rispettivamente a P e EXP − P, dove EXP rappresenta la classe di problemi risolubili mediante un **algoritmo esponenziale**, ovvero un algoritmo il cui numero di passi è al più esponenziale nella dimensione del dato in ingresso.[5] Talvolta gli algoritmi esponenziali sono utili per esaminare le caratteristiche di alcuni problemi combinatori sulla base della generazione esaustiva di tutte le istanze di piccola taglia.

Discutiamo due casi, che rappresentano anche un ottimo esempio di uso della ricorsione nella risoluzione dei problemi computazionali. Nel primo esempio, vogliamo generare tutte le 2^n sequenze binarie di lunghezza n, che possiamo equivalentemente interpretare come tutti i possibili sottoinsiemi di un insieme di n elementi. Per illustrare questa corrispondenza, numeriamo gli elementi da 0 a $n-1$ e associamo il bit in posizione b della sequenza binaria all'elemento b dell'insieme fornito (dove $0 \leqslant b \leqslant n-1$): se tale bit è pari a 1, l'elemento b è nel sottoinsieme così rappresentato; altrimenti, il bit è pari a 0 e l'elemento non appartiene a tale sottoinsieme. Durante la generazione delle 2^n sequenze binarie, memorizziamo ciascuna sequenza binaria A e utilizziamo la procedura `Elabora` per stampare A o per elaborare il corrispondente sottoinsieme. Notiamo che A viene riutilizzata ogni volta sovrascrivendone il contenuto ricorsivamente: il bit in posizione b, indicato con $A[b-1]$, deve valere prima 0 e, dopo aver generato tutte le sequenze con tale bit, deve valere 1, ripetendo la generazione. Il seguente codice ricorsivo permette di ottenere tutte le 2^n sequenze binarie di lunghezza n: inizialmente, dobbiamo invocare la funzione `GeneraBinarie` con input $b = n$.

```
1 GeneraBinarie( A, b ):          ⟨pre: i primi b bit in A sono da generare⟩
2   IF (b == 0) {
3     Elabora( A );
4   } ELSE {
5     A[b-1] = 0;
6     GeneraBinarie( A, b-1 );
7     A[b-1] = 1;
8     GeneraBinarie( A, b-1 );
9   }
```

Esempio 0.2. Considerando il caso $n = 4$, l'algoritmo di generazione delle sequenze binarie produce prima tutte le sequenze che finiscono con 0 per poi passare a produrre tutte quelle che finiscono con 1. Lo stesso principio viene applicato ricorsivamente alle sequenze di lunghezza 3, 2 e 1 ottenendo quindi tutte le sequenze binarie di quattro simboli:

[5] Volendo essere più precisi, i problemi intrattabili sono tutti i problemi decidibili che non sono inclusi in P: tra di essi, quindi, vi sono anche problemi che non sono contenuti in EXP. Nel resto di questo libro, tuttavia, non considereremo mai problemi che non ammettano un algoritmo esponenziale.

```
0000        1000        0100        1100
0010        1010        0110        1110
0001        1001        0101        1101
0011        1011        0111        1111
```

Il secondo esempio di un utile algoritmo esponenziale riguarda la generazione delle permutazioni degli n elementi contenuti in una sequenza A. Ciascuno degli n elementi occupa, a turno, l'ultima posizione in A e i rimanenti $n-1$ elementi sono ricorsivamente permutati. Per esempio, volendo generare tutte le permutazioni di $n=4$ elementi `a`, `b`, `c`, `d` in modo sistematico, possiamo generare prima quelle aventi `d` in ultima posizione (elencate nella prima colonna), poi quelle aventi `c` in ultima posizione (elencate nella seconda colonna) e così via:

```
a b c d     a b d c     a d c b     d b c a
b a c d     b a d c     d a c b     b d c a
a c b d     a d b c     a c d b     d c b a
c a b d     d a b c     c a d b     c d b a
c b a d     d b a c     c d a b     c b d a
b c a d     b d a c     d c a b     b c d a
```

Restringendoci alle permutazioni aventi `d` in ultima posizione (prima colonna), possiamo permutare i rimanenti elementi `a`, `b`, `c` in modo analogo usando la ricorsione su questi tre elementi. A tal fine, notiamo che le permutazioni generate per i primi $n-1=3$ elementi, sono identiche a quelle delle altre tre colonne mostrate sopra. Per esempio, se ridenominiamo l'elemento `c` (nella prima colonna) con `d` (nella seconda colonna), otteniamo le *medesime* permutazioni di $n-1=3$ elementi; analogamente, possiamo ridenominare gli elementi `b` e `d` (nella seconda colonna) con `d` e `c` (nella terza colonna), rispettivamente. In generale, le permutazioni di $n-1$ elementi nelle colonne sopra possono essere messe in corrispondenza biunivoca e, pertanto, ciò che conta sono il numero di elementi da permutare come riportato nel codice seguente. Invocando tale codice con parametro d'ingresso $p=n$, possiamo ottenere tutte le $n!$ permutazioni degli elementi in A:

```
1  GeneraPermutazioni( A, p ): ⟨pre: i primi p elementi di A sono da permutare⟩

2    IF (p == 0) {
3      Elabora( A );
4    } ELSE {
5      FOR (i = p-1; i >= 0; i = i-1) {
6        Scambia( i, p-1 );
7        GeneraPermutazioni( A, p-1 );
8        Scambia( i, p-1 );
9      }
10   }
```

Notiamo l'utilizzo di una procedura `Scambia` prima e dopo la ricorsione così da mantenere l'invariante che gli elementi, dopo esser stati permutati, vengono riportati al loro ordine di partenza, come può essere verificato simulando l'algoritmo suddetto.

0.5 Problemi NP-completi

La classificazione dei problemi decidibili nella Figura 2 ha in realtà una zona grigia localizzata tra i problemi trattabili e quelli intrattabili (le definizioni rigorose saranno date nell'ultimo capitolo del libro). Esistono decine di migliaia di esempi interessanti di problemi che giacciono in tale zona grigia: di questi ne riportiamo uno tratto dal campo dei solitari e relativo al noto gioco del *Sudoku*.

In tale solitario, il giocatore è posto di fronte a una tabella di nove righe e nove colonne parzialmente riempita con numeri compresi tra 1 e 9, come nell'istanza mostrata nella parte sinistra della Figura 3. Come possiamo vedere, la tabella è suddivisa in nove sotto-tabelle, ciascuna di tre righe e tre colonne. Il compito del giocatore

3	9							8
	7	1			3			
		8		4	9		6	
1			2	7				9
6								3
5				3	6			4
	4		1	5		9		
			9			8	2	
9							4	7

3	9	6	5	1	2	4	7	8
4	7	1	6	8	3	5	9	2
2	5	8	7	4	9	3	6	1
1	3	4	2	7	5	6	8	9
6	8	7	4	9	1	2	5	3
5	2	9	8	3	6	7	1	4
8	4	2	1	5	7	9	3	6
7	1	3	9	6	4	8	2	5
9	6	5	3	2	8	1	4	7

Figura 3 Un esempio di istanza del gioco del Sudoku e la corrispondente soluzione.