

Problemi Computazionali

Un problema computazionale è una collezione di

- **Istanze**(input)
 - **Risposte**(output)
 - **Criterio** per valutare la correttezza delle risposte
-

MCD

- **Input**: coppia di interi non negativi a e b non entrambi nulli
 - **Uscita**: un intero che soddisfa il seguente **criterio**:
 1. c divide sia a che b
 2. non esiste $d > c$ che divide sia a che b
-

Un problema è una **relazione binaria**

$$R = \{(input, output) | istanza, risposta \text{ soddisfano } \dots\}$$

Dominio della relazione:

$$dom(R) = \{ \exists r. (i, r) \in R \}$$

R è **univoca** se ogni istanza ammette una sola risposta

Esempi di Problemi Computazionali

- **Moltiplicazione** fra due interi
- **Fattorizzazione**
- **Sorting** (ordinamento)
- **Shortest Path** (percorso ottimo in un grafo)

Terminologia

- **Procedura**: sequenza finita di operazioni meccanicamente eseguibili che produce un'uscita a parte da certi ingressi

- **Algoritmo:** procedura che termina per ogni ingresso ammissibile

Peak Finding

- **Input:** un vettore $A[0..n-1]$ di interi positivi
- **Output:** un intero $0 \leq p < n$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ dove $A[-1]=A[n]=-\infty$
Ovvero nella posizione p si trova un picco

Left Peak Finding

PEAK-FIND-LEFT(A, n) $\triangleright n \geq 1$

$p \leftarrow 0$

$k \leftarrow 1$

while $k \leq n-1 \wedge A[p] < A[k]$ do

$p \leftarrow k$

$k \leftarrow k+1$

end while

return p

ritorna il picco più a sinistra di $A[0..n-1]$

Best Case: $p=0$ è un picco

Worst Case: $p=n-1$ (il vettore è in ordine crescente)

Max Peak Finding

PEAK-FIND-MAX(A, n) $\triangleright n \geq 1$

$p \leftarrow 0$

for $k \leftarrow 1$ to $n-1$ do

 if $A[p] < A[k]$ then

$p \leftarrow k$

 end if

end for

return p

ritorna il picco più alto

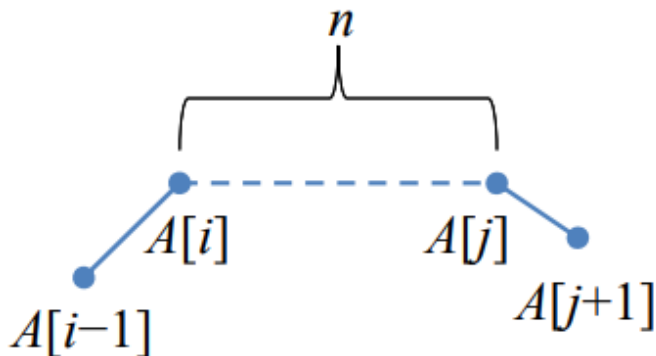
In ogni caso, si effettuano $n-1$ confronti \rightarrow lo stesso del worst case di left

peak finding.

Dunque è "migliore" questo algoritmo.

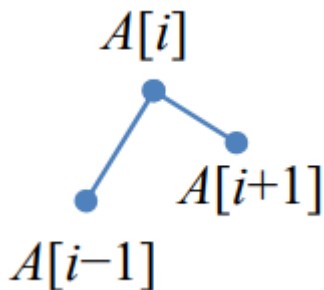
Teoremi del Peak Finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.



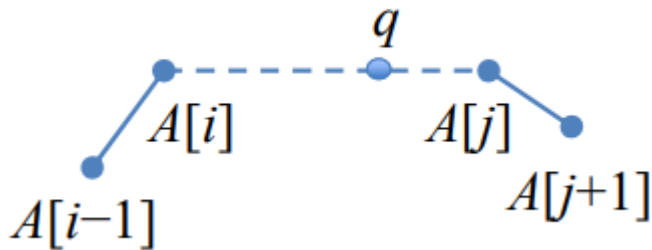
Ovvero, preso un intervallo qualunque, se $estremo_sinistro - 1 < estremo_sinistro$ e $estremo_destra > estremo_destra + 1$ allora esiste un picco in tale intervallo.

Nel caso in cui $i = j$ ovvero $n = 1$, il caso è banale. La posizione i stessa è un picco



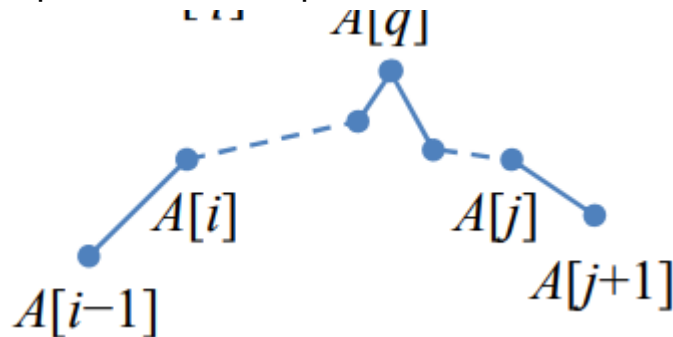
Non è altrettanto banale nel caso in cui $i < j$ ovvero $n > 1$

Scelgo una posizione q tale che $i \leq q \leq j$

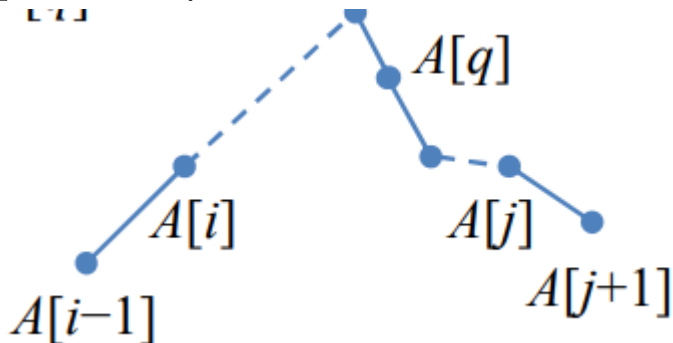


Abbiamo due casi:

- q stesso può essere un picco



- q non è un picco:



Dunque ripeterò il procedimento iniziale e mi scelgo di nuovo un intervallo

Posso :

- prendere come estremo sinistro $A[i]$ ed estremo destro $A[q - 1]$
- prendere come estremo sinistro $A[q + 1]$ ed estremo destro $A[j]$

Ricapitolando

- Se $n = 1$ il picco si trova in $i = j = p$
- Se $n < 1$ scelgo un q tale che $i \leq q \leq j$:
 - se q è picco mi fermo

- se q non è picco reitro sul segmento $i..q-1$ oppure sul segmento $q+1..j$.

Ciò garantisce di trovare un picco

Peak Finding Divide et Impera

Una scelta vantaggiosa di q potrebbe essere la posizione centrale:

```
PEAK-DI( $A, i, j$ )     $\triangleright i \leq j$   
 $p \leftarrow (i + j)/2$   
if  $A[p - 1] \leq A[p] \geq A[p + 1]$  then  
    return  $p$   
else     $\triangleright A[p - 1] > A[p] \vee A[p] < A[p + 1]$   
    if  $A[p - 1] > A[p]$  then  
        return PEAK-DI( $A, i, p - 1$ )  
    else  
        return PEAK-DI( $A, p + 1, j$ )  
    end if  
end if
```

```
PEAK-FIND-DI( $A, n$ )     $\triangleright n \geq 1$   
return PEAK-DI( $A, 0, n - 1$ )
```

Se $n = 1$, allora servirà soltanto 1 confronto per trovare il picco: il picco è, ovviamente, l'unico elemento presente.

Tuttavia se $n > 1$, ad ogni confronto, divido il range (che all'inizio sarà $[0, n - 1]$) in due.

Dunque:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{se } n > 1 \end{cases}$$

Se ad esempio $n = 8$:

$$\begin{aligned} T(8) &= T(4) + 1 \\ &= (T(2) + 1) + 1 \\ &= ((T(1) + 1) + 1) + 1 \\ &= 1 + 1 + 1 + 1 \\ &= 3 + 1 \\ &= \log_2 8 + 1 \end{aligned}$$

Nel caso peggiore quindi, dovremmo effettuare $\log_2 n + 1$ confronti dove n è il numero di elemento dell'array

$$\begin{aligned}T(n) &= T\left(\frac{n}{2}\right) + 1 \\&= T\left(\frac{n}{4}\right) + 1 + 1 \\&= T\left(\frac{n}{8}\right) + 1 + 1 + 1 \\&= T\left(\frac{n}{2^3}\right) + 3 \\&= T\left(\frac{n}{2^k}\right) + k \quad \text{per } 1 \leq k \leq \log_2 n \\&= T(1) + \log_2 n \\&= 1 + \log_2 n\end{aligned}$$

Analisi di Peak Divide et Impera

Left-Peak-Finding (nel caso peggiore) e **Max-Peak-Finding** effettuano $n - 1$ confronti mentre **Peak-Finding DI** (Divide et Impera) circa $\log_2 n$ confronti.

Con un vettore di 1000 elementi servono circa 10 confronti invece di 1000