Esercizio 1 La CINA (Compagnia Italiana per il Noleggio di Automobili) dispone di k automobili, tutte disponibili per n giorni. Una stessa automobile può avere costo di affitto diverso in giorni diversi: indicheremo con $c(a, g)$ il costo per il noleggio dell'automobile a nel giorno g, dove $1 \leq a \leq k$ e $1 \leq g \leq n$.

Un cliente si rivolge alla CINA per procurarsi un mezzo di locomozione per l'intero periodo. Per risparmiare e sfruttare i prezzi di noleggio migliori in ogni giornata, il cliente è disposto a cambiare macchina da un giorno all'altro. Per effettuare un cambio, però, è costretto dalla CINA a pagare una penale P, che si aggiunge al costo di noleggio. Il costo di una sequenza di noleggio è quindi dato dalla somma dei costi di noleggio e delle penali pagate per i cambi.

Esempio. Supponiamo che $n = 5$, $k = 2$, $P = 10$ e i costi di affitto sono specificati nella seguente tabella:

	giorno 1	giorno 2	giorno 3	giorno 4	giorno 5
automobile 1	16	3	5	30	6
automobile 2	2	20	40	8	50

In tal caso la sequenza di affitto migliore è 2 1 1 2 1, che consiste nel noleggiare l'automobile 1 nel secondo, terzo e quinto giorno, e l'automobile 2 nel primo e nel quarto giorno. Tale sequenza comporta quindi tre cambi, all'inizio del secondo, del quarto e del quinto giorno. Il costo totale è pertanto $c(2, 1)+P+c(1, 2)+c(1, 3) + P + c(2, 4) + P + c(1, 5) = 2 + 10 + 3 + 5 + 10 + 8 + 10 + 6 = 54$.

Dati k, n, i costi $c(a, g)$ e la penale P:

1. *(max 10 punti)* Proporre un algoritmo che in tempo $O(n \cdot k^2)$ calcoli la spesa minima per il cliente tra tutte le possibili sequenze di noleggio.

2. *(max 10 punti)* Modificare l'algoritmo proposto al punto 1 in modo da avere in output la sequenza di noleggio di costo minimo. La modifica deve avere costo additivo $O(n \cdot k)$.

Soluzione Esercizio 1 Si mantenga una tabella bidimensionale S di dimensione $n \times k$ tale che

> $S[i, j]$ = spesa minima per i primi i giorni di noleggio assumendo che nel giorno i il cliente affitti l'automobile j.

La soluzione al problema originario è il minimo dell'ultima riga della matrice, ovvero

$$\min_{1 \leq j \leq k} \{S[n, j]\}$$

La prima riga della matrice S può essere facilmente inizializzata ponendo

$$S[1, j] = c(j, 1)$$

per ogni $j \in [1, n]$.

Per costruire S, si può utilizzare la seguente regola di avanzamento:

$$S[i, j] = c(j, i) + \min\{\, S[i - 1, j], \min_{1 \leq a \leq k,\ a \neq j} \{S[i - 1, a] + P\}\, \}$$

Esempio. La tabella di programmazione dinamica costruita dall'algoritmo nell'esempio precedente è la seguente:

16	2
15	22
20	62
50	38
54	88

E' facile costruire la tabella per righe, spendendo tempo $\Theta(k)$ su ogni cella, e quindi ottenendo un tempo di esecuzione $O(n \cdot k^2)$. Lo pseudo-codice è il seguente:

```
COSTO-NOLEGGIO:
  INPUT gli interi n, k e P, la matrice dei costi c
  OUTPUT la matrice S ed il costo opt di una sequenza di noleggio ottima
   FOR j = 1 TO k S[1, j] ← c[j, 1]
   FOR i = 2 TO n
       FOR j = 1 TO k
           S[i, j] ← +∞
           FOR a = 1 TO k
               IF a ≠ j
                   IF c[j, i] + min{ S[i − 1, j], S[i − 1, a] + P } < S[i, j]
                   THEN S[i, j] ← c[j, i] + min{ S[i − 1, j], S[i − 1, a] + P }
   opt ← S[n, 1]
   FOR j = 2 TO k
       IF S[n, j] < opt THEN opt ← S[n, j]
   RETURN (opt, S)
```

Una implementazione più attenta permetterebbe anche di costruire la tabella in tempo $O(n \cdot k)$, spendendo solo tempo costante su ogni cella.

Data la spesa minima, una sequenza di noleggio ottima può essere facilmente ricostruita in tempo $O(n \cdot k)$ procedendo dall'ultima verso la prima riga della matrice e spendendo tempo $O(k)$ per riga.

```
SEQUENZA-NOLEGGIO:
  INPUT gli interi n, k e P, le matrici c ed S
  OUTPUT l'array SOL (di dimensione n) tale che SOL[i] è la macchina noleggiata dal cliente
   nell'i-esimo giorno
   j ← 1
   FOR a = 2 TO k
       IF S[n, a] < S[n, j]
       THEN j ← a
   SOL[n] ← j
   i ← n
   WHILE i > 1
       IF S[i, j] < c[j, i] + S[i − 1, j]
           FOR a = 1 TO k
               IF a ≠ j AND S[i, j] = c[j, i] + S[i − 1, a] + P
               THEN j ← a
       i ← i − 1
       SOL[i] ← j
   RETURN SOL
```

Esercizio 2 *(max 10 punti)* Scrivere in pseudo-codice una procedura che presi in input due interi n e k con $n \geq k \geq 2$ stampi tutti le stringhe binarie di lunghezza n le cui sottostringhe di simboli uguali e di lunghezza massima hanno tutte lunghezza diversa da k. Ad esempio per $n = 4$ e $k = 2$ la procedura deve stampare (non necessariamente in quest'ordine): $0000, 0001, 0101, 0111, 1000, 1010, 1110, 1111$.
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 assicurarsi che la complessità sia proporzionale al numero $D(n)$ di stringhe da stampare bisogna assicurarsi che una chiamata ricorsiva sarà effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare. Se la sottostringa di lunghezza i generata può essere estesa ad una stringa da stampare allora per mantenere questa proprietà nell'aggiungere il carattere $i + 1$ devo evitare che:

- se il carattere è diverso dall'ultimo finora inserito, allora la lunghezza del suffisso appena creato risulti k. Per questo basta controllare che $suf \neq k$, dove suf è la lunghezza del suffisso di simboli uguali della sottostringa i.

- se il carattere è uguale all'ultimo inserito, allora il suffisso che si sta creando dovrà poi poter avere una lunghezza diversa da k. Per questo basta controllare che $suf + 1 \neq k$ OR $i + 1 \neq n$.

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 di elementi della stringa creati fino a questo momento, la lunghezza suf del suffisso di simboli uguali di questa stringa. La prima chiamata sarà ABC(n, k SOL, 0, 0). Lo pseudo-codice dell'algoritmo è il seguente:

```
ABC:  INPUT gli interi n e k, il vettore SOL e gli interi i e suf
    IF i = n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
    ELSE
        IF i = 0 THEN
            SOL[1] ← 0
            ABC(n, k, SOL, 1, 1)
            SOL[1] ← 1
            ABC(n, k, SOL, 1, 1)
        ELSE
                IF suf + 1 ≠ k OR i + 1 ≠ n THEN
                    SOL[i + 1] ← SOL[i]
                    ABC(n, k, SOL, i + 1, suf + 1)
                ENDIF
                IF suf ≠ k THEN
                    SOL[i + 1] ← (SOL[i] + 1) mod 2
                    ABC(n, k, SOL, i + 1, 1)
                ENDIF
        ENDIF
    ENDIF
```