

Esercitazione7_Latex

October 25, 2023

1 Esercitazione 7

1.1 Es 1. Trovare l'eventuale punto fisso in un vettore crescente

Testo

Progettare un algoritmo che, preso un vettore ordinato V di n interi distinti, determini se esiste un indice i tale che $V[i] = i$ in $O(\log n)$ tempo.

Idea

Dato il punto medio m del vettore controllare se $V[m]$ è uguale ad m , in tal caso il punto fisso è stato trovato. Altrimenti: - se $V[m] > m$ allora cerco a sinistra - se $V[m] < m$ allora cerco a destra

Soluzione

```
[ ]: def utilfixpoint(V, start, end):  
    if start > end:  
        return False  
    m = (start + end) // 2  
    if V[m] == m:  
        return True  
    elif V[m] > m:  
        return utilfixpoint(V, start, m - 1)  
    else:  
        return utilfixpoint(V, m + 1, end)  
  
def fixpoint(V):  
    return utilfixpoint(V, 0, len(V) - 1)
```

Esecuzione

```
[ ]: V = [-10, -5, 0, 3, 7]  
print(fixpoint(V))  
V = [0, 2, 5, 8, 17]  
print(fixpoint(V))  
V = [-10, -5, 3, 4, 7, 9]  
print(fixpoint(V))
```

True

True

False

2 Es 2. Trovare gli zeri di un vettore continuo

Testo

Sia V un vettore di n interi. Si dice che V è continuo se per ogni $i = 1, 2, \dots, n-1$, $|V[i+1]-V[i]| \leq 1$. Si dice zero del vettore un indice k tale che $V[k] = 0$.

Progettare un algoritmo che, dato un vettore V di $n \geq 2$ interi continuo e tale che $V[1] < 0$ e $V[n] > 0$, trovi uno zero in $O(\log n)$ tempo.

Idea

Per risolvere questo problema in maniera ricorsiva, possiamo utilizzare un algoritmo di ricerca binaria ricorsivo. L'idea è quella di passare l'intervallo di ricerca come parametro della funzione, e poi chiamare la funzione stessa ricorsivamente dimezzando l'intervallo di ricerca.

Inizialmente, l'intervallo di ricerca è $[l, r]$ dove $l = 0$ e $r = n-1$. Ad ogni chiamata ricorsiva, calcoliamo l'indice medio $m = (l+r)//2$ e verifichiamo se $V[m]$ è uguale a 0. Se $V[m] > 0$, allora lo zero si trova nell'intervallo $[l, m-1]$, altrimenti si trova nell'intervallo $[m+1, r]$. Continuiamo a dimezzare l'intervallo di ricerca finché troviamo lo zero o fino a quando l'intervallo diventa vuoto.

Soluzione

```
[ ]: def util_find_zero(V, start, end):
    if start > end:
        return None
    m = (start + end) // 2
    if V[m] == 0:
        return m
    elif V[m] > 0:
        return util_find_zero(V, start, m - 1)
    else:
        return util_find_zero(V, m + 1, end)

def find_zero(V):
    return util_find_zero(V, 0, len(V) - 1)
```

Esecuzione

```
[ ]: V = [-3, -2, -1, 0, 1, 2, 3]
print(find_zero(V))
V = [-1, 0, 0, 1]
print(find_zero(V))
```

3

1

3 Es 3. Algoritmo per determinare la radice quadrata intera di n in tempo

Testo

Progettare un algoritmo che, dato un intero n, calcoli l'intero inferiore della radice di n in $O(\log n)$ tempo, usando solo operazioni aritmetiche.

Idea

Si calcola la radice quadrata intera di n come segue: - Se n è inferiore a 2, restituire n (poiché la radice quadrata di 0 o 1 è 0 o 1). - Altrimenti, calcolare la radice quadrata intera di $n // 4$ in modo ricorsivo e moltiplicarla per 2 per ottenere un valore iniziale per la parte intera della radice quadrata di n. - Verificare se left o right è la parte intera della radice quadrata di n, dove left è il valore iniziale moltiplicato per 2 e right è left + 1. - Se $right * right$ è maggiore di n, restituire left. Altrimenti, restituire right.

Soluzione

```
[ ]: def sqrt_int(n):  
    if n < 2:  
        return n  
    else:  
        left = sqrt_int(n // 4) * 2  
        right = left + 1  
        if right * right > n:  
            return left  
        else:  
            return right
```

Esecuzione

```
[ ]: print(sqrt_int(9))  
print(sqrt_int(81))  
print(sqrt_int(1000))  
print(sqrt_int(1024))  
print(sqrt_int(1025))
```

3
9
31
32
32

4 Es 4. Trovare il massimo in un vettore ordinato shiftato di k posizioni

Testo

Si supponga di avere in input un vettore ordinato di n interi il cui contenuto è stato ruotato di

k posizioni. Supponendo di conoscere solo n, progettare un algoritmo che restituisca l'elemento massimo del vettore in $O(\log n)$ tempo.

Idea

L'idea è di trovare il punto di rotazione del vettore, ovvero l'indice in cui il vettore viene spezzato e riordinato, e poi confrontare il primo elemento con l'ultimo elemento del vettore per determinare quale è il massimo.

Soluzione

```
[ ]: def util_findMax(arr, low, high):
    # If there is only one element left
    if (high == low):
        return arr[low]
    # Find mid
    mid = low + (high - low) // 2
    # Check if mid reaches 0, it is greater than next element or not
    if (mid == 0 and arr[mid] > arr[mid+1]):
        return arr[mid]
    # Check if mid itself is maximum element
    if (mid < high and arr[mid + 1] < arr[mid] and mid > 0 and
    ↪ arr[mid] > arr[mid-1]):
        return arr[mid]
    # Decide whether we need to go to the left half or the right half
    if (arr[low] > arr[mid]):
        return util_findMax(arr, low, mid - 1)
    else:
        return util_findMax(arr, mid + 1, high)

def findMax(arr):
    return util_findMax(arr, 0, len(arr) - 1)
```

```
[ ]: V = [4, 5, 6, 7, 1, 2, 3]
print(findMax(V))
V = [6, 5, 4, 3, 2, 1, 0]
print(findMax(V))
```

7

6

5 Es. 5. Contare il numero di inversioni in un vettore

Testo

Dato un array, trova il numero totale di inversioni di esso. Se $(i < j)$ e $(A[i] > A[j])$, quindi accoppia (i, j) è chiamata inversione di un array A. Dobbiamo contare tutte queste coppie nell'array.

Idea

Una soluzione semplice sarebbe per ogni elemento dell'array contare tutti gli elementi meno di esso alla sua destra e aggiungere il conteggio all'output. La complessità di questa soluzione è $O(n^2)$, dove n è la dimensione dell'input.

Questo è un classico problema che può essere risolto con la funzione di merge del mergesort. Fondamentalmente, per ogni elemento dell'array, conta tutti gli elementi in più alla sua sinistra e aggiungi il conteggio all'output.

Soluzione

```
[ ]: # Unisci due sottoliste ordinate `A[low ... mid]` e `A[mid+1 ... high]`
def merge(A, aux, low, mid, high):
    k = i = low
    j = mid + 1
    inversionCount = 0
    # mentre ci sono elementi nelle piste di sinistra e di destra
    while i <= mid and j <= high:
        if A[i] <= A[j]:
            aux[k] = A[i]
            i = i + 1
        else:
            aux[k] = A[j]
            j = j + 1
            inversionCount += (mid - i + 1)      # NOTA
        k = k + 1
    # copia gli elementi rimanenti
    while i <= mid:
        aux[k] = A[i]
        k = k + 1
        i = i + 1
    # copia indietro nell'elenco originale per riflettere l'ordine ordinato
    for i in range(low, high + 1):
        A[i] = aux[i]
    return inversionCount

# Ordina la lista `A[low...high]` usando la lista ausiliaria `aux`
def conta_inversioni(A, aux, low, high):
    # caso base
    if high <= low:      # se la dimensione della corsa <= 1
        return 0
    # trova il punto medio
    mid = low + ((high - low) >> 1)
    inversionCount = 0
    # divide in modo ricorsivamente le corse in due metà fino a quando la
    ↪ dimensione della corsa <= 1,
    # quindi li unisce e ritorna alla catena di chiamate
    inversionCount += conta_inversioni(A, aux, low, mid)      # divide/unisce
    ↪ la metà sinistra
```

```

    inversionCount += conta_inversioni(A, aux, mid + 1, high) # divide/unisci
    ↪ la metà destra
    inversionCount += merge(A, aux, low, mid, high)           # unisce le due mezze
    ↪ piste
    return inversionCount

```

Esecuzione

```

[ ]: A = [1, 9, 6, 4, 5]
print(conta_inversioni(A, A.copy(), 0, len(A) - 1))

A = [1, 2, 3, 4, 5]
print(conta_inversioni(A, A.copy(), 0, len(A) - 1))

A = [5, 4, 3, 2, 1]
print(conta_inversioni(A, A.copy(), 0, len(A) - 1))

```

5
0
10

6 Es. 6. Algoritmo che determina il sottovettore di lunghezza massima di spessore al più C

Testo

In un vettore V di interi si dice spessore del vettore la differenza tra il massimo e il minimo del vettore. Progettare un algoritmo che, preso un vettore V di n interi ed un intero C , trovi un sottovettore (una sequenza di elementi consecutivi del vettore) di, lunghezza massima tra quelli di spessore al più C . La complessità dell'algoritmo deve essere $O(n \log n)$.

Idea

Per risolvere questo problema, puoi utilizzare un approccio basato sull'algoritmo dei due punti (two-pointer). L'idea di base è di mantenere due puntatori che rappresentano l'inizio e la fine del sottovettore, e spostarli all'interno del vettore in modo intelligente. Inoltre, utilizzeremo una struttura dati che ci consentirà di trovare rapidamente il massimo e il minimo all'interno del sottovettore corrente.

Soluzione

```

[ ]: def findMaxThicknessSubarray(V, C):
    left = 0
    right = 0
    maxThickness = 0
    currentSet = set()
    while right < len(V):
        currentSet.add(V[right])
        max_in_set = max(currentSet)

```

```

min_in_set = min(currentSet)
if max_in_set - min_in_set <= C:
    if right - left + 1 > maxThickness:
        maxThickness = right - left + 1
        max_left = left
        max_right = right
    right += 1
else:
    currentSet.remove(V[left])
    left += 1
return V[max_left:max_right+1]

```

Esecuzione

```

[ ]: V = [2, 4, 7, 1, 5, 3]
    C = 3
    print(findMaxThicknessSubarray(V, C))

V = [2, 4, 7, 1, 5, 3]
C = 4
print(findMaxThicknessSubarray(V, C))

V = [2, 4, 7, 1, 5, 3]
C = 5
print(findMaxThicknessSubarray(V, C))

```

```

[2, 4]
[1, 5, 3]
[2, 4, 7]

```