Calcolare lo Span di un array di numeri

Altro esempio di come usare una pila come struttura dati ausiliaria per un algoritmo:

- **Dato un array *X*, lo span *S*[*i*] di *X*[*i*] è il massimo numero di elementi consecutivi *X*[*j*] immediatamente precedenti e tali che *X*[*j*] ≤ *X*[*i*]**
- **quindi il minimo span di X[i] è 1, perché lui stesso conta**
- **Gli Span hanno applicazioni in matematica finanziaria**

X	6	3	9	5	8
S	1	1	3	1	2

In altri termini S[i] = i - j
se X[j]>X[i] e X[j+1],…,X[i-1]≤X[i], visto che X[i]≤X[i]

Un Algoritmo per calcolare lo span

Algorithm *span1*(*X*)
  Input:array *X* **di** *n* **interi**
  Output: array *S* **di span di** *X*
  n ← **X.lenght**
  S ← nuovo array of *n* interi
  for *i* ← 0 **to** *n* − 1 **do**
    s ← 1
    while *s* ≤ *i* ∧ *X*[*i* − *s*] ≤ *X*[*i*] 1 + 2 + …+ (*n* − 1)
      s ← *s* + 1 1 + 2 + …+ (*n* − 1)
    S[*i*] ← *s*
  return *S*

Algoritmo *span1* ha complessità $O(n^2)$

Si può fare meglio?

X	6	3	9	5	8
S	1	1	3	1	2

S[i] = i - j
se X[j]>X[i] e X[j+1],…,X[i-1]≤X[i]

Osserviamo che
quando consideriamo X[i+1] può capitare che X[i+1] ≥ X[i] e allora è maggiore anche di X[i-1],…,X[j+1] e quindi basta confrontare X[i+1] con X[j] ed eventualmente ripetere lo stesso ragionamento con altri elementi precedenti

Se invece X[i+1] < X[i] allora S[i+1] = 1, perché non ci sono elementi precedenti minori o uguali a X[i+1].

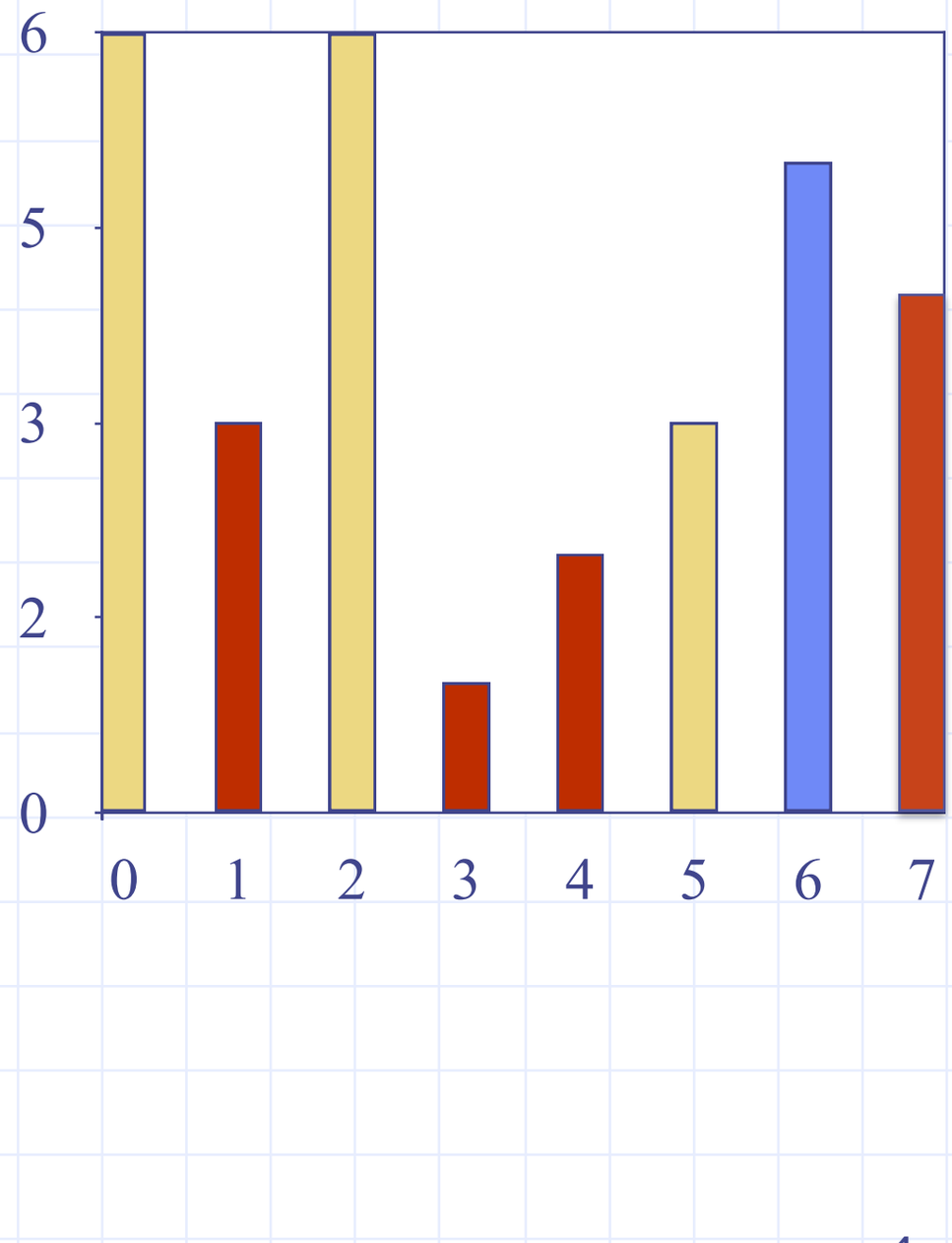
Si potrebbe allora conservare memoria solo degli elementi X[j] tali che X[j]>X[i] e X[j+1],…,X[i-1] ≤ X[i], per ogni i, in modo da limitare i confronti all'indietro a questi elementi X[j], che già maggiorano alcuni precedenti.

Usiamo una pila!

Guardando all'istogramma, potremmo fare in modo che nella pila si trovino gli indici degli elementi precedenti che "non ne nascondono altri", guardando all'indietro.

Per esempio nella figura, nel caso dell'elemento di indice 6, X[6], nella pila dobbiamo trovare 0,2 e 5, perché 1, 3 e 4 dovrebbero essere stati rimossi considerando rispettivamente l'elemento di indice 2 e quello di indice 5. Ora 5 deve essere rimosso dalla pila ma non 2.

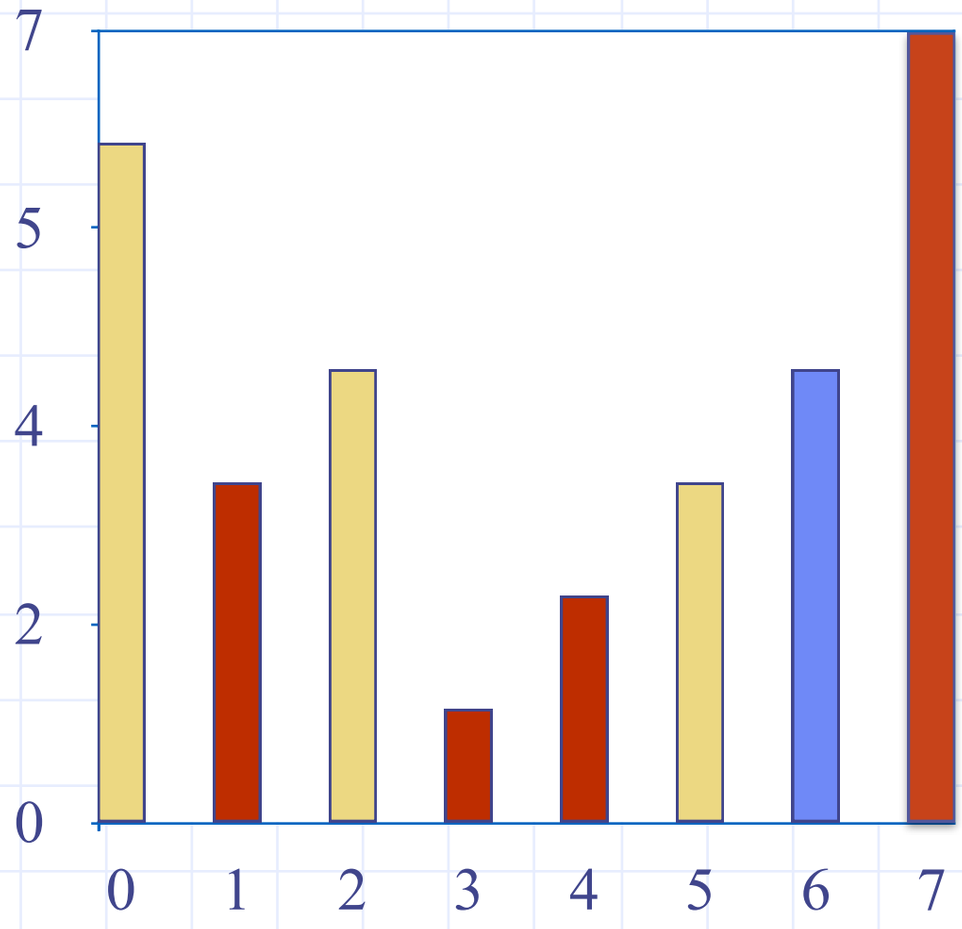
Allora S[6] = 6 - 2 = 4



Usiamo una pila!

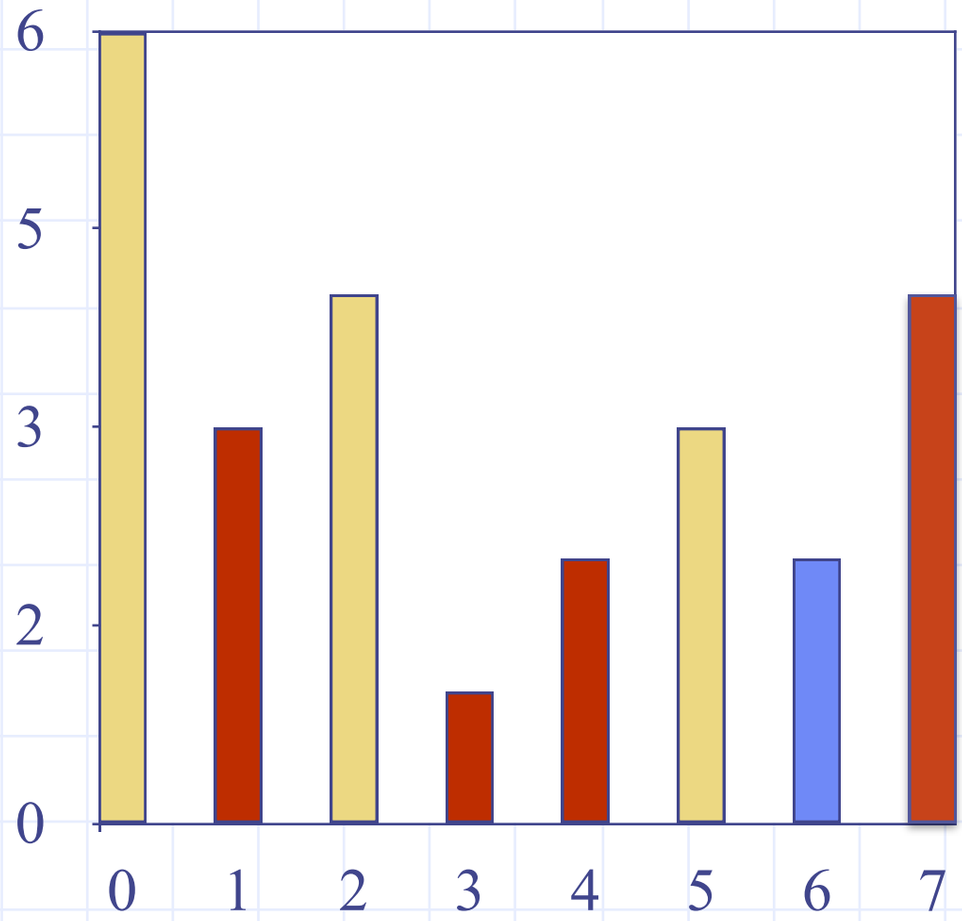
Ora nella figura, nel caso dell'elemento di indice 6, X[6], gli indici 5 e 2 (che sono stati impilati e non rimossi) possono essere rimossi dalla pila.

Allora S[6] = 6 - 0 = 6



Usiamo una pila!

Se X[6] < X[5] allora non si rimuove alcun indice dalla pila e S[6] = 1, perché non ci sono elementi più piccoli di X[6] e 6 viene impilato.



L'idea è quindi di leggere l'array da sinistra a destra, e quando si considera l'elemento di indice i, fare in modo che se j è il primo indice minore di i tale che X[j] > X[i] nella pila non ci siano quelli degli elementi X[k] tali che X[k] < X[i], per $j<k<i$ e $0 \leq j<i$.

L'algoritmo

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j]. Inoltre S[0],… S[i-1] contengono i corretti valori degli span.

Per mantenere l'invariante vero, quali operazioni si devono eseguire quando si considera l'elemento i-simo?

1. **si eliminano dalla pila gli indici j tale che X[i] ≥ X[j]**

Così facendo il primo confronto riguarda X[i-1], che è in cima alla pila se X[i] ≥ X[i-1], il prossimo confronto sarà tra X[k], che è maggiore di X[i-1], e X[i], perché X[i-1] ≥ X[i-2],…,X[k+1] e quindi per transitività anche X[i] è maggiore o uguale di questi elementi. Continuando con i confronti si otterrà in cima alla pila il primo (da destra) indice h tale che X[i] < X[h] oppure si sarà svuotata la pila

2. **se la pila è vuota si pone $S[i] = i+1$**
3. **altrimenti si pone $S[i] \leftarrow i - h$**
4. **si impila i**

Algoritmo lineare per span

```
Algorithm span2(X, n)
S ← nuovo array of n interi
    A ← nuova pila vuota
       for i ← 0 to n − 1 do
```

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j]. Inoltre S[0],…S[i-1] contengono i corretti valori degli span.

```
            while (¬A.isEmpty() ∧ X[A.top()] ≤ X[i] ) do
                    A.pop()
        if A.isEmpty()
            then
                    S[i] ← i + 1
            else
                    S[i] ← i − A.top()
        A.push(i)
    return S
```


Correttezza algoritmo per span

```
Algorithm span2(X, n)
S ← nuovo array of n interi
    A ← nuova pila vuota
        for i ← 0 to n − 1 do
```

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j].Inoltre S[0],…S[i-1] contengono i corretti valori degli span.

```
            while (¬A.isEmpty() ∧ X[A.top()] ≤ X[i] ) do
                    A.pop()
        if A.isEmpty()
            then
                S[i] ← i + 1
            else
                S[i] ← i − A.top()
        A.push(i)
    return S
```

inizializzazione : l'invariante è vero: la pila è vuota e nessun valore di S è stato ancora calcolato .

Correttezza algoritmo per span

```
Algorithm span2(X, n)
S ← nuovo array of n interi
    A ← nuova pila vuota
        for i ← 0 to n − 1 do
```

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j].Inoltre S[0],…S[i-1] contengono i corretti valori degli span.

```
            while (¬A.isEmpty() ∧ X[A.top()] ≤ X[i] ) do
                    A.pop()
        if A.isEmpty()
            then
                S[i] ← i + 1
            else
                S[i] ← i − A.top()
        A.push(i)
    return S
```

Conservazione : sia l'invariante vero, e consideriamo l'indice i.

Eseguiamo il codice e verifichiamo se l'invariante resta vero.

Se la pila è vuota vuol dire che siamo alla prima esecuzione del ciclo, perché l'ultima operazione dello stesso impila l'indice corrente. In questo caso si pone S[0]=1, correttamente e si impila 0.

Supponiamo che la pila non sia vuota, allora X[i] viene confrontato con X[A.top()] = X[i-1] e se X[i-1] > X[i] allora si pone S[i] = i-(i-1)=1 e si impila i, per cui l'invariante è ancora vero perché era vero per X[i-1] per ipotesi induttiva e non ci sono elementi (più piccoli di X[i]) prima di X[i-1].

Se invece X[i-1] ≤ X[i], X[i-1] viene eliminato dalla pila e il prossimo confronto è tra X[i] e X[k], dove X[k] > X[i-1] mentre tutti gli elementi di indice compreso tra k+1 e i-2 sono più piccoli di X[i-1] e dunque anche di X[i]. Se X[k] < X[i], X[k] viene eliminato della pila e il ragionamento si ripete, fino a trovare il primo elemento di indice minore di i più grande di X[i]. Se la pila non si è svuotata questo indice ora è in cima alla pila, quindi ponendo S[i] = i - A.top() si calcola correttamente S.

Il calcolo è corretto anche se la pila si svuota, perché in tal caso tutti i precedenti sono più piccoli di X[i] e quindi il numero di questi è proprio i+1. Infine i viene impilato e incrementato, ristabilendo l' invariante.

Correttezza algoritmo per span

```
Algorithm span2(X, n)
S ← nuovo array of n interi
    A ← nuova pila vuota
        for i ← 0 to n − 1 do
```

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j].Inoltre S[0],…S[i-1] contengono i corretti valori degli span.

```
            while (¬A.isEmpty() ∧ X[A.top()] ≤ X[i] ) do
                                A.pop()
        if A.isEmpty()
                then
                        S[i] ← i + 1
            else
                        S[i] ← i − A.top()
    A.push(i)
return S
```

Terminazione:All'uscita dal ciclo i = n e X[n-1] è in cima alla pila che contiene tra gli indici minori di n-1, quelli degli elementi X[j] tranne gli indici degli immediati precedenti più piccoli o uguali a X[j], l'indice j può essere n-2 al più se X[n-2] >X[n-1].

Poiché il valore di S è stato calcolato correttamente in ogni passo, l'algoritmo è corretto.

É lineare ?

```
Algorithm span2(X, n)
S ← nuovo array of n interi
    A ← nuova pila vuota
        for i ← 0 to n − 1 do
```

invariante: la pila è vuota o ha X[i-1] in cima e contiene, tra gli indici minori di i-1, quelli degli elementi X[j], per ogni j<i-1 tranne gli indici degli immediati precedenti più piccoli o uguali a X[j].Inoltre S[0],…S[i-1] contengono i corretti valori degli span.

```
            while (¬A.isEmpty() ∧ X[A.top()] ≤ X[i] )
do
                        A.pop()

        if A.isEmpty()
              then
                    S[i] ← i + 1
             else
                    S[i] ← i − A.top()
      A.push(i)
   return S
```

Ogni indice dell'array è impilato una volta sola ed è tolto dalla pila al più una volta.

le istruzioni all'interno del ciclo while quindi complessivamente sono eseguite al più n volte.

L'algoritmo *span2* ha complessità di tempo asintotica $O(n)$, ma anche complessità di spazio asintotica O(n).

