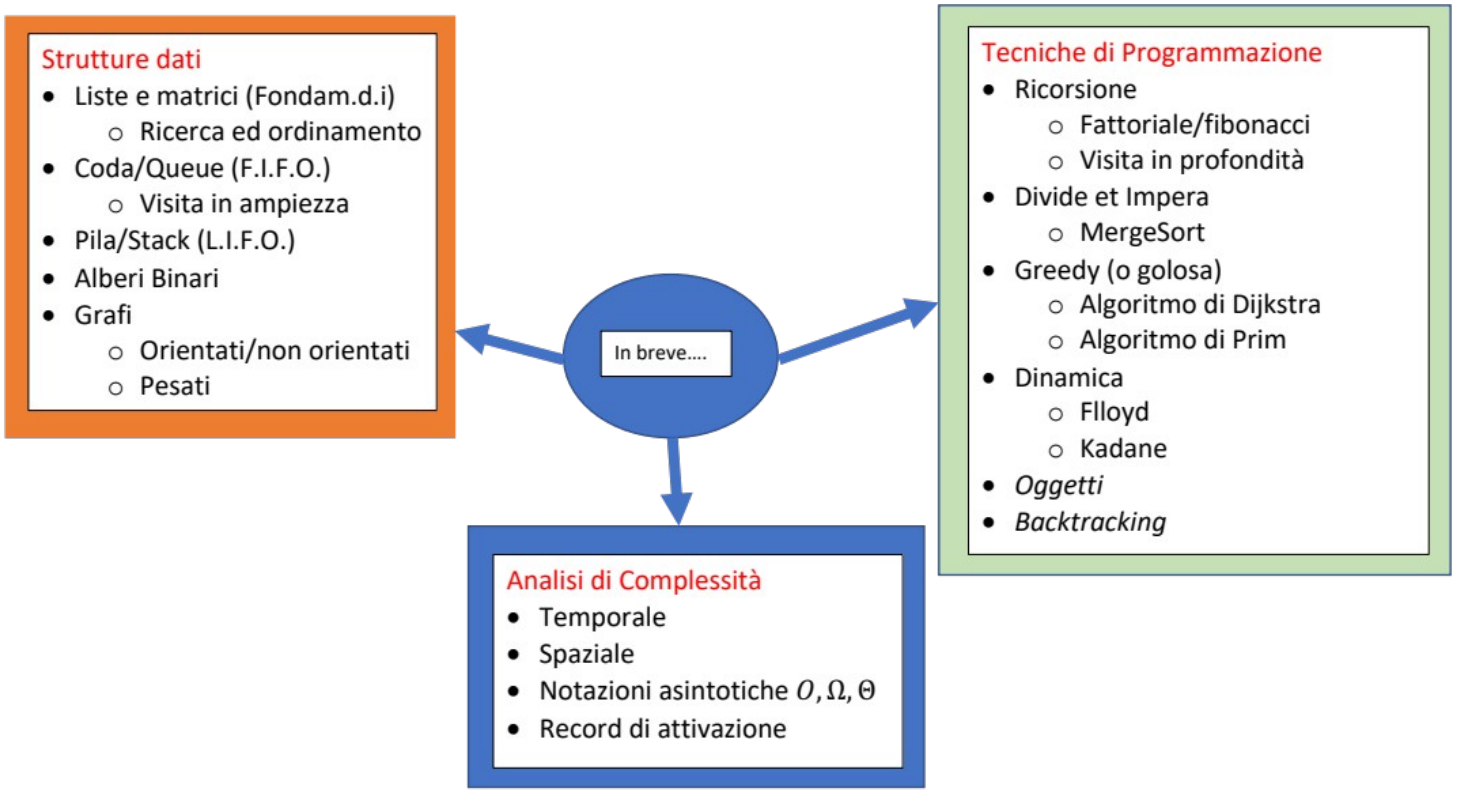
Medicina e Tecnologie TD
Lezione 8

Pierangelo Veltri
pierangelo.veltri@unical.it

Overview


Strutture dati
Liste e matrici (Fondam.d.i)
Ricerca ed ordinamento
Coda/Queue (F.I.F.O.)
Visita in ampiezza
Pila/Stack (L.I.F.O.)
Alberi Binari
Grafi
Orientati/non orientati
Pesati
In breve....
Tecniche di Programmazione
Ricorsione
Fattoriale/fibonacci
Visita in profondità
Divide et Impera
MergeSort
Greedy (o golosa)
Algoritmo di Dijkstra
Algoritmo di Prim
Dinamica
Flloyd
Kadane
Oggetti
Backtracking
Analisi di Complessità
Temporale
Spaziale
Notazioni asintotiche O, Ω, Θ
Record di attivazione

Problemi ed algoritmi su grafi

- Visita dei grafi
  - BFV(Breadth-first visit), DFV (deep-first-visit)
- Cammino minimo (a partire da un nodo) – Single-source shortest-path:
  - algoritmo di **Djakstra**
  - Bellman-ford algoritmo (anche per pesi negativi)
- Cammino minimo per ogni coppia di nodi
  - **Floyd-Warshall**
- Minumum Spanning Tree
  - Algo **Kruskal**
- Chiusura transitiva
  - Algoritmo di **Floyd-Warshall**
- Ordinamenti topologici (esempio per scheduling: se non ha terminato un evento non può partire il successivo)

Tecniche algoritmiche: la Programmazione Dinamica

- (riferimento capitolo 10 Demetrescu et al)
- Tecnica bottom-up:
- Identifica dei sottoproblemi del problema originario, procedendo logicamente dai problemi più piccoli verso quelli più grandi
- Utilizza una tabella per memorizzare le soluzioni dei sottoproblemi incontrati: quando si incontra lo stesso sottoproblema, sarà sufficiente esaminare la tabella
- Si usa quando i sottoproblemi non sono indipendenti, e lo stesso sottoproblema può apparire più volte
- Esempio: numeri di Fibonacci

Programmazione Dinamica

- La programmazione dinamica è un approccio che si può utilizzare in diversi casi per risolvere efficientemente un *problema di ottimizzazione*
  - Diverse soluzioni possibili per un problema
  - Ogni soluzione ha un costo, il problema è trovare *una* soluzione con il costo minimo o massimo
- Come divide-et-impera, la programmazione dinamica risolve un problema combinando le soluzioni di sottoproblemi
- Diversamente da divide-et-impera, la programmazione dinamica si applica anche quando i sottoproblemi non sono indipendenti
- La programmazione dinamica risolve i problemi in comune una sola volta, divide-et-impera più volte

Serie di Fibonacci: l’isola dei conigli

Leonardo da Pisa (anche noto come Fibonacci) si interessò di molte cose, tra cui il seguente problema di dinamica delle popolazioni:

> Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

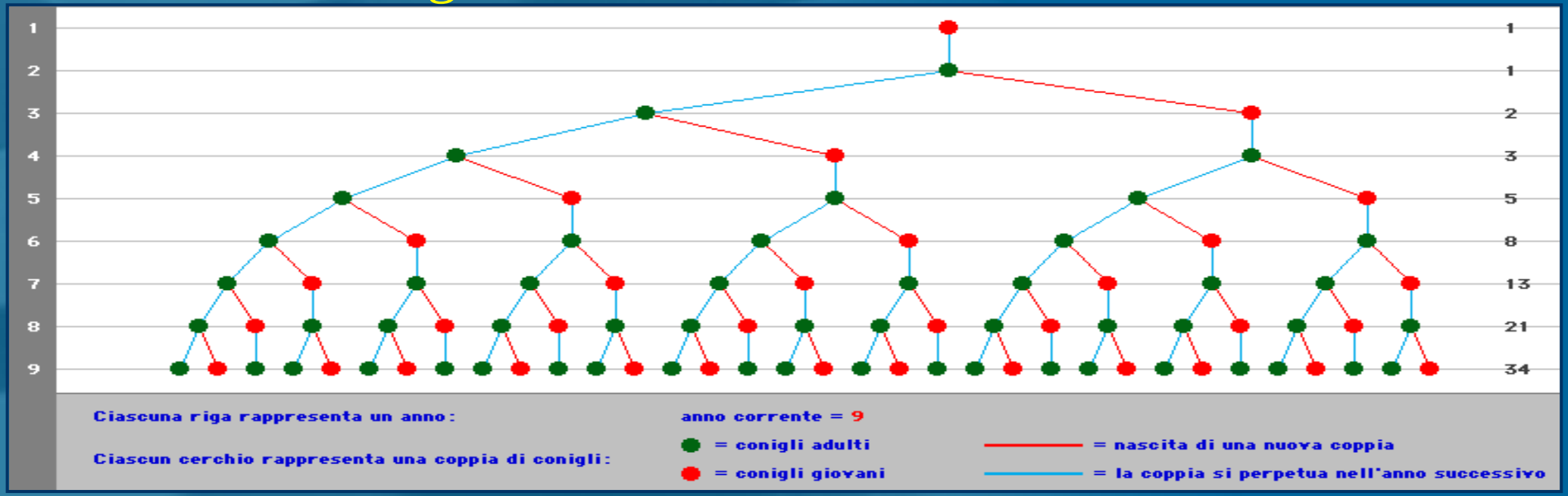
In particolare, partendo da una coppia di conigli in un’isola deserta, quante coppie si avrebbero nell’anno n?

Le regole di riproduzione

- Una coppia di conigli genera due coniglietti ogni anno
- I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita
- I conigli sono immortali

L’albero dei conigli

La riproduzione dei conigli può essere descritta in un albero come segue:



La regola di espansione

- Nell'anno n, ci sono tutte le coppie dell'anno precedente, e una nuova coppia di conigli per ogni coppia presente due anni prima
- Indicando con F_n il numero di coppie dell'anno n, abbiamo la seguente relazione di ricorrenza:

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{se } n \geq 3 \\ 1 & \text{se } n=1,2 \end{cases}$$

Algoritmo `fibonacci3`

- L’algoritmo si puo’ ottimizzare evitando di ricalcolare ripetutamente la soluzione dello stesso sottoproblema.
- Se memorizziamo il risultato parziale in un array le soluzioni dei sottoproblemi

```
algoritmo fibonacci3(intero n) → intero
    sia Fib un array di n interi
    Fib[1] ← Fib[2] ← 1
    for i = 3 to n do
        Fib[i] ← Fib[i-1] + Fib[i-2]
    return Fib[n]
```

Fibonacci: osservazioni

- Utilizza un array di dimensione n+1
- Memorizza le soluzioni dei sottoproblemi incontrati
- Quando incontro lo stesso problema sara’ sufficiente esaminare un elemento del vettore della programmazione dinamica
- Il vettore (o tabella) viene programmata dinamicamente

F[i-2] F[i-1] F[i]

Fibonacci: osservazioni:

- Identifichiamo innanzitutto dei sottoproblemi del problema originario: per numero di Fibonacci per 0<=i<=n il sottoproblema i-esimo consiste nel calcolo dell'i-esimo numero di Fibonacci.
- L'array momorizza le soluzioni dei sottoproblemi
- Si definiscono i valori iniziali dei sottopreoblemi (nel caso F[0] = 0; F[1]=1)
- Al generico passo i-esimo i>=2 si avanza sull 'array calcolando il valore in base al valore degli elementi precedentemente calcolati
- Restituiamo il valore di F[n]

Fibonacci in Python

fibonacci_numbers = [0, 1]

x – input

- If x >2
- for i in range(2,x):

  fibonacci_numbers.append(fibonacci_numbers[i-1]+fibonacci_numbers[i-2])

Sottovettore di valore massimo

Sottostruttura ottima

- Il problema esibisce una sottostruttura ottima quando una soluzione ottima del problema contiene al suo interno soluzioni ottime di sottoproblemi
- Questo è un segno che l'approccio della programmazione dinamica *può* essere impiegato
  - Ma si potrebbero impiegare anche divide-et-impera e approccio greedy (tecnica golosa, prossime lezioni!)
- La sottostruttura ottima varia da problema a problema in termini
  - del numero di sottoproblemi presenti nella soluzione ottima del problema originario (1 nel nostro esempio)
  - del numero di scelte che si hanno nel determinare quale/i sottoproblema/i è presente nella soluzione ottima (2 nel nostro esempio)

Sottostruttura ottima

- Il tempo di esecuzione di un algoritmo di programmazione dinamica è tipicamente il prodotto di due fattori:
  - Numero complessivo di sottoproblemi
  - Numero di sottoproblemi tra cui scegliere per ottenere la soluzione al problema originario
- La programmazione dinamica utilizza un approccio bottom-up
  - Gli algoritmi greedy un approccio top-down

Sottoproblemi sovrapposti

- È conveniente utilizzare la programmazione dinamica quando un algoritmo ricorsivo invocherebbe più volte uno stesso sottoproblema
  - Se ciò non accade, l'approccio divide-et-impera è adeguato
- La programmazione dinamica risolve ogni sottoproblema una sola volta e utilizza il valore calcolato quando si ripresenta lo stesso sottoproblema nell'approccio bottom-up

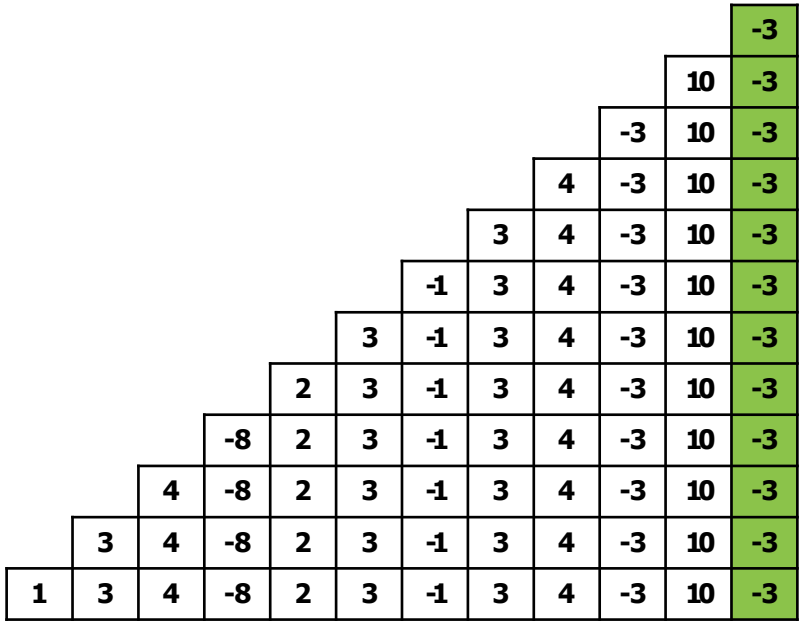
Sottovettore di somma massimale

1	3	4	-8	2	3	-1	3	4	-3	10	-3	2

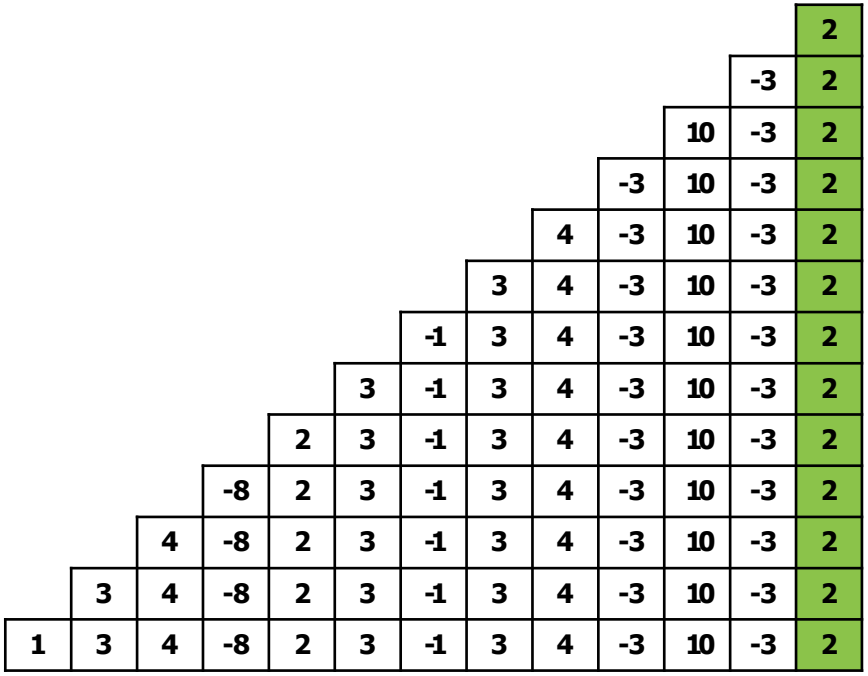
- Proviamo ad adottare un approccio *brute force*
- Calcoliamo la somma di ogni possibile sottovettore che termina in A[i]
- Dopo, calcoliamo la soma di ogni possibile sottovettore che termina in A[i+1]
- Il valore massimo tra tutte le somme identifica il nostro sottovettore di interesse

Sottovettore di somma massimale

1	3	4	-8	2	3	-1	3	4	-3	10	-3	2



1	3	4	-8	2	3	-1	3	4	-3	10	-3	2

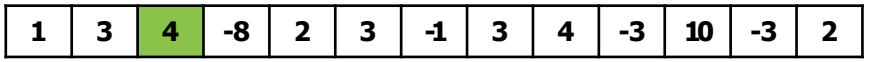


Implementiamolo in Python…(anche se inefficiente!)

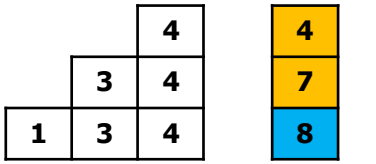
```
def somma_vett(v):
    return 0 if len(v)==0 else v[0]+somma_vett(v[1:])

def sotto_vett_max_cubic(v):
    n=len(v)
    somma_min=v[n-1]
    i_min=n-1
    j_min=n

    for i in range(n-1,-1,-1):
        for j in range(i-1,-1,-1):
            somma_tmp=somma_vett(v[j:i+1]
            if(somma_tmp)>somma_min):
                somma_min=somma_tmp
                i_min=i
                j_min=j
                print(v[j_min:i_min+1])

    return v[j_min:i_min+1]

print(sotto_vett_max_cubic([1,3,4,-8,2,3,-1,3,4,-3,10,-3,2]))
```

Sottovettore di somma massimale

- Concentriamoci sul terzo elemento del vettore



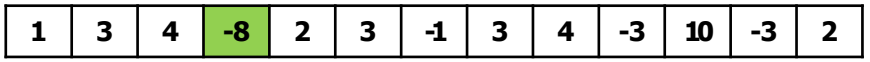
- Il “massimo locale” è 8, corrispondente al sottovettore dall'indice 0 all'indice 2



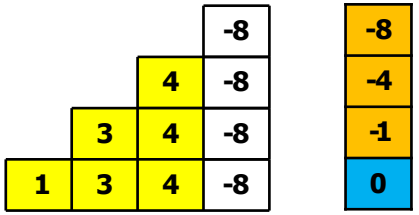
somme

Sottovettore di somma massimale

- Concentriamoci sul quarto elemento del vettore



- La parte gialla corrisponde all'insieme di vettori considerato nel caso precedente
- Se conosciamo già quelle somme, non è necessario ricalcolarle
- Se ci confrontiamo con il massimo precedente e troviamo uno zero (o un numero negativo), comincia una nuova fetta di sottovettore



somme

Programmazione dinamica

Programmazione dinamica

- Sia *maxHere[i]* il valore del sottovettore di somma massima che termina in posizione *A[i]*

$$\text{maxHere}[i] = \begin{cases} 0 & i < 0 \\ \max(\text{maxHere}[i-1] + A[i], 0) & i \geq 0 \end{cases}$$

- Viene tenuta traccia di quanto calcolato fino ad un certo punto di esecuzione dell'algoritmo

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere													
maxSoFar													
last													
start													
end													

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere													
maxSoFar													
last													
start													
end													

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0     # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	0												
maxSoFar	0												
last	0												
start	0												
end	0												

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =0

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	0												
maxSoFar	0												
last	0												
start	0												
end	0												

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =0

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1												
maxSoFar	0												
last	0												
start	0												
end	0												

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =0

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1												
maxSoFar	1												
last	0												
start	0												
end	0												

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =1

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	1											
maxSoFar	1	1											
last	0	0											
start	0	0											
end	0	0											

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =1

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4											
maxSoFar	1	1											
last	0	0											
start	0	0											
end	0	0											

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =1

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4											
maxSoFar	1	4											
last	0	0											
start	0	0											
end	0	1											

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =2

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	4										
maxSoFar	1	4	4										
last	0	0	0										
start	0	0	0										
end	0	1	1										

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =2

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8										
maxSoFar	1	4	1										
last	0	0	0										
start	0	0	0										
end	0	1	1										

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =2

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8										
maxSoFar	1	4	8										
last	0	0	0										
start	0	0	0										
end	0	1	2										

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =3

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	8									
maxSoFar	1	4	8	8									
last	0	0	0	0									
start	0	0	0	0									
end	0	1	2	2									

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =3

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0									
maxSoFar	1	4	8	8									
last	0	0	0	0									
start	0	0	0	0									
end	0	1	2	2									

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =3

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0									
maxSoFar	1	4	8	8									
last	0	0	0	4									
start	0	0	0	0									
end	0	1	2	2									

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =4

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	0								
maxSoFar	1	4	8	8	8								
last	0	0	0	4	4								
start	0	0	0	0	0								
end	0	1	2	2	2								

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =4

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2								
maxSoFar	1	4	8	8	8								
last	0	0	0	4	4								
start	0	0	0	0	0								
end	0	1	2	2	2								

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =5

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	2							
maxSoFar	1	4	8	8	8	8							
last	0	0	0	4	4	4							
start	0	0	0	0	0	0							
end	0	1	2	2	2	2							

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =5

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5							
maxSoFar	1	4	8	8	8	8							
last	0	0	0	4	4	4							
start	0	0	0	0	0	0							
end	0	1	2	2	2	2							

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =6

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	5						
maxSoFar	1	4	8	8	8	8	8						
last	0	0	0	4	4	4	4						
start	0	0	0	0	0	0	0						
end	0	1	2	2	2	2	2						

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =6

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4						
maxSoFar	1	4	8	8	8	8	8						
last	0	0	0	4	4	4	4						
start	0	0	0	0	0	0	0						
end	0	1	2	2	2	2	2						

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =7

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	4					
maxSoFar	1	4	8	8	8	8	8	8					
last	0	0	0	4	4	4	4	4					
start	0	0	0	0	0	0	0	0					
end	0	1	2	2	2	2	2	2					

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =7

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7					
maxSoFar	1	4	8	8	8	8	8	8					
last	0	0	0	4	4	4	4	4					
start	0	0	0	0	0	0	0	0					
end	0	1	2	2	2	2	2	2					

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =8

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	7				
maxSoFar	1	4	8	8	8	8	8	8	8				
last	0	0	0	4	4	4	4	4	4				
start	0	0	0	0	0	0	0	0	0				
end	0	1	2	2	2	2	2	2	2				

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =8

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11				
maxSoFar	1	4	8	8	8	8	8	8	8				
last	0	0	0	4	4	4	4	4	4				
start	0	0	0	0	0	0	0	0	0				
end	0	1	2	2	2	2	2	2	2				

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =8

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11				
maxSoFar	1	4	8	8	8	8	8	8	11				
last	0	0	0	4	4	4	4	4	4				
start	0	0	0	0	0	0	0	0	4				
end	0	1	2	2	2	2	2	2	8				

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =9

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	11			
maxSoFar	1	4	8	8	8	8	8	8	11	11			
last	0	0	0	4	4	4	4	4	4	4			
start	0	0	0	0	0	0	0	0	4	4			
end	0	1	2	2	2	2	2	2	8	8			

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =9

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8			
maxSoFar	1	4	8	8	8	8	8	8	11	11			
last	0	0	0	4	4	4	4	4	4	4			
start	0	0	0	0	0	0	0	0	4	4			
end	0	1	2	2	2	2	2	2	8	8			

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =10

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	8		
maxSoFar	1	4	8	8	8	8	8	8	11	11	11		
last	0	0	0	4	4	4	4	4	4	4	4		
start	0	0	0	0	0	0	0	0	4	4	4		
end	0	1	2	2	2	2	2	2	8	8	8		

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i = 10

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18		
maxSoFar	1	4	8	8	8	8	8	8	11	11	11		
last	0	0	0	4	4	4	4	4	4	4	4		
start	0	0	0	0	0	0	0	0	4	4	4		
end	0	1	2	2	2	2	2	2	8	8	8		

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =10

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18		
maxSoFar	1	4	8	8	8	8	8	8	11	11	18		
last	0	0	0	4	4	4	4	4	4	4	4		
start	0	0	0	0	0	0	0	0	4	4	4		
end	0	1	2	2	2	2	2	2	8	8	10		

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i = 11

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	18	
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	
last	0	0	0	4	4	4	4	4	4	4	4	4	
start	0	0	0	0	0	0	0	0	4	4	4	4	
end	0	1	2	2	2	2	2	2	8	8	10	10	

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i = 11

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	
last	0	0	0	4	4	4	4	4	4	4	4	4	
start	0	0	0	0	0	0	0	0	4	4	4	4	
end	0	1	2	2	2	2	2	2	8	8	10	10	

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =12

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	15
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	4
end	0	1	2	2	2	2	2	2	8	8	10	10	10

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

i =12

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	4
end	0	1	2	2	2	2	2	2	8	8	10	10	10

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	4
end	0	1	2	2	2	2	2	2	8	8	10	10	10

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	**4**
end	0	1	2	2	2	2	2	2	8	8	10	10	**10**

Programmazione dinamica (Kadane's Algorithm)

```
def maxsum4(A):
    maxSoFar = 0        # Maximum found so far
    maxHere = 0         # Maximum slice ending at the current pos
    start = end = 0 # Start, end of the maximal slice found so far
    last = 0            # Beginning of the maximal slice ending here
    for i in range(0, len(A)):
        maxHere = maxHere + A[i]
        if maxHere <= 0:
            maxHere = 0
            last = i+1
        if maxHere > maxSoFar:
            maxSoFar = maxHere
            start, end = last, i
    return (start, end)
```

A	1	3	4	-8	2	3	-1	3	4	-3	10	-3	2
maxHere	1	4	8	0	2	5	4	7	11	8	18	15	17
maxSoFar	1	4	8	8	8	8	8	8	11	11	18	18	18
last	0	0	0	4	4	4	4	4	4	4	4	4	4
start	0	0	0	0	0	0	0	0	4	4	4	4	4
end	0	1	2	2	2	2	2	2	8	8	10	10	10