Esercizio 1 Data una stringa binaria $X = x_1x_2 \dots x_n$ per ogni coppia di interi i, p con $1 \leq i, p \leq n$, la più lunga sottosequenza di X di soli uni e del tipo $x_i x_{i+p} x_{i+2p} \dots x_{i+m \cdot p}$ è detta *periodica* di periodo p e inizio i, in breve (i,p)-*periodica*. Si consideri il seguente problema: data una stringa X determinare la lunghezza massima tra quelle delle sottosequenze periodiche di X.
Ad esempio: nella stringa $X = 000101100\underline{1}0\underline{1}0\underline{1}0\underline{1}1\underline{0}0\underline{1}11$ la sottosequenza $(2,2)$-periodica ha lunghezza 0, la sottosequenza $(6,1)$-periodica ha lunghezza 2 come la sottosequenza $(6,8)$-periodica. La sottosequenza $(6,4)$-periodica ha lunghezza 5 ed è anche la sottosequenza periodica di lunghezza massima per X.

1. Descrivere in pesudo–codice un algoritmo che risolve il problema con complessità $O(n^2)$ dove n è la lunghezza della stringa X.

Soluzione Esercizio 1

Usiamo il metodo della programmazione dinamica.
Per ogni $1 \leq i, p \leq n$ sia $T[i,p]$ pari alla lunghezza massima per le sottosequenze periodiche di $x_1x_2 \dots x_i$ che terminano nella posizione i ed hanno periodo p.
Il calcolo dei $T[i,p]$ è facile:
Per ogni $1 \leq i, p \leq n$

$$T[i,p] = \begin{cases} 0 & x_i = 0 \\ 1 & x_i = 1 \text{ e } i \leq p \\ T[i-p,p] + 1 & \text{altrimenti} \end{cases}$$

Ovviamente la soluzione al nostro problema è data da $\max_{1 \leq i,p \leq n}\{T[i,p]\}$. Quindi l'algoritmo può essere così descritto:

```
LUN-PERIODICA:  INPUT una stringa binaria X di lunghezza n
    FOR i ← 1 TO n DO
        FOR p ← 1 TO n DO
            IF x_i = 0 THEN T[i,p] ← 0
            ELSE
                IF i ≤ p THEN T[i,p] ← 1
                ELSE T[i,p] ← T[i-p,p] + 1
        ENDFOR
    ENDFOR
    SOL ← 0
    FOR i ← 1 TO n DO
        FOR p ← 1 TO n DO
            IF SOL < T[i,p] THEN SOL ← T[i,p]
        ENDFOR
    ENDFOR
OUTPUT SOL.
```

Il calcolo del valore di una singola cella della tabella T richiede $O(1)$. Di conseguenza il costo per calcolare tutti i valori della tabella è $= O(n^2)$ che è anche il costo per trovare il massimo tra questi valori. La complessità dell'algoritmo è quindi $O(n^2)$.

Esercizio 2 Si ha una sequenza $(g_1, g_2, \ldots, g_n)$ di oggetti tali che g_i ha peso p_i, $1 \leq i \leq n$, ed una sequenza di scatole $(S_1, S_2, \ldots, S_n)$ tutte con una stessa capacità C (vale a dire ogni scatola può contenere oggetti per un peso totale al più C) con $p_i \leq C$, $1 \leq i \leq n$. Si vogliono inserire gli oggetti nelle scatole in modo che per ogni coppia di oggetti g_i e g_j con $i < j$ l'indice della scatola in cui viene inserito g_i è minore o uguale all'indice della scatola in cui viene inserito g_j. L'obiettivo è minimizzare l'indice delle scatole utilizzate.

Ad esempio: si consideri la sequenza di 7 oggetti $(g_1, g_2, \ldots .g_7)$ di pesi $4, 2, 3, 5, 1, 2, 3$ e la sequenza di scatole $(S_1, S_2, \ldots, S_7)$ tutte di capacità $C = 6$. Indicando gli inscatolamenti come sequenza di indici delle scatole, un inscatolamento ammissibile è $(1, 1, 2, 3, 4, 4, 5)$ un inscatolamento non ammissibile è $(1, 1, 1, 3, 2, 4, 4)$ (poiché nella scatola di indice 1 il peso degli oggetti supera la capacità inoltre l'oggetto g_5 è stato inserito in una scatola di indice inferiore rispetto all'oggetto g_4). Due inscatolamenti ottimi sono $(1, 1, 2, 3, 3, 4, 4)$ e $(1, 1, 2, 3, 4, 4, 4)$.
Per risolvere il problema si propone il seguente algoritmo greedy:

INSCATOLA: INPUT il numero n di oggetti, i loro pesi $(p_1, \ldots p_n)$ e la capacità C delle scatole
$j \leftarrow 1$
$R \leftarrow C$
FOR $i \leftarrow 1$ TO n DO
  IF $p_i > R$ THEN
    $j \leftarrow j + 1$
    $R \leftarrow C$
  ENDIF
  $R \leftarrow R - p_i$
  $SOL[i] \leftarrow j$
ENDFOR
OUTPUT SOL.

- Dire se l'algoritmo INSCATOLA risolve il problema e in caso affermativo dimostrare la correttezza dell'algoritmo mentre in caso negativo fornire un controesempio.

Soluzione Esercizio 2 L'algoritmo INSCATOLA risolve il problema.
Per provare che l'algoritmo è corretto è sufficiente dimostrare che ogni nuova estensione della soluzione parziale mantiene la proprietà di essere contenuta in una soluzione ottima. Sia SOL_i la soluzione prodotta dall'algoritmo dopo l'i-esima iterazione del FOR. Vogliamo dimostrare

$$\textit{Per ogni } i \in \{1, 2, \ldots n\} \textit{ esiste una soluzione ottima } SOL^* \textit{ tale che } SOL_i[j] = SOL^*[j] \textit{ per } j \in \{1, 2, \ldots i\}.$$

Dimostrazione: per induzione su i.
Per $i = 0$ è banalmente vero poichè $SOL_1[1] = 1$ e $SOL^*[1] = 1$ in qualunque soluzione ottima.
Supponiamo ora che sia vero per i e dimostriamolo per $i + 1$.

Per ipotesi induttiva esiste una soluzione ottima SOL^* tale che $SOL_i[j] = SOL^*[j]$ per ogni $j \leq i$. Poiché SOL_{i+1} coincide con SOL_i nei primi i assegnamenti vale anche $SOL_{i+1}[j] = SOL^*[j]$ per ogni $j \leq i$. Per quanto detto se vale anche $SOL_{i+1}[i + 1] = SOL^*[i + 1]$ allora abbiamo fatto.
Sia j la scatola in cui SOL_i e SOL^* hanno inserito l'i-esimo oggetto. All'$i + 1$-esima iterazione dell'algoritmo sono possibili due casi:
CASO 1: all'$i + 1$-esima iterazione l'algoritmo cambia scatola (vale a dire $SOL_{i+1}[i + 1] = j + 1$). Faremo vedere che in questo caso deve aversi anche $SOL^*[i + 1] = j + 1$.

Nota che se l'algoritmo ha cambiato scatola è perchè nella scatola j non c'è posto a sufficienza per l'oggetto g_{i+1} ma in base all'ipotesi induttiva nella soluzione ottima nella scatola j sono stati inseriti tutti gli oggetti che vi ha posto l'algoritmo e quindi anche nella soluzione ottima in j non c'è posto per l'oggetto g_{i+1}. Questo significa che nel passare da g_i a g_{i+1} la scatola cambia anche nella soluzione ottima. Per ipotesi induttiva sappiamo che $SOL^*[i] = j$, deve aversi quindi $SOL^*[i+1] = j'$ con $j' \geq j+1$ ma j' non può che essere $j+1$ altrimenti la soluzione ottima avrebbe saltato una scatola che non può essere recuperata e quindi non sarebbe una soluzione ottima.
CASO 2: all'$i+1$-esima iterazione l'algoritmo non cambia scatola (vale a dire $SOL_{i+1}[i+1] = SOL_{i+1}[i] = j$). Se anche $SOL^{\lfloor}i+1] = j$ abbiamo fatto. Sia allora $SOL^*[i+1] = j'$ con $j' \neq j$. In questo caso deve essere $j' > j$ in quanto SOL^* è una soluzione ammissibile per cui deve valere $SOL^*[i] \leq SOL^*[i+1]$ e per ipotesi induttiva sappiamo $SOL^*[i] = j$. Stando cosí le cose possiamo modificare SOL^* ponendo $SOL^*[i+1] = j$. La nuova soluzione è ancora ammissibile poiché la capacit à della scatola j con l'aggiunta dell'oggetto g_{i+1} non è violata contenendo gli stessi oggetti che ci mette l'algoritmo che non viola mai la capacità. Inoltre la soluzione resta ottima poiché la variazione non fa aumentare l'indice delle scatole utilizzate.