Informatica Generale
Homework 2/2019: Piramidi e Crivelli

Ivano Salvo – Sapienza Università di Roma – 11/5/2019

Esercizio 1. (Matrice a Piramide) Scrivere una funzione C di prototipo:

```
int** piramide(int n)
   /* PREC: n>0 */
```

che preso come parametro di input un numero intero $n > 0$ alloca una matrice dinamica di dimensione $n \times n$, caricandola a *piramide*, cioè con valori costanti sulle cornici della matrice e crescenti verso il centro.

Tutti gli elementi del bordo (che stanno quindi nella prima e ultima riga e nella prima e ultima colonna) devono essere caricati con 1. Gli elementi della cornice subito più interna col valore 2 e così via fino al centro (che è un quadrato per le matrici di lato pari e un singolo valore per le matrici di lato dispari).

Suggerimento: forse è più semplice trovare una funzione degli indici i, j e del lato n con cui calcolare il numero da mettere in posizione i, j.

Esempi: Ecco ad esempio le matrici a piramide di lato 1, 2, 3, 4 e 5:

```
1     1 1     1 1 1     1 1 1 1     1 1 1 1 1
      1 1     1 2 1     1 2 2 1     1 2 2 2 1
              1 1 1     1 2 2 1     1 2 3 2 1
                        1 1 1 1     1 2 2 2 1
                                    1 1 1 1 1
```

Esercizio 2 Sia P una lista non vuota che contiene ordinatamente i primi k numeri primi ($k > 0$). Scrivere il codice di una funzione C di prototipo:

```
int nthPrime(listFirstLast P, int n);
   /* PREC: P non vuota, contiene ordinatamente
      i primi k numeri primi */
```

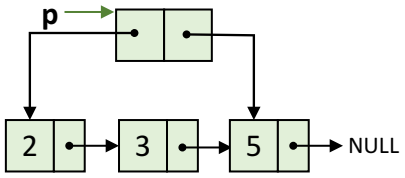
che restituisce l'n-esimo numero primo.

Se $n \leq k$, `nthPrime` si limita a ritornare il contenuto dell'n-esimo nodo di P. Altrimenti, oltre a ritornare l'n-esimo numero primo, modifica P aggiungendovi in coda altri $n - k$ numeri primi.

Il tipo `listFirstLast` è semplicemente il tipo `lista` visto a lezione, in cui invece del puntatore alla testa, avete un puntatore a un record contenente un puntatore alla testa ed uno alla coda: questo tipo di lista permette di accedere direttamente all'ultimo nodo e all'ultimo valore memorizzato, permettendo di rendere più efficienti la ricerca del prossimo elemento da inserire, così come il successivo inserimento.

Per esemplificare ulteriormente cosa deve fare la funzione `nthPrime`, spero risulti eloquente la Figura 1.

a) Sia `p` come in figura



b) risultato in memoria ***dopo*** l'esecuzione di `nthPrime(p,6);` (che torna `13`)

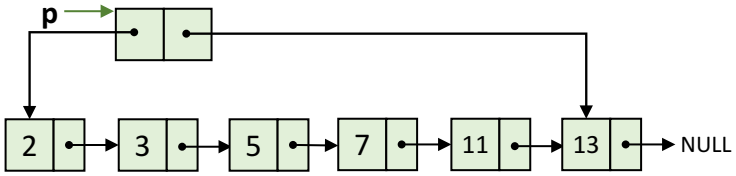


Figura 1: Esempio di esecuzione della funzione `nthPrime` (Esercizio **2**).

Potete usare le funzioni contenute nel file `list.c` che fornisce le operazioni base di inserimento, creazione e stampa di liste. Per compilare il tutto dovrete includere la libreria `list.h`.

SUGGERIMENTO: Potrebbe essere utile, nell'ottica di strutturare il codice, definirsi una funzione `void prossimoPrimo(listFirstLast P)`, che aggiunge in coda a `P` il prossimo numero primo nella sequenza dei primi.

Esercizio 3. (Crivello di Eulero) Tutti dovreste conoscere l'algoritmo del crivello di Eratostene per generare tutti i numeri primi fino a un certo n. In questo algoritmo, prima vengono scritti tutti i numeri fino a n e poi vengono cancellati i multipli di 2, poi quelli di 3, e in generale quelli del prossimo numero non ancora cancellato p (che è necessariamente un primo).

Anche cominciando un ciclo di cancellazione da p^2 e proseguendo a passo p (cancellando quindi $p^2 + ip$ fino a che $p^2 + ip \leq n$), viene fatto del lavoro inutile. Infatti, ad esempio, cancellando i multipli di 3, vengono cancellati il 12, il 18, il 24 etc. che sono anche multipli di 2. Lo stesso accade con i multipli di 5, in quanto vengono cancellati il 30, il 40, il 45 etc. Si può dimostrare che la complessità di questo algoritmo è $\mathcal{O}(n \ln \ln n)$ per generare i primi fino ad n. Evitando di ri-cancellare multipli già cancellati, possiamo ottenere un algoritmo, il *crivello di Eulero* appunto, di complessità $\mathcal{O}(n)$ in quanto $\ln \ln n$ è il numero medio di divisori primi di un numero composto (e quindi anche il numero medio di volte che un numero composto viene ricancellato durante l'esecuzione del Crivello di Eratostene).

Tuttavia non è ovvio "saltare" in modo efficiente i numeri già cancellati per trarre vantaggio da quest'idea. 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 viene 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 al solito # 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);
   /* PREC: n>2 */
```

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 efficienti le operazioni di cancellazione. Le cancellazioni sono 'problematiche' perché sono operazioni distruttive sulla struttura dati e potrebbero, se fatte troppo presto o 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...).

Note Pratiche

Dovete consegnare semplicemente la funzione richiesta, **rispettando il prototipo**. Il vostro file può contenere eventuali altre funzioni ausiliarie. Non dovete lasciare **nessuna** istruzione di input/output nel codice. Potete verificare se il vostro programma funziona, utilizzando i files `main-2019-2-1.c`, `main-2019-2-2.c` e `main-2019-2-3.c`, forniti nella pagina degli Homework.

Potete chiamare come volete il file `.c` contenenti la funzione richiesta (ed eventuali funzioni ausiliarie): il sistema li rinominerà automaticamente usando il vostro twiki name. Se il vostro twiki name fosse `LilyEvans`, essi verranno chiamati `LilyEvans.1.c`, `LilyEvans.2.c`, e `LilyEvans.3.c`. I comandi utilizzati per la compilazione saranno i seguenti (osservate che nel 2 verrà compilato anche il codice fornito nel file `list.c`):

```
gcc -std=c99 main-2019-2-1.c LilyEvans.1.c
gcc -std=c99 main-2019-2-2.c list.c LilyEvans.2.c
gcc -std=c99 main-2019-2-3.c LilyEvans.3.c
```

Osservate che i files dovrebbero poter essere compilati separatamente con l'opzione `-c` che traduce in linguaggio macchina, ma senza generare un eseguibile (`gcc -c -std=c99 LilyEvans.1.c` e `gcc -c -std=c99 main-2019-2-1.c` dovrebbero generare con successo un file chiamato rispettivamente `LilyEvans.1.o` e `main-2019-2-1.o`.