TPFI 2020/21
Hw 5: A taste of C

assegnato: 25 maggio 2021, consegna 8 giugno 2021

Esercizio 1 (Aritmetica Intera) Cosa restituisce la funzione `pred` iterativa che non termina (vedi slides Lezione **17**)? E l'analoga funzione in Python? E quella ricorsiva?

Cosa ne deducete? Scrivere anche dei programmi che determino il massimo valore rappresentabile nei tipi `int`, `unsigned int` e `long long int` (i risultati saranno dipendenti dalla macchina su cui eseguono questi programmi).

Esercizio 2 (Liste I) Scrivere la funzione `removeIt(int x, lista L)` (vedi slides Lezione **22**) usando la tecnica di usare una lista di appoggio per il risultato, come visto qualche slide prima per la funzione `reverse`.

Esercizio 3 (Liste II) Scrivere una funzione di ordinamento Selection Sort.

1. in una prima versione, calcolate il minimo della parte non ancora ordinata e scambiate il campo informazione;

2. in una seconda versione, provate calcolare invece i massimi, "appoggiarli" in una lista ausiliaria in cui aggiungerli via via in testa (staccandoli dalla lista di input).

Esercizio 4 Data la matrice:

$$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

si può far vedere (per induzione, per esempio) che:

$$F^{n-1} = \begin{pmatrix} fib_n & fib_{n-1} \\ fib_{n-1} & fib_{n-2} \end{pmatrix}$$

Usare questo fatto e lo schema della moltiplicazione egiziana applicata all'esponenziale di matrici, per calcolare in modo efficiente fib_n, l'$n$esimo numero di Fibonacci. Qual la complessit ottenuta?

Esercizio 5 (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.

Definiamo un tipo `Pair` che è una coppia di interi `succ` e `prec` e definiamo un vettore di `Pair` (vedi file `eulero.h`). 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'):

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

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

pos	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
succ	1	2	#	2	#	4	#	#	#	2	#	4	#	#	#	2	#	4	#	#	#	2	#
prec	#	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 (vedi funzione `printPrimes` nel main fornito).

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 ($\ln \ln n$) è modesto e fa operazioni più complicate. Occorrerà provarlo per un qualche n sufficientemente grande (ordine di migliaia o milioni...).