Laboratorio 6 – Esercizi

Elementi di Informatica e Programmazione

Lab 6 – Es 1 (1/4)

Scrivere un modulo dedicato all'elaborazione di sottostringhe **mysubstring.py** contenente queste funzioni.

- **getAllSubstrings(s)** genera e restituisce una lista contenente tutte le possibili sottostringhe della stringa s, comprese quelle "improprie" (cioè la stringa vuota e la stringa s), disposte in ordine di lunghezza crescente
- **numSubstrings(s)** restituisce la lunghezza della lista che verrebbe restituita da **getAllSubstrings(s)**, senza costruire le sottostringhe né invocare **getAllSubstrings(s)**; ad esempio, **numSubstrings("") = 1**, **numSubstrings("tu") = 4**, **numSubstrings("casa") = 11**, ecc.
- **areUnique(ss)** riceve una lista di stringhe (eventualmente vuota) e restituisce **True** se e solo se i suoi elementi sono tutti diversi
- **isSortedByIncreasingLength(ss)** riceve una lista di stringhe (eventualmente vuota) e restituisce **True** se e solo se i suoi elementi sono disposti in ordine di lunghezza crescente (o, meglio, non decrescente) al crescere dell'indice
- **isSortedByDecreasingLength(ss)** riceve una lista di stringhe (eventualmente vuota) e restituisce **True** se e solo se i suoi elementi sono disposti in ordine di lunghezza decrescente (o, meglio, non crescente) al crescere dell'indice
- **isForwardSorted(ss)** riceve una lista di stringhe (eventualmente vuota) e restituisce **True** se e solo se i suoi elementi sono disposti, al crescere dell'indice, secondo l'ordinamento imposto dal criterio lessicografico

Lab 6 – Es 1 (2/4)

- **isBackwardSorted(ss)** riceve una lista di stringhe (eventualmente vuota) e restituisce **True** se e solo se i suoi elementi sono disposti, al crescere dell'indice, secondo l'ordinamento **inverso** di quello imposto dal criterio lessicografico
- **isSorted(ss)** svolge la stessa elaborazione svolta dalla funzione **isForwardSorted** (risolvere il problema **senza** copiare il codice di quella funzione)
- **isSubstring(s1, s2)** riceve due stringhe e restituisce True se e solo se s1 è una sottostringa (eventualmente impropria) di s2 (è ammesso l'uso dell'operatore **in**)
- **isSubsequence(s1, s2)** riceve due stringhe e restituisce True se e solo se s1 è una sottosequenza (eventualmente impropria) di s2 (come visto nell'Esercizio 4-6, una stringa s1 è una sottosequenza di un'altra stringa s2 se e solo se tutti i caratteri di s1 sono presenti in s2 nello stesso ordine, anche se in posizioni diverse e non consecutive)
- **getAllSubsequences(s)** genera e restituisce una lista contenente tutte le possibili sottosequenze della stringa s, comprese quelle "improprie" (cioè la stringa vuota e la stringa s), utilizzando l'algoritmo illustrato al termine di questo elenco
- **numSubsequences(s)** restituisce la lunghezza della lista che verrebbe restituita da **getAllSubsequences(s)**, senza costruire le sottostringhe né invocare **getAllSubsequences(s)**; ad esempio, **numSubsequences("") = 1**, **numSubsequences("tu") = 4**, **numSubsequences("casa") = 16**, ecc.

Lab 6 – Es 1 (3/4)

Generare tutte le sottosequenze di una stringa non è semplice, ma si può progettare un algoritmo di soluzione ragionando sulla natura della sottosequenza. Se immaginiamo che la sequenza sia memorizzata in una lista, come possiamo descrivere i valori che appartengono a una specifica sottosequenza? Possiamo farlo mediante le posizioni, all'interno della lista, dei caratteri che appartengono alla sottosequenza: tali posizioni sono, ovviamente, un sottoinsieme delle posizioni valide all'interno della lista (i due sottoinsiemi impropri corrispondono alle sottosequenze improprie: nessuna posizione e tutte le posizioni). Che alternative abbiamo per specificare un sottoinsieme delle posizioni di una sequenza? Possiamo generare una nuova sequenza contenente, come valori, tutti e soli gli indici corrispondenti alle posizioni che fanno parte del sottoinsieme. Oppure, possiamo utilizzare una sequenza di valori booleani, avente la stessa lunghezza della sequenza che stiamo elaborando, dove ciascun valore corrisponde al fatto che il valore della sequenza che si trova nella posizione corrispondente faccia parte del sottoinsieme (oppure no): una lista di questo tipo viene solitamente chiamata "vettore caratteristico" del sottoinsieme che vuole rappresentare.

Esempio: sequenza = "abcdef", sottosequenza = "acd", vettore caratteristico = [True, False, True, True, False, False].

Lab 6 – Es 1 (4/4)

A volte, per semplicità, invece di una lista di valori booleani si usa una lista di valori numerici zero o uno (dove, ad esempio, zero corrisponde a False). L'esempio precedente diventa: [1, 0, 1, 1, 0, 0]. Il primo valore 1 corrisponde al fatto che la lettera "a" (la prima della sequenza) fa parte della sottosequenza. Il primo valore 0 corrisponde al fatto che la lettera "b" (la seconda della sequenza) NON fa parte della sottosequenza. Osserviamo che, ovviamente, questa modalità di descrizione delle sottosequenze è valida anche quando la sequenza contiene eventuali elementi replicati, perché non fa riferimento ai valori contenuti nella sequenza, bensì alle loro posizioni.

Dopo aver osservato ciò, come possiamo generare tutte le sottosequenze di una sequenza? Generiamo tutti i vettori caratteristici che le descrivono! Quali sono tali vettori? Sono tutte (e sole!) le liste di lunghezza uguale alla lunghezza della sequenza e contenenti valori 0 e 1, in tutte le combinazioni possibili. Quante sono le possibili sequenze di n valori, ciascuno dei quali viene scelto tra due possibilità? Sono 2^n. Come le genero? Sono le rappresentazioni binarie posizionali dei numeri interi che vanno da zero a $(2^n - 1)$... vediamo una descrizione dell'algoritmo in pseudocodice:

```
n = len(s)
subs = [ ] # lista vuota
for i in range(2**n) :
   converti i in binario con n bit (aggiungendo zeri a sinistra se necessario)
   sub = ""
   for j in range(n) :
      se il bit j-esimo della conversione è 1
         sub += s[j]
   subs.append(sub)
```

Lab 6 – Es 2

Scrivere, poi, un programma di collaudo, **testSubstring.py**, che svolga le seguenti elaborazioni:

- per ogni stringa appartenente alla lista **["", "a", "ab", "abc", "abcdefghilm"]** genera tutte le sottostringhe invocando **getAllSubstrings** e ottenendo la lista **ss**, poi:
  - visualizza la lunghezza di **ss** prevista dalla funzione **numSubstrings**
  - visualizza la lunghezza effettiva di **ss**, evidenziando un errore se i due valori differiscono
  - passa **ss** alla funzione **areUnique** e segnala un errore se viene restituito il valore **False**
  - ordina la lista **ss** usando il metodo **sort**
  - passa **ss** alla funzione **isForwardSorted** e segnala un errore se viene restituito il valore **False**
  - inverte il contenuto della lista **ss** usando la funzione **reverse**, da progettare all'interno del programma (tale funzione riceve una lista e ne inverte il contenuto)
  - passa **ss** alla funzione **isBackwardSorted** e segnala un errore se viene restituito il valore **False**
- per ogni stringa appartenente alla lista **["", "a", "ab", "abc", "abcdefghilm"]** genera tutte le sottosequenze invocando **getAllSubsequences** e ottenendo la lista **ss**, poi:
  - visualizza la lunghezza di **ss** prevista dalla funzione **numSubsequences**
  - visualizza la lunghezza effettiva di **ss**, evidenziando un errore se i due valori differiscono
  - passa **ss** alla funzione **areUnique** e segnala un errore se viene restituito il valore **False**

Lab 6 – Es 3 (1/2)

Scrivere il programma (non grafico) **tictactoe.py** che gestisca, per conto di due giocatori umani, il gioco del "tris" (tic-tac-toe in inglese), il classico gioco in cui due giocatori dispongono alternativamente un proprio contrassegno in una casella di una scacchiera 3 per 3 finché uno dei due non pone tre contrassegni in una fila orizzontale, verticale o diagonale, vincendo la partita.

Il programma inizia visualizzando la scacchiera vuota, come segue (ogni puntino rappresenta una casella vuota):

```
. . .
. . .
. . .
```

e chiedendo al primo giocatore di inserire le coordinate della casella in cui vuole porre il suo contrassegno (che sarà un carattere **X**).

Le coordinate si indicano con due numeri interi (valori ammessi: 0, 1 o 2), il primo numero essendo l'indice di riga a partire dall'alto ed il secondo l'indice di colonna a partire da sinistra.

Lab 6 – Es 3 (2/2)

Il programma deve verificare se la casella richiesta è libera oppure no. Nel primo caso visualizza la scacchiera aggiornata e chiede all'altro giocatore di inserire la propria mossa (verrà usato il contrassegno **O**). Nel secondo caso, invece, il programma fornisce una segnalazione d'errore, visualizza nuovamente la stessa scacchiera visualizzata in precedenza e chiede al giocatore di inserire una nuova mossa; la stessa cosa avviene se il giocatore introduce delle coordinate non valide, ma il messaggio d'errore deve essere diverso.

Il programma deve essere in grado di segnalare la vittoria di uno dei due giocatori qualora questa avvenga, oppure di porre fine alla partita quando la scacchiera è piena senza che uno dei due giocatori abbia raggiunto la vittoria, segnalando la parità.

Al termine di una partita, il programma chiede se si intende giocare un'altra partita oppure no: nel secondo caso il programma termina.

Risolvere il problema definendo le funzioni che si ritengono utili.

Lab 6 – Es 4

Scrivere il modulo **myinput.py** che contenga le seguenti funzioni di utilità per l'acquisizione di dati dall'utente:

- **inputYesNo(message, yes, no)** restituisce True se l'utente ha scritto la stringa fornita come secondo argomento, restituisce False se l'utente ha scritto la stringa fornita come terzo argomento; ignora la differenza tra maiuscole e minuscole (es. di utilizzo: **if inputYesNo("Vuoi continuare? (S/N) ", "S", "N") : # vuole continuare**)
- **inputStringStartingWith(message, startingString)** restituisce una stringa che inizia con la stringa **startingString**; se **startingString** è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- **inputStringEndingWith(message, endingString)** restituisce una stringa che termina con la stringa **endingString**; se **endingString** è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- **inputStringContaining(message, substring)** restituisce una stringa che contiene la stringa **substring**; se **substring** è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- **isDecimalInteger(s)** restituisce True se e solo se la stringa s contiene un numero intero decimale, che ha questo formato: zero o più spazi iniziali, un eventuale segno meno, una o più cifre decimali (da 0 a 9), zero o più spazi finali (quindi, ad esempio, non ci può essere uno spazio tra il segno meno e la prima cifra del numero)
- **inputDecimalInteger(message)** restituisce un numero intero; la funzione deve utilizzare in modo opportuno la funzione **isDecimalInteger**
- **inputPositiveDecimalInteger(message)** restituisce un numero intero positivo; la funzione deve utilizzare in modo opportuno la funzione **inputDecimalInteger**
- **inputNegativeDecimalInteger(message)** restituisce un numero intero negativo; la funzione deve utilizzare in modo opportuno la funzione **inputDecimalInteger**
- **inputNonPositiveDecimalInteger(message)** restituisce un numero intero non positivo; la funzione deve utilizzare in modo opportuno la funzione **inputDecimalInteger**
- **inputNonNegativeDecimalInteger(message)** restituisce un numero intero non negativo; la funzione deve utilizzare in modo opportuno la funzione **inputDecimalInteger**
- **isFloating(s)** restituisce True se e solo se la stringa s contiene un numero decimale in virgola mobile, che ha questo formato: zero o più spazi iniziali, un eventuale segno meno, una o più cifre decimali (da 0 a 9), un eventuale separatore decimale (il carattere "punto") seguito da una o più cifre decimali, un'eventuale lettera "e" (maiuscola o minuscola) seguita da un eventuale segno meno e da una o più cifre decimali, zero o più spazi finali
- **inputFloating(message)** restituisce un numero in virgola mobile; la funzione deve utilizzare in modo opportuno la funzione **isFloating**

Tutte le funzioni di tipo **input...** devono chiedere ripetutamente il dato all'utente finché questo non rispetta le specifiche della funzione, riproponendo il messaggio **message** (senza visualizzare messaggi d'errore). Tutte le funzioni di tipo **is...**, invece, non devono avere alcuna interazione con l'utente (né in input né in output).

Lab 6 – Es 5

Scrivere il programma **nim2.py** che abbia esattamente lo stesso comportamento del programma **nim.py** visto in precedenza.

Nella soluzione, però, è richiesta la definizione e l'utilizzo delle seguenti **funzioni**: **main** deve utilizzare i seguenti metodi ausiliari:

- **askUserBalls(numBalls)** comunica al giocatore umano il numero di biglie presenti nel mucchio (numBalls) e chiede quante ne vuole prendere, ripetendo la domanda finché non viene indicata una quantità che rispetti le regole del gioco; a quel punto, tale quantità viene restituita dalla funzione
- **takeComputerBalls(computerPlayerIsExpert, numBalls)** calcola e restituisce il numero di biglie prese dal computer, sulla base del numero di biglie presenti nel mucchio (**numBalls**) e sulla modalità di gioco del computer stesso (che gioca in modo "intelligente" se e solo se **computerPlayerIsExpert** ha il valore **True**); la funzione non deve visualizzare informazioni in output
- **playAgain()** restituisce il valore **True** se e solo se l'utente vuole fare un'altra partita
- **randomBoolean()** genera e restituisce un valore booleano casuale (con distribuzione di probabilità uniforme)

Il programma principale, poi, va definito all'interno della funzione **main**. Per quanto possibile, riutilizzare e adattare il codice già scritto per il programma **nim.py**.

Lab 6 – Es 6

Scrivere il programma **newPassword.py** che chieda all'utente una nuova password e la accetti soltanto se rispetta i requisiti di sicurezza, che, in questo esempio, sono:

- deve contenere almeno 8 caratteri
- deve contenere almeno una lettera maiuscola
- deve contenere almeno una lettera minuscola
- deve contenere almeno una cifra numerica
- deve contenere almeno un carattere che non sia una lettera né una cifra
- non deve contenere spazi
- non deve contenere caratteri ripetuti (cioè tutti i caratteri devono essere diversi)

Se la password è accettabile il programma termina, altrimenti visualizza un messaggio d'errore (che deve specificare quali delle regole sono state violate) e ripete la richiesta. Se la password viola più regole, queste vanno elencate tutte.

Definire una funzione per ciascuna regola, che riceva la password e restituisca **True** se e solo se la regola corrispondente è rispettata (senza avere alcuna interazione con l'utente, né in input né in output).

Definire, inoltre, una funzione che riceva la password e restituisca **True** se e solo se la password è valida, visualizzando un messaggio d'errore per ciascuna regola violata ma senza richiedere una nuova password.

Lab 6 – Es 7

Scrivere il programma **numberName.py** che acquisisca un numero intero positivo minore di un milione e ne visualizzi la descrizione in italiano, seguendo le regole riportate qui: https://it.wiktionary.org/wiki/Appendice:Tutti_i_numeri_in_lettere (attenzione ai molti casi particolari...).

Definire funzioni opportune in modo da evitare, per quanto possibile, la duplicazione di codice.