**Esercizio 1** Dato un insieme di stringhe binarie dette *primitive* ed una stringa binaria $X$, si vuole determinare se la stringa $X$ si può ottenere dalla concatenazione di stringhe primitive.
Ad esempio: dato l'insieme di primitive $\{01, 10, 011, 101\}$, per la stringa $X = 0111010101$ la risposta è sì (due possibili soluzioni sono $011-10-10-101$ e $011-101-01-01$) mentre per la stringa $X = 0110001$ la risposta è no.

1. Descrivere in pesudo–codice un algoritmo che risolve il problema con complessità $O(n \cdot m \cdot l)$ dove $n$ è la lunghezza della stringa $X$, $m$ è il numero delle primitive e $l$ è la lunghezza massima per le stringhe primitive. Spiegare perchè l'algoritmo proposto è corretto. (Suggerimento: programmazione dinamica).

2. Modificare l'algoritmo proposto in modo che, nel caso $X$ sia ottenibile dalla concatenazione di primitive, vengano prodotti in output gli indici delle primitive la cui concatenazione genera $X$.

   Ad esempio: data la stringa $X = 0111010101$ e le primitive $Y_1 = 01, Y_2 = 10, Y_3 = 011$ e $Y_4 = 101$, l'algoritmo deve produrre in output la sequenza $3, 2, 2, 4$ o la sequenza $3, 4, 1, 1$.

**Soluzione Esercizio 1**

**a)** Usiamo il metodo della programmazione dinamica.

Per ogni $0 \leq i \leq n$ sia $T[i]$ pari a *vero* o $falso$ a seconda che la stringa $x_1 x_2 \ldots, x_i$ dei primi $i$ simboli di $X$ sia generabile dalle primitive o meno.

Notiamo che $T[0] = vero$ mentre per $i \geq 1$ $T[i]$ è vero se e solo se la stringa $x_1 x_2 \ldots, x_i$ è scomponibile in un prefisso $x_1 x_2, \ldots x_j$ che è ottenibile con le primitive ed un suffisso $x_{j+1} x_{j+2} \ldots, x_i$ che è una primitiva.

Il calcolo dei $T[i]$ è facile:

Per ogni $i \geq 1$

$$T[i] = \begin{cases} vero & \text{se c'è una primitiva } Y_j \text{ con } |Y_j| \leq i \text{ tale che } T[i - |Y_j|] = vero \text{ e } x_{i-|Y_j|+1} \ldots x_i = Y_j \\ falso & \text{altrimenti} \end{cases}$$

Ovviamente la soluzione al nostro problema è data da $T[n]$. Quindi l'algoritmo può essere così descritto:

```
CONCATENAZIONE:  INPUT una stringa X e le primitive Y_1, Y_2, ... Y_m
    T[0] ← vero
    Calcola le lunghezze l_1, l_2, ..., l_m delle primitive Y_1, Y_2, ... Y_m
    FOR i ← 1 TO n DO
        T[i] ← falso
        j ← 1
        WHILE j ≤ m AND T[i] = falso DO
            IF l_j ≤ i AND T[i − l_j] = vero AND x_{i−l_j+1} ... x_i = Y_j THEN T[i] ← vero
            j ← j + 1
        ENDWHILE
    ENDFOR
OUTPUT T[n].
```

Il calcolo delle lunghezze delle $m$ primitive richiede $O(l \cdot m)$. Ogni iterazione del FOR richiede al più $m$ iterazioni del WHILE ciascuna delle quali ha costo al più $O(l)$ (che è il costo per eseguire il test $x_{i-l_j+1} \ldots x_i = Y_j$). La complessità dell'algoritmo è quindi $O(n \cdot m \cdot l)$.

**b)** Una possibile soluzione sta nell'utilizzo di una tabella ausiliaria $Z$ dove in $Z[i]$ si registra l'indice della primitiva che ha prodotto il valore vero per $T[i]$ ($Z[i]$ resta indefinito se $T[i]$ ha valore falso). Una volta calcolato $T[n]$ e verificato che ha valore vero allora la sequenza degli indici delle primitive la cui concatenazione genera $X$ si può costruire scandendo la tabella $Z$ in modo opportuno.

L'algoritmo può essere così descritto:

CONCATENAZIONE2: INPUT una stringa $X$ e le primitive $Y_1, Y_2, \ldots Y_m$
  $T[0] \leftarrow vero$
  Calcola le lunghezze $l_1, l_2, \ldots, l_m$ delle primitive $Y_1, Y_2, \ldots Y_m$
  FOR $i \leftarrow 1$ TO $n$ DO
    $T[i] \leftarrow falso$
    $j \leftarrow 1$
    WHILE $j \leq m$ AND $T[i] = falso$ DO
      IF $l_j \leq i$ AND $T[i - l_j] = vero$ AND $x_{i-l_j+1} \ldots x_i = Y_j$ THEN
        $T[i] \leftarrow vero$
        $Z[i] \leftarrow j$
      ENDIF
      $j \leftarrow j + 1$
    ENDWHILE
  ENDFOR
  IF $T[n] = vero$ THEN
    $SOL = []$ (*lista vuota*)
    $i \leftarrow n$
    WHILE $i > 0$ DO
      $SOL \leftarrow [Z[i], SOL]$ (*aggiunge in testa* )
      $i \leftarrow i - l_{Z[i]}$
    ENDWHILE
OUTPUT $SOL$.

**Esercizio 2** Scrivere in pseudo–codice una procedura che presi in input due interi positivi $n$ e $k$, con $k \leq n$, stampi tutte le sequenze binarie lunghe $n$ che contengono almeno $k$ simboli 1i consecutivi. Ad esempio se $n = 4$ e $k = 2$ allora la procedura deve stampare (non necessariamente in quest'ordine): $0011, 0110, 1100, 1110, 1101, 1011, 0111, 1111$. La complessità della procedura **deve essere** $O(nD(n))$, dove $D(n)$ è il numero di sequenze da stampare. **Motivare** la procedura proposta.

**Soluzione Esercizio 2** Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. L'algoritmo CONS, essendo ricorsivo, prende in input l'intero $n$, il vettore $SOL$ in cui viene memorizzata la sequenza da stampare, il numero $k$ di elementi consecutivi che devono essere presenti nella soluzione da stampare, il numero $i$ di elementi inseriti nella sequenza fino a questo momento, il numero $t$ di simboli 1 con cui termina questa soluzione parziale ed una variabile booleana $flag$ che indica se nella soluzione parziale sono presenti almeno $k$ simboli 1 consecutivi. Quindi la prima chiamata sarà CONS($n$, $SOL$, $k$, 0, 0, $falso$).

```
CONS: INPUT l'intero n, il vettore SOL gli interi k, i e t, la variabile booleana flag
    IF i = n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
    ELSE
        IF flag = vero THEN
            SOL[i+1] ← 0
            CONS(n, SOL, k, i+1, t, vero)
            SOL[i+1] ← 1
            CONS(n, SOL, k, i+1, t+1, vero)
        ELSE
            IF n − (i+1) ≥ k THEN
                SOL[i+1] ← 0
                CONS(n, SOL, k, i+1, 0, falso)
            ENDIF
            SOL[i+1] ← 1
            IF t = k − 1 THEN CONS(n, SOL, k, i+1, k, vero)
            ELSE CONS(n, SOL, k, i+1, t+1, falso)
        ENDIF
    ENDIF
```

Per decidere se inserire o meno un elemento nella soluzione parziale senza compromettere la possibilità di ottenere una sequenza da stampare si segue la seguente strategia:

Se $flag = vero$ (vale a dire nella soluzione parziale sono presenti almeno $k$ simboli 1 adiacenti) allora sia aggiungere un simbolo 0 che aggiungere un simbolo 1 permetterà comunque di ottenere stringhe da stampare quindi non viene eseguito alcun controllo ed entrambe le scelte vengono effettuate.

Se $flag = falso$ (vale a dire nella soluzione parziale non sono presenti $k$ simboli 1 consecutivi) allora: la scelta di inserire uno zero può rendere impossibile il completamento della soluzione parziale in una sequenza da stampare (questo accade unicamente se per completare la sequenza restano da inserire meno di $k$ simboli) per evitare questo si esegue il test $n - (i + 1) \geq k$ e lo 0 viene inserito solo se il test ha esito positivo.

La scelta di inserire un 1 non può compromettere il completamento della soluzione parziale in una sequenza da stampare quindi questa scelta va sempre fatta tuttavia la scelta di inserire un 1 può determinare la presenza di $k$ simboli 1 consecutivi nella soluzione parziale e quindi causare il cambiamento di valore della variabile $flag$. Per tener conto di questo va effettuato il test $t = k - 1$ in modo da aggiornare opportunamente il valore di $flag$ ( vale a dire porre il $flag$ a $vero$ se il test è soddisfatto, lasciarlo invariato altrimenti).

Per quanto detto una chiamata ricorsiva sarà effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare. Il costo di un nodo dell'albero di

decisione è $O(1)$ nel caso di nodo interno ed $O(n)$ nel caso di una foglia, l'altezza dell'albero è $n$. La complessità risultante sarà $\mathrm{O}(n \cdot D(n))$.