

TPFI 2022/23

Hw 5: A taste of C

assegnato: 27 maggio 2023, consegna 12 giugno 2023

Esercizio 1 Scrivere una funzione C che elimina i duplicati da una lista. Ad esempio, ricevendo in ingresso la lista $[7, 1, 1, 3, 7, 1, 3]$ occorre produrre come risultato la lista $[7, 1, 3]$ *mantenendo l'ordine delle prime occorrenze* di ciascun valore.

L'algoritmo elementare per risolvere questo problema è $\Theta(n^2)$, dove n è la lunghezza della lista.

[FACOLTATIVO] Sarebbe interessante scrivere una soluzione di complessità $\Theta(n \log n)$. Altrettanto interessante confrontare tale soluzione con la soluzione che si può dare in Haskell allo stesso problema.

Esercizio 2 (ALBERO DELLE CHIAMATE RICORSIVE). Considerate la seguente funzione ricorsiva che calcola i coefficienti binomiali.

```
int cbin(int n, int k){
    if (n==k || n==0) return 1;
    return cbin(n-1,k-1)+cbin(n-1,k);
}
```

Dare prima la definizione di un tipo `cBinTree` adatto a memorizzare in un albero binario i valori dei parametri n e k e il valore calcolato dalla in tale chiamata (ogni nodo di un `cbinTree` contiene quindi tre valori.).

Scrivere poi una funzione C che genera l'albero delle chiamate ricorsive della funzione `int cbin(int n, int k)`, memorizzando in ogni nodo i valori dei parametri n e k e il valore calcolato dalla in tale chiamata.

ESEMPIO Se fosse invocata con paramentri $n = 5$ e $k = 3$, la funzione dovrebbe produrre l'albero in Fig. 1, dove il valore ritornato è scritto in grassetto.

[FACOLTATIVO] Siccome, come potete osservare, ci sono numerosi sotto-alberi ripetuti nell'albero delle chiamate, è interessante costruire il grafo aciclico delle chiamate ricorsive, ma questa volta allocando un unico nodo per eventuali chiamate ripetute (con gli stessi valori per i parametri n e k), evitando quindi la replicazione di sotto-alberi (vedi Fig. 2). Osservate che in

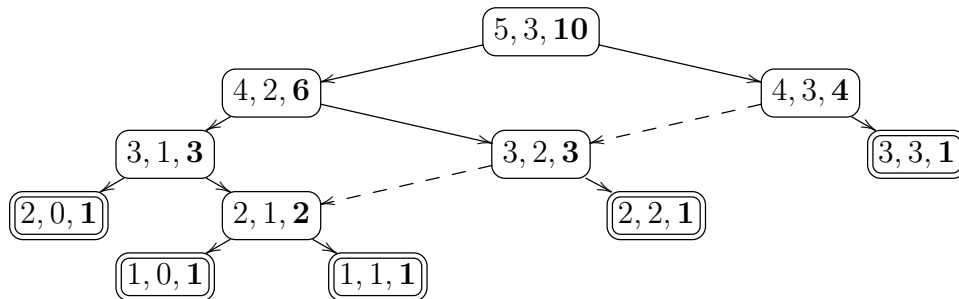
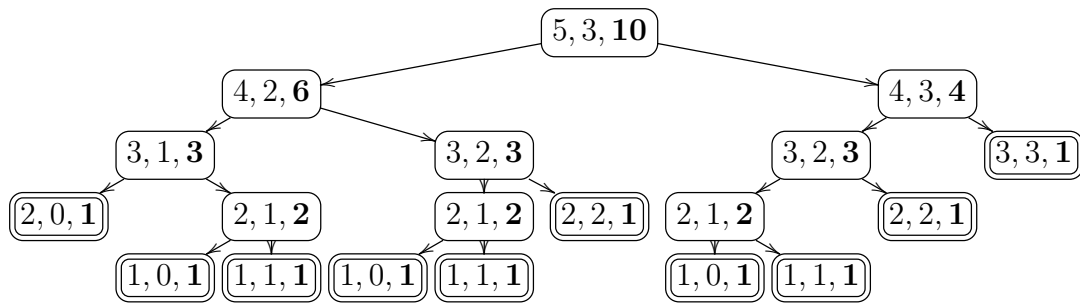


Figura 2: “Albero” (grafo aciclico) che deve essere generato dalla funzione `cBinInvocationSharing(5,3)`, evitando replicazione di sottoalberi.

questo caso ci sono cammini diversi che finiscono nello stesso nodo e osservate anche che non occorre cambiare la definizione del tipo `cBinTree`, né funzioni di stampa o di visita.

ESEMPIO: In Fig. 2, l’“albero” risultante sullo stesso esempio. Gli archi tratteggiati sono quelli che “riportano” su nodi già precedentemente allocati.

Esercizio 3 (QUELLO CHE IN HASKELL NON SI PUÒ FARE) Scrivere una funzione C che implementi il *crivello di Eulero*. Non è ovvio “saltare” in modo efficiente i numeri già cancellati per trarne vantaggio nelle successive cancellazioni. La soluzione che vi propongo di implementare consiste nell’usare un vettore di coppie di naturali, *succ* e *prec*, come una *lista doppiamente concatenata* in cui nella posizione *i*, se *i* non è stato cancellato, *succ* è il numero di posizioni che occorre saltare per andare al prossimo numero non cancellato, mentre *prec* è il numero di posizioni che occorre saltare (all’indietro) per andare al precedente numero non cancellato.

Potete ad esempio definire un tipo `Pair` che è una coppia di interi `succ` e `prec` e definiamo un vettore di `Pair`. Questo vettore va inizializzato con tutti 1 (che significa appunto che tutti i numeri sono ancora potenziali primi). Quindi lo stato del vettore, inizialmente è il seguente (dove `#` significa ‘non rilevante’):

<i>pos</i>	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
<i>succ</i>	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
<i>prec</i>	#	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Dopo aver cancellato i multipli di due, il vettore avrà i seguenti valori:

<i>pos</i>	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
<i>succ</i>	1	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#
<i>prec</i>	#	1	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#	2	#

Ora, partendo da 3 posso facilmente saltare sui numeri non cancellati. Moltiplicando questi per 3 ottengo quelli da cancellare in questa iterazione, e cioè 9, 15, 21, ottenendo la seguente situazione:

<i>pos</i>	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
<i>succ</i>	1	2	#	2	#	4	#	#	#	2	#	4	#	#	#	2	#	4	#	#	#	2	#
<i>prec</i>	#	1	#	2	#	2	#	#	#	4	#	2	#	#	#	4	#	2	#	#	#	4	#

Nell’esempio, a questo punto ho finito, perché il prossimo numero non cancellato è il 5 e $5^2 > 24$. Partendo da 2 e scorrendo il vettore usando i puntatori *succ* posso stampare tutti i numeri non cancellati che sono a questo punto necessariamente primi (scrivere una funzione `printPrimes` allo scopo).

Voi dovete scrivere una funzione: `Pair* eulerSieve(int n);` che restituisce un vettore di coppie da cui sia possibile ricostruire tutti i numeri primi da 2 a n .

OSSERVAZIONI: I puntatori *prev* servono essenzialmente per effettuare in modo efficiente le operazioni di cancellazione. Le cancellazioni sono ‘problematiche’ perché sono operazioni distruttive sulla struttura dati e potrebbero, se fatte con poca cura, rendere inconsistente lo stato del vettore.

SPERIMENTAZIONI: Verificare che questo programma risulta effettivamente più efficiente del crivello di Eratostene. Ovviamente, il guadagno asintotico è modesto ($\Theta(n)$ invece che $\Theta(n \ln \ln n)$) e fa operazioni più complicate. Occorrerà provarlo per un qualche n sufficientemente grande (ordine di migliaia o milioni...).