uma fila de inteiros
14 queue<int> q; // O(1)
15
16 // Adiciona o elemento 5 ao final da fila
17 q.push(5); // O(1)
18
19 // Remove o elemento do início da fila
20 q.pop(); // O(1)
21
22 // Retorna o elemento do início da fila
23 q.front(); // O(1)
```

Deque<T>

O que é? Uma fila de final duplo, isso é, permite remoção na frente e no fundo em O(1).

Quando usar? Quando precisar alterar a frente o fundo de forma rápida e eficiente.

Observações: É possível fazer acesso por índice na deque em O(1).



Maratona

CIn

Centro de
Informática
UFPE



deque<T>.cpp

```
1 // Cria uma deque de inteiros
2 deque<int> dq; // O(1)
3
4 // Adiciona o elemento 5 ao final da deque
5 dq.push_back(5); // O(1)
6
7 // Remove o elemento do início da deque
8 dq.pop_front(); // O(1)
9
10 // Retorna o elemento do início da deque
11 dq.front(); // O(1)
12
13 // Adiciona o elemento 10 ao início da deque
14 dq.push_front(10); // O(1)
15
16 // Remove o elemento do final da deque
17 dq.pop_back(); // O(1)
18
19 // Retorna o elemento do final da deque
20 dq.back(); // O(1)
```

Deque<T>

O que é? Uma fila onde o elemento de maior prioridade (por padrão, o maior valor) está sempre na frente, pronto para ser removido.

Quando usar? Em algoritmos gulosos, como Dijkstra ou quando você precisa processar sempre o "melhor" item de uma coleção.



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
 VIRTUS IMPAVIDA

priority_queue<T>.cpp

```
1 // Cria uma fila de prioridade (max-heap) de inteiros
2 priority_queue<int> pq; // O(1)
3
4 // Cria uma fila de prioridade (min-heap) de inteiros
5 priority_queue<int, vector<int>, greater<>> min_pq; // O(1)
6
7 // Adiciona o elemento 5 à fila de prioridade
8 pq.push(5); // O(log(N))
9
10 // Remove o elemento com maior prioridade (topo) da fila
11 pq.pop(); // O(log(N))
12
13 // Retorna o elemento com maior prioridade (topo) da fila
14 pq.top(); // O(1)
```

Pair<T, T>

O que é? Uma par de elementos de tipos arbitrários com operadores pré-definidos.

Quando usar? Para unir dados dentro de outra estrutura de dados, por exemplo, unir informações que precisam ser ordenadas juntas.



Maratona

CIn



Centro de
Informática
UFPE



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

C++ pair<T, T>.cpp

```
1 // Cria um par de inteiro e string
2 pair<int, string> p; // O(1)
3
4 // Inicializa o par com valores
5 p = {5, "C++"}; // O(1)
6
7 // Acessa e modifica os valores do par
8 p.first = 10; // O(1)
9 p.second = "Lua"; // O(1)
10
11 // Desestrutura o par em duas variáveis
12 auto [num, lang] = p; // O(1)
```

Tuple<T, ...>

O que é? Uma tupla de n elementos de tipos arbitrários com operadores pré-definidos.

Quando usar? Para unir mais de 2 dados dentro de outra estrutura de dados, por exemplo, unir informações que precisam ser ordenadas juntas.



Maratona

CIn

Centro de
Informática
UFPE



tuple<T, ...>.cpp

```
1 // Cria uma tupla de int, string e double
2 tuple<int, string, double> t; // O(1)
3
4 // Inicializa a tupla com valores
5 t = {5, "C++", 3.14}; // O(1)
6
7 // Acessa e modifica os valores da tupla
8 get<0>(t) = 10; // O(1)
9 get<1>(t) = "Lua"; // O(1)
10 get<2>(t) = 2.71; // O(1)
11
12 // Desestrutura a tupla em três variáveis
13 auto [num2, lang2, val] = t; // O(1)
```

Funções Gerais

algorithms.cpp

```
1 sort(v.begin(), v.end()); // O(N log(N))
2 stable_sort(v.begin(), v.end()); // O(N log(N))
3
4 v.erase(v.begin(), v.begin() + 3); // O(N)
5 v.erase(unique(v.begin(), v.end()), v.end()); // O(N)
6
7 binary_search(v.begin(), v.end(), 20); // O(log(N))
8 lower_bound(v.begin(), v.end(), 20); // O(log(N))
9 upper_bound(v.begin(), v.end(), 20); // O(log(N))
10
11 accumulate(v.begin(), v.end(), 0LL); // O(N)
12 reverse(v.begin(), v.end()); // O(N)
13
14 min_element(v.begin(), v.end()); // O(N)
15 max_element(v.begin(), v.end()); // O(N)
16
17 next_permutation(v.begin(), v.end()); // O(N)
```

Além dos contêineres, a maior força da STL está em sua vasta coleção de algoritmos prontos, eficientes e testados.

Eles operam sobre sequências de dados, geralmente definidas por um par de iteradores, como `v.begin()` e `v.end()`.

Sort

O que faz? Ordena os elementos em uma faixa de valores. Não garante estabilidade.

Complexidade? $O(N \log N)$ em média. Para a versão estável é $O(N \log^2 N)$ ou $O(N \log N)$ se houver memória extra disponível.

Observações: Você pode ordenar ao contrário usando `rbegin()` e `rend()` ou passar um função customizada como 3º argumento.



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
 VIRTUS IMPAVIDA

sorting.cpp

```
1  vector<int> v = {4, 2, 5, 2, 1};  
2  sort(v.begin(), v.end());  
3  // v agora é {1, 2, 2, 4, 5}  
4  
5  // Para ordenar em ordem decrescente:  
6  sort(v.rbegin(), v.rend());  
7  
8  vector<pair<int,int>> arr;  
9  arr = { {2, 4}, {2, 2}, {1, 3} };  
10  
11 // Ordenar somente pelo primeiro elemento do par  
12 auto f = [](auto &a, auto &b) {  
13     return a.first < b.first;  
14 };  
15  
16 stable_sort(arr.begin(), arr.end(), f);  
17 // arr agora é { {1, 3}, {2, 4}, {2, 2} }
```

Erase

O que faz? A função `erase` é o método padrão para remover elementos da maioria dos contêineres da STL (como `vector`, `string`, `map`, `set`). Ela pode remover um único elemento ou uma faixa de elementos.

Complexidade? $O(N)$ para containers de acesso contínuo (`vector`), $O(\log N)$ para containers de acesso ordenado (`set`).



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
 VIRTUS IMPAVIDA

C++ erase.cpp

```
1  vector<int> v = {10, 20, 30, 40, 50};  
2  
3  // Remove o terceiro elemento  
4  // (índice 2, valor 30)  
5  v.erase(v.begin() + 2); // O(N)  
6  
7  // Agora v = {10, 20, 40, 50}  
8  
9  // Remove os elementos do segundo  
10 // ao terceiro (índices 1 a 2, valores 20, 40)  
11 v.erase(v.begin() + 1, v.begin() + 2); // O(N)  
12  
13 // Agora v = {10, 50}
```

Unique

O que faz? Remove elementos duplicados consecutivos de uma faixa. Precisa que a faixa esteja pré-ordenada.

É preciso usar `erase` para de fato remover os elementos extras. Caso contrário, eles apenas vão para o final.

Complexidade? $O(N)$.



Maratona

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
 VIRTUS IMPAVIDA

unique.cpp

```
1 vector<int> v = {1, 2, 2, 3, 3, 3, 2, 4, 4};  
2 // Primeiro, ordene para agrupar os duplicados  
3 sort(v.begin(), v.end());  
4 // v = {1, 2, 2, 2, 3, 3, 3, 4, 4}  
5  
6 // Aplique o padrão erase-unique  
7 v.erase(unique(v.begin(), v.end()), v.end());  
8  
9 // v agora é {1, 2, 3, 4}
```

Buscas binárias

O que fazem?

binary_search: Verifica se um elemento existe na faixa (retorna true ou false).

lower_bound: Retorna um iterador para o primeiro elemento não menor que o valor buscado.

upper_bound: Retorna um iterador para o primeiro elemento maior que o valor buscado.

Complexidade? $O(\log N)$



Maratona

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
VIRTUS IMPAVIDA

C++ binary_search.cpp

```
1 // Já ordenado!
2 vector<int> v = {10, 20, 20, 30, 40};
3
4 bool existe = binary_search(v.begin(), v.end(), 20);
5 // existe = true
6
7 // Pega o índice do primeiro 20
8 auto it_lower = lower_bound(v.begin(), v.end(), 20);
9 int index = it_lower - v.begin(); // index = 1
10
11 // Pega o índice do elemento após o último 20
12 auto it_upper = upper_bound(v.begin(), v.end(), 20);
13
14 int count = it_upper - it_lower;
15 // count = 2 (existem dois '20')
```

Accumulate

O que faz? Soma todos os elementos em uma faixa, partindo de um valor inicial.

Complexidade? $O(N)$.

Reverse

O que faz? Inverte a ordem dos elementos.

Complexidade? $O(N)$.



MaratonaCIn

Centro de
Informática
UFPE



accumulate.cpp

```
1 vector<int> v = {1, 2, 3, 4, 5};  
2 // soma = 15  
3 ll soma = accumulate(v.begin(), v.end(), 0LL);  
4  
5 // 0LL é importante para evitar overflow  
6 // se a soma for grande!
```

reverse.cpp

```
1 vector<int> v = {1, 2, 3, 4, 5};  
2 reverse(v.begin(), v.end(), 0LL);
```

Min_element

O que faz? Encontra o menor elemento dentro da faixa.

Complexidade? $O(N)$.

Max_element

O que faz? Encontra o maior elemento dentro da faixa.

Complexidade? $O(N)$.



MaratonaCIn

Centro de
Informática
UFPE



min_element.cpp

```
1 vector<int> v = {3,2,7,1,8};  
2 mn = *min_element(v.begin(), v.end());
```



max_element.cpp

```
1 vector<int> v = {3,2,7,1,8};  
2 mx = *max_element(v.begin(), v.end());
```

Next_permutation

O que faz? Transforma uma sequência na sua próxima permutação lexicograficamente maior.

Retorna true se uma próxima permutação foi encontrada, e false se a sequência já está na maior permutação possível (ex: [3, 2, 1]).

Complexidade? $O(N)$ por chamada $O(N!)$ no loop do-while.



Maratona

CIn



Centro de
Informática
UFPE



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

next_permutation.cpp

```
1  vector<int> v = {1, 2, 3};  
2  // 1. Garanta que o vetor esteja ordenado  
3  sort(v.begin(), v.end());  
4  
5  // 2. Use o loop do-while para iterar  
6  // por todas as permutações  
7  do {  
8      for (auto e : v) cout << e << ' ';  
9      cout << endl;  
10 } while (next_permutation(v.begin(), v.end()));
```

For loops

Desde o C++11, existe uma forma mais simples e legível de percorrer todos os elementos de um contêiner, como um vector. Em vez de usar índices ou iteradores manuais, você pode usar um range-based for loop.

A forma como você declara a variável muda a eficiência e o que você pode fazer dentro do loop.



Maratona

CIn

Centro de
Informática
UFPE



C++ for loops.cpp

```
1  vector<int> v = {1, 2, 3};  
2  
3  // Passa pelos elementos da cópia de v  
4  // Não permite modificação  
5  // O(N) da cópia + O(N) do loop  
6  for (auto x : v) {  
7      cout << x << ' ';  
8  } cout << endl;  
9  
10 // Passa pelos elementos de v  
11 // Somente O(N) do loop  
12 for (auto &x : v) {  
13     x *= 2; // Permite modificação  
14 }  
15  
16 // Não permite modificação mas evita cópia  
17 // Somente O(N) do loop  
18 for (auto const&x : v) {  
19     cout << x << ' ';  
20 } cout << endl;
```



Prefix Sum

Seu primeiro algoritmo

Importância

Imagine um problema comum:
você recebe um array com N
números e depois Q consultas.
Cada consulta te dá um intervalo
[L,R] e pede a soma de todos os
elementos desde o índice L até o
índice R.



Exemplo:

array = {2, 8, 3, 5, 1, 9}

Consulta 1: Qual a soma do
intervalo [1,3]? ($8 + 3 + 5 = 16$)

Consulta 2: Qual a soma do
intervalo [0,2]? ($2 + 8 + 3 = 13$)

Solução ingênuua

A primeira ideia é, para cada consulta, percorrer o array de L até R com um for e somar os valores.

Análise: Se N e Q podem chegar a 10^5 , essa abordagem teria uma complexidade de $O(N \cdot Q)$, ou seja, $10^5 \times 10^5 = 10^{10}$ operações. Isso é lento demais!



MaratonaCIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO
 VIRTUS IMPAVIDA



prefix sum.cpp

- □ ×

```
1 // Para cada consulta [L, R]
2 long long soma = 0;
3 for (int i = L; i <= R; i++) {
4     soma += array[i];
5 }
```

Forma eficiente



Maratona**CIn**



A ideia da Soma de Prefixos é gastar um tempo inicial ($O(N)$) para pré-calcular as somas acumuladas até cada índice do array.

Com isso, podemos responder a qualquer consulta de soma em tempo constante ($O(1)$). Só precisamos olhar a soma acumulada até os índices e fazer algumas operações matemáticas.

Forma eficiente

C++ prefix_sum.cpp

- □ ×

```
1 vector<int> v = {1, 5, 3, ...};  
2 // Tamanho N+1, inicializado com 0  
3 vector<ll> psum(N + 1, 0);  
4  
5 for (int i = 0; i < N; i++) {  
6     psum[i + 1] = psum[i] + v[i];  
7 }
```

1. Construindo o Array de Soma de Prefixo (Prefix Sum)

Criamos um array psum onde psum [i] guarda a soma de todos os elementos do array original do índice 0 até i-1.

É comum usar 1-based indexing para o array psum para simplificar a fórmula.

Forma eficiente

prefix_sum.cpp

- □ ×

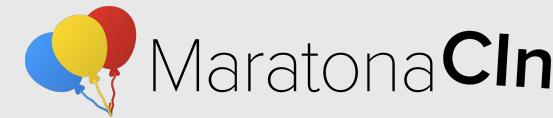
```
1 vector<int> v = {1, 5, 3, ...};  
2 // Tamanho N+1, inicializado com 0  
3 vector<ll> psum(N + 1, 0);  
4  
5 for (int i = 0; i < N; i++) {  
6     psum[i + 1] = psum[i] + v[i];  
7 }  
8  
9 for (int i = 0; i < Q; i++) {  
10    int l, r; cin >> l >> r;  
11    cout << psum[r+1] - psum[l] << endl;  
12 }
```

2. A Mágica: Respondendo Consultas em O(1)

A soma do intervalo [L,R] (inclusivo) é simplesmente:

$$\text{soma}(L,R) = \text{pref}[R+1] - \text{pref}[L]$$

Forma eficiente



Por que funciona?

$\text{pref}[R+1]$ é a soma de $\text{array}[0]$ até $\text{array}[R]$.

$\text{pref}[L]$ é a soma de $\text{array}[0]$ até $\text{array}[L-1]$.

Subtraindo um do outro, sobram apenas os elementos de $\text{array}[L]$ até $\text{array}[R]$.

[0 ... L-1 | L ... R | R+1 ... N-1]

$\text{pref}[R+1]$ (soma de tudo até R) - $\text{pref}[L]$ (soma de tudo até L-1) = soma(L,R)

Forma eficiente



Maratona

CIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO


Análise Final:

- Tempo de pré-processamento: $O(N)$
- Tempo por consulta: $O(1)$
- Complexidade Total: $O(N+Q)$. Para $N,Q=10^5$, isso é cerca de $2 \cdot 10^5$ operações. Extremamente rápido!



MaratonaCIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

VIRTUS IMPAVIDA

Dúvidas?



MaratonaCIn

Centro de
Informática
UFPE

UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

VIRTUS IMPAVIDA

Dúvidas?

Próximo passo:
Fazer o Homework!



MaratonaCIn

Centro de
Informática
UFPE



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

That's all, Folks!

