Programmazione e Algoritmi 1

A.A. 2023/24 — Compito del 12 febbario 2024

prof. Gianluca Amato

Gli esercizi di programmazione saranno valutati sulla base della correttezza, efficienza e comprensibilità della soluzione proposta. In generale, **se volete usare una funzione o un metodo non è stato presentato a lezione, chiedete prima al docente se è consentito**.

Esercizio 1 (5 punti)

Scrivere la funzione `remove_duplicate` che prende come parametro in input una lista e restituisce una nuova lista in cui tutti i duplicati sono stati rimossi. Ad esempio, `remove_duplicate([1, 10, 9, "abc", 10, "cde", "abc"])` restituisce la lista `[1, 10, 9, "abc", "cde"]`.

Esercizio 2 (5 punti)

Scrivere alcuni test nel framework `pytest` per verificare il corretto funzionamento di `remove_duplicate`. In particolare, si deve controllare che il valore di ritorno sia corretto per:

- la lista di esempio dell'Esercizio 1;
- la lista vuota;
- una lista in cui tutti gli elementi sono uguali: sia l'elemento da inserire nella lista sia la lunghezza della lista devono essere generati casualmente.

Esercizio 3 (5 punti)

Scrivere una funzione `positive_pairs` che prende due parametri: una lista di interi `l` e un numero intero positivo *opzionale* `k`. Se il parametro `k` non viene fornito, la funzione restituisce una nuova lista con tutte le coppie di numeri interi positivi presenti in `l`. Ad esempio `positive_pairs([-5, 6, -11, -3, -9, 15, -1, 90, -21, 3])` restituisce la lista `[ (6, 15), (6, 90), (6, 3), (15, 90), (15, 3), (90, 3) ]`. Si noti che consideriamo solo le coppie (x, y) dove x viene prima di y nella lista `l`.

Se invece il parametro `k` viene fornito in input, la funzione restituisce solo le prime `k` coppie di numeri interi positivi presenti in `l`. Ad esempio `positive_pairs([-5, 6, -11, -3, -9, 15, -1, 90, -21, 3], 4)` restituisce la lista `[ (6, 15), (6, 90), (6, 3), (15, 90) ]`. Se il numero di coppie presenti in `l` è minore di `k`, la funzione restituisce tutte quelle presenti in `l`, come se il parametro `k` non fosse stato fornito.

Esercizio 4 (5 punti)

Scrivere un programma che legge un file di testo dal nome `input.txt`. Il file deve essere composto da varie righe, ognuna delle quali contenenti un numero intero. Il programa legge il file specificato e inserisce i numeri in una lista, dopo di che chiama la funzione `positive_pairs` dell'Esercizio 3. Il risultato di `positive_pairs` deve essere salvato su un file che si chiama `output.txt`, una riga per ogni coppia, e gli elementi della coppia separati da uno spazio.

Se il file di input dovesse contenere una riga che non è convertibile in un numero, il programma deve ignorarla e continuare normalmente a leggere le righe successive. Ad esempio, se il file di input contiene:

```
-5
6
15
ciao
-1
90
```

il file di output generato dovrà contenere:

```
6 15
6 90
15 90
```

Esercizio 5 (5 punti)

Illustrare il funzionamento degli algoritmi di ricerca lineare e ricerca binaria, prima in termini generali e poi applicandoli alla ricerca del numero 10 nella lista che contiene, in ordine, tutti i numeri da 1 a 100. Discutere vantaggi e svantaggi dei due algoritmi.

Esercizio 6 (8 punti)

Si consideri il seguente codice Python (che, per la cronaca, implementa la *funzione di Ackermann*):

```python
def ackermann(m, n):
    if m == 0:
        return n + 1
    elif n == 0:
        return ackermann(m - 1, 1)
    else:
        x = ackermann(m, n - 1)
        return ackermann(m - 1, x)

print(ackermann(1, 2))
```

Eseguire passo passo il programma (senza l'ausilio di un computer) e fornirne la traccia di esecuzione.