Algoritmi ordinamento

A CURA DI LINO POLO

L'ordinamento si riferisce alla disposizione dei dati in un formato particolare. L'algoritmo di ordinamento specifica il modo in cui disporre i dati in un ordine particolare. Gli ordini più comuni sono in ordine numerico o lessicografico.

L'importanza dell'ordinamento risiede nel fatto che la ricerca dei dati può essere ottimizzata a un livello molto alto, se i dati vengono memorizzati in modo ordinato. L'ordinamento viene utilizzato anche per rappresentare i dati in formati più leggibili. Di seguito vediamo cinque di queste implementazioni di ordinamento in Python.

Bubble Sort

Unisci ordinamento

Ordinamento di inserzione

Shell Sort

Ordina selezione

Bubble Sort

È un algoritmo basato sul confronto in cui viene confrontata ogni coppia di elementi adiacenti e gli elementi vengono scambiati se non sono in ordine.

Bubble Sort

```
def bubblesort(list):

# Swap the elements to arrange in order
    for iter_num in range(len(list)-1,0,-1):
        for idx in range(iter_num):
            if list[idx]>list[idx+1]:
                temp = list[idx]
                list[idx] = list[idx+1]
                list[idx+1] = temp


list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

Unisci ordinamento

Unisci ordinamento divide prima l'array in due metà uguali, quindi le combina in modo ordinato.

Unisci ordinamento - 1

```
def merge_sort(unsorted_list):
    if len(unsorted_list) <= 1:
        return unsorted_list
# Find the middle point and devide it
    middle = len(unsorted_list) // 2
    left_list = unsorted_list[:middle]
    right_list = unsorted_list[middle:]

    left_list = merge_sort(left_list)
    right_list = merge_sort(right_list)
    return list(merge(left_list, right_list))
```

Unisci ordinamento - 2

```
# Merge the sorted halves

def merge(left_half,right_half):

    res = []
    while len(left_half) != 0 and len(right_half) != 0:
        if left_half[0] < right_half[0]:
            res.append(left_half[0])
            left_half.remove(left_half[0])
        else:
            res.append(right_half[0])
            right_half.remove(right_half[0])
    if len(left_half) == 0:
        res = res + right_half
    else:
        res = res + left_half
    return res
```

Unisci ordinamento

```
unsorted_list = [64, 34, 25, 12, 22, 11, 90]

print(merge_sort(unsorted_list))
```

Ordinamento di inserzione

L'ordinamento per inserzione implica la ricerca del posto giusto per un dato elemento in un elenco ordinato. Quindi all'inizio confrontiamo i primi due elementi e li ordiniamo confrontandoli. Quindi scegliamo il terzo elemento e troviamo la sua posizione corretta tra i due precedenti elementi ordinati. In questo modo continuiamo gradualmente ad aggiungere altri elementi alla lista già ordinata mettendoli nella posizione corretta.

Ordinamento di inserzione

```
def insertion_sort(InputList):
    for i in range(1, len(InputList)):
        j = i-1
        nxt_element = InputList[i]
# Compare the current element with next one

        while (InputList[j] > nxt_element) and (j >= 0):
            InputList[j+1] = InputList[j]
            j=j-1
        InputList[j+1] = nxt_element

list = [19,2,31,45,30,11,121,27]
insertion_sort(list)
print(list)
```

Shell Sort

Shell Sort implica l'ordinamento di elementi che sono lontani da ech other. Ordiniamo una grande sottolista di una data lista e continuiamo a ridurre la dimensione della lista finché tutti gli elementi non sono ordinati. Il programma seguente trova il divario equiparandolo alla metà della lunghezza della dimensione dell'elenco e quindi inizia a ordinare tutti gli elementi in esso. Quindi continuiamo a ripristinare il divario fino a quando l'intero elenco non viene ordinato.

Shell Sort

```
def shellSort(input_list):

    gap = len(input_list) // 2
    while gap > 0:

        for i in range(gap, len(input_list)):
            temp = input_list[i]
            j = i
# Sort the sub list for this gap

            while j >= gap and input_list[j - gap] > temp:
                input_list[j] = input_list[j - gap]
                j = j-gap
            input_list[j] = temp

# Reduce the gap for the next element

        gap = gap//2

list = [19,2,31,45,30,11,121,27]

shellSort(list)
print(list)
```

Ordina selezione

Nell'ordinamento di selezione iniziamo trovando il valore minimo in un dato elenco e lo spostiamo in un elenco ordinato. Quindi ripetiamo il processo per ciascuno degli elementi rimanenti nell'elenco non ordinato. L'elemento successivo che entra nell'elenco ordinato viene confrontato con gli elementi esistenti e posizionato nella posizione corretta. Quindi alla fine vengono ordinati tutti gli elementi dell'elenco non ordinato.

Ordina selezione

```
def selection_sort(input_list):

    for idx in range(len(input_list)):

        min_idx = idx
        for j in range( idx +1, len(input_list)):
            if input_list[min_idx] > input_list[j]:
                min_idx = j
# Swap the minimum value with the compared value

        input_list[idx], input_list[min_idx] = input_list[min_idx], input_list[idx]



l = [19,2,31,45,30,11,121,27]
selection_sort(l)
print(l)
```