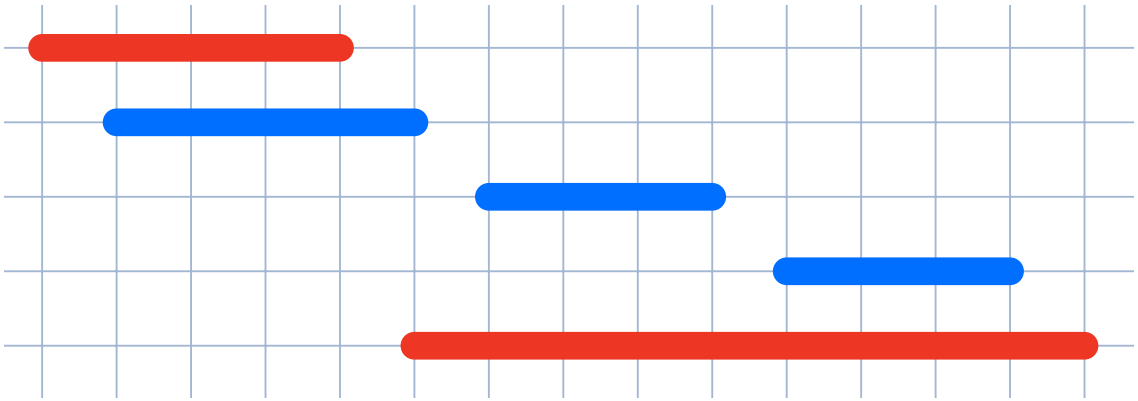
Massima copertura

Si considerino n segmenti sulla retta delle ascisse, dove l'i-esimo segmento inizia nella coordinata $a[i]$ (inclusa) e termina nella coordinata $b[i]$ (esclusa).

Scrivere un algoritmo che prenda in input i vettori a, b e la dimensione n, e restituisca il sottoinsieme di segmenti indipendenti (che non si intersecano) di copertura massimale, ovvero che copre la porzione più grande della retta delle ascisse.

Valutare il costo computazionale dell'algoritmo proposto.



Palindroma

Una stringa si dice palindroma se è uguale alla sua trasposta, cioè se è identica se letta da sinistra e destra o da destra a sinistra.
Scrivere un algoritmo minpal(ITEM[] s) che ritorna il numero minimo di caratteri da inserire in s per rendere s palindroma.

Per esempio, input: "casacca":

- $n = 7$ caratteri: "casaccaACCASAC"
- $n = 6$ caratteri: "casaccaCCASAC"
- $n = 3$ caratteri: "casaccaSAC"
- $n = 2$ caratteri: "ACcasacca"

Notate che non necessariamente i caratteri si inseriscono in testa o in fondo; per esempio, "anta" $\rightarrow$ "antNa".

D20

Siano dati n dadi, con il dado i-esimo dotato di $F[i]$ facce numerate da 1 a $F[i]$. Scrivere un algoritmo che restituisca il numero di modi diversi con cui è possibile ottenere una certa somma X sommando i valori di tutti i dadi.

- Ad esempio, avendo due dadi a quattro facce numerati da 1 a 4, il valore 7 è ottenibile in un solo modo non contando le possibili permutazioni: $3 + 4$.
- Avendo tre dadi, i primi due a 4 e l'ultimo a 6 facce, il valore 8 è ottenibile in cinque modi diversi non contando le possibili permutazioni: $1 + 1 + 6$, $1 + 2 + 5$, $1 + 3 + 4$, $2 + 2 + 4$, $2 + 3 + 3$.

Permutazioni: contatele oppure no, ma siatene consapevoli...

Spoiler alert!

Massima copertura (Compito 05/06/2014)

Questo problema è molto simile al problema dell'insieme indipendente di intervalli pesati visto a lezione, dove il peso $w[i]$ è pari a $b[i] - a[i]$.

Si risolve quindi con la costruzione di un vettore w di pesi e una singola chiamata alla soluzione già proposta, con costo computazionale pari a $\Theta(n \log n)$ (dovuto all'ordinamento, se i vettori a, b non sono già ordinati per tempo di fine).

SET segmentcover(**int**[ ] a, **int**[ ] b, **int** n)

---

int[ ] w = **new int**[$1 \ldots n$]
for $i = 1$ **to** n **do**
  $w[i] = b[i] - a[i]$
return maxinterval(a, b, w, n)

Palindroma (Soluzione 1.15 di 13-pd.pdf)

- Se $s = \mathrm{a}s'\mathrm{a}$ è composta da due identici caratteri "a" iniziale e finale, allora:

  $$minpal(s) = minpal(s')$$

- Se $s = \mathrm{a}s'\mathrm{b}$ ha due caratteri iniziale e finale diversi
  - o aggiungiamo o un carattere "b" in testa (e consideriamo il problema $\mathrm{a}s'$, eliminando virtualmente il carattere b),
  - oppure aggiungiamo un carattere "a" in coda (e consideriamo il problema $s'\mathrm{b}$, eliminando virtualmente il carattere a).

  Scegliamo fra le due possibilità quella con costo minore. In entrambi i casi, dobbiamo sommare 1 per il carattere aggiunto.

  $$minpal(s) = \min\{minpal(\mathrm{a}s'), minpal(s'\mathrm{b})\} + 1$$

- Se s contiene un carattere solo o nessuno carattere, s è palindroma e bisogna rispondere 0.

Palindroma

Sia $DP[i][j]$ il numero di caratteri che è necessario inserire per rendere palindroma la stringa $s[i \dots j]$; può essere calcolato in modo ricorsivo nel modo seguente:

$$DP[i][j] = \begin{cases} 0 & i \geq j \\ DP[i+1][j-1] & i < j \textbf{ and } s[i] = s[j] \\ \min(DP[i+1][j], DP[i][j-1]) + 1 & i < j \textbf{ and } s[j] \neq s[j] \end{cases}$$

Palindroma

```
int minpal(Item[] s, int n)
int[][] DP = new int[1...n][1...n] = {-1}    % Initialized to -1
return minpalRec(s, DP, 1, n)
```

```
int minpalRec(Item[] s, int[][] DP, int i, int j)
if j ≤ i then
    return 0
else
    if DP[i][j] < 0 then
        if s[i] == s[j] then
            DP[i][j] = minpalRec(s, i + 1, j - 1)
        else
            DP[i][j] = min(minpalRec(s, i, j-1), minpalRec(s, i+1, j))+1
    return DP[i][j]
```

Palindroma

Un possibile approccio alternativo si basa sull'algoritmo LCS.
Data una stringa s, si consideri la più lunga sottosequenza comune $LCS(s, s')$, dove s' è l'inversa della stringa s. Per esempio, se $s =$ `"ANTA"`, $s' =$ `"ATNA"`, $LCS(s, s') =$ `ANA`. Tale LCS è la più lunga sottosequenza palindroma contenuta in s. Si tratta quindi di aggiungere i caratteri che mancano per rendere palindroma s; se m è la lunghezza di tale LCS, restituiremo $n - m$.

D20 (Compito 02/09/13)

Utilizziamo la programmazione dinamica. Sia $DP[x][i]$ il numero di modi in cui è possibile ottenere un valore x con i primi i dadi:

$$DP[x][i] = \begin{cases} \sum_{j=1}^{F[i]} DP[x-j][i-1] & x > 0 \textbf{ and } i > 0 \\ 1 & x = 0 \textbf{ and } i = 0 \\ 0 & \text{altrimenti, incluso } x < 0 \end{cases}$$

D20

Il problema di questa versione è che conta tutte le possibili permutazioni; per ovviare a questo problema, è possibile aggiungere un terzo parametro m che indica il valore minimo del dado che può essere considerato, che deve essere più alto o uguale dei valori già ottenuti:

$$DP[x][i][m] = \begin{cases} \sum_{j=m}^{F[i]} DP[x-j][i-1][j] & x > 0 \textbf{ and } i > 0 \\ 1 & x = 0 \textbf{ and } i = 0 \\ 0 & \text{altrimenti, incluso } x < 0 \end{cases}$$

D20

```
int diceRec(int[] F, int i, int x, int m, int[][][] T)
if x == 0 and i == 0 then
    return 1
else if x > 0 and i > 0 then
    if DP[x][i][m] < 0 then                          % Memoization check
        DP[x][i][m] = 0
        for j = m to F[i] do
            DP[x][i][m] = DP[x][i][m] + diceRec(F, i − 1, x − j, j, DP)
    return DP[x][i][m]
else
    return 0
```

D20

int dice(**int**[ ] F, **int** n, **int** X)

$M = \max(F, n)$
int[ ][ ][ ] DP = **new int**$[1 \dots n][1 \dots X][1 \dots M] = \{-1\}$
return diceRec$(F, n, X, 1, DP)$

Il costo è pari a $O(nXM^2)$, dove M è il dado con il maggior numero di facce. Questo perchè ci sono nXM celle da riempire, ognuna delle quali viene riempita con costo $O(M)$.

D20 - Si può fare meglio di così

$$DP[x][i][m] = \begin{cases} DP[x-m][i-1][m] + DP[x][i][m+1] & i > 0 \textbf{ and } x > 0 \textbf{ and } m < F[i] \\ DP[x-m][i-1][m] & i > 0 \textbf{ and } x > 0 \textbf{ and } m = F[i] \\ 1 & x = 0 \textbf{ and } i = 0 \\ 0 & \text{altrimenti} \end{cases}$$

- La semantica di $DP[x][i][m]$ è la stessa di quella precedente.
- Se $m < F[i]$, ci sono ancora dadi e c'è un valore $x > 0$ da ottenere, è possibile scegliere fra: selezionare il valore m per il dado i-esimo, togliendo tale valore da x e considerando quindi cosa succede con $i - 1$ dadi; oppure considerare il dado i innalzando il valore di m.
- Se $m = F[i]$, il secondo caso non è possibile.
- I casi base sono uguali
- Costo per riempire la tabella in questo modo: $O(nXM)$.