Esercizio 1

Si dispone di uno ZAINO di capacità C in cui è possibile inserire oggetti di n tipi diversi. Ciascun oggetto è caratterizzato da un valore v ed un peso p, diversi per ogni oggetto. È possibile inserire nello zaino quante copie si vuole di ciascun oggetto a patto che il peso complessivo non superi la capacità dello zaino. Il valore dello zaino è dato dalla somma dei valori degli oggetti inseriti. Si vuole individuare quanti oggetti di ciascun tipo inserire in modo da ottenere lo zaino di valore massimo

Esempio. Supponiamo che $n = 3$, $C = 18$ e i valori ed i pesi dei tre tipi di oggetto sono specificati nella seguente tabella:

	oggetto 1	oggetto 2	oggetto 3
peso	9	5	4
valore	6	4	3

In tal caso la combinazione di oggetti che massimizza il valore dello zaino è quella che prevede zero oggetti di tipo 1, due oggetti di tipo 2 e due oggetti di tipo 3. Tale combinazione comporta un peso dello zaino pari a $0 \times 9 + 2 \times 5 + 2 \times 4 = 18$ ed un valore dello zaino pari a $0 \times 6 + 2 \times 4 + 2 \times 3 = 14$.

Dati i pesi $p_1, \dots p_n$ e i valori $v_1, \dots v_n$ degli oggetti e la capacità C dello zaino:

1. *(max 10 punti)* Proporre un algoritmo che in tempo $O(nC)$ calcoli il valore massimo ottenibile tra gli inserimenti possibili nello zaino. (sull'istanza dell'esempio deve calcolare 14).

2. *(max 10 punti)* Modificare l'algoritmo proposto al punto 1 in modo da avere in output gli oggetti da inserire per avere lo zaino di valore massimo. La modifica deve avere costo additivo $O(C)$. (sull'istanza dell'esempio deve produrre il vettore $\boxed{0 \mid 2 \mid 2}$)

Soluzione Esercizio 1 Si mantenga un vettore T di dimensione C tale che

$T[i]$ = valore massimo che si può ottenere con uno zaino di capacità i.

La soluzione al problema originario è il valore $T[C]$.
Per costruire T si può utilizzare la seguente regola:

$$T[i] = \begin{cases} 0 & \text{se } i < \min\{p_1, \dots p_n\} \\ \max_{1 \le j \le n, p_j \le i}\{T[i - p_j] + v_j\} & \text{altrimenti} \end{cases}$$

E' facile calcolare i valori del vettore T spendendo tempo $O(n)$ su ogni cella e quindi ottenendo un tempo di esecuzione $O(n \cdot C)$. Lo pseudo-codice è il seguente:

```
VALORE-ZAINO:
  INPUT l'intero C i pesi p_1,... p_n e i valori v_1,... v_n
  OUTPUT il valore dell'inserimento di valore massimo
    FOR i = 0 TO C DO
        T[i] ← 0
        FOR j = 1 TO n DO
            IF p_j ≤ i AND T[i − p_j] + v_j > T[i] THEN T[i] ← T[i − p_j] + v_j
        ENDFOR
    ENDFOR
    OUTPUT T[C]
```

Dati i valori del vettore T, gli oggetti da inserire nello zaino per ottenere il valore $T[C]$ possono essere facilmente individuati in tempo $O(C)$ procedendo dall'ultima verso la prima cella del vettore e spendendo tempo $O(1)$ per cella.

```
OGGETTI:
  INPUT l'intero C, i pesi p_1, ... p_n, i valori v_1, ... v_n e il vettore T
  OUTPUT l'array SOL (di dimensione n) tale che SOL[i] è il numero di oggetti di tipo i
   presi dalla soluzione
   capacita ← C
   FOR i = 1 TO n DO
       SOL[i] ← 0
       WHILE capacita ≥ p_i AND T[capacita − p_i] = T[capacita] − v_i DO
           SOL[i] ← SOL[i] + 1
           capacita ← capacita − p_i
       ENDWHILE
   ENDFOR
   RETURN SOL
```

Esercizio 2 *(max 10 punti)* Scrivere in pseudo-codice una procedura che presi in input due interi n e k con $k \leq \frac{n}{2}$ stampi tutte le stringhe binarie di lunghezza n in cui il numero di uni è almeno pari al numero di zeri ed il numero di zeri è almeno k . Ad esempio per $n = 5$ e $k = 2$ la procedura deve stampare (non necessariamente in quest'ordine): $00111, 01011, 01101, 01110, 10011, 10101, 10110, 11001, 11010, 11100$.
La complessità della procedura **deve essere** $O(n \cdot D(n))$, dove $D(n)$ è il numero di stringhe da stampare.

Soluzione Esercizio 2 Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. Per ottenere una complessità legata al numero $D(n)$ di stringhe da stampare basta garantire che una chiamata ricorsiva venga effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare.
L'algoritmo ABC, essendo ricorsivo, prende in input gli interi n e k, il vettore SOL in cui viene memorizzata la stringa da stampare, il numero i che indica la posizione in cui inserire l'elemento, il numero $numzeri$ di zeri presenti nella stringa finora costruita. La prima chiamata sarà ABC(n, k SOL, 1, 0).
Notiamo che:

- lo zero può essere inserito nella stringa se non ho ancora inserito k zeri o è possibile poi completare la stringa in modo che il numero di uni superi il numero di zeri ($numzeri < k$ OR $n \geq 2(numzeri+1)$).
- l'uno può essere inserito nella stringa se il numero di zeri inseriti è gia almeno k o è possibile poi completare la stringa in modo che il numero di zeri superi k ($numzeri \geq k$ OR $n-i-numzeri \geq k$).

Lo pseudo-codice dell'algoritmo è il seguente:

```
ABC:  INPUT gli interi n e k, il vettore SOL e gli interi i e numzeri
   IF i > n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
   ELSE
      IF numzeri < k OR n ≥ 2(numzeri + 1) THEN
         SOL[i] ← 0
         ABC(n, k, SOL, i + 1, numzeri + 1)
      ENDIF
      IF numzeri ≥ k OR n − i − numzeri ≥ k THEN
         SOL[i] ← 1
         ABC(n, k, SOL, i + 1, numzeri)
      ENDIF
   ENDIF
```