TestiGara - DiciottesimaEdizione

Diciottesima 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: Lo scorciatore. [Punti 1] In Informatica, una *struttura dati* è un modo di organizzare le informazioni nella memoria di un calcolatore in modo da facilitare l'esecuzione di un insieme predefinito di operazioni. La struttura dati più semplice è la *sequenza*, che può essere facilmente rappresentata da una sequenza di simboli sul nastro. Sulle sequenze si possono facilmente programmare molte operazioni, per esempio lo *scorciamento*. Si scriva un programma per macchina di Turing che, ricevuta sul nastro di input una sequenza di simboli A, lasci sul nastro la stessa sequenza, accorciata di una A.

NASTRO INIZIALE	*NASTRO FINALE*
AAAA	AAA
A	
AAAAAAAAAA	AAAAAAAAA

Esercizio 2: L'inseritore. [Punti 3] Si scriva un programma per macchina di Turing che, ricevuta sul nastro di input una sequenza di simboli sull'alfabeto {A, B}, lasci sul nastro la stessa sequenza, in cui dopo ogni A è stata inserita una C.

NASTRO INIZIALE	*NASTRO FINALE*
ABABAAB	ACBACBACACB
BBBB	BBBB
BBABAAA	BBACBACACAC
A	AC

Esercizio 3: Il sommatore. [Punti 5] Si scriva un programma per macchina di Turing che, ricevuta in ingresso una sequenza di simboli sull'alfabeto {0, 1, 2, 3, 4} di lunghezza pari, lasci sul nastro la sequenza ottenuta sommando a due a due gli elementi della sequenza di output (usando come alfabeto le cifre decimali}.

NASTRO INIZIALE	*NASTRO FINALE*
12344321	3773
0413223240	44454

00114140223244	0254458
00	0

Esercizio 4: La pila [Punti 7] Un'altra struttura dati molto popolare è la *pila* o *stack*. Una pila è una sequenza in cui nuovi elementi vengono aggiunti all'inizio della sequenza (operazione *push*, che indicheremo con P seguito da un valore sull'alfabeto 0-9), e prelevati dall'inizio (operazione *pop*, che indicheremo con Q), col che vengono rimossi dalla pila. Si scriva un programma per macchina di Turing che, ricevuta sul nastro di input una sequenza di operazioni di push e pop, lasci sul nastro la sequenza di valori che descrivono lo stato finale della pila, dopo aver applicato tutte le operazioni di push e pop, da sinistra a destra, partendo dalla pila vuota. Viene garantito che la sequenza di operazioni non tenterà di eseguire delle pop con la pila vuota.

NASTRO INIZIALE	*NASTRO FINALE*
P3P4P1	143
P3P4QP1	13
P8P3QQP3P8P8	883
P2P4P5QQQ	
P1QP1QP1P2P3Q	21

Esercizio 5: Un calcolatore modulare. [Punti 10] Un uso frequente delle pile consiste nel valutare espressioni (per esempio, espressioni aritmetiche). In questa applicazione, gli operandi vengono inseriti su una pila, mentre gli operatori (per esempio: +) estraggono due operandi dalla pila, calcolano il risultato, e lo inseriscono in cima alla pila. Nel nostro caso, vogliamo effettuare operazioni in *aritmetica modulare*, in cui i risultati sono sempre calcolati modulo un valore detto base. In particolare, in $\mathbb{Z}_4$ gli unici valori possibili sono 0, 1, 2 e 3 (ciascuno rappresentato dalla cifra corrispondente); le operazioni vengono calcolate come di consueto, ma poi il risultato è dato modulo 4 (ovvero: resto della divisione per 4). Per esempio, in $\mathbb{Z}_4$ abbiamo che 1+1=2, 3+2=1, 3-1=2, 2-2=0, 1-2=3, 3×3=1, 2×2=0. Si scriva un programma per macchina di Turing che, data una pila sull'alfabeto 0-3, seguita da un simbolo $ e da una sequenza di operatori sull'alfabeto {A, S, M} (che stanno per: Addizione, Sottrazione, Moltiplicazione), lasci sul nastro la pila risultante dopo aver eseguito le operazioni, in ordine da sinistra a destra, oppure “E” (che sta per: Errore) nel caso si tenti di eseguire un'operazione senza che siano presenti almeno due operandi sulla pila.

NASTRO INIZIALE	*NASTRO FINALE*
01$A	1
0123$AAA	2
22$M	0
2323$M	223
321$SA	2
3031$MMA	1
21$SA	E

Esercizio 6: L'albero. [Punti 15] Le strutture dati ad *albero* sono anch'esse molto popolari in Informatica. Gli alberi sono insiemi di *nodi*, in cui ogni nodo (tranne uno, detto *radice*) ha esattamente un *padre*, e può avere 0 o più *figli* (di cui, ovviamente, sarà padre). La radice non ha padre, ma può avere figli. I nodi con 0 figli sono detti *foglie*. Un tipo particolare di albero, detto *albero binario* è definito con il vincolo aggiuntivo che ogni nodo può avere al più 2 figli. Possiamo rappresentare su nastro un albero con la convenzione seguente: una foglia è rappresentata da un simbolo sull'alfabeto A-Z; gli altri nodi sono rappresentati da un simbolo sullo stesso alfabeto, seguito da una coppia di parentesi che racchiudono 1 o 2 figli. Per esempio, la sequenza **F(B(AD(CE))G(I(H)))** rappresenta l'albero dato, in forma grafica, nella figura accanto.

fig1

Sugli alberi si definiscono molti algoritmi interessanti. Uno dei più semplici è il calcolo della *profondità*, che si ottiene contando la lunghezza del più lungo cammino dalla radice a una foglia. Per esempio, l'albero in figura ha profondità 4 (che è la lunghezza dei cammini equivalenti F-B-D-C, F-B-D-E o F-G-I-H). Si scriva un programma per macchina di Turing che, ricevuta sul nastro di input la rappresentazione di un albero come descritto sopra, lasci sul nastro la sua profondità.

NASTRO INIZIALE	*NASTRO FINALE*
F(B(AD(CE))G(I(H)))	4
R	1
R(AB)	2
R(A(BC(D(F(G))E))H(IJ))	6
X(A(A)Z(Z))	3

Esercizio 7: Visita per livelli. [Punti 17] Sugli alberi, definiti come nell'esercizio precedente, sono definite anche varie operazioni di *visita*, che consistono nell'elencare i nomi dei nodi che si incontrano seguendo una determinata procedura. Per esempio, la *visita in profondità*, o in *post-ordine*, consiste nel visitare prima i figli di un nodo, e poi il nodo stesso, partendo dalla radice dell'albero. Una visita in profondità applicata all'albero illustrato nella figura relativa all'Esercizio 6 produrrebbe quindi la sequenza: ACEDBHIGF. La *visita per livelli* consiste invece nell'elencare i nodi a seconda della loro distanza dalla radice (ovvero: la *profondità* di cui all'esercizio precedente), partendo dalla radice stessa. Una visita per livelli applicata all'albero illustrato nella figura relativa all'Esercizio 6 produrrebbe quindi la sequenza: FBGADICEH. Si scriva un programma per macchina di Turing che, ricevuto sul nastro di input un albero codificato come sopra, lasci come risultato la relativa visita per livelli.

NASTRO INIZIALE	*NASTRO FINALE*
F(B(AD(CE))G(I(H)))	FBGADICEH
R(A(BC(D(F(G))E))H(IJ))	RAHBCIJDEFG
X(AB(C))	XABC
X(A(C)B)	XABC
R	R

Esercizio 8: Potature e innesti. [Punti 21] Sugli alberi, definiti come nell'esercizio precedente, definiamo le operazioni *pota* e *innesta* come segue: dato un albero *T*, e un nodo *n* (rappresentato da un simbolo sull'alfabeto A-Z), l'operazione di *pota* elimina da *T* tutti i sottoalberi radicati in nodi etichettati *n*, incluso il nodo *n* stesso, e restituisce l'albero potato risultante. L'operazione *innesta* ha come argomenti un albero *T*, un nodo *n* (che deve essere una foglia) e un secondo albero *T'*. L'operazione sostituisce ogni occorrenza di *n* in *T* con una copia di *T'*, e restituisce l'albero innestato risultante. Si scriva un programma per macchina di Turing che, ricevuti sul nastro un albero iniziale *T*, e una sequenza di operazioni di pota (codificata come: "$*n*") e innesta (codificata come: "#*nT*""), lasci sul nastro l'albero risultante dopo aver applicato tutte le operazioni in sequenza.

NASTRO INIZIALE	*NASTRO FINALE*
F(B(AD(CE))G(I(H)))$D	F(B(A)G(I(H)))
F(B(AD(CE))G(I(H)))#CX(YZ)	F(B(AD(X(YZ)E))G(I(H)))
F(B(AD(CE))G(I(H)))CE#DW	F(B(AW)G(I(H)))
X(AB)#AA(RS(T))$R	X(A(S(T))B)

Esercizio 9: La mappa [Punti 23] Una *mappa* (anche detta *array associativo*) è una struttura dati che associa un elemento, detto *chiave*, a un altro, detto *valore*. Noi useremo chiavi sull'alfabeto A-Z, e valori sull'alfabeto 0-9. Sulle mappe sono definite due operazioni: *put*, che riceve come argomento una chiave e

un valore e stabilisce l'associazione fra la chiave e il valore nella mappa, e *get*, che riceve come argomento una chiave, e restituisce il valore corrispondente alla chiave nella mappa, se presente, oppure il valore 0 nel caso la chiave non sia presente. Per questo esercizio, aggiungeremo una operazione non standard, *add*, che riceve come argomento una chiave e un valore, e inserisce nella mappa, in corrispondenza della chiave, il valore precedentemente contenuto, sommato al valore dato in modulo 10. Se la chiave non era presente nella mappa, *add* si comporta come *put*.

In questo esercizio, la codifica della mappa sul nastro è a vostra discrezione: dovete progettare voi la struttura dati. La codifica delle operazioni è invece come segue: *put* è indicato da P*kv*, *get* è indicato da G*k*, e *add* è indicato da A*kv*, dove *k* è la chiave, e *v* è il valore.

Si scriva un programma per macchina di Turing che, ricevuta sul nastro una sequenza di operazioni put, get e add, codificate come sopra, le esegua in ordine da sinistra a destra, partendo dalla mappa vuota, e lasci sul nastro al termine della computazione la sequenza di valori estratti dalle operazioni get, in ordine di esecuzione.

NASTRO INIZIALE	*NASTRO FINALE*
PA3PB4AA3GAGB	64
AX5	
AX5AX4GXAX1GX	90
PS3GSGSGSAS1GSGQ	33340
PA0AA0PB3GAGBPB8GBGA	0380

Esercizio 10: L'albero binario di ricerca. [Punti 30] Un *albero binario di ricerca* è un albero binario con la proprietà che, in ogni nodo, il figlio sinistro (se esiste) ha un valore minore del padre, e il figlio destro (se esiste) ha un valore maggiore del padre. Se un nodo ha un solo figlio, assumiamo che sia quello sinistro. Ovviamente, sui valori associati ai nodi deve essere definito un ordinamento di qualche tipo, per esempio alfabetico o numerico. Dato un insieme di nodi, esistono molti possibili alberi binari di ricerca che soddisfano il criterio enunciato. Per esempio, l'insieme di nodi {A, B, C, D} con ordinamento alfabetico sul nome potrebbe essere rappresentato da uno qualunque di questi alberi:

fig 2

Per questo esercizio, considereremo come albero *canonico* quello in cui tutti i livelli sono riempiti, tranne al più l'ultimo: e in questo caso, tutti i nodi del livello più basso sono accumulati a sinistra. In un albero canonico di profondità *n*, tutti i nodi nei livelli da 1 a *n*-2 hanno esattamente due figli; i nodi del livello *n*-1 avranno: i nodi più a sinistra esattamente due figli (che saranno foglie, al livello *n*), poi se il numero totale di nodi è dispari avremo un nodo con un solo figlio (che sarà una foglia, al livello *n*), e infine tutti i nodi più a destra avranno zero figli – ovvero, saranno foglie essi stessi, al livello *n*-1. Nell'esempio precedente, l'albero centrale è canonico, mentre i due laterali non lo sono.

Si scriva un programma per macchina di Turing che, ricevuta sul nastro di input una sequenza di operazioni *put* e *add* su una mappa (come nell'Esercizio 9), lasci sul nastro l'albero binario di ricerca canonico i cui nodi sono le chiavi della mappa costruita applicando la sequenza di operazioni data, partendo dalla mappa vuota, e l'ordinamento fra i nodi segua l'ordine numerico dei valori corrispondenti alle chiavi nella mappa. L'albero dovrà essere codificato come descritto nell'Esercizio 6. Si assuma che al termine della costruzione della mappa, non vi saranno nella mappa due chiavi con lo stesso valore numerico.

NASTRO INIZIALE	*NASTRO FINALE*
PA1PB2PC3	B(AC)
PA1PB2PC3PD4PE5	D(B(AC)E)

PA3PB2PC1	B(CA)
PA1PB2PC3AA5	C(BA)
AA8PB3PC1PD1AC5PD4PE0PF9	C(B(ED)F(A))
PR3	R