Batterie

State viaggiando con una modernissima auto elettrica su un'autostrada, entrate al km 0 con la batteria carica ed dovete uscire al km N. L'autonomia della batteria è r km; esistono n aree di servizio, ai km $d[1 \ldots n]$, dove la vostra batteria può essere sostituita con una carica. Descrivete un algoritmo che minimizzi il numero di soste, dimostrandone la correttezza e discutendone la complessità.

Donald Trump

La Route 66 è una strada che collega Chicago a Los Angeles e che contiene un totale di n città. Nel suo prossimo tour elettorale Donald Trump (The Donald) ha deciso di seguire la Route 66 in una direzione, senza mai tornare indietro sui propri passi. The Donald non può tenere un comizio in ogni città, ma deve sceglierne un sottoinsieme.

Date due città i, j in cui terrà comizi, esse devono trovarsi ad una distanza superiore o uguale a D. La città i-esima si trova al miglio $m[i]$; quindi la distanza fra i e j è pari a $m[j] - m[i]$, con $i < j$.

Seguendo queste regole, The Donald vorrebbe parlare al maggior numero di elettori. Si stima che al comizio nella città i-esima saranno presenti $e[i]$ elettori.

Donald Trump

Sorprendentemente, The Donald non ha grandi conoscenze informatiche, e ha chiesto a voi di risolvere il problema; in particolare, vorrebbe un algoritmo che restituisca il maggior numero di elettori che possono essere presenti seguendo le regole di cui sopra, dati un vettore di posizioni delle città $m[]$ e un vettore di numero di elettori $e[]$. Scrivere l'algoritmo e commentatene correttezza e complessità. Nel caso vi rifiutaste di risolvere il problema per conto di The Donald, riceverete $+1$ punto bonus.[1]

[1] Non è vero, è uno scherzo.

Scheduling

Siano dati n job da sottomettere ad un processore; il job i-esimo ($1 \leq i \leq n$):

- ha una deadline positiva intera $D[i]$
- un guadagno positivo intero $G[i]$
- un tempo di esecuzione pari a 1 (uguale per tutti)

Se il job i è eseguito entro l'istante $D[i]$ darà un guadagno $G[i]$, altrimenti il guadagno è 0. Trovare una sequenza di esecuzione che massimizzi il guadagno.

---

maxgain(**integer**[ ] D, **integer**[ ] G, **integer** n)

```
{ ordina i vettori D, G per guadagno decrescente }
integer t = 1
for i = 1 to n do
    if t ≤ D[i] then                % Job i può essere eseguito al tempo t
        print i
        t = t + 1
```

- Provare che l'algoritmo proposto non è corretto

Sciatori

Siano dati n sciatori di altezza $p_1, \ldots, p_n$, e n paia di sci di lunghezza $s_1, \ldots, s_n$. Il problema è assegnare ad ogni sciatore un paio di sci, in modo da minimizzare la differenza totale fra le altezza degli sciatori e la lunghezza degli sci; ovvero, se allo sciatore i è assegnato il paio di sci $h(i)$, minimizzare la seguente quantità:

$$\sum_{i=1}^{n} |p_i - s_{h(i)}|$$

Si consideri il seguente algoritmo greedy. Si individui la coppia (sciatore, sci) con la minima differenza. Si assegni allo sciatore questo paio di sci. Si ripete con gli sciatori restanti fino a quando non si è terminato.
Provare la correttezza di questo algoritmo o trovare un controesempio.

Prima elementare

A mia figlia (prima elementare) è stato chiesto di disegnare tutte le possibili sequenze composte da tre pallini rossi e due pallini gialli.

1. Scrivere un algoritmo che stampa tutte le possibili stringhe composte da n caratteri R e da m caratteri G, per un totale di $n + m$ caratteri.
2. Scrivere un algoritmo che conta tutte queste possibile stringhe – ovviamente senza generarle tutte e poi contandole.

Complessità correttezza blah blah

Spoiler alert!

Batterie

Per minimizzare il numero di fermate, si prosegue fino all'ultima stazione di servizio possibile, prima di rimanere a secco. Per semplificare il codice, supponiamo che esista un 'ultima fermata $D[n+1]$, come sentinella. Inoltre, assumiamo che non esistano due stazioni di servizio distanti più di r km (altrimenti l'autostrada non può essere percorsa).

SET fermate(**integer**[ ] D, **integer** n)

```
deadline = r
SET stops = Set()
for i = 1 to n do
    if D[i] ≤ deadline and D[i+1] > deadline then
        stops.insert(i)
        deadline = D[i] + r
return stops
```

Donald Trump

$$best[j] = \begin{cases} e[1] & j = \\ \max\{best[j-1], e[j] + \max_{1 \le i < j \wedge m[j] - m[i] \ge D} best[i]\} & j > \end{cases}$$

Donald Trump

integer serveTheDonald(**integer**[ ] m, **integer**[ ] e, **integer** n, **integer** D)

$best[1] \leftarrow e[1]$
for $j \leftarrow 2$ **to** n **do**
    $best[j] \leftarrow best[j-1]$
    integer $i \leftarrow 1$
    while $i < j$ **and** $m[j] - m[i] \geq D$ **do**
        $best[j] \leftarrow \max(best[j], e[j] + best[i])$
        $i \leftarrow i + 1$
return $best[n]$

Scheduling

1. Si considerino due job, uno con guadagno 2 e deadline 2 e uno con guadagno 1 e deadline 1. Eseguendo prima il primo job, come da algoritmo, si può eseguire solo quello e il guadagno è 2; eseguendo invece prima il secondo e poi il primo, si ottiene un guadagno di 3. Questo dimostra che l'algoritmo greedy non è corretto.

Sciatori

Si consideri il seguente input: $p = \{5, 10\}$, $s = \{9, 14\}$. Secondo l'algoritmo, associamo $p[2]$ ad $s[1]$ e $p[1]$ ad $s[2]$ (ovvero $h(1) = 2$, $h(1) = 2$. La differenza totale è $|10 - 9| + |14 - 5| = 10$. Se l'associazione fosse $h(1) = 1$ e $h(2) = 2$, la differenza totale sarebbe: $|9 - 5| + |10 - 14| = 8$. Quindi l'algoritmo proposto non è corretto.

Prima elementare

stampaCombinazioni(**char**[ ] V, **integer** i, **integer** n, **integer** m)

if $n = 0$ **and** $m = 0$ **then**
  print V

if $n > 0$ **then**
  $V[i] =$ "R"
  stampaCombinazioni$(V, i + 1, n - 1, m)$

if $m > 0$ **then**
  $V[i] =$ "G"
  stampaCombinazioni$(V, i + 1, n, m - 1)$

Prima elementare

calcolaCombinazioniRic(**integer** n, **integer** m)

if $n == 0$ **or** $m == 0$ **then**
    return 1
else
    return
        calcolaCombinazioniRic$(n-1, m)$ + calcolaCombinazioniRic$(n, m-1)$

Prima elementare

calcolaCombinazioni(**integer** n, **integer** m)

integer[ ][ ] $M =$ **new integer**$[0 \ldots n][0 \ldots m]$
for $i = 0$ **to** n **do**
  $M[i, 0] = 1$
for $j = 0$ **to** m **do**
  $M[0, j] = 1$
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    $M[i, j] = M[i - 1, j] + M[i, j - 1]$
return $M[n, m]$