Il numero di Sarah

Sommario

Un classico problema logico tratto da un best seller di enigmi e puzzle per C64. Viene proposta e sviluppata una soluzione molto più completa di quella fornita dal testo, applicabile ad una vastissima gamma di problemi analoghi ma anche ad innumerevoli situazioni di programmazione: dai gestionali all'automazione industriale, dai protocolli alla grafica.

«È più facile quadrare un circolo che arrotondare un Matematico»
(Augustus de Morgan, 1806-1871)

1 Introduzione.

Scopo del presente articolo è l'illustrazione di un semplice problema che molti di noi «retroprogrammatori» si sono all'epoca divertiti a risolvere con l'ausilio del proprio PET preferito: una tipologia di problema che peraltro caratterizza anche molti rompicapo logici proposti dalle più note riviste di enigmistica. Il linguaggio è intenzionalmente informale, divulgativo e didascalico, scevro da particolari pretese di rigore, al fine di raggiungere il più vasto uditorio informatico possibile.
L'articolo è organizzato in tre parti principali. Nella prima parte si illustra il problema nella sua forma originale, proponendone anche una immediata traduzione dall'inglese.
Nella seconda parte si presenta la scarna soluzione originale del testo. Nella terza e ultima parte si discute una soluzione completamente autonoma da parte del calcolatore, senza alcun passaggio manuale, rivelando la reale natura del problema e dell'algoritmo risolutivo, che si presta ad una scelta totalmente automatizzabile della soluzione corretta tra quelle potenzialmente valide.

2 Sarah's Number.

Il problema originale, a pagina 12 del testo [LS83], è caratterizzato dal numero progressivo 11. Eccone il testo completo:

Last week while we were out sailing on the Henderson's yacht, Julie, Ken, Liz, Morris and Naomi were trying to remember Sarah's phone number.
They all agreed on the prefix, but none of them could remember exactly the last four digits. They came up with these statements:

Julie:	*There is a 9 in it.*
Ken:	*It is definitely higher than 5000.*
Liz:	*It is palindromic: it reads the same forwards as backwards*
Morris:	*The number is even.*
Naomi:	*It has a digital root of 9*.*

On arriving home, I looked up the phone number and found that they were all correct except for one. Had I known who was wrong, I could have discovered the number. What was Sarah's number, and who made the incorrect statement?
**The digital root of a number is obtained by adding together the digits until a single digit remains, e.g. the digital root of 8765 is:*
$8 + 7 + 6 + 5 = 26,\ 2 + 6 = 8$

Per chi avesse bisogno della traduzione: la scorsa settimana, mentre stavamo navigando sullo yacht degli Henderson, Julie, Ken, Liz, Morris e Naomi (ovvero J, K, L, M, N) cercavano di ricordare il numero di telefono di Sarah. Tutti d'accordo sulla parte iniziale, ma nessuno ricordava esattamente le ultime quattro cifre. Ciascuno di loro fece una affermazione [inerente le sole ultime quattro cifre]:

J Il numero contiene un 9.
K Il numero è molto maggiore di 5000.
L Il numero è palindromo (non cambia se letto da destra a sinistra o viceversa).
M Si tratta di un numero pari.
N Il suo modulo gaussiano (la somma ricorsiva delle cifre) è pari a nove.

All'arrivo a casa, ho controllato il numero sulla rubrica e ho scoperto che tutte le affermazioni erano corrette, tranne una. Se avessi saputo chi aveva sbagliato, avrei potuto risalire immediatamente al numero in questione. Qual è dunque il numero di Sarah, e chi ha prodotto l'asserzione sbagliata?

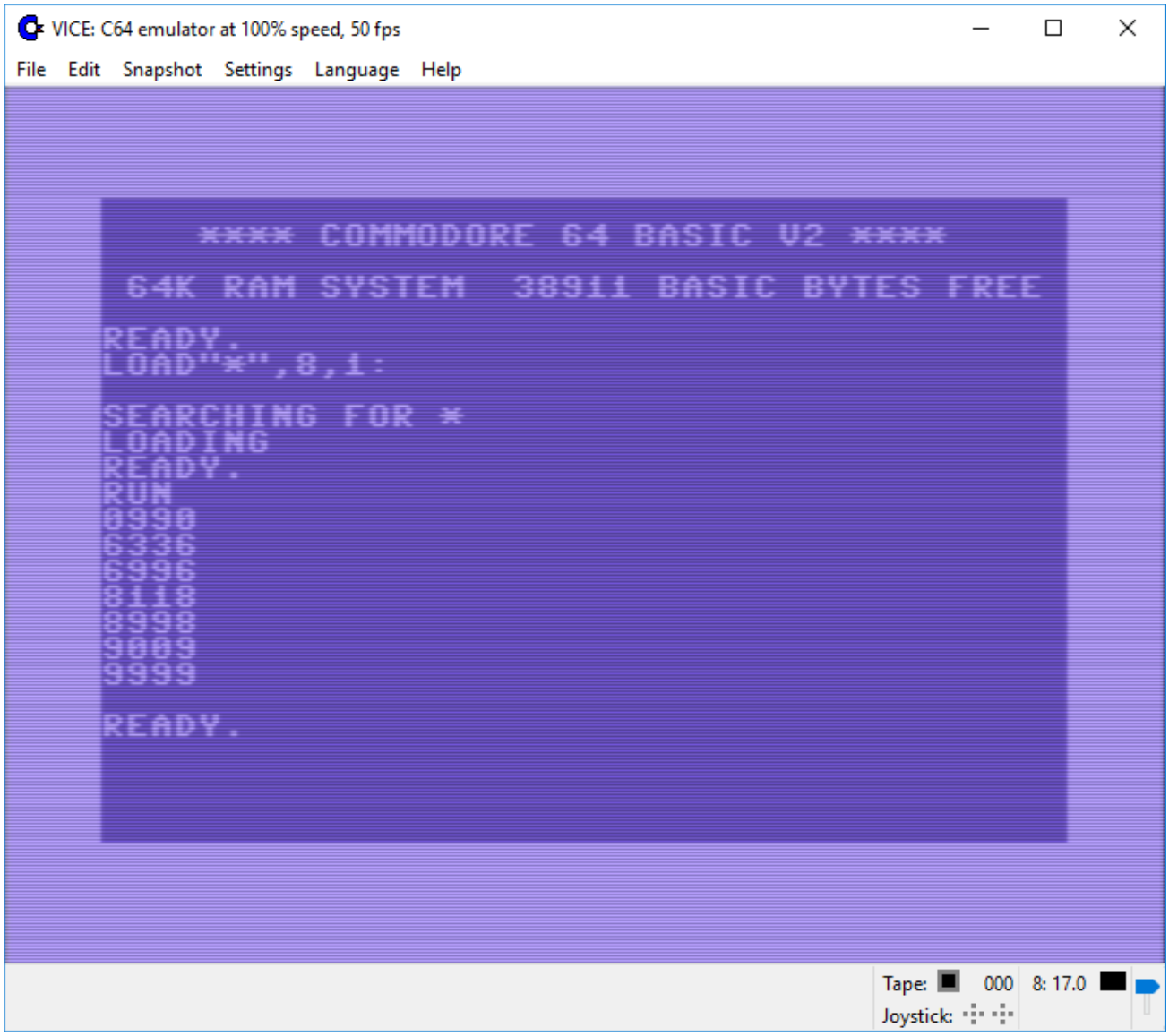
3 La soluzione originale.

La soluzione proposta a pag. 62 di [LS83] si limita sostanzialmente a generare tutti i numeri di quattro cifre, selezionando quelli che soddisfano esattamente quattro delle cinque condizioni stabilite dalle asserzioni J, K, L, M, N. L'unica euristica adottata consiste nel limitare a priori la ricerca ai soli numeri palindromi (condizione L), il che delimita il range di generazione ai valori tra 0 e 99 inclusi, rendendo immediatamente il loop principale più efficiente di un fattore cento rispetto all'implementazione più *naïf*. Il semplice programma contiene varie ingenuità implementative, probabilmente per invogliare il lettore ad applicare le necessarie migliorie, e usa il più banale dei pattern per la verifica di proprietà binarie multiple, ossia un contatore threshold a somma non pesata. L'utilità del programma proposto dagli autori è ulteriormente limitata dal fatto che esso fornisce in output un mero elenco dei valori, senza indicazione esplicita delle condizioni di volta in volta verificate. Proponiamo di seguito le considerazioni che accompagnano il codice, seguite dal relativo listato BASIC V2.

It is possible to check all the numbers between 0000 and 9999 against a permutation of four out of the five statements to see which gives a unique answer, but doing so would be a rather lengthy process. However, we know that only one of the statements is incorrect, so the other four must be correct. Let's look at Liz's statement. Now, there are only 100 possible fourdigit palindromic numbers - those having the first two digits 00 to 99 followed by these two digits in reverse order. If we assume that Liz's statement is one of the correct ones, we need only test these hundred possibilities against the other four statements. In fact, we can deduce that her statement is correct as it is quite easy to find many possibilities that satisfy all of the other conditions. In other words, there is more than one number that is more than 5000, has a "9" in it, is even, and has a digital root of 9. Two such numbers that can be easily discovered are 7398 and 7938. As this fails to provide an unique answer, Liz's statement must be correct, and one of the others is the wrong one. Therefore, we can use this as a basis for our program. The program checks all four-digit palindromic numbers and prints out any that are found to agree with three of the four remaining statements.

```
10 FOR N = 0 TO 99
20 N$ = MID$(STR$(N),2)
30 IF (N < 10) THEN N$ = "0" + N$
40 N$ = N$+RIGHT$(N$,1)+LEFT$(N$,1)
50 T = 0
100 FOR M = 1 TO 4
110 IF (MID$(N$,M,1) = "9") THEN T = 1
120 NEXT
200 IF (VAL(N$) > 5000) THEN T = T+1
300 IF (VAL(N$)/2 = INT(VAL(N$)/2) THEN T = T+1
400 C = 0
410 FOR M = 1 TO 4
420 C = C + VAL(MID$(N$,M,1)
430 NEXT
```

```
440 IF (C = 9) OR (C = 18) OR (C = 27) OR (C = 36) THEN T = T+1
500 IF (T = 3) THEN PRINT N$
510 NEXT
```



Nell'appendice dedicata alle soluzioni vere e proprie, a pag. 118, gli autori riportano poi una soluzione tabulare completa, calcolata con carta e penna. Riproponiamo qui tale tabella, con una veste grafica leggermente diversa per evidenziare la natura di matrice booleana del problema, in quanto tale perfettamente computabile con minimo sforzo algoritmico. A tale scopo lavoriamo in logica negata e marchiamo con un simbolo unicamente le asserzioni che risultano **false** per ciascun valore numerico trovato, riportato nella colonna più a sinistra. Ad esempio, il valore 6996 verifica a priori la condizione L e anche le condizioni J, K ed M in quanto rispettivamente:

L) Si tratta di un numero palindromo;
J) Una delle sue cifre è pari a 9;
K) Risulta maggiore di 5000;
M) Si tratta di un numero pari.

In tabella risulterà quindi marcata la sola condizione falsa N, in quanto il modulo gaussiano di 6996 è pari a 3.

	Julie	Ken	Morris	Naomi
0990		✖		
6336	✖			
6996				✖
8118	✖			
8998				✖
9009			✖	
9999			✖	

La soluzione è quindi 0990 e l'affermazione falsa è quella di Ken, perché la negazione di qualsiasi altra condizione non porta a soluzioni univoche, il che è caratteristica implicita nella descrizione del problema: «*Had I known who was wrong, I could have discovered the number.*». A questo punto risulta decisamente ovvia la natura del problema come ricerca dei valori singolari **per colonne** in una matrice binaria: un problema molto comune, che il nostro C64 può risolvere in piena autonomia, senza alcun passaggio manuale.

4 Soluzione alternativa.

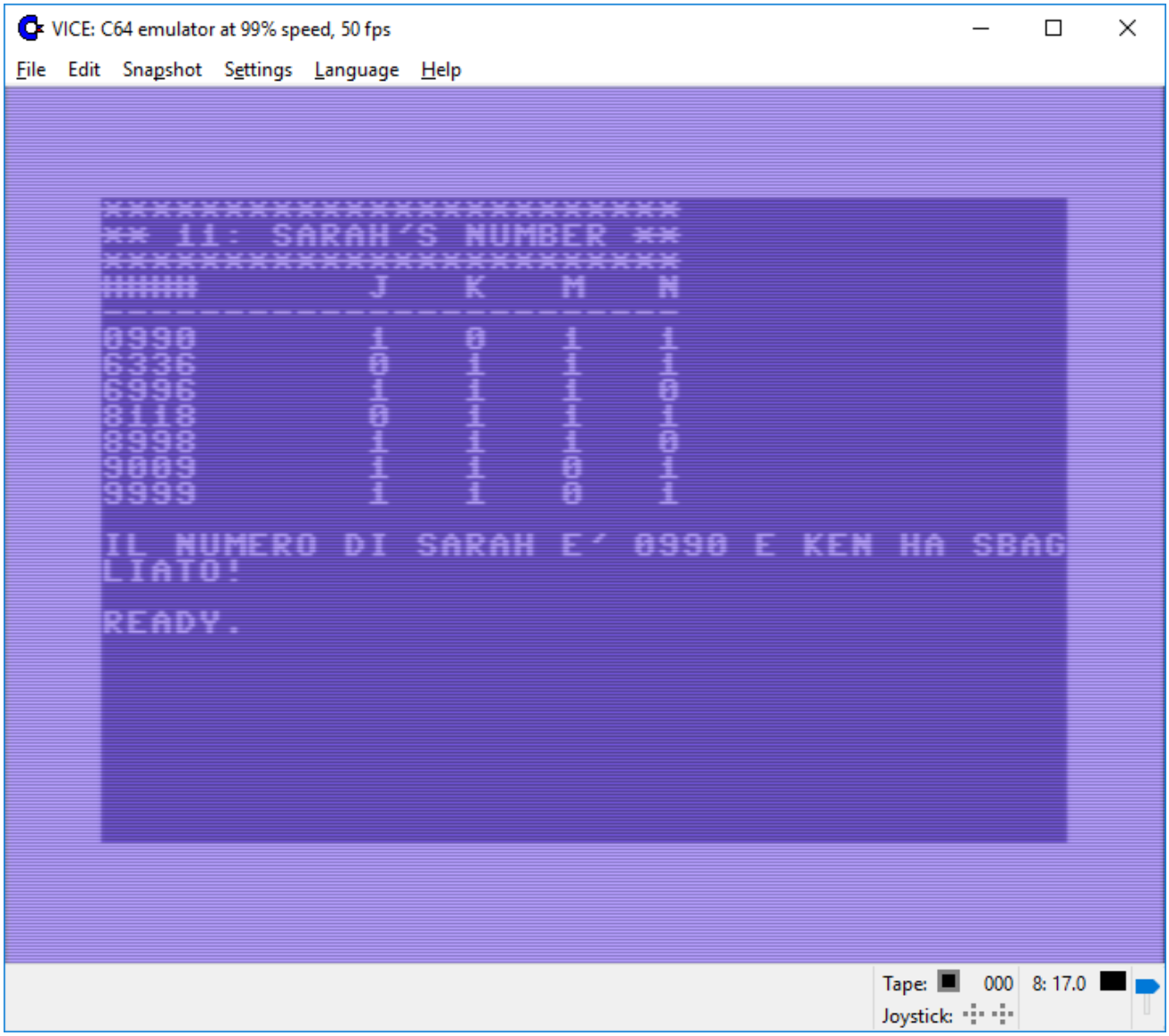
Si propone nel seguito una soluzione completa in BASIC V2 al quesito del testo, in grado di memorizzare i risultati parziali in una opportuna coppia di array ed analizzare autonomamente l'unicità della soluzione. In questo modo sarà il calcolatore a fornire direttamente la risposta finale. Si noti che questo genere di risoluzione si applica a decine di problemi eterogenei riconducibili alla medesima natura, da passatempi e rompicapo a reali problemi di ottimizzazione, logistica, automazione industriale, contabilità e molto altro, quindi vale la pena di considerare seriamente il metodo, seppure presentato in un contesto puramente ludico e con una implementazione puramente didattica.

Qualche breve nota sull'implementazione, che con le sue 30 linee di codice dovrebbe essere perfettamente intuitiva anche per i neofiti. Si è sostanzialmente mantenuta l'impostazione fortemente didattica e didascalica della soluzione originale. La struttura dati principale destinata a contenere la matrice booleana è denominata `co%()`ovvero `conditions`. Associato indirettamente alla matrice per indici di riga corrispondenti l'array `so$` (lapalissianamente `solutions`) dei numeri che soddisfano esattamente tre delle quattro condizioni previste. Si è deliberatamente scelto di mantenere elementare e intuitiva l'intera implementazione, evitando di ricorrere ad una vera matrice binaria (bit array), come invece si farebbe in altri contesti e linguaggi, al solo fine di rendere più facile e immediata la comprensione dell'algoritmo e delle indicizzazioni. Lo stesso vale per la memorizzazione del numero telefonico (a rigore, la sua parte finale) sotto forma di stringa, con la preservazione degli zeri iniziali. Il tutto senza preoccuparsi di comprimere il tempo di esecuzione e/o il costo di memorizzazione i quali, date le irrisorie dimensioni dello spazio delle soluzioni esplorato, sono decisamente poco rilevanti e passano in secondo piano rispetto alla didatticità e chiarezza del codice.

Si noti infine che la scansione della matrice binaria si arresta alla prima colonna che contenga un singolo valore nullo, il che è una concessione puramente didattica alla fiducia nella corretta costruzione della matrice e in ultima analisi nella esistenza e unicità della soluzione: nel mondo reale un controllo di congruenza esaustivo sarebbe quantomeno doveroso per la robustezza dell'applicazione e la coerenza dei dati. Allo stesso modo si è dato per scontato il dimensionamento iniziale degli array: ma confidiamo che qui anche il lettore meno famigliare col «Testo Sacro» [Sta86] e poco a suo agio nel calcolo a priori della dimensione dello spazio delle soluzioni concorderà sulla ragionevolezza di una tale assunzione, alla quale si può facilmente giungere anche col più empirico dei metodi, ossia una esecuzione preliminare del loop di generazione ed enumerazione, strutturato in modo simile alla soluzione originale degli autori Lee e Scrimshaw.

```
100 print "{clear}************************"
101 print "** 11: sarah's number **"
102 print "************************"
103 print "####       j   k   m   n"
105 print "-------------------------"
110 so = 1 : dim co%(7,4), so$(7), na$(4)
120 na$(1) = "julie": na$(2) = "ken": na$(3) = "morris": na$(4) = "naomi"
```

```
130 for n = 0 to 99 140 n$ = right$("0" + mid$(str$(n), 2), 2)
150 n$ = n$ + right$(n$, 1) + left$(n$, 1)
160 for i = 1 to 4: co%(so, i) = 0: next
170 t = 0
200 if (mid$(n$, 1, 1) = "9") or (mid$(n$, 2, 1) = "9") then t = 1 : co%(so, 1) = 1
210 if (val(n$) > 5000) then t = t + 1 : co%(so, 2) = 1
220 if (val(n$) / 2 = int(val(n$) / 2)) then t = t + 1: co%(so, 3) = 1
230 c = 0
240 for m = 1 to 4
250 c = c + val(mid$(n$, m, 1))
260 next m
270 if (c > 9) then c = val(right$(mid$(str$(c),2), 1)) + val(left$(mid$(str$(c),2), 1))
280 if (c = 9) then t = t + 1: co%(so, 4) = 1
290 if (t <> 3) then goto 330
300 so$(so) = n$ : ? n$,
310 for i = 1 to 4: ? co%(so, i); " ";: next i:?
320 so = so +1
330 next n
400 for c = 1 to 4: t = 0
410 for r = 1 to 7
420 if co%(r, c) = 0 then t = t +1 : rs = r
430 next r
440 if t = 1 then print : print "il numero di sarah e' ";so$(rs);" e ";
na$(c); " ha sbagliato!": end
450 next c
```



Riferimenti bibliografici

[LS83] Gordon Lee and Nevin B. Scrimshaw, *The commodore puzzle book - basic brainteasers*, Birkhauser, Boston, USA, 1983.

[Sta86] Richard P. Stanley, *Enumerative combinatorics*, vol. 1, Wadsworth Publ. Co., Belmont, CA, 1986.