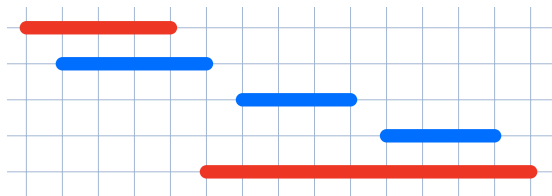


Massima copertura

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 maggior parte della retta delle ascisse.

Valutare il costo computazionale dell'algoritmo proposto.



I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda rincomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?

I Promessi Sposi

"Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene non interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del rientrare di quelli, vien, quasi a un tratto, a restringersi, e a prender corso e figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte; e il ponte, che ivi congiunge le due rive, par che renda ancor più sensibile all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda ricomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo, lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni."

Quante volte questo testo contiene la sottosequenza "lucia"?

Alcune considerazioni:

- Due sottosequenze sono distinte (e quindi vanno contate separatamente) se esiste almeno una differenza negli insiemi di caratteri utilizzati.
- Esempio: "did you go" contiene due volte la sottosequenza "dog"....

I Promessi Sposi

Scrivere un algoritmo che prenda in input una stringa testo T di n caratteri e una stringa pattern P di m caratteri e restituisca il numero di volte distinte che la stringa P appare come sottosequenza di T .

Discutere correttezza e complessità dell'algoritmo.

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 a quella soluzione, in un tempo 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 .. n]
```

```
for i = 1 to n do
```

```
     $w[i] = b[i] - a[i]$ 
```

```
return maxinterval(a, b, w, n)
```

I Promessi Sposi

Sia $DP[i][j]$ il numero di occorrenze del prefisso j -esimo del pattern $P(j)$ come sottosequenza del prefisso i -esimo del testo $T(i)$.

$$DP[i][j] = \begin{cases} 0 & i = 0 \text{ and } j > 0 \\ 1 & j = 0 \\ DP[i-1][j] + DP[i-1][j-1] & i > 0 \text{ and } j > 0 \text{ and } T[i] = P[j] \\ DP[i-1][j] & i > 0 \text{ and } j > 0 \text{ and } T[i] \neq P[j] \end{cases}$$

- Se il testo è finito ($i = 0$) e il pattern non è vuoto ($j > 0$), non siamo riusciti a trovare il pattern
- Se il pattern è vuoto ($j = 0$), significa che siamo riusciti a trovare il pattern, e lo contiamo per uno
- Se l'ultimo carattere del testo e del pattern sono uguali, possiamo sfruttare quest'uguaglianza oppure no; i due casi vanno sommati
- Se l'ultimo carattere del testo e del pattern sono diversi, ignoriamo l'ultimo carattere del testo.

I Promessi Sposi

Utilizziamo un vettore $DP[0 \dots m]$ invece che una matrice, in quanto il valore si ottiene a partire dalla sola riga precedente, che viene memorizzata in DP' .

Complessità in tempo: $\Theta(mn)$, complessità in spazio $\Theta(m)$

```
int lucia(ITEM[]  $T$ , ITEM[]  $P$ , int  $n$ , int  $m$ )
```

```
int[]  $DP$  = new int[0... $m$ ]
```

```
int[]  $DP'$  = new int[0... $m$ ]
```

```
 $DP[0] = 1$ 
```

```
for  $j = 1$  to  $m$  do  $DP[j] = 0$ 
```

```
for  $i = 1$  to  $n$  do
```

```
    for  $j = 0$  to  $m$  do  $DP'[j] = DP[j]$ 
```

```
    for  $j = 1$  to  $m$  do
```

```
        if  $T[i] == P[j]$  then
```

```
             $DP[j] = DP'[j] + DP'[j - 1]$ 
```

```
        else
```

```
             $DP[j] = DP'[j]$ 
```

```
return  $DP[m]$ 
```

5 Maggio e Promessi Sposi

Subject: Sottosequenza Promessi sposi
Date: Fri, 11 Dec 2015 00:25:41 +0100
To: Alberto Montresor <alberto.montresor@unitn.it>

Il 5 maggio nei promessi sposi senza considerare spazi, punteggiatura e numeri, ma considerando gli accenti ci sta:

```
21975465301516630979573617593825769513857583563025262379789778337947615817191757
43398428321975621542396623347442197637158184228160098596758725678010178001365659
73566045203119340799723612777962220975263078675519750712637479432237655210391918
48601874737423942438531018213728179566210700422537584195776715536664949343794694
74341304486367721199869205484517178136400988317581077715393614892844560303556628
57753722957658724970416074137670807561854959314707635812057348445333068267511860
81613043221797286605904378408112853795888506693820006728695515750235630153301285
93082577269020619288952011873970086359263850877345042074027243309950696549344467
34109698508036913355586284550592994928187284736568396263368466837671143105426910
99608601301418040383501823489133955578653269527834272234364741431604038516826654
82045722458282856692545688317000656065623166181081105288235575975572352074726528
75237693750708795898738364470324968401496146509587976160485483512050002468507459
5216118769351618590697862390987987264814086004487458833092023307
```

Allego lo script in python che ha generato il risultato (con testo completo dei promessi sposi e del 5 maggio). Ovviamente ha tempi assurdi per svariati motivi.

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:

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

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:

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

```

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

```

```
int dice(int[]  $F$ , int  $n$ , int  $X$ )
```

```
 $M = \max(F, n)$ 
```

```
int[][][]  $DP = \text{new int}[1 \dots n][1 \dots X][1 \dots M] = \{-1\}$ 
```

```
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

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

- 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: $O(nXM)$.