Facultad de Ingeniería

Universidad de Buenos Aires

Teoría de Algoritmos (75.29/95.06)

Trabajo Práctico N°1

2^{er} Cuatrimestre, 2018

	00000	
	00000	
Geloso, Federico Pedro	98138	gelosofederico@gmail.com
	00000	

1. Parte 1: Variante de Gale Shapley

2. Parte 2: Complejidad algorítmica

Para esta sección se consideró el problema de calcular todos los números primos menor que un valor N. Se consideraron 2 algoritmos para resolverlo: un algoritmo por fuerza bruta, que prueba la divisibilidad con todos los números menores, y el algoritmo de la criba de Erastótenes.

El primer algoritmo es fácil de ver y analizar. En pseudocódigo:

```
PrimosNaive(N):
    Sea L una lista.
    EsPrimo := Verdadero.
    Para cada i entre 2 y N:
        j = 2.
        Mientras j < i y EsPrimo = Verdadero:
            Si i es divisible por j:
                EsPrimo := Falso.
            j = j + 1.
        Si EsPrimo = Verdadero:
            Agregar i al final de L.
    Devolver L
```

En esto se puede ver que L contendrá todos los números que pasaron el chequeo. La complejidad temporal del algoritmo se puede ver que es $\mathcal{O}(n^2)$, donde n es el número de entrada. Es así ya que itera por cada número todos los números menores que este, teniendo entonces $T(n) = \sum_{i=2}^{N} \sum_{j=2}^{i} 1 = \sum_{i=2}^{N} i - 2 = (N-1)(N-2)/2$.

Este algoritmo se podría mejorar, pero se va a analizar el otro algoritmos propuesto. En pseudocódigo:

```
PrimosCriba(N):
    Sea L una lista de booleanos de 0 a N-1, seteados en Verdadero.
    Para cada i de 2 a ⌊√N⌋:
        Si L[i] = Verdadero:
            Para cada j de i² a N de a pasos de a i:
                L[j] := Falso
    Sea P la lista de números i tal que L[i] = Verdadero
    Devolver P
```

Se puede ver que para cada primo p menor que $\sqrt{N}$, hace $\frac{N}{p}$ operaciones. A partir del segundo teorema de Mertens, se tiene que:

$$\lim_{n\to\infty} \sum_{p<x} \frac{1}{p} = ln\ ln\ n + M \tag{1}$$

Siendo p los números primos y M una constante que a efectos del orden asintótico no es relevante. La optimización de hacer hasta $\sqrt{N}$ no es relevante en el orden asintótico tampoco, ya que estaría dentro del logaritmo y sale multiplicando del primer logaritmo y sumando del segundo. Con esto, queda que $N \sum_{p<\sqrt{N}} \frac{1}{p} \in \mathcal{O}(N \cdot log\ log\ N)$.

Todo esto asume que el acceso a la lista L es $\mathcal{O}(1)$. Al programarlo se utilizó la estructura *list* de Python, que asegura el acceso en tiempo constante. Dado que se pide devolver una lista con los números, al final se recorre esta estructura y se agregan a la lista final a devolver todos los que terminaron siendo Verdaderos, lo cual tiene un coste lineal. A su vez se tuvo en cuenta una optimización más, que es que los primos i después del 2 se pueden recorrer desde i^2 a N en pasos de a $2i$, ya que se sabe que i es impar (el único primo par es 2) y entonces $i^2 + n \cdot i$ es par si n es impar, lo cual ya se sabe que no es primo. Por esto se separaron el caso del 2 del resto.

2.1. Documentación del programa

Para ejecutar el programa se deben pasar por línea de comandos el número N, el método como E (Erastótenes) o F (fuerza bruta), y si se quieren imprimir los datos, se pasa -T para escribir el tiempo, y -L para la lista. Esto se escribirá en la salida estándar, por lo que para escribir en un archivo se puede redirigir el flujo de salida.

2.2. Corridas de prueba