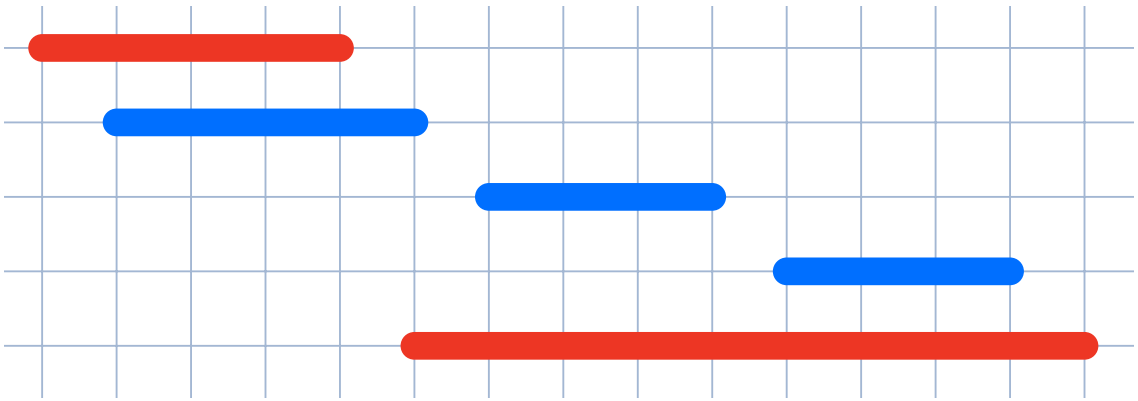
Massima copertura

Si considerino n segmenti sulla retta reale, dove l'i-esimo segmento inizia nella coordinata $s[i]$ (inclusa) e termina nella coordinata $f[i]$ (esclusa).

Scrivere una funzione SET maxCover(**int**[] s, **int**[] f, **int** n) che prenda in input i vettori s, f e la dimensione n, e restituisca un sottoinsieme di segmenti disgiunti la cui lunghezza totale coperta è massima.

Valutare correttezza e complessità dell'algoritmo proposto.



Palindroma

Una stringa si dice palindroma se è identica se letta da sinistra e destra o da destra a sinistra. Scrivere un algoritmo

int minpal(ITEM[] S, **int** n)

che restituisca il numero minimo di caratteri da inserire in S per rendere S palindroma.

Esempio: input `"casacca"`:

- 7 caratteri: `"casaccaaccasac"`
- 6 caratteri: `"casaccaccasac"`
- 3 caratteri: `"casaccasac"`
- 2 caratteri: `"accasacca"` (ottimo)

I caratteri possono essere inseriti in qualsiasi posizione, non solo all'inizio o alla fine della stringa. Esempio: `"anta"` $\rightarrow$ `"antna"`.

Quadrato binario

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 $f[i] - s[i]$.

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 s, f non sono già ordinati per tempo di fine).

SET maxCover(**int**[ ] s, **int**[ ] f, **int** n)

---

int[ ] w = **new int**[$0 \ldots n-1$]
for $i = 0$ **to** $n-1$ **do**
  $w[i] = f[i] - s[i]$
return maxinterval(s, f, 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:

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

Palindroma

```
int minpal(Item[] s, int n)
int[][] DP = new int[0...n-1][0...n-1] = {-1}   % Initialized to -1
return minpalRec(s, DP, 0, n-1)
```

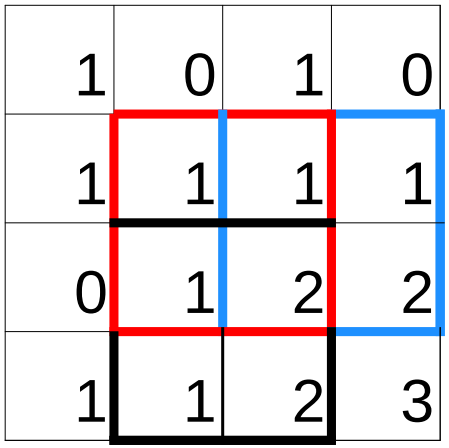
```
int minpalRec(Item[] s, int[][] DP, int i, int j)
if j ≤ i then
    return 0
else
    if DP[i][j] < 0 then
        if s[i] == s[j] then
            DP[i][j] = minpalRec(s, DP, i+1, j-1)
        else
            DP[i][j] =
              min(minpalRec(s, DP, i, j-1), minpalRec(s, DP, i+1, j)) + 1
```

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 $n - m$.

Quadrato binario (Compito 10/09/12)

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



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)

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

Quadrato binario

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

Quadrato binario – Alcuni errori tipici

- Utilizzare max() al posto di min()
- Non inizializzare il bordo