Algoritmi e Strutture Dati 1 SA 2020-2021

Esercizio 1 (stima difficoltà: ★☆☆☆☆)

Luca sta valutando di comprare un nuovo smartphone. Sai che, nel suo giro di shopping, visiterà `N` negozi in ordine e che l'`i`-esimo di questi negozi espone in vetrina uno smarpthone di valore `prezzo[i]`. Essendo appassionato di tecnologia, ogni qual volta Luca visita un negozio e vede un telefono di valore maggiore di quello che usa attualmente, decide subito di comprarlo e iniziare ad usarlo. Assumi che, prima di visitare il primo negozio, il valore dello smartphone di Luca sia 0. Quanto spende in tutto?

Esempio:

```
>>> prezzi = [400, 1000, 800, 1200, 1100]
>>> shopping(prezzi)
2600
```

Spiegazione. Luca comprerà lo smartphone esposto nel primo negozio (400 CHF), nel secondo (1000 CHF) e nel quarto (1200 CHF), spendendo in tutto 2600 CHF.

Esercizio 2 (stima difficoltà: ★★☆☆☆)

Sapendo che la polizia lo sta cercando, Luca ha trovato una lunga via composta da `N` case (la prima ha numero `0`, l'ultima `N-1`) e vuole nascondersi. La polizia lo sta inseguendo ma non sa in che casa si trova, quindi ha intenzione, partendo dalla prima casa, di interrogare gli abitanti di ogni casa finché non rivelano il numero civico della prossima casa da controllare. Gli agenti quindi si spostano, di casa in casa, seguendo le istruzioni date dagli abitanti, sperando di trovare Luca. Sapendo che Luca si nasconde nella casa `K` (secondo parametro della funzione `trova_criminale`), dopo quanti tentativi falliti la polizia riuscirà a trovarlo? Se non riuscirà mai, ritorna `-1`.

Esempio:

```
>>> trova_criminale([1, 3, 0, 2, 2], 2)
3
>>> trova_criminale([2, 1, 0, 1, 4], 0)
0
>>> trova_criminale([1, 2, 0, 0, 4], 4)
-1
>>> trova_criminale([2, 4, 1, 0, 4], 4)
3
```

Spiegazione. Nel primo caso, la polizia inizia dalla casa `0`, poi va nella casa `1`, poi alla `3` e infine trova Luca nella `2` dopo tre tentativi falliti. Nel secondo caso, Luca è nella prima casa, quindi la polizia lo trova dopo zero tentativi falliti. Nel terzo caso, la polizia inizia dalla casa `0`, poi `1`, poi `2` e poi indietro a `0` continuando così per sempre: Luca è al sicuro! Nel quarto caso, la polizia inizia dalla casa `0`, poi va alla casa `2`, poi va alla casa `1` e infine va alla casa `4`, dove si nasconde Luca; per trovarlo ha fallito tre tentativi prima di arrivare alla casa giusta.

Esercizio 3 (stima difficoltà: ★☆☆☆☆ oppure ★★☆☆☆)

Alcuni bambini stanno giocando con dei cubotti morbidi 1m x 1m x 1m in palestra, disposti in pile come nella seguente immagine: il gioco è partire da sinistra e arrivare in fondo a destra. Conosci il numero di cubi presenti in ciascuna pila, visti da sinistra a destra (nell'immagine sarebbero quindi `[4,2,2,5]`).

Sorge tuttavia un problema: mentre si può "lasciarsi cadere" oppure camminare tranquillamente per passare da una pila a quella successiva (se quella successiva ha altezza minore oppure uguale), serve una scala per passare da una pila ad una successiva più alta. La scala deve essere lunga *almeno* tanto quanto la differenza in altezza tra le due pile. Qual è la *più corta* scala che un bambino può portarsi con sé per riuscire a completare il percorso, partendo dall'inizio (ad altezza 0, nessun cubotto)?

Esempio:

```
>>> scala([4, 2, 2, 5])
4
>>> scala([2, 6, 3, 7, 0, 0, 7, 3, 6, 4])
7
```

Spiegazione. Il primo esempio corrisponde all'immagine sopra; una scala di lunghezza 4 metri è sufficiente per superare i due dislivelli e non si può fare di meglio. Il secondo esempio è raffigurato nella prossima immagine: serve almeno una scala di 7 metri.

Nota. Questo problema può essere risolto con complessità lineare. Se hai realizzato una soluzione avente complessità maggiore, prova a pensare a quella lineare!

Esercizio 4 (stima difficoltà: ★★☆☆☆ oppure ★★★☆☆)

Per festeggiare il diploma appena ricevuto da USI, Luca ha comprato una grande torta rettangolare. Vuole invitare quanti più amici possibile e offrire a ciascuno un pezzo della torta, ma è anche molto pigro e vuole fare *al massimo* `N` tagli. Ogni taglio è *orizzontale* oppure *verticale* e si estende lungo tutta la torta. Quanti pezzi di torta riuscirà ad ottenere Luca al massimo?

Esempio:

```
>>> taglia(4)
9
>>> taglia(1)
2
```

Spiegazione. Nel primo caso una soluzione ottima prevede di fare due tagli verticali e due orizzontali, ottenendo nove pezzi, come mostrato nella figura seguente.

Nel secondo caso, possiamo ottenere due pezzi con un unico taglio verticale o orizzontale.

Nota. Questo problema può essere risolto con complessità quadratica, lineare o costante. Se hai realizzato una soluzione avente complessità quadratica o lineare, prova a pensare a come si può fare di meglio!

Esercizio 5 (stima difficoltà: ★★☆☆☆ oppure ★★★☆☆)

Luca sta provando a cucinare dei pancake: ne ha disposti `N` in fila e li ha decorati con alcune farciture. Questo significa che ora ogni pancake ha una bontà associata `B[i]` (che può essere anche negativa, se la farcitura l'ha reso sgradevole). Luca ha a disposizione una paletta larga `K` (secondo parametro della funzione) con cui prendere

esattamente `K` pancake e ovviamente vuole *massimizzare la bontà totale* di quelli che prende. Quanto vale questa bontà massima?

Esempio:

```
>>> bonta([10, -3, -1, 6, 4, 1, -10], 3)
11
>>> bonta([1, -2, 4, -8, 16], 2)
8
```

Spiegazione. Nel primo caso la soluzione ottima prevede di prendere i pancake di bontà 6, 4, 1 per un totale di 11. Nel secondo caso, visto che si è obbligati a prendere `K=2` pancake, la scelta migliore è -8 e 16 per un totale di 8.

Nota. Questo problema può essere risolto con complessità lineare (`O(N)`). Se hai realizzato una soluzione avente complessità maggiore, prova a pensare a come si può fare di meglio!

Esercizio 6 (stima difficoltà: ★☆☆☆☆ oppure ★★★☆☆)

Luca si spazientisce spesso quando fa la spesa in Italia e la persona in coda alla cassa prima di lui vuole pagare con i *buoni pasto*. Un buono pasto funziona come una banconota di un certo valore, ma il supermercato impone delle restrizioni per spenderli: non puoi usare due buoni pasto dello stesso valore e non puoi superare il valore della spesa con il valore totale dei buoni pasto che usi.

La persona di fronte a lui deve pagare una spesa di `T` euro (primo parametro della funzione) e ha a disposizione buoni pasto di tutti i valori da `1` fino a `T` e oltre! Sembra che le persone si divertano a far perdere tempo a Luca usando quanti più buoni pasto possibile (nel rispetto delle regole del supermercato). Luca si chiede: quanti buoni pasto può usare al massimo la persona di fronte a lui?

Esempio:

```
>>> buoni_pasto(9)
3
>>> buoni_pasto(10)
4
```

Spiegazione. Nel primo caso si possono usare i buoni pasto di valore 5, 3 e 1 per un totale di 9 euro; non è possibile usarne quattro senza superare 9 euro e quindi 3 è il massimo. Nel secondo caso l'unica combinazione valida è 1+2+3+4 = 10.

Nota. Questo problema può essere risolto facilmente con complessità lineare (`O(N)`) oppure un po' meno facilmente in tempo costante facendo alcune osservazioni matematiche (non richiedono niente più di cose che abbiamo visto nel corso, ma non perdete troppo tempo nel pensarci se prima non avete già fatto gli altri esercizi).

Esercizio 7 (stima difficoltà: ★★☆☆☆)

Luca è molto ordinato e metodico. Quando trova un foglio di carta, lo piega ripetutamente per poi metterlo da parte. Essendo metodico, Luca piega sempre un foglio secondo il lato più lungo. Per esempio, Luca piega un foglio di dimensione 80x60 in modo da ottenere un foglio (piegato) di dimensione 40x60. Tuttavia, siccome Luca non crede nei numeri frazionari, Luca piega sempre i fogli il più vicino possibile alla metà, ma comunque in modo da ottenere una dimensione intera. Quindi per esempio, Luca piega un foglio di dimensione 15x12 in modo da ottenere la dimensione 8x12. Nota che la dimensione del foglio piegato è data dalla maggiore delle due parti. Nell'esempio precedente, le parti piegate sono lunghe 8 e 7 (somma 15), quindi la dimensione risultante è 8.

Scrivi un algoritmo `piega_foglio(w,h,k)` che, data la dimensione iniziale `w`x`h` di un foglio e un numero `k`, stampa le dimensioni risultanti dopo `k` pieghe.

Esempio:

```
>>> piega_foglio(50,30,3)
13 15
>>> piega_foglio(100,32,6)
7 8
>>> piega_foglio(80,10,3)
10 10
```

Esercizio 8 (stima difficoltà: ★★★☆☆ oppure ★★★★☆)

Una fabbrica di scarpe produce separatamente le scarpe sinistre e quelle destre. Luca deve occuparsi di metterle assieme. Luca riceve un gruppo di scarpe sinistre di varie taglie e un gruppo di scarpe destre pure di varie taglie. Luca deve accoppiarle in modo che ogni paio contenga una scarpa sinistra e una destra della stessa taglia.

Scrivi un algoritmo `paia_di_scarpe(S,D)` che, date due sequenze `S` e `D` che contengono le taglie delle scarpe rispettivamente sinistre e destre, ritorna il numero di paia di scarpe correttamente inscatolate.

Esempio:

```
>>> paia_di_scarpe([39, 38, 38, 36], [37, 38, 39, 40])
2
>>> paia_di_scarpe([41, 40, 39, 40], [40, 41, 42, 40])
3
```

Esercizio 9 (stima difficoltà: ★★★★★)

Luca adora passeggiare lungo il sentiero *Infinito* e, essendo metodico, tiene una tabella dove ogni volta che fa una passeggiata segna nella colonna `A` il punto di partenza e nella colonna `B` il punto di arrivo. I punti sono identificati da un numero (non necessariamente intero) che indica il chilometraggio lungo il sentiero.

Un giorno Antonio chiede a Luca quanti chilometri non ha mai visto del sentiero *Infinito*, e siccome il sentiero è appunto infinito, Antonio specifica che si riferisce alla parte del sentiero che parte dal punto `p` e finisce al punto `q`.

Scrivi un algoritmo `chilometri_mai_visti(A,B,p,q)` che, date le due colonne `A` e `B` della tabella di Luca e dato una parte del sentiero che inizia al chilometro `p` e finisce al chilometro `q`, ritorna la distanza totale di sentiero tra `p` e `q` che Luca non ha mai visitato nelle sue passeggiate.

Esempio:

```
>>> chilometri_mai_visti([6,7,3],[8,11,6],0,10)
3
>>> chilometri_mai_visti([10,-10],[15,2.5],0,20)
12.5
```

Esercizio 10 (stima difficoltà: ★★☆☆☆)

Luca è un tipo sportivo. Quindi nel fine settimana fa sport per mantenersi in forma. Per non annoiarsi, Luca alterna tra le sue attività sportive preferite, che sono il canottaggio, la corsa, e le trazioni, in questo ordine. Ossia, la prima attività in assoluto sarà il canottaggio. Poi, se l'ultima attività che ha fatto è la corsa, la prossima sarà le trazioni, e se l'ultima è le trazioni, la prossima sarà il canottaggio, ecc.

Luca vuole mantenere un regime calorico di 2000 calorie al giorno e quindi tiene traccia del numero di calorie assimilate ogni giorno. Poi, ogni domenica, dopo un'attività sportiva che fa comunque, aggiunge un'altra attività per ogni mille calorie in eccesso al suo fabbisogno settimanale. Per esempio, se nella prima settimana ha assimilato 15500 calorie, nella prima domenica farà `canottaggio` e `corsa`. Se poi la settimana seguente mangia per un totale di 14600 calorie, allora farà `trazioni`.

Per aiutare Luca a pianificare le sue attività sportive della domenica, scrivi un algoritmo `sport_della_domenica(C)` che, data una sequenza `C` di valori calorici giornalieri, stampa per ogni settimana la lista di attività sportive per quella settimana. Le attività di una settimana devono essere stampate sulla stessa riga.

Esempio:

```
>>> sport_della_domenica([2000, 3000, 2000, 3000, 2500, 2000, 1800, 2000, 2000, 2000])
canottaggio corsa trazioni
>>> sport_della_domenica([2000, 3000, 2000, 1400, 2500, 2000, 1800, 2000, 2000, 2000])
canottaggio
>>> sport_della_domenica([2000, 3000, 2000, 2200, 2500, 2000, 1800, 2000, 2000, 2000])
canottaggio corsa
>>> sport_della_domenica([2000, 3000, 2000, 2200, 2500, 2000, 1800, 2000, 2000, 2000, 2000, 1800,
2000, 2000])
canottaggio corsa
trazioni
>>> sport_della_domenica([2000, 3000, 2000, 2200, 2500, 2000, 1800, 2000, 2000, 2000, 2000, 1800,
2000, 2000, 2000, 2000, 3000, 3000, 2000, 2300, 3000, 5000, 2000])
canottaggio corsa
trazioni
canottaggio corsa trazioni canottaggio
```

Esercizio 11 (stima difficoltà: ★☆☆☆☆)

Antonio è un programmatore dilettante ma crede di essere sopra la media dei programmatori. Per mettersi alla prova, Antonio partecipa ad una competizione di programmazione e poi ottiene i risultati di tutti i partecipanti in una sequenza `R`. Antonio sa che il prorio risultato è nella posizione `k`, ma non riesce a verificare se è effettivamente sopra alla media.

Per aiutare Antonio, scrivi un algoritmo `sopra_la_media(R,k)` che, data una sequenza `R` di risultati, ritorna ~True~ se il valore in posizione `k` è maggiore della media di tutti i valori di `R`.

Esempio:

```
>>> sopra_la_media([32, 78, 30, 50, 65, 34, 54, 43, 67], 3)
False
>>> sopra_la_media([10], 0)
False
>>> sopra_la_media([32, 78, 30, 51, 65, 34, 54, 43, 67], 3)
True
```

Esercizio 12 (stima difficoltà: ★★★☆☆)

Antonio ha adottato un coniglio e lo tiene sulla tetto della sua casa. Sul tetto ci sono anche un po' di vasi con piante e erbette di cui il coniglio è ghiotto e che Antonio vuole che il coniglio possa mangiare a piacimento. Antonio però non vuole che il coniglio cada dal tetto e quindi ha deciso di tenere il coniglio al guinzaglio. Antonio vuole trovare un punto dove fissare il guinzaglio in modo che si possa regolare il guinzaglio in modo che il coniglio possa raggiungere tutti i vasi ma non possa cadere dal tetto. Il tetto di Antonio è un rettangolo di dimensioni `dx` e `dy`. Antonio non crede che esistano altri numeri fuorché gli interi, quindi ha posizionato i vasi in punti di coordinate intere e vuole trovare un punto di coordinate intere `(x,y)` in cui fissare il guinzaglio. Nota che le coordinate partono da `(0,0)` e arrivano a `(dx,dy)`. Nota inoltre che il guinzaglio non può essere fissato in un punto dove c'è un vaso.

Scrivi un algoritmo `coniglio_al_guinzaglio(dx,dy,X,Y)` che, data la dimensione del tetto `l` e data una sequenza di coordinate `(X[i],Y[i])` di vasi, stampi le coordinate di un punto in cui fissare il guinzaglio. Se un tale punto non esiste, l'algoritmo deve stampare la scritta "impossibile". Se ci sono più punti in cui è possibile fissare il guinzaglio, l'algoritmo deve stampare quello con coordinata `X` minore, e se anche di quelli ce ne fossero più di uno, l'algoritmo deve stampare quello con coordinata `Y` minore.

Esempio:

```
>>> coniglio_al_guinzaglio(10,10,[6,5],[6,4])
3 6
>>> coniglio_al_guinzaglio(20,20,[1,19],[1,19])
impossibile
>>> coniglio_al_guinzaglio(20,8,[1,1,1],[1,2,3])
2 2
```

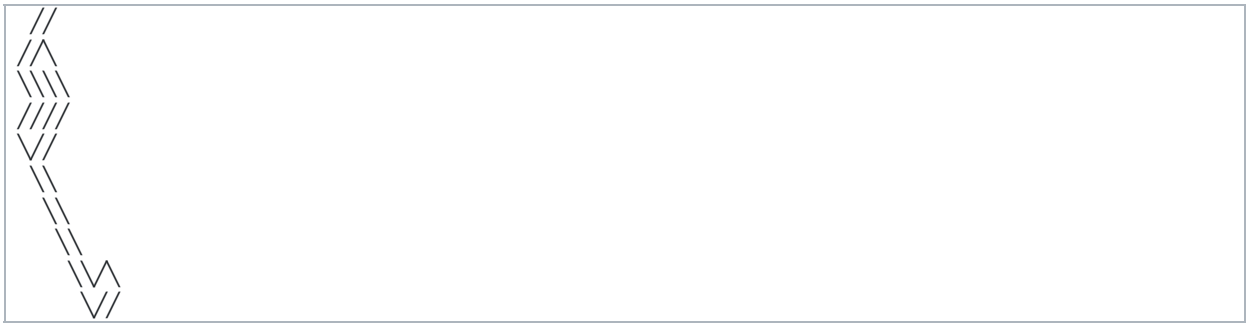
Esercizio 13 (stima difficoltà: ★★★★☆)

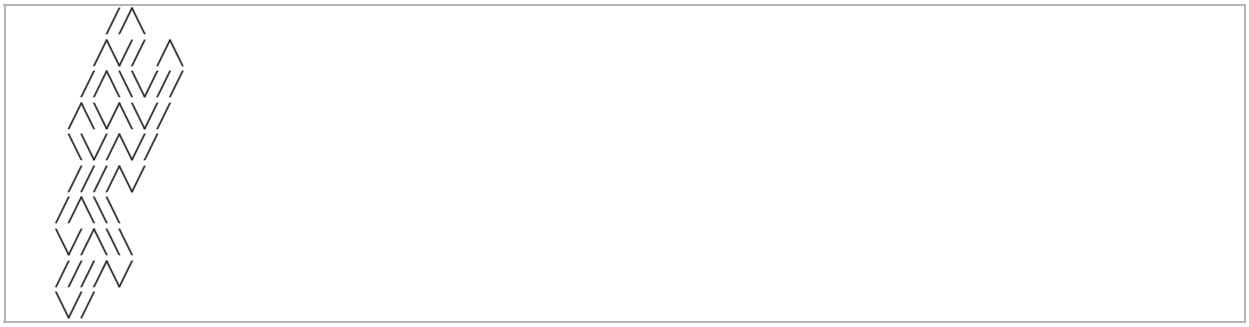
Aiuta l'esploratore a trovare il mitico passaggio da Nord a Sud. O magari quello mitico era il passaggio a Nord-Ovest, ma comunque qui è da Nord a Sud. Scrivi un algoritmo `passaggio_nord_sud(M)` che data una mappa `M` ritorna `True` se esiste un percorso che attraversa la mappa da Nord a Sud. La mappa è una sequenza di stringhe di caratteri `╱` o `╲`, ossia ogni carattere è uno sbarramento che va da Sud-Ovest verso Nord-Est (carattere `╱`, Unicode `'\u2572'`), oppure da Sud-Est verso Nord-Ovest (carattere `╲`, Unicode `'\u2571'`).

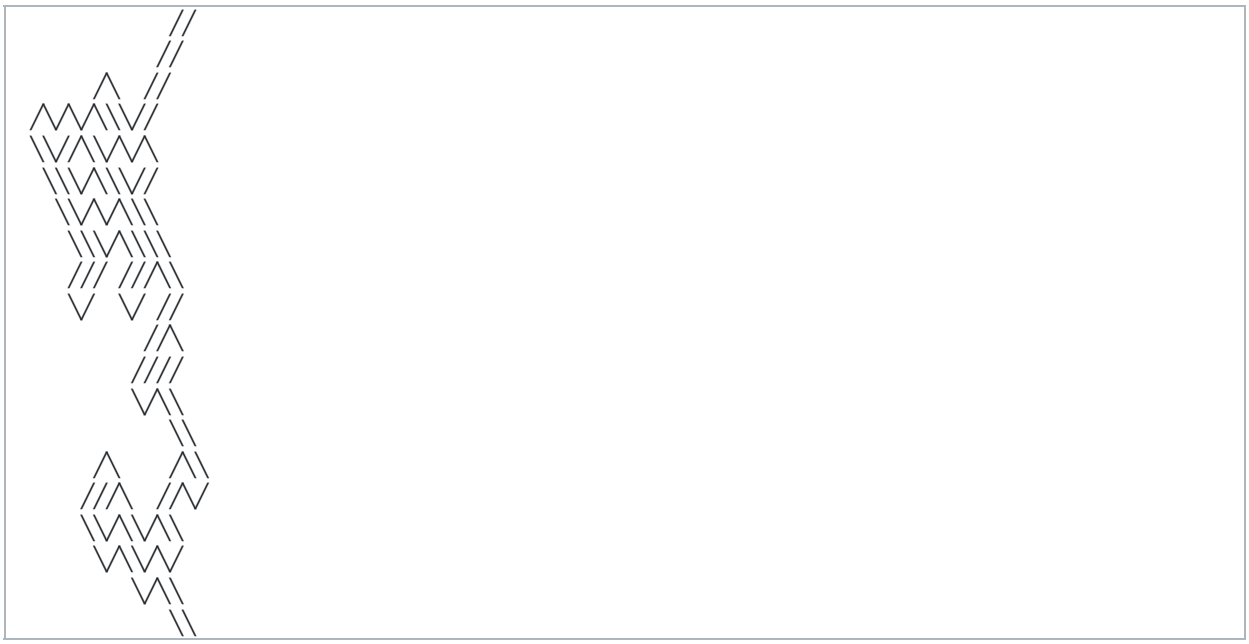
Esempio:

```
>>> M = ['\///\/\\//',
         '\//\//\/\\',
         '\\\\\/\///',
         '\////\/\\/',
         '/\//\///\/',
         '\\\\\\///\',
         '///\\\//\/',
         '////\\////',
         '\\///\\/\/',
         '//\\//\///']
>>> passaggio_nord_sud(M)
True
>>> M = ['\\///\////',
         '/\\\\/////',
         '\\///\/\//',
         '//\\///\/\',
         '/\/\///\//',
         '\\\//\//\/',
         '\//\/\\\\/',
         '//\\/////\',
         '///\/\//\\',
         '/\\//\\\/\']
>>> passaggio_nord_sud(M)
False
>>> M = ['\\/\\\/\//\///\/\///',
         '/\/\/\\\/\///\\/\\//',
         '//\\/////\\/\\/\//\',
         '\/\\\/\\/\\//\/\////',
         '\\/\/\\/\/\///\\/\/\',
         '\/\\///\/\\///\\/\//',
         '\/\//\\\/\\/\\\//\/\',
         '\//\/\//\\\//\/\/\\\',
         '\/\\///\/\/\//\\//\\',
         '//\/\\/\\//\\\//\\/\']
>>> passaggio_nord_sud(M)
True
>>> passaggio_nord_sud(
['\\/\//\\//\\/\\\\//\/////\/\///\\///\//\//\/\///////\/////\',
 '\/\\\\///\//\//\/\/\/\/\\/\/////\////\//\\/\\/\\/\/\\/\/\',
 '/\/////\/////\/\/\/\//\/\/\//////\////\/\/\/\///\\\/\\\',
 '\//\/\\\/\/\/\//\/\/\/\/\\///\/\///\\\/\/\/\//\\/\//\',
 '\/\//\/\/\//\\/\\\\\\\\\\/\\\\//\////\/\\///\/////',
 '/\/\\/\\/////\\\\///\/\/\///\\\/\/\/\/\/\\\\\',
 '\//\\/\/\\/\//\//\/\\/\/\/\/\/\/\/////\\/\//\/\/',
 '//\/\\\/\\\\\\\\\\\//\/\/\\/\/\/\/\/\//\//\/\\//',
 '\\\\//////\\///\/\/\\\//\/\/\/\//\/\/\\\//\/\//\/',
 '/\\/\/\/\///\/\/\\\\\/\/\//\\\/\///\\/\/\/\\\/\/\/',
 '\/\/\///\///\\/\\\\\\/\/\/\\\/\/\/\///\/\/\\//\\\',
 '\//\\\/\////\\\\\/\/\//\\\/\/\/\/\/\/\/\\\/\/\/\',
 '\\\/\\\/\/\/\/////\//\/\\\/\//\\\/\/\/\/\/\//\/\/\\',
 '///\/\/\/\/\///\/////\/\/\/\//\\\/\/\/\/\\//',
 '/\///\/\/\/\/\///\/\\/\/\/\/\\/\\\\\\\/\//\/\//\',
 '\\\/\//\/\//\/\\\/\/\\/\/\/\//\/\\///\/\/\\/\/\',
 '\///\\\/\/\\\\/\//\//\/\/\/\/\\\/\/\\/\\\/\//',
 '/\///\\/\/\/\/\/\\/\/\//\////\\\\/\//\\/\\\/\//',
 '\\/\\\/\/\/\/\/\//\\\\/\/\//\\/\/\/\/\/\//\/\\\/\/',
 '/\\/\\\/\\\/\\\\/\\\/\/\\/\/\/\/\/\\\\\/\/\/\/\\\/\'])
... True
```

Le soluzioni del primo, terzo e quarto esempio sono







Esercizio 14 (stima difficoltà: ★★☆☆☆)

Antonio e Luca vogliono parlare senza farsi capire da altri e quindi hanno sviluppato un codice che hanno chiamato *Anima*. Il codice trasforma ogni parola aggiungendo dopo ogni vocale la lettera `'s'` e la poi la stessa vocale. Per esempio, la parola `'anima'` diventa `'asanisimasa'`.

Scrivi un algoritmo `anima(T)` che decodifica un testo `T` passato come sequenza di parole (stringhe). L'algoritmo deve stampare il testo originale, stampando al meglio anche le parole che non sono codificate correttamente, come nel secondo esempio qua sotto.

Esempio:

```
>>> anima(['pisizzasa', 'masargheserisitasa'])
pizza margherita
>>> anima(['un asalgosoritmoso mosoltoso beselloso'])
un algoritmo molto bello
```

Esercizio 15 (stima difficoltà: ★★☆☆☆)

Luca deve andare a fare delle commesse in vari uffici in la lunga via del centro. Luca viene da fuori città, parcheggia in centro e poi si muove a piedi tra tutti gli uffici dove deve andare. Ed essendo metodico, Luca parcheggia in una

◀ Soluzioni dell'esame intermedio Jump to... Soluzioni degli esercizi 1-8 ▶

Credits | : eLab / USI / SUPSI
Get the Moodle mobile app
powered by eLab USI

```
>>> percorso_minimo([24,15,89,37])
152
>>> percorso_minimo([7, 30, 41, 14, 39, 42])
70
```

Last modified: Thursday, 21 January 2021, 4:00 PM