Soluzioni per il Problem Set 1

Davide Lights, Ginevra Bru, Alexandra asdf

04-12-2024

1 Problema 1

1.1 Codice

```
algoritmo bilancia(seq S):
    Sia n il numero di elementi in S

    h = 0
    k = 0
    for i = 0 to n do:
        if S[n] == '(' then
            h+=1
        else
            if h > 0 then
                h-=1
            else k+=1

    if h == k then
        return h
    else
        return inf
```

1.2 Tempo di Esecuzione

Il codice viene eseguito in tempo $\Theta(n)$, dato che itero su ogni carattere della sequenza. La memoria ausiliaria usata e' costante, non vado a richiedere memoria all'interno del ciclo 'for'.

1.3 Correttezza

1.3.1 Lemma

Una sequenza S, e' k-bilanciabile se e' solo se ha lo stesso numero h di parentesi aperte (che devono essere chiuse) e k di parentesi chiuse (che devono essere aperte).

Dimostrazione Per assurdo: $k \neq h \rightarrow$ la sequenza S e' k-bilanciabile. Provo a bilanciare la S con k. Indico con k' e h' le parentesi aperte e chiuse (da chiudere e aprire) dopo che ho bilanciato la sequenza. Posso ricavare k' e h' cosi:

- $k' = k - k = 0$, quindi ho chiuso tutte le parentesi aperte!
- $h' = h - k > 0$, quindi ho almeno una parentesi chiusa che deve essere aperta, la dimostrazione termina qui.

Al contrario, per assurdo: la sequenza S e' k-bilanciabile $\rightarrow k \neq h$. Indico con k' e h' le parentesi aperte e chiuse (da chiudere e aprire) dopo che ho bilanciato la sequenza. Dato che S e' k-bilanciabile mi aspetto che $k' = h' = 0$. Come prima mi posso ricavare k' e h':

- $k' = 0 = k - k$
- $h' = 0 = h - k$, ma $h \neq k$ quindi deve necessariamente essere che $k == h$

La dimostrazione funziona anche se provo a bilanciare con h parentesi.

1.3.2 Conclusione

La correttezza dell'algoritmo viene dal Lemma che viene applicato nel codice

2 Problema 2

2.1 Codice

```
algoritmo find_delta(d, t, M, j):
    Sia n il numero di elementi nell array.

    # -- CASO BASE --
    if n-j == 1:
        d1 = M - t[n-1] - 1
        if d1 < d:
            return -1
        return d1

    # -- CASO RICORSIVO --
    d1 = find_delta(d, t, M, j+1)

    if t[j] + d1 > t[j+1]:
        d1 = (M - t[j] - 1) // (n-j)
        if d1 < d:
            return -1

    return d1


# metodo wrapper per find_delta
procedura alg_find_delta(d, t, M):
    return find_delta(d, t, M, 0)
```

2.2 Tempo di Esecuzione

E' stato utilizzato un metodo ricorsivo, la ricorsione e' del tipo:

$$T(k) = \begin{cases} O(1) & \text{se } k = n-1 \\ T(k+1) + O(1) & \text{altrimenti} \end{cases}$$

per $0 \leq k < n$, segue che l'algoritmo viene eseguito in tempo $\Theta(n)$, verifichiamo che sia un tempo accettabile secondo l'upper-bound $o(n \cdot M)$, usando la definizione di o-piccolo.

Per definizione $M \geq n \cdot \Delta$, quindi M cresce almeno come n (e quindi $M = \Omega(n)$). Ipotizzando che M cresca esattamente come n ($M \sim n$) allora, dalla definizione di o-piccolo:

$$\lim_{n\to\infty} \frac{n}{n \cdot M} = \frac{1}{M} \sim \frac{1}{n} = 0$$

Nel caso in cui M cresca ancora più velocemente di n, per esempio $n^2, n^3, \ldots$ il limite tende sempre a 0, tenendo presente che M non e' mai $< \Delta \cdot n$.

2.3 Correttezza

Nel codice, i tempi di esecuzione si trovano in un array dove l'indice 0 corrisponde al primo cliente, e l'indice $n-1$ corrisponde all'ultimo cliente. Possiamo dimostrare la correttezza dell'algoritmo analizzando il caso base e il caso induttivo.

2.3.1 Caso Base

Per $k = n - 1$, ricado nel caso base, calcolo il valore massimo di Δ' per servire un solo cliente (l'ultimo), questo valore e' $\Delta' = M - t_{\text{n-1}} - 1$.

2.3.2 Caso Induttivo

Ipotizzando che Δ'_{k+1} sia il valore massimo trovato al passo $k+1$, devo vedere se questo valore va bene anche al passo k o se devo cambiarlo, ho due scenari possibili.

Scenario 1 $t_k + \Delta'_{k+1} > t_{k+1}$: questo vuol dire che se servo il cliente k con tempo Δ'_{k+1}, il cliente $k+1$ verrà messo in coda. Se servo i clienti da k a n con il tempo Δ'_{k+1} andrò sicuramente oltre il tempo limite M, il tempo in cui "inizio a servire" almeno uno dei clienti dopo k e' cambiato, devo trovare un nuovo Δ'. Similmente a come faccio per il caso base, il delta massimo possibile e': $\Delta'_k = (M - t_k - 1)/(n - k)$, dove $\Delta'_k < \Delta'_{k+1}$.

Scenario 2 $t_k + \Delta'_{k+1} \leq t_{k+1}$: questo vuol dire che se servo il cliente k con tempo Δ'_{k+1}, mi avanza del tempo che potrei sfruttare per non fare nulla. Ma non posso incrementare Δ'_{k+1} dato che questo e' il delta massimo per servire i clienti da $k+1$ a n entro il tempo limite M, quindi: $\Delta'_k = \Delta'_{k+1}$.