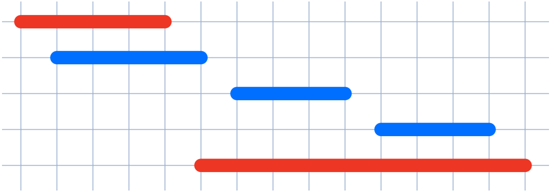
Massima copertura (Compito 05/06/2014)

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 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"?
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.

D20 (Compito 02/09/13)

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 sempre a 4 facce, il valore 6 è ottenibile in tre modi diversi non contando le possibili permutazioni: $1 + 1 + 4$, $1 + 2 + 3$, $2 + 2 + 2$.

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

Costo partizione di un vettore (Compito 05/06/14)

- Il costo $C(i, j)$ di un sottovettore $V[i \ldots j]$ di V è pari alla somma dei suoi elementi
  - Esempio: $V = [1, 2, 4, 8, 16, 2]$, $C(2, 4) = 2 + 4 + 8 = 14$.
- Una k-partizione di V è una divisione di V in k sottovettori non vuoti, contigui che coprono totalmente il vettore e non si sovrappongono.
  - Esempio: $V = [2, 3, 7, -7, 15, 2]$, una delle $n - 1$ 2-partizioni è $[2, 3, 7], [-7, 15, 2]$.
- Il costo della k-partizione è il costo massimo dei suoi sottovettori.
  - Nella 2-partizione $[2, 3, 7], [-7, 15, 2]$, il costo dei sottovettori è
    $2 + 3 + 7 = 12$,
    $-7 + 15 + 2 = 10$,
    quindi il costo della 2-partizione è pari a 12.

Costo partizione di un vettore (Versione formale)

- Il costo $C(i, j) = \sum_{t=i}^{j} V[t]$
- k-partizione di V è una divisione di V in k sottovettori contigui $V[j_0 + 1 \ldots j_1], V[j_1 + 1 \ldots j_2], \ldots, V[j_{k-1} + 1 \ldots j_k]$ con $j_0 = 0, j_k = n$ e $j_t < j_{t+1}, \forall 1 \leq t < k$.
- Il costo della k-partizione è pari a:

$$\max\{C(j_{t-1} + 1, j_t) : \forall 1 \leq t < k\}$$

Costo partizione di un vettore (Compito 05/06/14)

Scrivere un algoritmo che prenda in input un vettore V contenente n interi e un intero k tale che $2 \leq k \leq n$ e restituisca il costo della k-partizione di V di costo minimo.

Esempio: $V = [2, 3, 7, -7, 15, 2]$, $k = 3$
$[2, 3, 7], [-7, 15], [2]$ costo 2+3+7=12
$[2, 3], [7][-7, 15, 2]$ costo -7+15+2=10

1. Soluzione per $k = 2$ (facile: $O(n^2)$, meglio in $O(n)$).
2. Soluzione per $k = 3$ (facile: $O(n^3)$, meglio in $O(n^2)$).
3. Soluzione generale (facile: $O(n^k)$, meglio in $O(kn^2)$).

Quadrato binario (Compito 10/09/12)

Sia A una matrice $n \times n$ di valori booleani 0/1. Scrivere un algoritmo che prenda in input la matrice A e la sua dimensione n, e restituisca la dimensione del più grande quadrato composto da valori 1 contenuto nella matrice. Ad esempio, nella matrice seguente, i quadrati di dimensione massima sono grandi 4×4 (ve ne sono due, di cui uno evidenziato in rosso).

```
1 0 1 0 1 0 0
1 0 1 1 1 1 0
0 1 1 1 1 1 0
0 0 1 1 1 1 0
1 1 1 1 1 1 0
1 1 1 1 1 1 0
```

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 $T(i)$.

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

- 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.

```
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] == T[j] then
            DP[j] = DP'[j] + DP'[j − 1]
        else
            DP[j] = DP'[j]
return DP[m]
```

5 Maggio e Promessi Sposi

```
---- Forwarded Message ----
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
```

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

$\textbf{int}\ \mathsf{dice}(\textbf{int}[\,]\ F, \textbf{int}\ n, \textbf{int}\ X)$
$M = \mathsf{max}(F, n)$
$\textbf{int}[\,][\,][\,]\ DP = \textbf{new int}[1 \ldots n][1 \ldots X][1 \ldots M] = \{-1\}$
$\textbf{return}\ \mathsf{diceRec}(F, n, X, 1, DP)$

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: $O(nXM)$.

Costo partizione di un vettore (Compito 05/06/14)

- Vediamo innanzitutto un approccio generico al problema, estremamente inefficiente per valori grandi di k.
- Proponiamo poi soluzioni efficienti "ad-hoc" per $k = 2$ e $k = 3$, "propedeutiche" alla costruzione di una soluzione generale.
- Infine, vediamo una soluzione generica che può essere utilizzata per qualunque valore di k, basata su programmazione dinamica

Costo partizione di un vettore (Versione $\Theta(n^k)$)

int $\mathsf{partitionK}(\textbf{int}[\,]\ V, \textbf{int}\ n)$

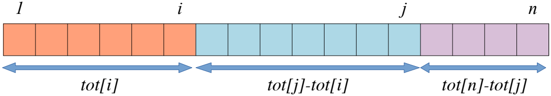
int $minSoFar = +\infty$
for $i_1 = 1$ **to** n **do**
  for $i_2 = i_1 + 1$ **to** n **do**
    for $i_3 = i_2 + 1$ **to** n **do**
      for $\ldots = \ldots + 1$ **to** n **do**
        for $i_{k-1} = i_{k-2} + 1$ **to** n **do**
          $tot_1 = \mathsf{sum}(V, 1, i_1)$
          $tot_2 = \mathsf{sum}(V, i_1 + 1, i_2)$
          $tot_3 = \mathsf{sum}(V, i_2 + 1, i_3)$
          $\ldots$
          $tot_k = \mathsf{sum}(V, i_{k-1} + 1, n)$
          $minSoFar = \mathsf{min}(minSoFar, \mathsf{max}(tot_1, \ldots, tot_k))$

2-partizione

```
int partition2(int[] V, int n)
int tot = 0
for i = 1 to n do
    tot = tot + V[i]
int sumSoFar = 0
int minSoFar = +∞
for i = 1 to n − 1 do
    sumSoFar = sumSoFar + V[i]
    minSoFar = min(minSoFar, max(sumSoFar, tot − sumSoFar))
return minSoFar
```

3-partizione

```
int partition3(int[] V, int n)
int[] tot = new int[1 . . . n]
tot[1] = V[1]
for i = 2 to n do
    tot[i] = tot[i − 1] + V[i]
int minSoFar = +∞
for i = 1 to n − 2 do
    for j = i + 1 to n − 1 do
        int temp = max(tot[i], tot[j] − tot[i], tot[n] − tot[j])
        minSoFar = min(minSoFar, temp)
return minSoFar
```



k-partizione

Sia $DP[i][t]$ il minimo costo associato al sottoproblema di trovare la migliore t-partizione nel vettore $V[1 \dots i]$. Il problema iniziale corrisponde a $DP[n][k]$ – ovvero trovare la migliore k-partizione in $V[1 \dots n]$. Sfruttiamo un vettore di appoggio *tot* definito come nel caso $k = 3$.

$$DP[i][t] = \begin{cases} +\infty & t > i \\ tot[i] & t = 1 \\ \min_{1 \leq j < i} \max(DP[j][t-1], tot[i] - tot[j]) & \text{altrimenti} \end{cases}$$

k-partizione

int partition(**int**[] V, **int** n, **int** k)

---

int[][] DP = **new int**$[1 \ldots n][1 \ldots k] = \{-1\}$ % Initizialized to -1
int[] tot = **new int**$[1 \ldots n]$
$tot[1] = V[1]$
for $i = 2$ **to** n **do**
    $tot[i] = tot[i-1] + V[i]$
return partitionRec(V, tot, DP, n, k)

k-partizione

```
int partitionRec(int[] V, int[] tot, int[][] DP, int i, int t)
if t > i then
    return +∞
else if t == 1 then
    return tot[i]
else if DP[i][t] < 0 then
    int DP[i][t] = +∞
    for j = 1 to i − 1 do
        int cost = max(partitionRec(V, tot, DP, j, t − 1), tot[i] − tot[j])
        DP[i][t] = min(DP[i][t], cost)
return DP[i][t]
```

k-partizione

Costo computazionale:

- La tabella ha dimensione $n \times k$
- Ogni cella richiede tempo $O(n)$ per essere riempita
- Il costo computazionale è quindi $O(kn^2)$

Quadrato binario (Compito 10/09/12)

1	0	1	0
1	1	1	1
0	1	1	1
1	1	1	1

1	0	1	0
1	1	1	1
0	1	2	2
1	1	2	3

Quadrato binario

$DP[i][j]$ contiene la dimensione del più grande quadrato composto da soli 1 il cui angolo in basso a destra sia nella posizione (i, j)

$$DP[i][j] = \begin{cases} 0 & A[i][j] = 0 \\ 1 & A[i][j] = 1 \wedge (i = 1 \vee j = 1) \\ \min\{DP[i-1][j], & \\ \qquad DP[i-1][j-1], & \\ \qquad DP[i][j-1]\} + 1 & \text{altrimenti} \end{cases}$$

Quadrato binario

```
int maxSquare(boolean[][] A, int n)
---------------------------------------------------------------
int[][] DP = new int[1...n][1...n]
for i = 1 to n do
    DP[i][1] = A[i][1]
    DP[1][i] = A[1][i]
for i = 2 to n do
    for j = 2 to n do
        if A[i][j] == 0 then
            DP[i][j] = 0
        else
            DP[i][j] = min(DP[i-1][j], DP[i-1][j-1], DP[i][j-1]) + 1
return max(A, n)                % Return max value in matrix, Θ(n^2)
```