**Esercizio 1** Una pedina è posizionata sulla prima casella in alto a sinistra di una scacchiera $n \times n$ e, con un cammino composto da una sequenza di $2 \times n - 2$ mosse, deve raggiungere l'ultima casella in basso a destra. Ad ogni casella della scacchiera è assegnato un valore ed il valore del cammino eseguito dalla pedina è dato dalla somma dei valori delle $2 \times n - 1$ caselle toccate. Ad ogni mossa la pedina può spostarsi sulla casella adiacente alla sua destra (posto che questa casella esista) o sulla casella immediatamente più in basso (posto che questa casella esista).

- *(max 10 punti)* Descrivere in pseudo-codice un algoritmo che, dato l'intero $n$ ed una matrice $S$ di $n$ righe ed $n$ colonne con i valori delle varie caselle della scacchiera, determina il cammino di valore massimo. L'algoritmo deve essere **ben motivato** ed avere complessità $O(n^2)$.
- *(max 10 punti)* Modificare l'algoritmo proposto in modo che, con costo addizionale $O(n)$, venga prodotta una sequenza di $2 \times n - 2$ mosse che determinano un cammino di valore massimo (la sequenza può essere rappresentata con una stringa binaria dove 0 indica uno spostamento a destra ed 1 uno spostamento in basso).

**Soluzione Esercizio 1**

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

Per ogni $0 \leq i \leq n$ e $0 \leq j \leq n$ sia $M(i,j)$ pari al valore massimo per un cammino che porta la pedina dalla casella in alto a sinistra alla casella della $i$-esima riga e $j$-esima colonna. Il calcolo degli $M(i,j)$ è facile:

Per ogni $i, j \geq 1$

$$M(i,j) = \begin{cases} S[i,j] & \text{se } i = 1 \text{ e } j = 1 \\ M[i,j-1] + S[i,j] & \text{se } i = 1 \text{ e } j > 1 \\ M[i-1,j] + S[i,j] & \text{se } i > 1 \text{ e } j = 1 \\ \max\{M[i-1,j], M[i,j-1]\} + S[i,j] & \text{se } i > 1 \text{ e } j > 1 \end{cases}$$

Ovviamente la soluzione al nostro problema è data da $M(n,n)$. Quindi l'algoritmo può essere così descritto:

```
CAMMINO:  INPUT l'intero n e la matrice S.
    FOR i ← 1 TO n DO
        FOR j ← 1 TO n DO
            IF i = 1 AND j = 1 THEN M[i,j] ← S[i,j]
            ELSE IF i = 1 THEN M[i,j] ← M[i,j-1] + S[i,j]
                ELSE IF j = 1 THEN M[i,j] ← M[i-1,j] + S[i,j]
                    ELSE IF M[i-1,j] ≥ M[i,j-1] THEN M[i,j] ← M[i-1,j] + S[i,j]
                        ELSE M[i,j] ← M[i,j-1] + S[i,j]
        ENDFOR
    ENDFOR
OUTPUT M[n,n].
```

La complessità dell'algoritmo è determinata dai due 'FOR' nidificati ed è quindi $O(n^2)$.

**b)** Come è tipico degli algoritmi basati sulla tecnica della programmazione dinamica, per ricostruire la soluzione a partire dal suo valore basta ripercorrere le decisioni che hanno portato a determinare il valore della soluzione. Nel nostro caso, per ottenere il cammino basterà scorrere all'indietro la matrice $M$, aggiungendo all'algoritmo le istruzioni:

```
k ← 2 · n − 1
i ← n
j ← n
WHILE k > 0 DO
    IF j = 1 THEN
        BEGIN
        i ← i − 1
        SOL[k] ← 0
        END
    ELSE
        IF i = 1 THEN
            BEGIN
            j ← j − 1
            SOL[k] ← 1
            END
        ELSE
            IF M[i − 1, j] + S[i, j] = M[i, j] THEN
                BEGIN
                i ← i − 1
                SOL[k] ← 0
                END
            ELSE
                BEGIN
                j ← j − 1
                SOL[k] ← 1
                END
    k ← k − 1
ENDWHILE
OUTPUT SOL.
```

ed il costo addittivo dell'ultimo WHILE dell'algoritmo è $O(n)$.

**Esercizio 2** *(max 10 punti)* Scrivere in pseudo–codice una procedura che presi in input due interi positivi $n$ e $k$, stampi tutte le sequenze di lunghezza $2 \cdot n$ dei numeri in $\{1, 2, \ldots k\}$ con la proprietà che il numero che compare in posizione $i$, con $i \leq n$, ha valore inferiore al numero che compare in posizione $i+n$. Ad esempio se $n = 2$ e $k = 3$ allora la procedura deve stampare (non necessariamente in quest'ordine): $\{1,1,2,2\}$, $\{1,1,2,3\}$, $\{1,1,3,2\}$, $\{1,1,3,3\}$, $\{1,2,2,3\}$, $\{1,2,3,3\}$, $\{2,1,3,2\}$, $\{2,1,3,3\}$,$\{2,2,3,3\}$. La complessità della procedura **deve essere** $\mathrm{O}(n \cdot k \cdot D(n))$ dove $D(n)$ è il numero di sequenze da stampare.

**Soluzione Esercizio 2** Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. L'algoritmo ABC, essendo ricorsivo, prende in input gli interi $n$ e $k$, il vettore $SOL$ in cui viene memorizzata la sequenza da stampare, il numero $i$ di elementi inseriti nella sequenza fino a questo momento, Quindi la prima chiamata sarà ABC($n$, $k$, $SOL$, 0).

```
ABC:  INPUT gli interi n e k, il vettore SOL e l' intero i
    IF i = 2n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[2n]
    ELSE
        IF i + 1 ≤ n THEN
            FOR j ← 1 TO k − 1 DO
                SOL[i + 1] ← j
                ABC(n, k, SOL, i + 1)
            ENDFOR
        ELSE
            FOR j ← SOL[i + 1 − n] + 1 TO k DO
                SOL[i + 1] ← j
                ABC(n, k, SOL, i + 1)
            ENDFOR
        ENDIF
    ENDIF
```

Nelle soluzioni parziali gli elementi sono inseriti in modo tale che nelle prime $n$ posizioni non è mai inserito il numero $k$ mentre nella posizione $i$ con $i > n$ può comparire un qualsiasi numero superiore al numero inserito in posizione $i - n$.
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 interno dell'albero di decisione è $O(k)$ mentre il costo di un nodo foglia è $O(n)$ e l'altezza dell'albero è $n$. La complessità risultante sarà $\mathrm{O}(n \cdot k \cdot D(n))$.