Parentesi k-bilanciate

Idea

L'obiettivo è determinare se una stringa di parentesi può essere bilanciata aggiungendo al massimo k parentesi chiuse e aperte. Per farlo, dobbiamo verificare due proprietà:

1. Il numero totale di parentesi aperte deve essere uguale al numero di parentesi chiuse.
2. Durante la scansione, ogni parentesi chiusa deve essere accoppiata a una parentesi aperta già incontrata (o potenzialmente aggiunta).

Algoritmo

Algorithm 1 Ricerca del k

```
function KBILANCIO(S)
    par_aperte ← 0
    par_chiuse ← 0
    counter ← 0
▷ Prima scansione per contare le parentesi
    for x ∈ S do
        if x = ( then
            par_aperte ← par_aperte + 1
        else if x = ) then
            par_chiuse ← par_chiuse + 1
        end if
    end for
▷ Controllo del bilanciamento delle parentesi
    if par_chiuse ≠ par_aperte then
        return +∞
    end if
▷ Seconda scansione per determinare il bilanciamento
    for x ∈ S do
        if x = ( then
            counter ← counter + 1
        else if x = ) and counter ≠ 0 then
            counter ← counter − 1
        end if
    end for
▷ Restituisce il numero di parentesi necessarie per bilanciare
    return counter
end function
```

Complessità

La complessità temporale risulta $O(n)$ per scansionare la sequenza e $O(n)$ per la scansione della sequenza mentre eseguiamo le operazioni di addizione e sottrazione sul counter per un totale di $O(2n) \sim O(n)$.
La complessità spaziale risulta $O(1)$ in quanto l'algoritmo utilizza memoria ausiliaria costante dipendente dal numero costante di variabili utilizzate.

Correttezza

La correttezza dell'algoritmo è conseguenza della corretta gestione dei diversi casi in input. Entrando più nello specifico, le problematiche gestite riguardano principalmente due casi:
1. $|par_{aperte}| \neq |par_{chiuse}|$

oppure

2. S risulta k-bilanciabile

1) Numero delle parentesi aperte diverso dal numero delle parentesi chiuse

Dopo il primo ciclo for (in cui vengono contate le parentesi chiuse e aperte), nel caso in cui $|par_{aperte}| \neq |par_{chiuse}|$ l'algoritmo restituisce $+\infty$ in quanto aggiungendo k parentesi chiuse e aperte, la sequenza di parentesi rimarrebbe comunque dispari lasciando **almeno** una parentesi *non accoppiata*. Un esempio banale può essere la seguente stringa: **(- (- (- (-) -)**
Contando le parentesi notiamo che sono presenti:
- quattro parentesi *aperte*
- due parentesi *chiuse*

Ci accorgiamo immediatamente che le parentesi *non accoppiate* sono le prime due. Proviamo ad aggiungere $k = 2$ parentesi per accoppiarle.
La nostra stringa diventa: **(- (- (- (- (- (-) -) -) -)**
Rendiamola più leggibile e evidenziamo gli accoppiamenti: **(- (- (((( ))))**
La stringa rimane non bilanciata dato che per ogni *k-aggiunta* la sequenza avrà sempre *almeno una* parentesi non accoppiata, quindi:

$$\nexists k \in \mathbb{N} \text{ tale che } S \text{ è una sequenza } k\text{-}bilanciabile$$

2) La stringa risulta bilanciata o "apparentemente" bilanciata

Nel primo caso la stringa data in input è già bilanciata. Tramite la nostra implementazione del counter l'algoritmo restituisce *counter = 0* in quanto abbiamo bisogno di $k = 0$ parentesi per bilanciare la sequenza 0-*bilanciabile*.
Esempio: **(- (-) -)**

Nel secondo caso, dato che non possiamo aggiungere un numero negativo di parentesi, gestiamo il problema del $counter \leq 0$.
Il problema si pone quando $counter = 0$ e incontriamo una parentesi chiusa. In questo caso ci si aspetta che $counter = -1$ ma con la nostra implementazione il counter non è soggetto a sottrazioni nel caso in cui dovesse diventare negativo.
Un esempio di stringa inerente è la seguente: **) -) - (- (**
Senza il nostro "*workaround*" il *counter* sarebbe stato:
1. $counter = -1$
2. $counter = -2$
3. $counter = -1$
4. $counter = 0$

Il $counter = 0$ e la sequenza risulta apparentemente 0-*bilanciabile* ma in realtà è 2-*bilanciabile*.