Come allenarsi e diventare cintura nera di STL

- Thinking in Patterns -

Marco Arena

Al ristorante…

- ti siedi,
- ti danno il menù,
- scegli cosa prendere,
- vengono a prendere l'ordinazione,
- Ecc…

Un copione ben consolidato

Il cervello spiegato da un informatico

- Cervello = macchina anticipatrice
- Per ragioni di efficienza, si tiene una «cache» col proprio modello del mondo (*box*)
- Per ogni decisione da prendere, usa il box per scartare le alternative
  (il *lobo frontale* decide quali opzioni considerare)
- Per ogni «sorpresa» raffina il modello

A lezione dai maestri di scacchi

Quante mosse da qui in avanti riesci a calcolare?

"Solo una, la migliore." – José Capablanca

A lezione dai maestri di scacchi

- Gli esperti scacchisti sono bravi a riconoscere i **pattern** che emergono da un certo stato del gioco.
- Questo consente loro di «scartare» mosse.
- Tuttavia, se da un dato stato non emergono pattern, gli esperti mediamente non hanno vantaggi di memoria rispetto ai principianti.

Thinking *inside* the box

```
for (auto i=0u; i<N; ++i)
{
   cout << v[i] << "\n";
}
```

Thinking *inside* the box

```
for (auto j=0; j<N; ++j)
{
     list.addItem(v[j]);
}
```

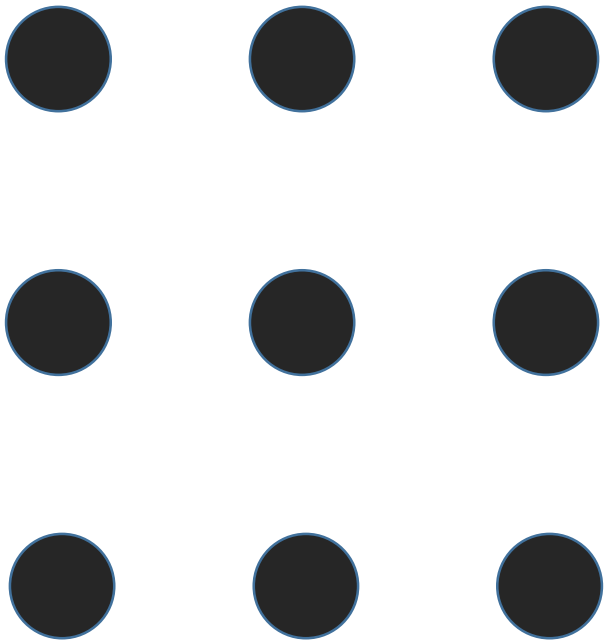
Thinking *inside* the box

```
while (i != limit)
{
     f(arr[i] * arr[i]);
     ++i;
}
```

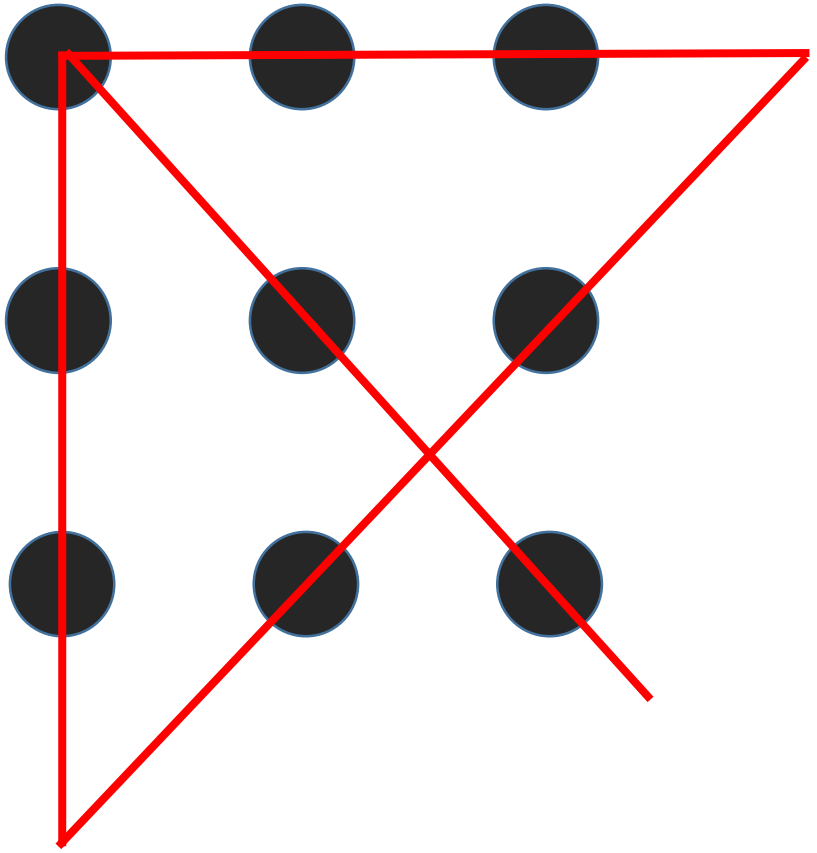
Thinking *inside* the box

```
for (i=0; i<N; ++i)
{
     cart.add(price[i] - discount[i]);
}
```

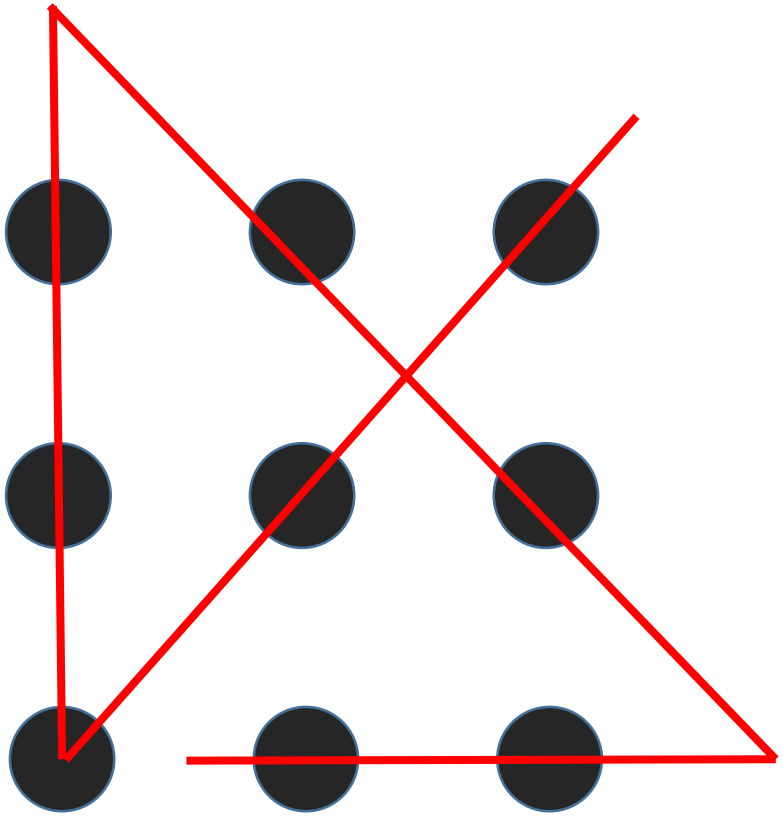
Nine dot problem



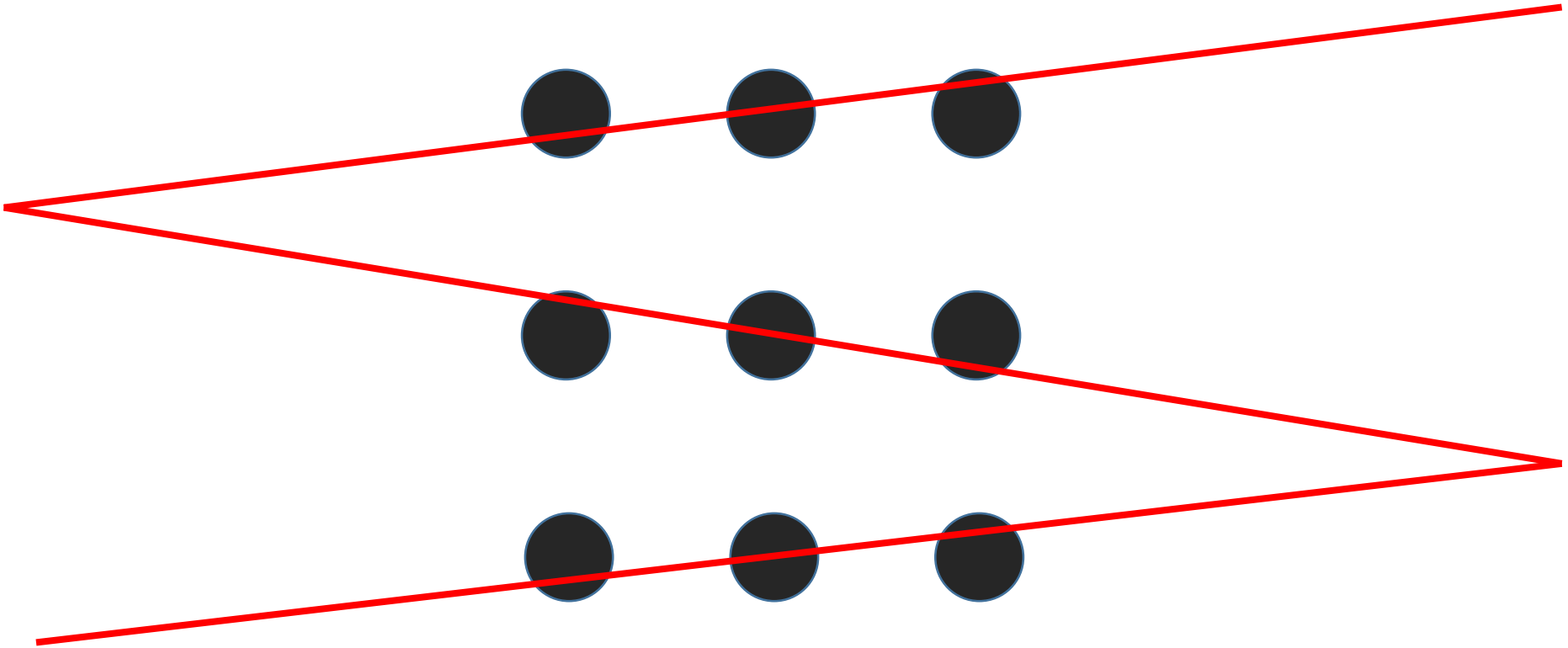
Nine dot problem



Nine dot problem



Nine dot problem



Thinking outside the box

- Quando il box non è abbastanza bisogna uscirne
- Non è un processo totalmente «volontario»
- Si devono creare le *condizioni* giuste

Thinking inside the box++

- Tuttavia, possiamo *raffinare e migliorare* il proprio box, ovvero la propria conoscenza di *modelli* del mondo e la propria capacità di riconoscerli e usarli

- Da un certo punto di vista, espandere il proprio box è una forma di creatività

- La pratica e le **condizioni** sono fondamentali

Le condizioni per allenarsi

- Ansia e scadenze riducono la creatività
- Ricompense e creatività
  - Una ricompensa economica non migliora la creatività
- Sbagliare deve far parte del processo di allenamento
- Uscire volontariamente della comfort zone
- *Conoscersi* è importante (**Pratica Consapevole**)

Torniamo all’informatica

Coding Pattern

- Soluzione standard a problema standard (modello)
- Intuitivo ed espressivo
- Precondizioni e post-condizioni (invarianti)
- Customization points
- Se ne possono comporre tanti insieme
- Alcuni pattern sono intuitivi anche per i "non addetti ai lavori"
  (*e.g. papà, mi ordini gli yogurt per data di scadenza?*)

$Your Coding Pattern

- Proprio del tuo ecosistema
- Esempio:

  Linguaggio proprietario per manipolare segnali

  No cicli

  Solo funzioni e operazioni matematiche

```
LastSeconds(10, Integrate(RemoveNoise(input * gain)))
```

Allenarsi con i pattern

- Usarli – applicare quelli noti
- Impararne di nuovi
- Applicarli «a posteriori» (soluzione alla mano)

  Può richiedere uno «sforzo»
- Provare linguaggi/paradigmi diversi/sconosciuti

Coding Patterns in C++

La triade del C++

- Contenitori
- Iteratori
- Algoritmi

Containers

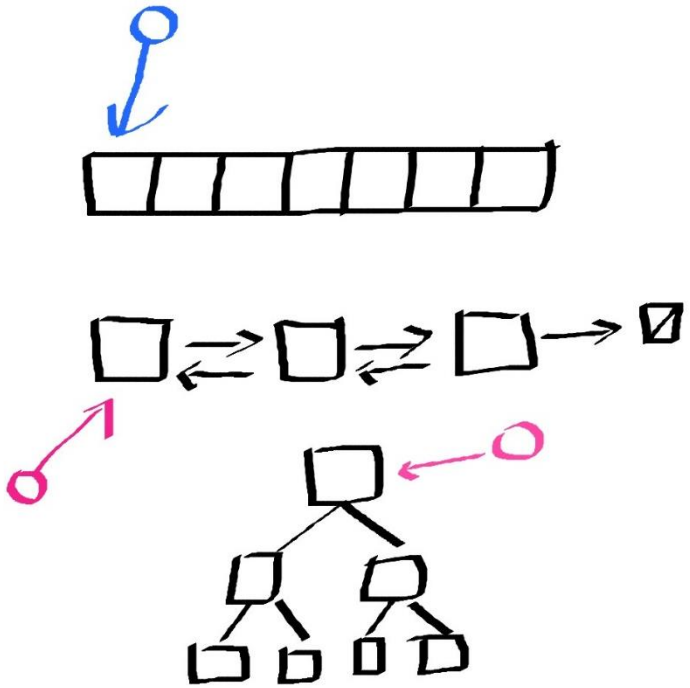
- Strutture dati *generiche*
- Hanno requisiti sui tipi che contengono (Concepts)

Esempi:

```
vector<T>
map<K, V>
```

Iteratori

- Astrazioni sull'accesso ad una struttura dati
- *Categorie* diverse in base alle operazioni esposte



Algoritmi

- Funzioni generiche su una *coppia* di iteratori [begin, end)

Esempi:

```
std::accumulate
std::find
std::remove_if
```

Algoritmi STL e Coding Patterns

- Molti algoritmi STL sono implementazioni di pattern più generali (e.g. accumulate, find)

- Io considero **tutti gli algoritmi pattern** da combinare insieme per risolvere problemi

Perché usare gli algoritmi STL

- Soliti motivi: efficienza, affidabilità, stato dell'arte
- Espressività: ogni «pattern» esprime i suoi intenti in maniera piuttosto chiara e dichiarativa
  «Un linguaggio nel linguaggio»
- **Parallel STL**: la maggior parte degli algoritmi STL hanno una versione parallela/vettoriale
- **Ranges (C++20):** view ed action dei range sono delle implementazioni range-based degli algoritmi STL

Perché usare gli algoritmi STL

Per restare nel mercato

Pratica

Ciò che dobbiamo imparare a fare, lo impariamo facendolo.
- Aristotele -

Dove trovare gli esercizi

- Competitive Programming
  - Hacker Rank
  - Leet Code
  - Top Coder
  - Coding Game
  - Interview Bit
  - …
- Variazioni di problemi che incontriamo al lavoro (richiede creatività!)
- Coding Gym :-)

Ogni esercizio ha valore

- Contesto di allenamento:
  - No ansia, no stress
  - Non c'è una «ricompensa»
  - Posso sbagliare, gli errori sono un valore
  - Posso allocare tempo per sperimentare

  Create il contesto **giusto per voi**

- Usare ogni esercizio **creativamente** e **liberamente**

- Imparare a fare *domande di valore* sugli esercizi

Ogni esercizio ha valore

- Esempi di domande *di valore*:
  - Posso applicare uno o più pattern?
  - Posso evitare iterazioni/scansioni?
  - Posso sostituire i cicli con funzioni standard?
  - Posso salvare spazio e/o tempo?
  - Cambia qualcosa se ordino/disordino i dati?
  - Cosa cambia se *tolgo* un'operazione (e.g. divisione)?
  - Come posso sfruttare il dominio?
  - Come posso generalizzare la soluzione?
  - Esistono soluzioni peggiori/migliori di questa?
  - Cosa succede se la scala dell'input cambia?
  - …

Pattern

```
for (auto i=0; i<N; ++i)
   if (v[i] == 10)
        return i;
```

```
while (begin != end)
   if (*begin++ == 10)
        return begin;
```

```
while (begin != end)
{
   if (pred(*begin))
        return begin;
   begin++;
}
```

find

find_if

Pattern

```
vector<int> destination(N);
for (auto i=0; i<N; ++i)
   destination[i] = v[i];
```

copy

```
while (begin != end)
   *out++ = *begin++;
```

copy_n

```
while (begin != end) {
   if (*begin % 2 == 0)
        *out++ = *begin;
    begin++;
}
```

copy_if

```
while (begin != end) {
   if (pred(*begin))
        *out++ = *begin;
   begin++;
}
```

Pattern

transform

```
vector<int> out(N);
for (auto i=0; i<N; ++i)
   out[i] = 2 * v[i];


for (auto i=0; i<N; ++i)
   v[i] = z[i] * v[i];


string s = "ciao";
for (auto& c : s)
        c = std::toupper(c);


while (begin != end)
   *out++ = fun(*begin++);
```

Pattern

accumulate

```
auto sum = 0;
for (auto i=0; i<N; ++i)
   sum += v[i];


auto mul = 1;
while (begin != end)
   mul *= *begin++;


auto min = 0;
while (begin != end)
   min = std::min(*begin++, min);
```

Allenatevi a modificare la possibile implementazione di un pattern (e.g. en.cppreference.com) in modo da somministrare più casi concreti al vostro cervello.

https://en.cppreference.com/w/cpp/algorithm

Warm up – Array Sum

```
int N;
cin >> N;
vector<int> v(N);
for (auto i=0; i<N; ++i)
   cin >> v[i];

auto sum = 0;
for (auto i=0; i<N; ++i)
   sum += v[i];

cout << sum;
```

Cerchiamo più valore

Proviamo a togliere i cicli

Warm up – Array Sum

```
int N;
cin >> N;
vector<int> v(N);
copy_n(istream_iterator<int>(cin), N, bgin(v));

auto sum = 0;
for (auto i=0; i<N; ++i)
   sum += v[i];

cout << sum;
```

Warm up – Array Sum

```
int N;
cin >> N;
vector<int> v(N);
copy_n(istream_iterator<int>(cin), N, bgin(v));

auto sum = accumulate(begin(v), end(v), 0);


cout << sum;
```

Warm up – Array Sum

```
int N;
cin >> N;
```

```
auto sum = accumulate(istream_iterator<int>(cin),
                      istream_iterator<int>(), 0);
```

```
cout << sum;
```

Warm up – Array Sum

```
cout << accumulate(next(istream_iterator<int>(cin)),
                   istream_iterator<int>(), 0);
```

Allenarsi con le sequenze ordinate

Papà, mi trovi un nome sull'elenco telefonico?

Pairs

Dato un array di interi, trovare quante coppie danno come differenza K.

Esempio:

`[2 3 7 1 9]`

K=2

Risposta: 2. Ovvero `(9,7), (3, 1)`

Pairs

Algoritmo brute-force facile ma inefficiente (quadratico):

```
auto cnt = 0;
for (auto i : v)
{
	for (auto j : v)
	{
		if (i-j == K)
			cnt++;
	}
}
cout << cnt;
```

Pairs

```
sort(begin(v), end(v));

auto cnt = 0;
for (auto i : v)
{
	if (binary_search(begin(v), end(v), i-K))
		cnt++;
}
cout << cnt;
```

i - j = k

→ j = i - k

Pairs

```
sort(begin(v), end(v));

cout << count_if(begin(v), end(v), [&](int i) {

        return binary_search(begin(v), end(v), i-K);

});
```

https://en.cppreference.com/w/cpp/algorithm/count

Pairs

Alcune varianti:

- Binary search più «ottimizzata»
- Usare strutture dati con lookup efficiente (e.g. set, unordered_set)

https://coding-gym.org/challenges/pairs/

Altri pattern sulle sequenze ordinate

- `lower_bound, upper_bound, equal_range`
- `set_* (operazioni sugli insiemi)`
- `merge, includes`
- `is_sorted, is_sorted_until`
- `partial_sort, stable_sort, ecc`
- `nth_element`

Allenatevi con la binary search

Per casa:

Beautiful Triplets

Oltre a quella con la ricerca binaria, questo esercizio lascia spazio a tante altre soluzioni…

Soluzioni e discussione:

https://coding-gym.org/challenges/beautiful-triplets/

Fate amicizia con `lower_bound`

Per casa:

Minimum Loss

Esistono almeno due soluzioni. Per allenarvi con `lower_bound`, provate a pensare a come usare un `std::set` di supporto…

Soluzioni e discussione:

https://coding-gym.org/challenges/minimum-loss

Allenarsi con un altro linguaggio

Farsi ispirarsi da Python

Squares of a Sorted Array

Dato un array di numeri ordinati in modo crescente, stampare l'array dei quadrati anch'esso ordinato in modo crescente. Trovare un algoritmo lineare.

Esempi:

```
in:   [-4 -1 0 3 10]
out:  [0 1 9 16 100]
```

Squares of a Sorted Array

```
vector<int> res(A.size());
// locate the first positive value
auto it = find_if(begin(A), end(A), [](int i) { return i >= 0; });
auto pos = (int)distance(begin(A), it);
auto neg = pos - 1;

int len = (int)A.size();
for (auto i = 0; i < len; ++i)
{
    // negative values finished
    if (neg < 0)
    {
        res[i] = square(A[pos++]);
    }
    // positive values finished
    else if (pos >= len)
    {
        res[i] = square(A[neg--]);
    }
    // positive value is bigger
    else if (square(A[pos]) > square(A[neg]))
    {
        res[i] = square(A[neg--]);
    }
    // negative value is bigger
    else
    {
        res[i] = square(A[pos++]);
    }
}
return res;
```

Squares of a Sorted Array

Esistono diverse soluzioni:

https://coding-gym.org/challenges/squares-of-a-sorted-array/

Squares of a Sorted Array

Tra le varie soluzioni, consideriamo questa in Python:

```
positive = []
negative = []

for v in values:
    (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```

Dimentichiamoci che sia meno efficiente di altre ma usiamola per allenarci con la nostra STL.

Squares of a Sorted Array

Tra le varie soluzioni, consideriamo questa in Python:

```
positive = []
negative = []

for v in values:
    (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```

reverse

Squares of a Sorted Array

Tra le varie soluzioni, consideriamo questa in Python:

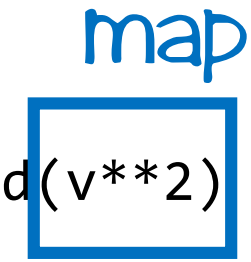
```
positive = []
negative = []

for v in values:
    (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```

merge

Squares of a Sorted Array

Tra le varie soluzioni, consideriamo questa in Python:

```
positive = []
negative = []

for v in values:
    (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```



Squares of a Sorted Array

Tra le varie soluzioni, consideriamo questa in Python:

```
positive = []
negative = []

for v in values:
    (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```

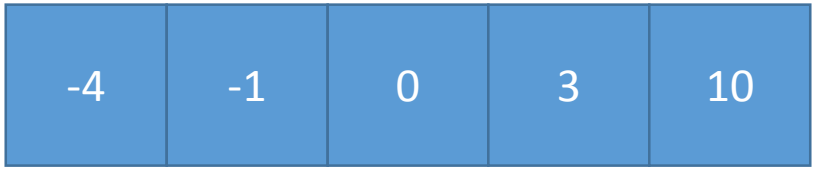
partition

Squares of a Sorted Array

```
positive = []
negative = []

for v in values:
   (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```

```
vector<int> negatives, positives, res(v.size());


partition_copy(
           begin(v), end(v),
           back_inserter(negatives),
           back_inserter(positives),
           [](auto i) { return i<0; });

transform(
           begin(negatives), end(negatives),
           begin(negatives),
           [](auto i){ return i*i; });

transform(
           begin(positives), end(positives),
           begin(positives),
           [](auto i){ return i*i; });

reverse(begin(negatives), end(negatives));


merge(begin(negatives), end(negatives),
      begin(positives), end(positives),
      begin(res));
```

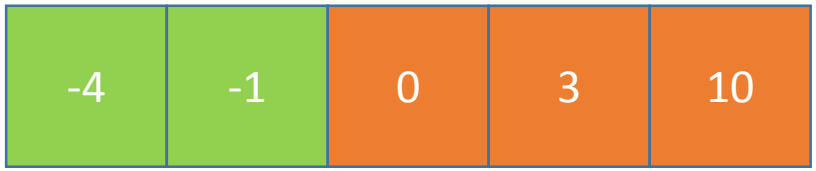
Squares of a Sorted Array



Squares of a Sorted Array

Squares of a Sorted Array

```
positive = []
negative = []

for v in values:
   (negative, positive)[v >= 0].append(v**2)

res = merge(reversed(negative), positive)
```



primo positivo

```
vector<int> res(v.size());

auto pos = find_if(begin(v), end(v),
           [](auto i){ return i>=0; });

transform(begin(v), end(v), begin(v),
          [](auto i){return i*i;});

merge(
    make_reverse_iterator(firstPos), rend(v),
    firstPos, end(v),
    begin(res));
```

va indietro di uno

Allenarsi con pattern meno noti

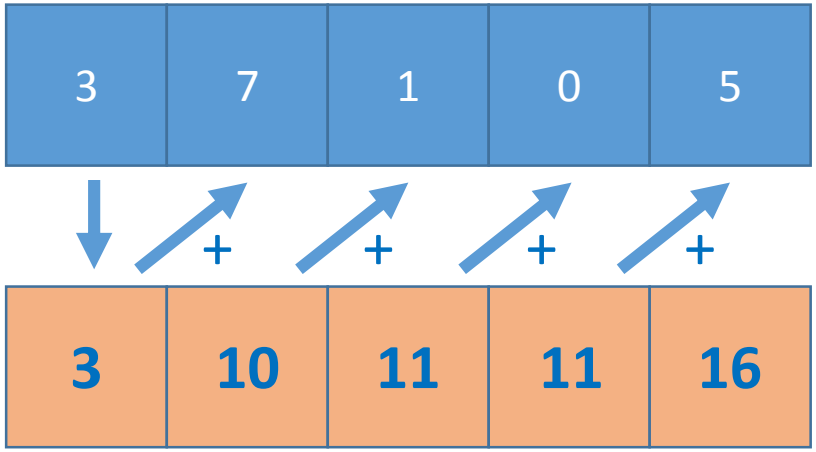
Impariamo un nuovo pattern

Prefix Sum

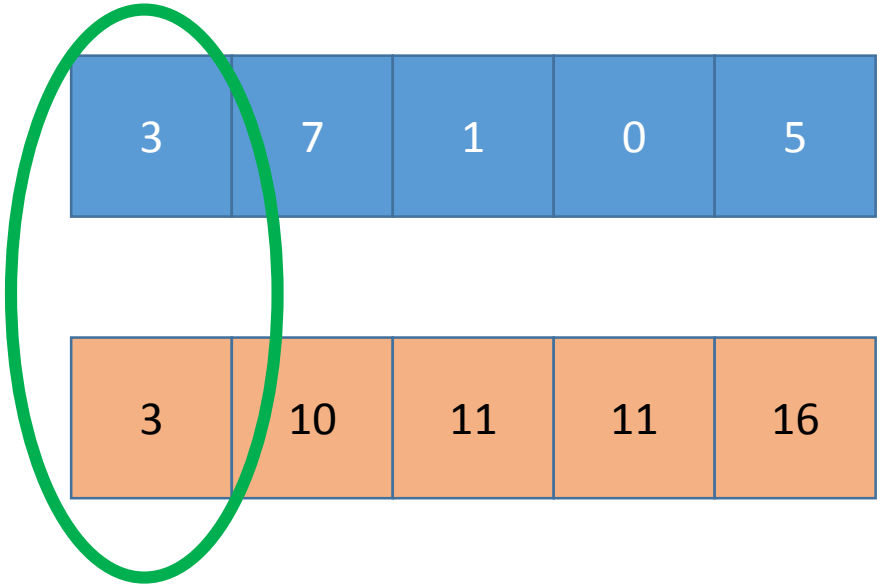
partial_sum

```
auto acc = *begin;
*out = acc;
while (++begin != end)
{
    acc = acc + *begin;
    *++out = acc;
}
```

Prefix Sum



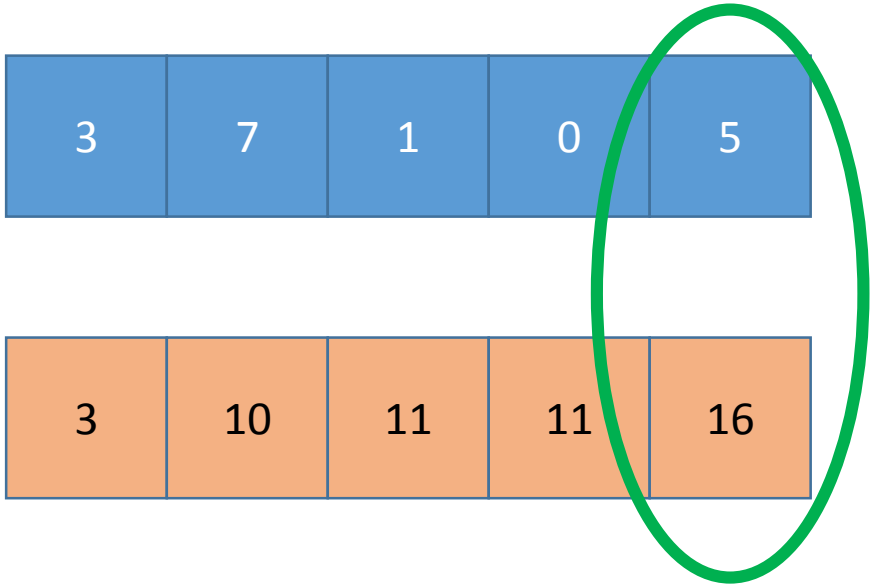
Prefix Sum



Invariante

`psum[0] = in[0]`

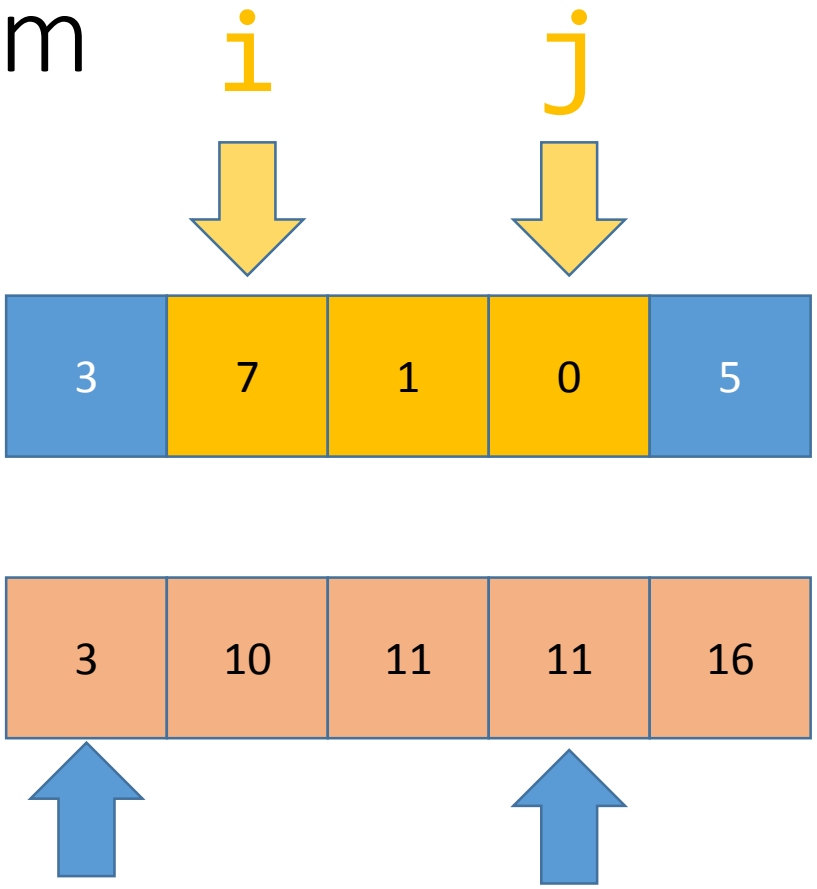
Prefix Sum



Invariante

```
psum[N-1] = accumulate(in)
```

Prefix Sum



Invariante

`somma in[i, j] = psum[j] – psum[i-1]`

Prefix Sum: Esempio

```
int N, Q; cin >> N >> Q;
vector<int> prefix(N + 1);
copy_n(istream_iterator<int>(cin), N, begin(prefix));
partial_sum(next(begin(prefix)), end(prefix), begin(prefix));
int i, j;
while (Q--)
{
    cin >> i >> j;
    cout << (prefix[j] - prefix[i-1]) << "\n";
}
```

The Equilibrium Index

Dato un array, trovare l'indice dell'elemento che divide l'array in due parti con la stessa somma.

Esempi:

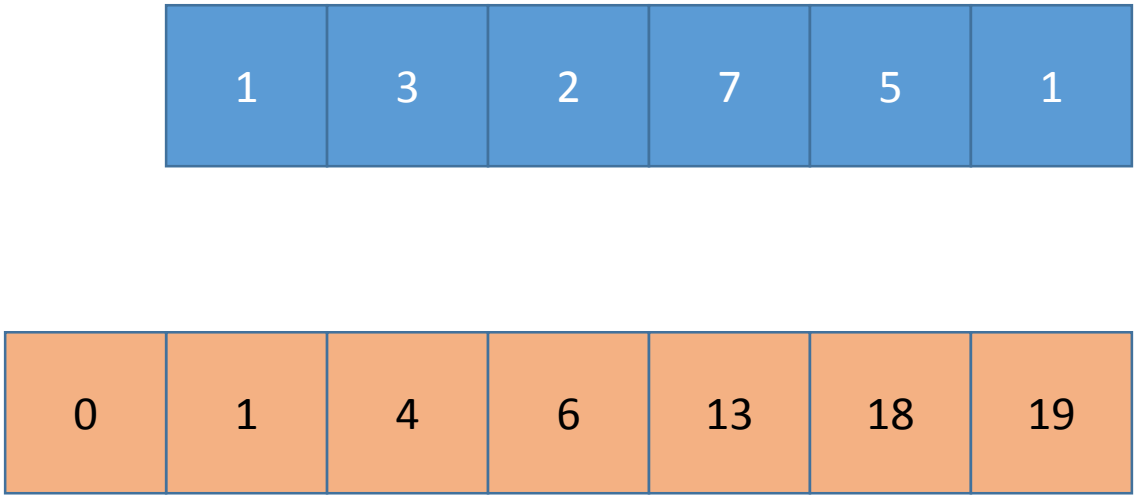
```
[1 3 2 7 5 1]      [1 0 1]
       ^               ^
```

The Equilibrium Index

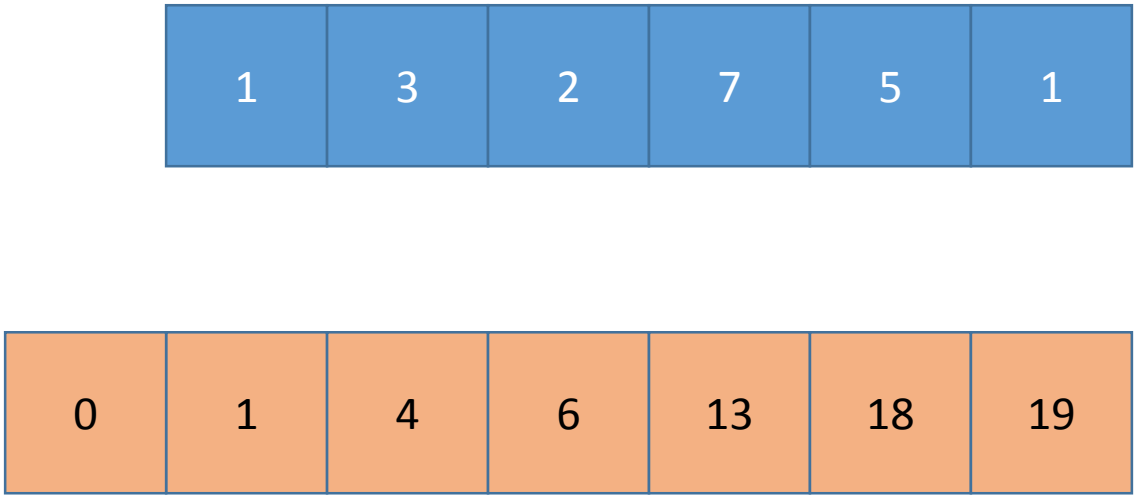
Esistono diverse soluzioni:

https://coding-gym.org/challenges/sherlock-and-array/

The Equilibrium Index



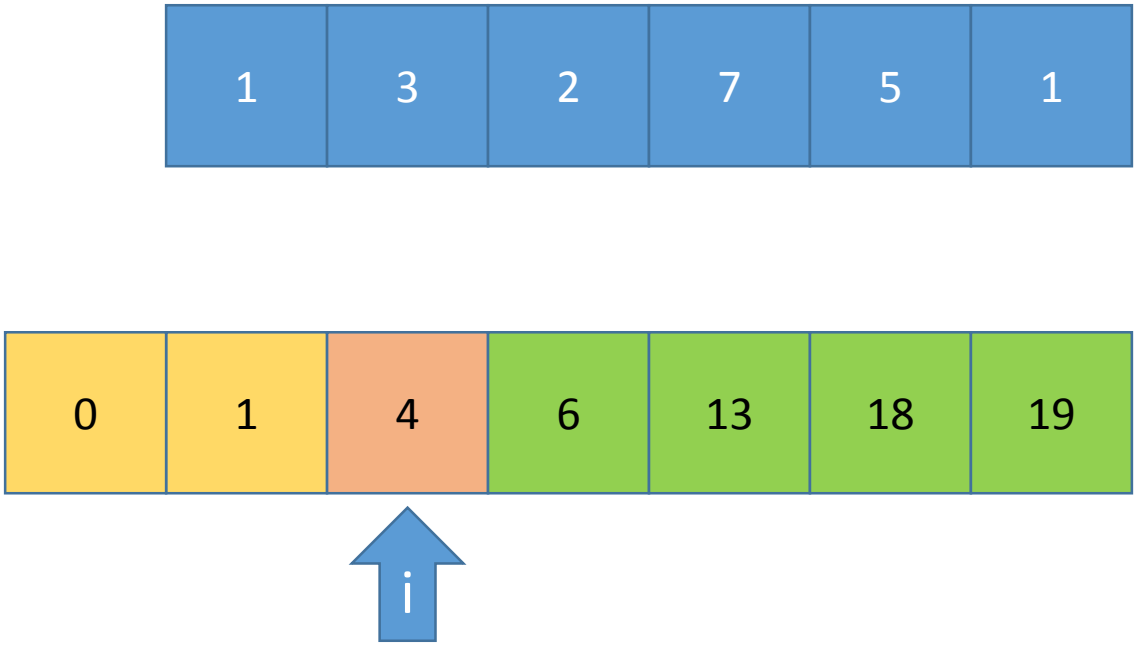
The Equilibrium Index



Invariante

`somma in[i, j] = psum[j] – psum[i-1]`

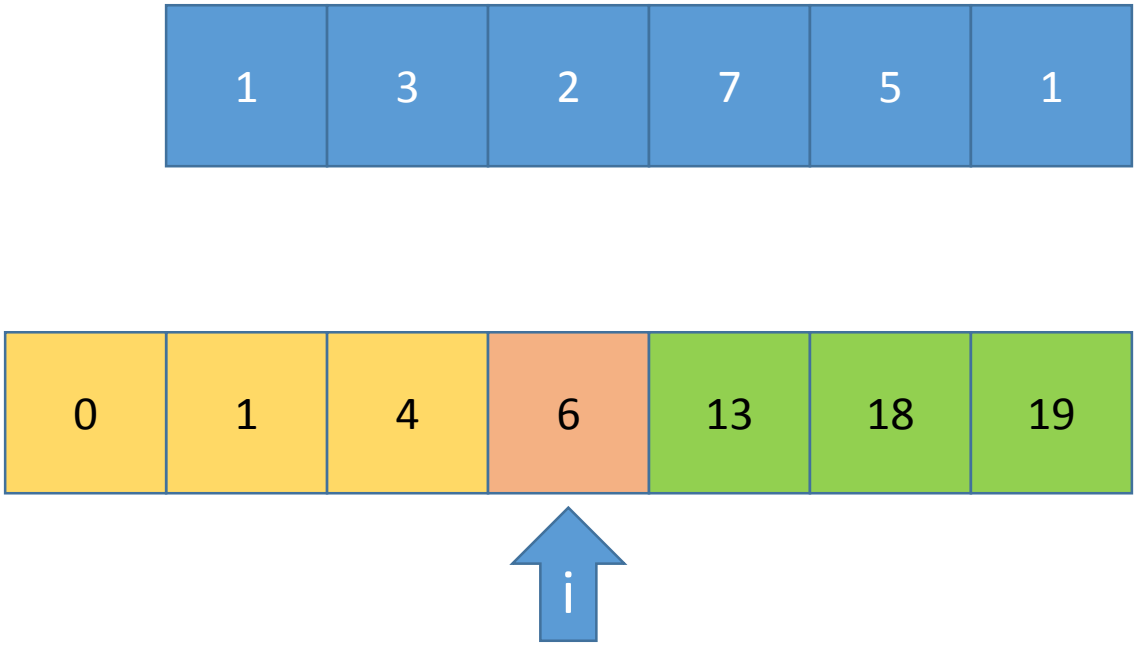
The Equilibrium Index



Invariante

```
left_sum = psum[i-1] = 1
right_sum = psum.last – psum[i] = 15
```

The Equilibrium Index



Invariante

```
left_sum = psum[i-1] = 4
right_sum = psum.last – psum[i] = 13
```

The Equilibrium Index

1	3	2	7	5	1

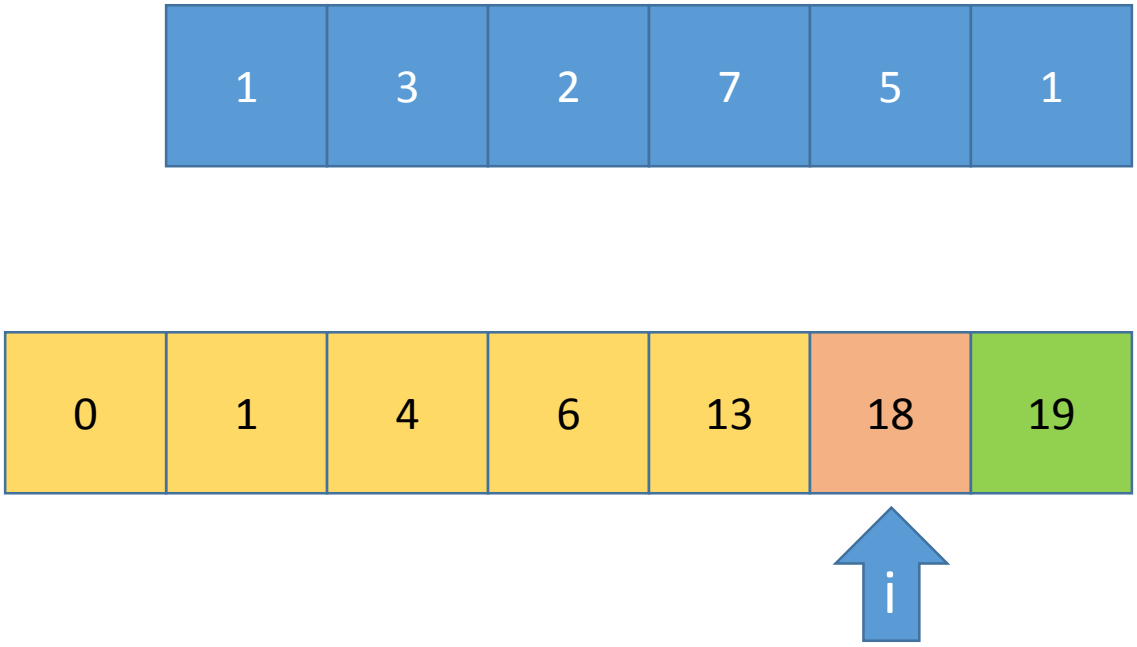
0	1	4	6	13	18	19

i

Invariante

```
left_sum = psum[i-1] = 6
right_sum = psum.last – psum[i] = 6
```

The Equilibrium Index



Invariante

```
left_sum = psum[i-1] = 13
right_sum = psum.last – psum[i] = 1
```

The Equilibrium Index

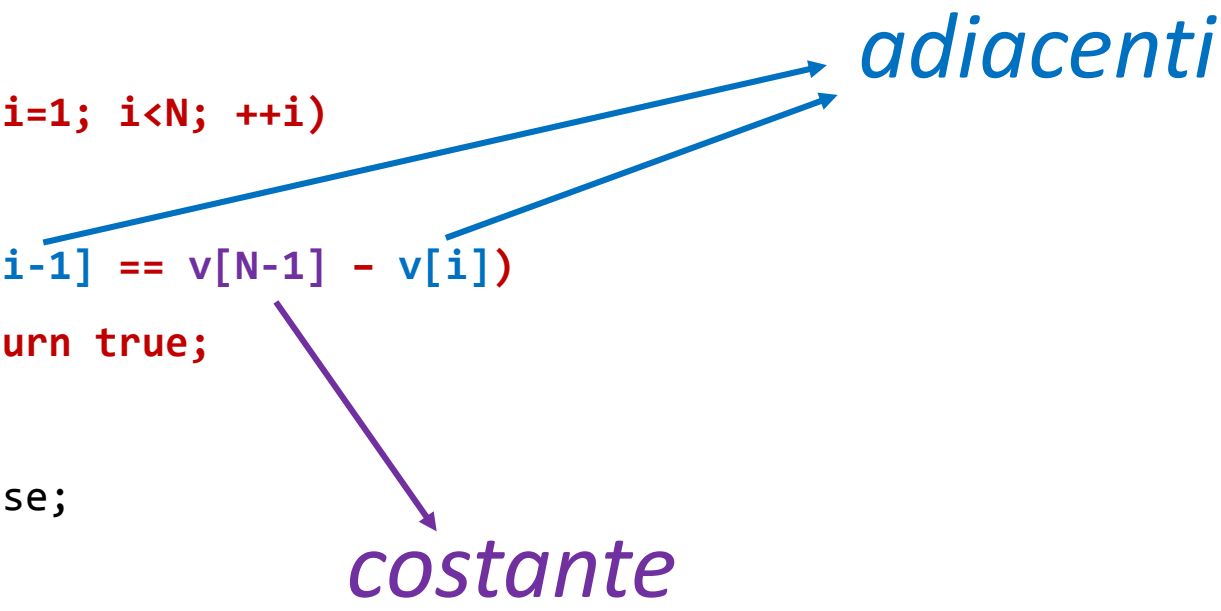
```
partial_sum(next(begin(v)), end(v), begin(v));


for (auto i=1; i<N; ++i)
{
    if (v[i-1] == v[N-1] – v[i])
       return true;
}
return false;
```

The Equilibrium Index

```
partial_sum(next(begin(v)), end(v), begin(v));

for (auto i=1; i<N; ++i)
{
    if (v[i-1] == v[N-1] - v[i])
       return true;
}
return false;
```

The Equilibrium Index

```
partial_sum(next(begin(v)), end(v), begin(v));


for (auto i=1; i<N; ++i)
{
    if (v[i-1] == v[N-1] - v[i])
       return true;
}
return false;
```



The Equilibrium Index

```
partial_sum(next(begin(v)), end(v), begin(v));

for (auto i=1; i<N; ++i)
{
    if (v[i-1] == v[N-1] - v[i])
        return true;
}
return false;
```

se la proprietà è verificata, esco

The Equilibrium Index

```
partial_sum(next(begin(v)), end(v), begin(v));
```

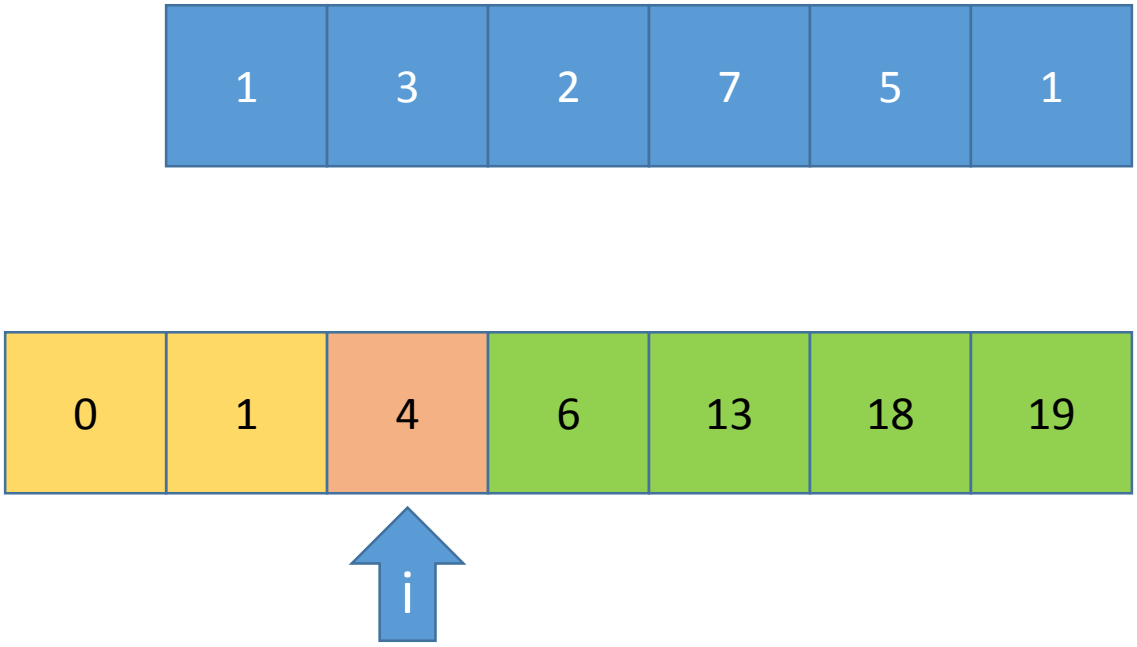
```
for (auto i=1; i<N; ++i)
{
    if (v[i-1] == v[N-1] - v[i])
        return true;
}
return false;
```

ricerca tra due adiacenti che soddisfano una proprietà

The Equilibrium Index

```
partial_sum(next(begin(v)), end(v), begin(v));

return adjacent_find(begin(v), end(v),
                     [last=v.back()](auto left, auto right) {
                                    return left == last - right;
                     }) != end(v);
```

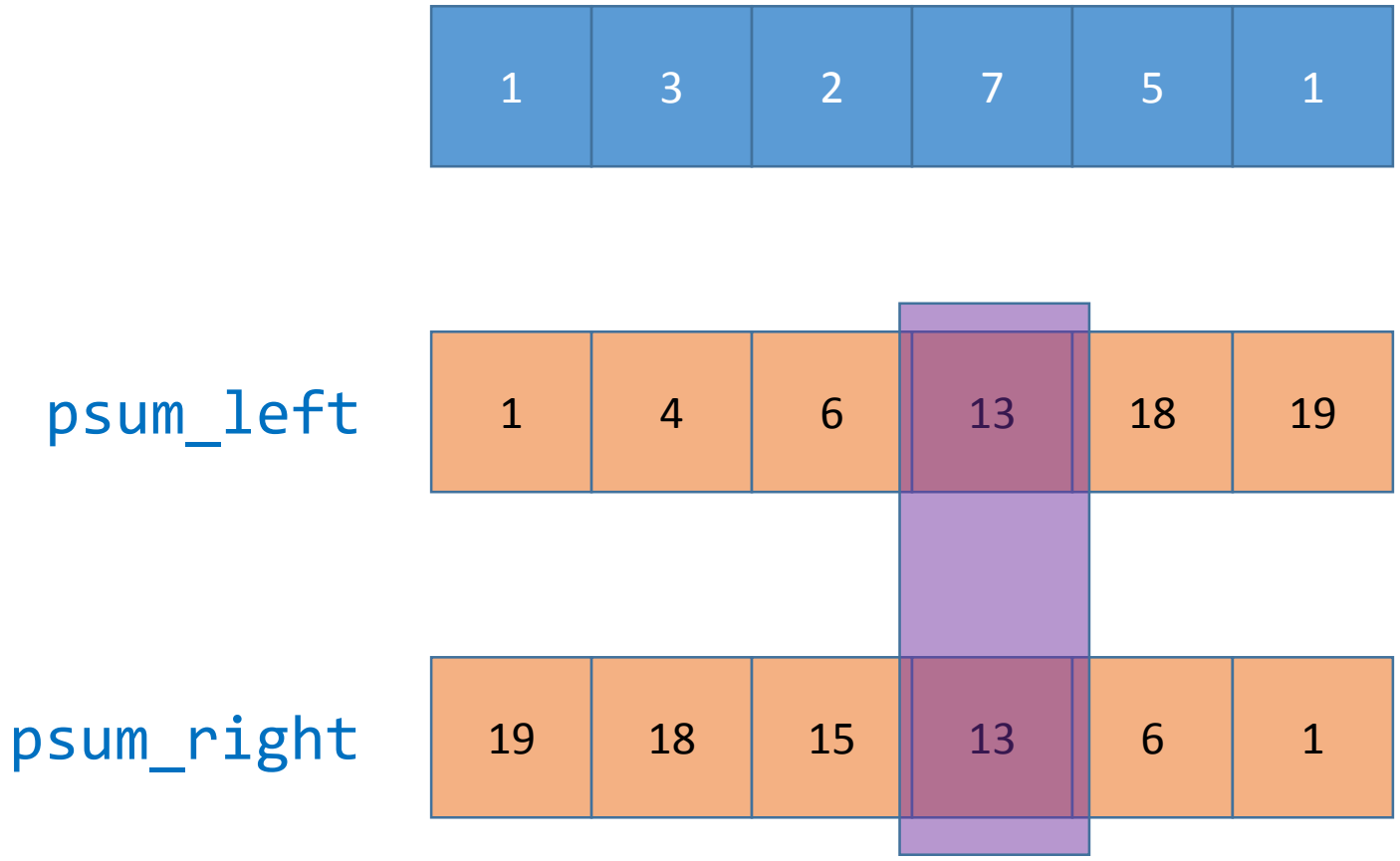
The Equilibrium Index: Variante



Invariante

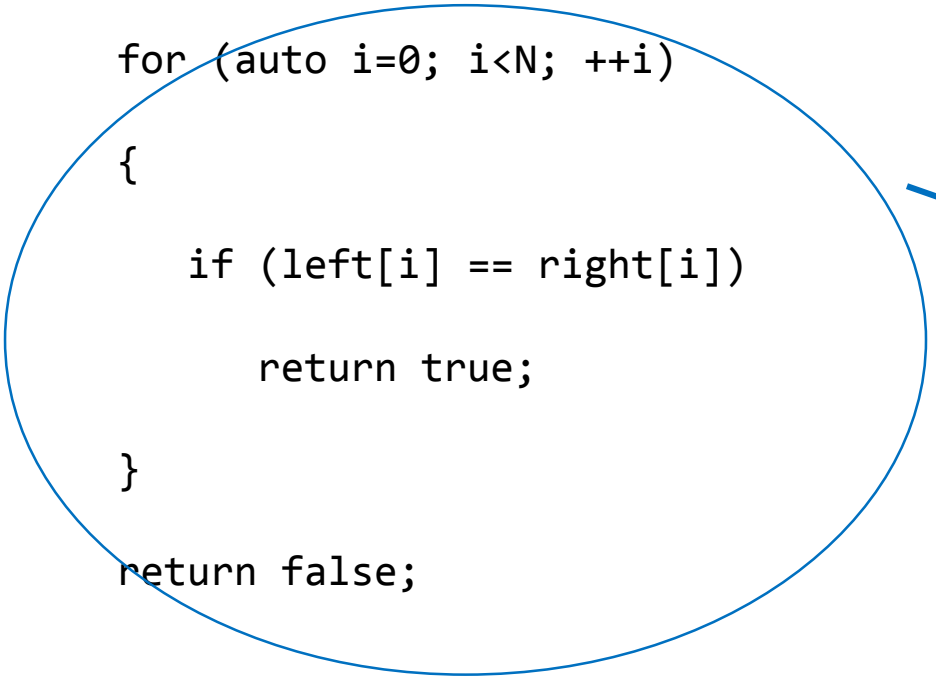
```
left_sum = psum[i-1] = 1
```

The Equilibrium Index: Variante



The Equilibrium Index: Variante

```
vector<int> left(N), right(N);

partial_sum(begin(v), end(v), begin(left));

partial_sum(rbegin(v), rend(v), rbegin(right));

for (auto i=0; i<N; ++i)

{

   if (left[i] == right[i])

      return true;

}

return false;
```



prima coppia che soddisfa una proprietà

The Equilibrium Index: Variante

```
vector<int> left(N), right(N);

partial_sum(begin(v), end(v), begin(left));

partial_sum(rbegin(v), rend(v), rbegin(right));

return mismatch(begin(left), end(left), begin(right),
                not_equal_to<>{}).first
                != end(left);
```

Prefix Sum solo con la «somma»?

Per casa:

The Crazy Broker

Potrete allenarvi sapendo già di dover applicare la prefix sum.

Come?!

Soluzioni e discussione:

https://coding-gym.org/challenges/the-crazy-broker

Allenarsi con i Pattern Funzionali

reduce([1,2,3,4], sum) = 10

map([1,2,3,4], square) = [1, 4, 9, 16]

filter([1,2,3,4], odd) = [1, 3]

zip([1,2,3], [A,B,C]) = [(1,A), (2,B), (3,C)]

Zip, Map, Reduce, Filter in C++

Map: `std::transform`

Reduce: `std::accumulate`

Filter: `std::copy_if`

Zip: …vediamo…

Esempio:
Massima differenza assoluta tra due array

bs=[6, 10, 13, 5, 1]

ac=[6, 9, 10, 3, 2]

Δ= [0, 1, 3, 2, 1]

Esempio:
Massima differenza assoluta tra due array

```
auto maxDelta = 0.0;

for (auto i=0; i<N; ++i)

{

    maxDelta = max(maxDelta, fabs(ac[i]-bs[i]));

}
```

Esempio:
Numero di caratteri adiacenti ripetuti

ABAAACCBDBBB

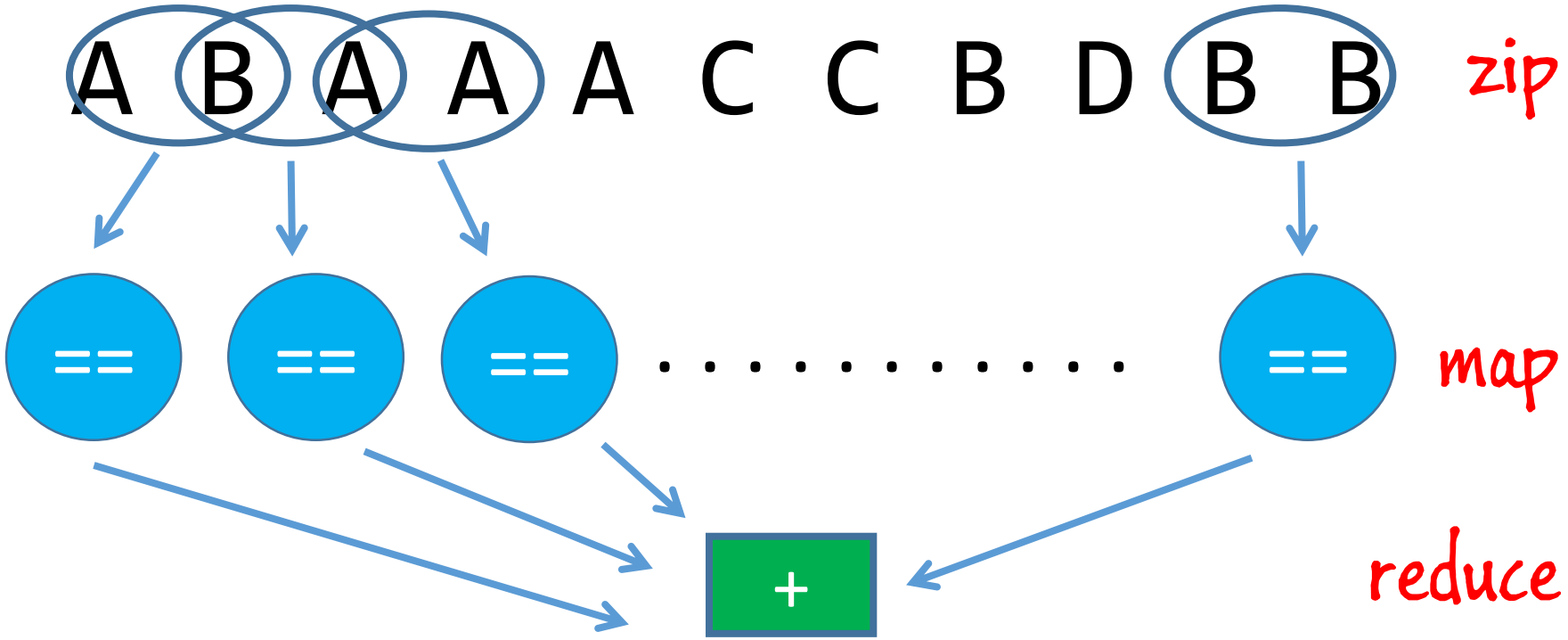
4

Esempio:
Numero di caratteri adiacenti ripetuti

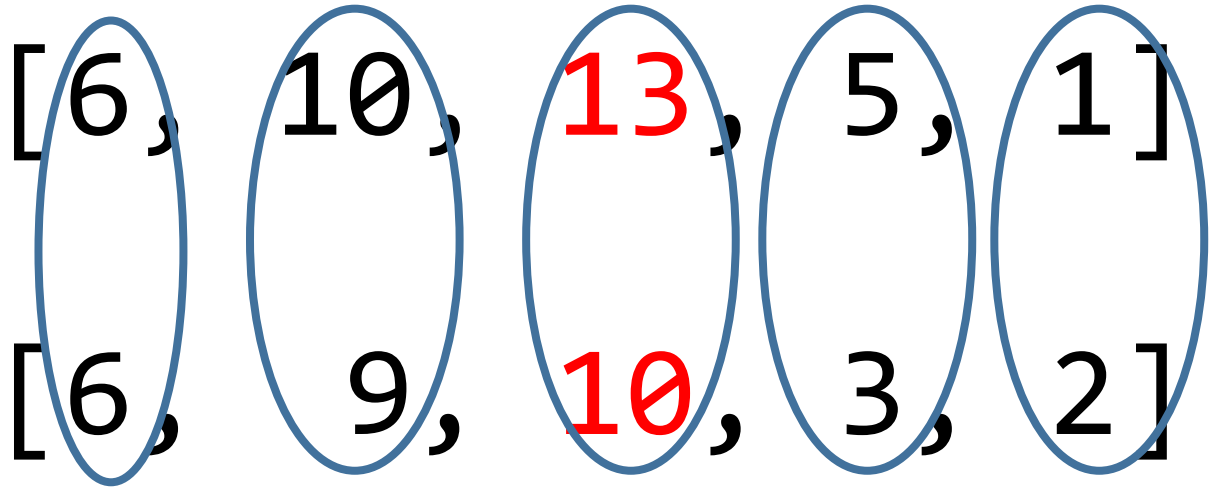
```
auto cnt = 0;

for (auto i=1; i<N; ++i)

{

    if (s[i]==s[i-1])

        cnt++;

}
```

Vedete il pattern?

Esempio:
Numero di caratteri adiacenti ripetuti



Esempio:
Massima differenza assoluta tra due array



Δ= [0, 1, 3, 2, 1]

Zip | Map | Reduce

Zip: "accoppia" gli elementi per posizione

Map: trasforma le coppie in valori intermedi

Reduce: accumula i valori intermedi

Zip | Map | Reduce

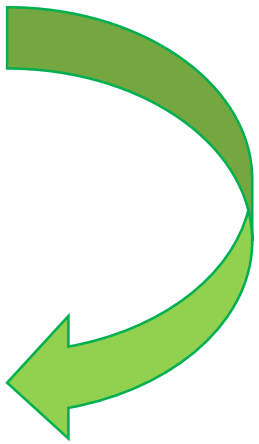
```
1   2   3   4
10 20 30 40

1*10 + 2*20 + 3*30 + 4*40
```

Zip | Map | Reduce in C++

`inner_product`
`transform_reduce (C++17)`

Esempio:
Massima differenza assoluta tra due array

```
auto maxD = 0.0;

for (auto i=0; i<N; ++i)

    maxD = max(maxD, fabs(ac[i]-bs[i]));
```



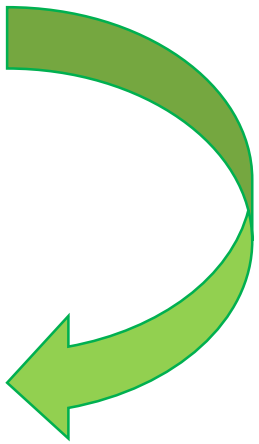
```
auto maxD = inner_product(begin(bs), end(bs),
                          begin(ac),
                          0,
                          [](auto a, auto b) { return max(a,b); },
                          [](auto bsV, auto acV) { return fabs(acV-bsV); }
            );
```

reduce

map

Esempio:
Numero di caratteri adiacenti ripetuti

```
for (auto i=1; i<N; ++i)
    if (s[i]==s[i-1])
        cnt++;
```



```
auto cnt = inner_product(next(begin(s)), end(s),
                         begin(s),
                         0,
                         plus<>{},
                         equal_to<>{});
```

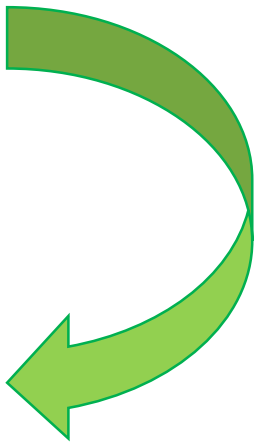
reduce

map

Esempio:
Numero di caratteri adiacenti ripetuti

C++17
Parallel STL

```
for (auto i=1; i<N; ++i)
    if (s[i]==s[i-1])
        cnt++;
```



```
auto cnt = transform_reduce(std::execution::par,
                            next(begin(s)), end(s),
                            begin(s),
                            0,
                            plus<>{},
                            equal_to<>{});
```

Un mio articolo su inner_product

A hidden gem: `inner_product`

https://marcoarena.wordpress.com/2017/11/14/a-hidden-gem-inner_product/

Allenatevi con *zip | map | reduce*

Per casa:

The Love Letter Mystery

Per i coraggiosi: provate ad usare anche i range del C++20!

Soluzioni e discussione:

https://coding-gym.org/challenges/the-love-letter-mystery/

Un accenno ai `ranges-v3` (C++20)

Problemi degli iteratori:

- Composizione difficile
- Hanno bisogno di una "fine"

https://ericniebler.github.io/range-v3/

Un accenno ai `ranges-v3` (C++20)

Un range è una **sequence abstraction:**

- Coppia di iteratori
- Iteratore + counter
- Iteratore + condizione
- …

Un accenno ai `ranges-v3` (C++20)

View adaptors:

- Riferimenti ai dati senza ownership
- Lazy
- Componibili (pipeline-able)

Un accenno ai `ranges-v3` (C++20)

Actions:

- Mutano una sequenza in-place
- Eager
- Componibili (pipeline-able)

Esempi:

```
auto rg = v
        | view::reverse
        | view::filter([](auto i){ return i%2; });

auto vec = std::move(v)
        | action::sort
        | action::unique
        | action::reverse;
```

Praticate e sperimentate a casa:

www.hackerrank.com/thinking-in-stl-patterns

Trovate soluzioni e discussioni sul sito di Coding Gym:
https://coding-gym.org/challenges

Situazioni di allenamento

- Da Zero
  - Possono emergere pattern al livello del pensiero
- Da codice funzionante
  - Possono emergere pattern dal codice oppure dall'idea che ti sei fatto dell'algoritmo
- Da pattern certo
  - Esercizi di «potenziamento»
  - Generano automatismi

Grazie!