



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Algoritmi STL – Introduzione e algoritmi di ricerca

Stefano Ghidoni



Agenda

- Introduzione e panoramica sugli algoritmi STL
- Un primo esempio: `find()`



Algoritmi STL

- STL offre un grande numero di algoritmi (circa 80!) $= 7.1569457046263802294811533723186532165584657342365752577109 \times 10^{118}$
lol
- Impossibile conoscerli tutti, ma:
 - Conoscerne alcuni permette di abituarci alla sintassi
 - Buona base di partenza per usare anche altri algoritmi
- Conoscere gli algoritmi ci permette di risparmiare:
 - Tempo
 - Sforzi



- Molti algoritmi disponibili:
 - Ricerca
 - Ordinamento
 - Conteggio
 - Copia con/senza duplicati
 - Fusione
- Algoritmi compatibili con tutti i container STL
- **#include <algorithm>**



Algoritmi STL – una selezione

Algoritmo	Significato
<code>r = find(b, e, v)</code>	<code>r</code> punta alla prima occorrenza di <code>v</code> in <code>[b:e)</code>
<code>r = find_if(b, e, p)</code>	<code>r</code> punta al primo elemento <code>x</code> in <code>[b:e)</code> tale che <code>p(x) == true</code>
<code>x = count(b, e, v)</code>	<code>x</code> è il numero di occorrenze di <code>v</code> in <code>[b:e)</code>
<code>x = count_if(b, e, p)</code>	<code>x</code> è il numero di elementi in <code>[b:e)</code> per cui <code>p(x) == true</code>
<code>sort(b, e)</code>	Ordina <code>[b:e)</code> usando <code><</code>
<code>sort(b, e, p)</code>	Ordina <code>[b:e)</code> usando <code>p</code>
<code>copy(b, e, b2)</code>	Copia <code>[b:e)</code> in <code>[b2:b2+(e-b))</code> – è necessario sufficiente spazio a destinazione
<code>unique_copy(b, e, b2)</code>	Come sopra, elimina i duplicati
<code>merge(b, e, b2, e2, r)</code>	Fonde due sequenze ordinate <code>[b2:e2)</code> e <code>[b:e)</code> in <code>[r:r+(e-b)+(e2-b2))</code>
<code>r = equal_range(b, e, v)</code>	<code>r</code> è la sottosequenza del range ordinato <code>[b:e)</code> contenente <code>v</code> (ricerca binaria)



Algoritmi STL – numerici

Algoritmo	Significato
<code>equal(b, e, b2)</code>	Confronta tutti gli elementi in $[b:e)$ e $[b2:b2+(e-b))$
<code>x = accumulate(b, e, i)</code>	$x = i + \text{la somma di tutti gli elementi in } [b:e)$
<code>x = accumulate(b, e, i, op)</code>	Come sopra, ma con la somma calcolata usando <code>op</code>
<code>x = inner_product(b, e, b2, i)</code>	Prodotto scalare di $[b:e)$ e $[b2:b2+(e-i))$
<code>x = inner_product(b, e, b2, i, op, op2)</code>	Come sopra, ma con <code>op</code> e <code>op2</code> al posto di <code>+ e *</code>

find()



find()

- `find()` cerca un elemento con un dato valore in una sequenza

```
template<typename In, typename T>
    // In: iteratore di input
    // T deve essere confrontabile
In find(In first, In last, const T& val)
{
    while(first != last && *first != val) ++first;
    return first;
}
```

- Implementazione efficiente e veloce



find()

- La sequenza su cui opera `find()` è definita mediante due iteratori
 - `[first, last)`
- Ritorna un iteratore
 - Alla prima occorrenza trovata
 - `end()` se il valore non è stato trovato
- In STL, ritornare `end()` spesso indica che un elemento non è stato trovato

Ma alcuni algoritmi restituiscono proprio la fine



Caratteristiche di find()

- **find()** è generico
 - Rispetto al tipo di container
 - Rispetto all'elemento contenuto
- La chiamata a **find()** non cambia in funzione di questi due elementi
- Flessibilità comune agli algoritmi STL
 - Effetto dell'**interfaccia standard** gestita tramite iteratori
 - Vedi esempio slide successiva



Flessibilità

```
void f(vector<int>& v, int x)
{
    vector<int>::iterator p = find(v.begin(), v.end(), x);
    if (p != v.end())
    {
        // x è stato trovato
    }
}
```

```
void f(list<string>& v, string x)
{
    list<string>::iterator p = find(v.begin(), v.end(), x);
    if (p != v.end())
    {
        // x è stato trovato
    }
}
```



Flessibilità

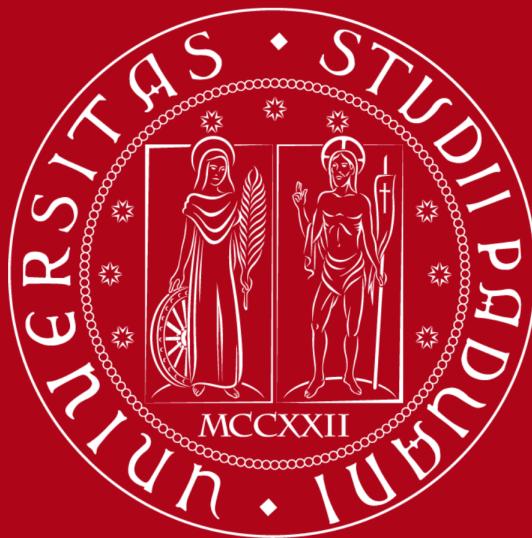
```
void f(vector<int>& v, int x)
{
    auto p = find(v.begin(), v.end(), x);
    if (p != v.end())
    {
        // x è stato trovato
    }
}
```

```
void f(list<string>& v, string x)
{
    auto p = find(v.begin(), v.end(), x);
    if (p != v.end())
    {
        // x è stato trovato
    }
}
```



Recap

- Introduzione agli algoritmi STL
- `find()`
- Caratteristiche di flessibilità di `find()`



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Algoritmi STL – Introduzione e algoritmi di ricerca

Stefano Ghidoni