Ventiduesima Gara di Informatica per studenti delle Scuole Superiori

Esercizi di gara

AVVISI:

- Se non specificato altrimenti negli esercizi, le sequenze iniziali su nastro si intendono *non vuote*, ovvero contenenti almeno un simbolo.
- Per numero decimale si intende un numero positivo o nullo rappresentato con le cifre 0, 1, 2, ..., 9, senza zeri iniziali non significativi; per esempio 0 e 19 sono numeri validi, mentre 0032 deve essere scritto come 32.
- Nel fornire le soluzioni, ricordarsi di pulire il nastro finale da ogni simbolo che non costituisca la risposta!
- Ogni volta che si salva la soluzione di un esercizio con il simulatore della macchina di Turing, il "timestamp" dell'esercizio viene aggiornato con il tempo trascorso fino a quel momento.

Esercizio 1: Algebra Booleana [Punti 1]. Alla base del funzionamento di qualunque calcolatore elettronico c'è la capacità di effettuare operazioni in *algebra booleana*, ovvero l'algebra che opera su due soli valori distinti (che possiamo indicare con 0 e 1, vero e falso, bianco o nero, 0 volt e +5 volt di corrente… noi useremo i simboli "F" e "T") e con operatori quali **and**, **or**, **not**, e simili. Ciascuno di questi operatori può essere definito da una *tabella di verità*, che indichi per ogni possibile insieme di operandi, quale sarà il valore del risultato. Per esempio, a fianco vedete la tabella di verità dell'operatore **and** (interpretando T come vero, e F come falso, il risultato di *a* **and** *b* sarà vero solo se entrambi *a* e *b* sono veri). Si scriva un programma per macchina di Turing che, ricevuti sul nastro di input due simboli sull'alfabeto {F,T}, lasci sul nastro il risultato dell'operazione di **and** sui valori dati.

a	*b*	*a* **and** *b*
F	F	F
F	T	F
T	F	F
T	T	T

NASTRO INIZIALE	*NASTRO FINALE*
`FF`	`F`
`FT`	`F`
`TT`	`T`

Esercizio 2: La lunghezza non conta! [Punti 2]. Una volta che si sa come fare un'operazione fra due valori, è spesso banale estenderla a una sequenza arbitraria di valori. Per esempio, possiamo calcolare l'and fra 4 valori come ((*a* **and** *b)* **and** *c)* **and** *d*. Si scriva un programma per macchina di Turing che, ricevuta sul nastro una sequenza lunga a piacere di valori in {F, T}, lasci sul nastro il risultato dell'**and** a catena sulla sequenza, come descritto.

NASTRO INIZIALE	*NASTRO FINALE*
`TF`	`F`
`TT`	`T`
`FTFFT`	`F`
`T`	`T`
`TTTTT`	`T`

Esercizio 3: Più sequenze [Punti 3]. Un altro modo di estendere l'algebra Booleana è di applicarla "a coppie", sugli elementi di più sequenze. Date due sequenze $s_1=a_k...a_2a_1a_0$ e $s_2=b_k...b_2b_1b_0$, il risultato dell'operazione di **and** fra le sequenze sarà (s_1 **and** s_2) = (a_k **and** b_k) … (a_2 **and** b_2)(a_1 **and** b_1)(a_0 **and** b_0). Si scriva un programma per macchina di Turing che, ricevute sul nastro di ingresso due sequenze (di uguale lunghezza) di valori Booleani, separate dal simbolo #, lasci sul nastro il risultato dell'and fra le sequenze.

NASTRO INIZIALE	*NASTRO FINALE*
FFTTF#TTTFT	FFTFF
T#T	T
TTTT#FFFF	FFFF
TTTTTTTF#FTTFTTFT	FTTFTTFF

Esercizio 4: Numeri [Punti 5]. Assumiamo di operare su sequenze di valori Booleani di lunghezza fissata. Se interpretiamo F come 0 e T come 1, e assegnamo il significato di numerazione con sistema posizionale alla sequenza, possiamo esprimere numeri arbitrari in base 2 tramite sequenze di Booleani. Per esempio, con sequenze di lunghezza 4 possiamo esprimere numeri fra 0_{10} e 15_{10}, ovvero fra 0000_2 e 1111_2. Su queste sequenze si può operare con l'algebra booleana per ottenere risultati che, interpretati poi nuovamente come numeri, hanno significato aritmetico. Si scriva un programma per macchina di Turing che, ricevuto sul nastro un numero decimale, fra 0 e 255, lasci sul nastro il corrispondente numero espresso in binario con una sequenza di Booleani di lunghezza esattamente 8. Una simile sequenza è quella che si usa chiamare *byte*.

NASTRO INIZIALE	*NASTRO FINALE*
15	00001111
1	00000001
192	11000000

Esercizio 5: Testi [Punti 12]. Una sequenza di valori Booleani può essere usata per rappresentare un testo. È sufficiente concordare una *codifica* che associ ogni carattere dell'alfabeto a una particolare sequenza. Per esempio, potremmo considerare le 21 lettere dell'alfabeto Italiano standard, più 10 simboli per le cifre decimali, e un simbolo per lo spazio, per arrivare a una codifica con 32 valori come quella indicata nella tabella accanto. Codifiche di questo tipo sono già sufficienti per esprimere semplici testi. In effetti, codifiche non troppo diverse da questa erano usati nei primissimi calcolatori. Si scriva un programma per macchina di Turing che, ricevuto sul nastro un testo sull'alfabeto {0-9, A-Z, spazio) come da codifica, terminata da un simbolo ".", lasci sul nastro la corrispondente codifica come sequenza Booleana, senza separatori (e senza tradurre il simbolo "." terminale).

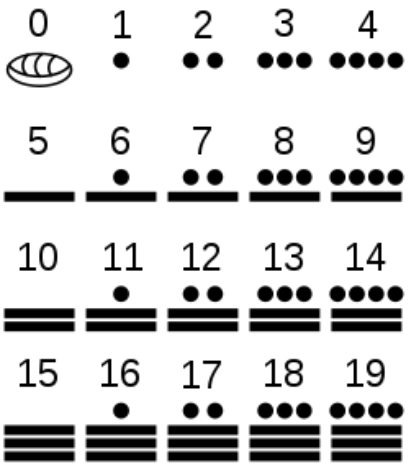
0	00000	**8**	01000	**G**	10000	**Q**	11000
1	00001	**9**	01001	**H**	10001	**R**	11001
2	00010	**A**	01010	**I**	10010	**S**	11010
3	00011	**B**	01011	**L**	10011	**T**	11011
4	00100	**C**	01100	**M**	10100	**U**	11100
5	00101	**D**	01101	**N**	10101	**V**	11101
6	00110	**E**	01110	**O**	10110	**Z**	11110
7	00111	**F**	01111	**P**	10111	*spazio*	11111

NASTRO INIZIALE	*NASTRO FINALE*
CIAO.	01100100100101010110
ADD 3.	010100110101101111100011
INC R0.	10010101010110011111111001000000
2 PIU 2.	000101111110111100101110011111100010
15.	0000100101
S .	11010111111111111111

Esercizio 6: Immagini [Punti 8]. Una codifica molto naturale per le immagini (ci limitiamo al bianco e nero) prevede di organizzare l'immagine in una griglia regolare di *pixel*, ciascuno dei quali può essere bianco (che indicheremo con 0) o nero (che indicheremo con 1). Una griglia di questo tipo è detta *bitmap*. Su questi dati è possibile svolgere semplicemente operazioni di natura grafica. Per esempio, assumendo che un'immagine sia composta da 8x8 pixel, e che contenga una figura chiusa convessa di cui è disegnato soltanto il bordo, è possibile definire l'operazione *fill* che riempe di nero la parte interna (bianca) della figura. Si scriva un programma per macchina di Turing che, ricevuta sul nastro una immagine in formato bitmap, espressa mettendo in sequenza 8 gruppi di 8 simboli su {0, 1} ciascuna, lasci sul nastro la corrispondente immagine "riempita" con l'operazione di fill. *Nota: negli esempi seguenti, le bitmap sono rappresentate su due dimensioni e con caselle colorate per chiarezza. Sul nastro, le righe saranno tutte consecutive.*

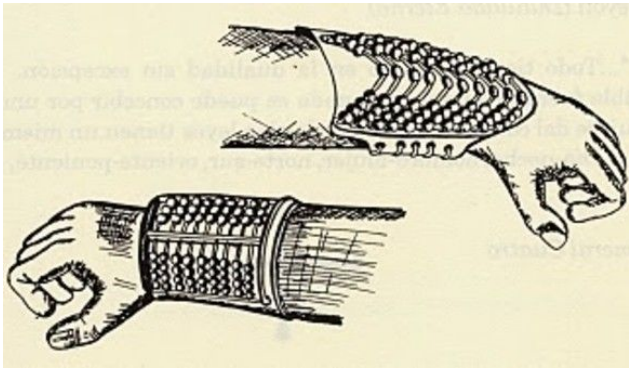
NASTRO INIZIALE	*NASTRO FINALE*
0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0

Esercizio 7: La fine del mondo è vicina! [Punti 20]. Abbiamo visto come si possano usare sequenze di valori booleani per codificare numeri e testi. Con opportuni sistemi di codifica, è possibile codificare anche immagini, suoni, film, libri contabili, poesie, reti di contatti fra le persone, forme di caratteri tipografici, e ogni altro tipo di informazione che si presti ad avere una rappresentazione scritta. In effetti, con un po' di fantasia, si può codificare quasi tutto. Consideriamo per esempio il sistema di numerazione dei Maya. Si tratta di un sistema posizionale a base mista, organizzato su due livelli. La base posizionale è 20, per tutte le posizioni tranne che per la seconda, per cui è 18[1]. Le singole cifre (ciascuna delle quali con valore fra 0 e 19) sono a loro volta rappresentate con un sotto-sistema a due posizioni in base 5, in cui la cifra delle unità è rappresentata in notazione unaria (un pallino per ogni unità), mentre la cifra delle cinquine è anch'essa rappresentata in notazione unaria, ma con un simbolo diverso (una linea per ogni cinquina). Un simbolo speciale, a forma di conchiglia, denota lo 0. Nel sistema Maya originale, le cifre sono scritte dall'alto verso il basso; noi per praticità le scriveremo da sinistra a destra, con le cifre di ordine più basso a destra, separate da un simbolo "#". Useremo il simbolo "0" per la conchiglia, "/" per le linee, e "." per il pallino. Così, la cifra 17 verrà rappresentata su nastro come "///..". Possiamo così codificare facilmente iscrizioni Maya (numeriche). Si scriva un programma per macchina di Turing che, ricevuta in ingresso una iscrizione Maya, lascia sul nastro il corrispondente numero decimale (che, come abbiamo visto nell'esercizio 4, può a sua volta essere trasformato in una sequenza di 0 e 1; anche i Maya amano George Boole).



NASTRO INIZIALE	*NASTRO FINALE*
///..	17
0	0
..#0	40
///#/..#/...	5548
.#0#//.#///...	7438

Esercizio 8: Il Nepohualtzintzin [Punti 15]. Il sistema Maya si prestava particolarmente bene ad eseguire operazioni aritmetiche con un abaco particolare, detto Nepohualtzintzin. Il Nepohualtzintzin era costituito da una serie di cordicelle; ogni cordicella rappresentava una cifra, e su ogni cordicella erano presenti quattro palline "punto" e tre palline "linea", di solito distinte dal colore o da un segno inciso. Le palline "punto" veniva normalmente tenute da un lato della cordicella (poniamo a sinitra), e le palline "linea" dal lato opposto (poniamo a destra). Per indicare sull'abaco una cifra, si spostavano verso il centro della cordicella la quantità di palline "punto" e "linea" necessarie a rappresentare il valore interessato. A quel punto, era facile eseguire per esempio addizioni e sottrazioni: bastava impostare sull'abaco il primo addendo, e poi aggiungere palline "punto" e "linea", cifra per cifra, in corrispondenza del valore del secondo addendo; ogni volta che raggiungevano al centro di una cordicella quattro punti, si riportavano tutte le palline "punto" a sinistra, e si aggiungeva una ulteriore pallina "linea"; al raggiungimento di tre palline "linea", si portavano tutte le palline linee della cordicella a destra, e si aggiungeva una pallina "punto" alla cordicella soprastante. I Maya erano calcolatori provetti: un abaco di questo tipo era spesso realizzato sotto forma di bracciale e indossato comodamente, fungendo sia da ornamento che da calcolatore. Insomma, un po' come un iPhone. Un bracciale con 10 cordicelle, come quello mostrato in figura, consentiva di eseguire rapidamente addizioni e sottrazioni di numeri fino a $1{,}024\times10^{13}$ – che in binario sarebbe 10010101000000101111100100000000000000000000. Si scriva un programma per macchina di Turing che, dati in ingresso due numeri Maya espressi come nell'esercizio precedente, separati dal simbolo "+", lasci sul nastro il risultato della loro addizione, sempre in formato Maya. Per questo esercizio, si consideri che la base di numerazione di tutte le cifre sia 20 (si sospetta che anche i Maya usassero una base uniforme 20 per i calcoli quotidiani, riservando la base mista 20/18/20 ... per i calcoli di tempo – ma tutte le iscrizioni pervenute fino a noi usano la base mista).



1 Questa scelta è ovviamente molto irregolare, ma gode di un importante vantaggio pratico: consente infatti di svolgere con grande facilità calcoli basati su 18 ventine, ovvero con base 360: una base comoda per il computo del tempo. E come si sa, i Maya avevano un debole per i calendari… Come conseguenza, nella seconda posizione non sono ammesse le cifre 18 e 19.

NASTRO INIZIALE	*NASTRO FINALE*
`..#0+///..`	`..#///..`
`///..+///..`	`.#//....`
`.#0#0+..#0#0`	`...#0#0`
`0+..#/`	`..#/`
`///+.`	`///.`
`/#//...#+/`	`/#///...`
`...+/#...`	`/#/.`
`//+//`	`.#0`

Esercizio 9: Programmi [Punti 25]. Anche un programma può essere codificato con una sequenza di Booleani (tipicamente molto lunga!). Si consideri per esempio un semplice processore a 4 bit, dotato di 16 registri da 4 bit ciascuno (ogni registro può contenere un numero da 0 a 15), che indicheremo con R0-R15 e di un set di istruzioni molto regolare, con ogni istruzione codificata in 8 bit, divisi in 4 bit per l'*opcode*, che indica quale operazione deve essere eseguita, e 4 bit per l'*operando*. Il nostro processore eseguirà, in ogni istante, una di queste istruzioni. Normalmente, dopo aver eseguito un'istruzione, il processore passerà ad eseguire la successiva; alcune istruzioni (dette *di salto*) però alterano questo ordine naturale. In particolare, nelle istruzioni di salto si intende che le istruzioni siano indicizzate a partire da 0, nell'ordine in cui compaiono, e che per i salti relativi, un valore di 0 indica l'istruzione di salto stessa. Per praticità, useremo il sistema di numerazione esadecimale (ovvero: a base 16) per esprimere il valore di un gruppo di 4 bit. Tutte le operazioni aritmetiche si intendono effettuate modulo 16 (quindi, i valori calcolati rimangono esprimibili in 4 bit).

La lista delle istruzioni disponibili, con il corrispondente significato e il loro codice numerico è il seguente:

Istruzione	Codifica	Effetto
INC R*n*	1*n*	Incrementa di 1 il valore contenuto nel registro *n*
DEC R*n*	2*n*	Decrementa di 1 il valore contenuto nel registro *n*
LDR R*n*	3*n*	Copia il valore contenuto nel registro *n* all'interno del registro 0
STR R*n*	4*n*	Copia il valore contenuto nel registro 0 all'interno del registro *n*
LDR N*n*	5*n*	Copia il valore *n* all'interno del registro 0
ADD R*n*	*6n*	Somma il valore del registro *n* a quello del registro 0, mettendo il risultato nel registro 0
JMP N*n*	7*n*	Salta all'istruzione di indice *n* (solo le prime 16 istruzioni sono raggiungibili in questo modo)
JMP R*n*	*8n*	Salta all'istruzione di indice pari al valore contenuto nel registro *n* (solo le prime 16 istruzioni sono raggiungibili in questo modo)
FWD R*n*	9*n*	Salta in avanti di un numero di istruzioni pari al contenuto del registro *n*
BWD R*n*	A*n*	Salta indietro di un numero di istruzioni pari al contenuto del registro *n*
JZE N*n*	B*n*	Salta avanti di *n* istruzioni, se il contenuto di R0 è 0
JNE Nn	C*n*	Salta avanti di *n* istruzioni, se il contenuto di R0 è diverso da 0
OUT R*n*	D*n*	Manda in output il valore contenuto nel registro *n*
STOP	00	Termina l'esecuzione
...	...	(altre istruzioni che non considereremo)

Per esempio, ecco alcuni programmi scritti per il nostro processore, con la relativa codifica del programma e il risultato che si ottiene eseguendo il programma a partire dalla sua prima istruzione:

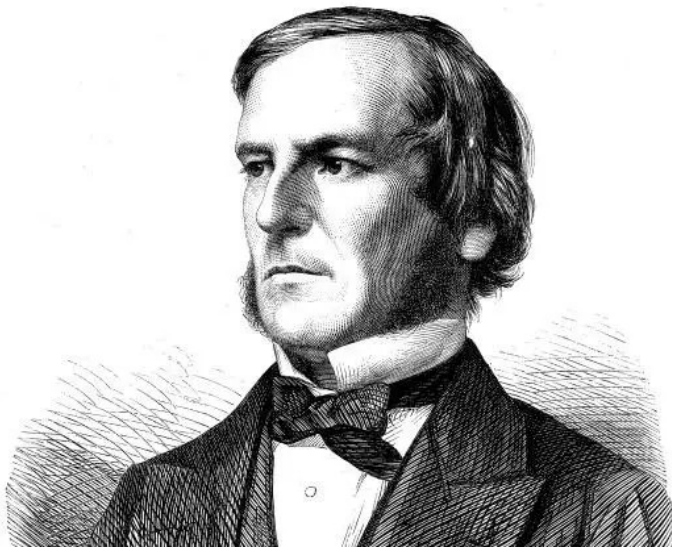
Programma	Codifica	Risultato
LDR NA STR R1 LDR N4 STR R2 OUT R1 INC R1 DEC R0 JZE N2 BWD R2 STOP	5A 41 54 42 D1 11 20 B2 A2 00	ABCD
LDR N1 ADD R0 ADD R0 ADD R0 OUT R0 STOP	51 60 60 60 D0 00	8
LDR N8 OUT R0 DEC R0 JZE N2 JMP N1 STOP	58 D0 20 B2 71 00	87654321
STOP	00	
LDR N3 OUT R0 DEC R0 STOP	53 D0 20 00	3

Si scriva un programma per macchina di Turing che, ricevuta sul nastro una sequenza nel formato *registri@programma*, in cui registri è una sequenza di 16 cifre esadecimali che rappresenta i valori inizialmente contenuti nei 16 registri, in ordine numerico crescente, e *programma* è la codifica esadecimale di un programma per il nostro processore, termini lasciando sul nastro il risultato in output dell'esecuzione del programma. Si assuma che i programmi in input siano *ben formati*, ovvero che non contengano istruzioni sconosciute, errori di salto fuori dal programma, ecc.

NASTRO INIZIALE	*NASTRO FINALE*
0000000000000000@5A415442D11120B2A200	ABCD
0000000000000000@51606060D000	8
2440AB78AA0F3640@58D020B27100	87654321
12345678901234 56@1414D4DA20D000	710

Esercizio 10: Un assemblatore [Punti 22]. Ci siamo quasi! Ora che sappiamo come codificare numeri, testi, immagini e programmi in sequenze di booleani, ed eseguire questi programmi, ci rimane solo da procurarci una ultima comodità. Vogliamo scrivere un *assemblatore*, ovvero un programma che trasformi il testo di un programma in formato *sorgente* (che sappiamo essere pur sempre una sequenza di Booleani, magari con le convenzioni di codifica come visto nell'esercizio 4) in *linguaggio macchina* eseguibile, come quello visto nell'esercizio 9 (anche il linguaggio macchina è, a meno di una traduzione da esadecimale a binario, una sequenza di Booleani). In effetti, un assemblatore è una forma semplice di *compilatore,* caratterizzata dal fatto che ogni istruzione del sorgente viene codificata in esattamente una istruzione in linguagigo macchina. Si scriva dunque un programma per macchina di Turing che, ricevuto sul nastro di input un programma in formato sorgente (con le istruzioni una dopo l'altra, separate da spazi), termini lasciando sul nastro il corrispondente programma eseguibile, secondo le convenzioni di cui all'esercizio 9. Si assuma che i programmi sorgente in input siano corretti.

NASTRO INIZIALE	*NASTRO FINALE*
LDR NA STR R1 LDR N4 STR R2 OUT R1 INC R1 DEC R0 JZE N2 BWD R2 STOP	5A415442D11120B2A200
LDR N1 ADD R0 ADD R0 ADD R0 OUT R0 STOP	51606060D000
LDR N8 OUT R0 DEC R0 JZE N2 JMP N1 STOP	58D020B27100



George Boole, 1815-1864. Figlio di un calzolaio, non andò mai oltre le elementari, rimase orfano a 16 anni, e studiò matematica, greco, latino e tedesco da autodidatta per diventare insegnante e mantenere così i tre fratelli più piccoli. A 19 anni fondò una sua scuola. A 25 anni venne nominato preside di una scuola più antica e prestigiosa. A 34 anni divenne il primo professore di Matematica del Queen's College, Cork, che era stato appena fondato. Morì a 49 anni, padre di 5 figli, dopo aver posto con il suo lavoro le basi dell'elettronica e dell'informatica moderne.