75.29 - Teoría de Algoritmos

TP Nº 1

93542 - Bollero, Carlos
98059 - Czop, Guillermo
98017 - Errázquin, Martín
85826 - Escobar, Cynthia

1 Variante de Gale Shapley

En la ciudad de Buenos Aires se llevarán a cabo N recitales en diferentes barrios. Se ha realizado una convocatoria a bandas de música del underground. Como resultado, se han presentado M bandas.

Cada banda tiene diferentes preferencias sobre en cual recital tocar. Asimismo cada comité organizador del recital tiene un listado de preferencias de bandas. Los organizadores se comunican con las bandas para ofrecerles participar y pueden contratar como mucho a X bandas distintas. De la misma manera, una banda puede participar en como mucho Y recitales.

Se solicita:

1. Utilizar una variante de Gale-Shapley para resolver el problema. Explicarlo y presentar el pseudocódigo.
2. Analizar y justificar la complejidad del algoritmo
3. Analice las condiciones para que el algoritmo propuesto retorna un matching estable y/o perfecto.
4. Suponga que los organizadores pueden tener preferencias similares sobre diferentes bandas y viceversa. ¿Cómo afecta esto en el algoritmo? ¿Considere que en caso de empate el involucrado decida desempatar tirando una moneda. ¿Cómo se ve afectado el proceso?
5. Programe la solución propuesta en el punto 1. Genere archivos random y ejecute el programa para los siguientes valores de N, M, X e Y. ¿Qué ocurre en cada caso?:
   - N = 10, M = 10, X = 1, Y = 1
   - N = 10, M = 5, X = 2, Y = 2
   - N = 10, M = 5, X = 2, Y = 1
6. Compare la complejidad teórica con la del algoritmo programado.

Información adicional:

- Cada recital contará con un archivo llamado “recital_[nro]”. en cada línea estarán en forma ordenada decreciente sus preferencias de bandas.
- Cada banda contará con un archivo llamado “banda_[nro]”. En cada línea estarán en forma ordenada decreciente sus preferencias de recitales.
- Las bandas estarán identificadas por números entre el 1 y el M.
- Los recitales estarán identificados por números entre el 1 y el N.
- Al programa se le deben pasar por parametro de inicio los valores numéricos enteros en el siguiente orden: N M X Y

1.

S = {}

Mientras haya recitales que todavía puedan contratar bandas y no se haya comunicado con todas las bandas:

Seleccionar recital **r** de los que cumplen la condición anterior

Sea **b** la banda que mejor rankea en la lista de preferencias de **r**, y **b** no fue contactada previamente por **r**

Si **b** participa en **k** recitales, con 0<**k** < **Y** entonces:

se agrega **(b, r)** a **S**

se aumenta la cantidad de bandas que contrató el recital **r**

se aumenta la cantidad de recitales que aceptó la banda **b**

Si **b** ya participa en **Y** recitales, entonces:

Sea **~r** el recital menos preferido de **b**

Por cada recital **r'** previamente aceptado por **b**

Si **b** prefiere más al recital **r'** que al nuevo **r**, continúo

Si **b** prefiere más el nuevo recital **r** que al existente **r'**

Si **~r** no está seteado o si **b** prefiere más a **~r** que a **r'** -> **~r** = **r'**

Si existe **~r** entonces:

b rechaza a **~r** -> quita **(b,~r)** de **S**

~r se encola para ser procesada posteriormente

se disminuye la cantidad de bandas que tiene contratadas **~r**

b acepta a **r** -> se agrega **(b, r)** a **S**

se aumenta la cantidad de bandas que contrató el recital **r**

retorna **S**

2. Partiendo de que la complejidad de Gale-Shapley es O(n^2), podemos decir que para esta variante tenemos:

   (a) Inicialmente, una complejidad O(NxX), ya que sabemos que cada organizador de los N recitales posibles va a intentar llenar el cupo máximo de X bandas que puede contratar.

   (b) Para completar el cupo de bandas a ser contratadas se deberán recorrer a su vez, en el peor de los casos, las M bandas configuradas en orden de preferencia. Pero como para llenar cada cupo siempre partiremos desde la última banda verificada, en realidad en el peor de los casos recorreremos las M bandas, independientemente de la cantidad de cupos a completar, con lo que tenemos una complejidad de O(NxM).

   (c) Finalmente, si agregamos la variación de que cada banda puede participar de Y recitales, en el peor de los casos deberíamos validar contra cada uno de los Y recitales ya aceptados de cada banda, para comprobar que una propuesta nueva no es mejor. Con esto podemos decir que la complejidad final sería O(NxMxY)

3. Para esta versión de Gale-Shapley consideramos que una solución es perfecta si todo recital logra contratar alguna banda, y toda banda puede participar en algún recital.

   En el único caso en el que habrá match perfecto es cuando N==M y X==Y.

   Para N==M o N<>M:

   (a) X < Y, no habrá match perfecto ya que algunas bandas podrían quedar sin participar en algún recital.

   (b) X > Y, no habrá match perfecto ya que algunos recitales podrían quedarse sin bandas.

   Independientemente del la cantidad de bandas y recitales y de las restricciones, la solución es estable ya que los pares banda-recital formados son inmejorables, incluso en los casos en los que quedan recitales o bandas libres.

4. El algoritmo actual no soporta que un organizador tenga la misma preferencia para bandas diferentes (o visceversa) dado que nos basamos en el orden de inserción en las distintas estructuras de datos para determinar el orden de preferencia. Si se quisiera soportar este comportamiento, deberíamos cambiar las estructuras que estamos utilizando actualmente.

   Para este tipo de problema donde existe la posibilidad de indiferencia o empate en el ranking de preferencias, quedan determinados distintos niveles de estabilidad, donde el emparejamiento puede ser *débilmente estable*, *fuertemente estable* o *super-estable*.

   - **Debilmente estable:** si no existe un par (b, r) no unido, tal que ambos se prefieran estrictamente más que su emparejamiento actual.
   - **Fuertemente estable:** si no existe un par (b, r) no unido, tal que uno de los dos prefiere estrictamente más al otro antes que a su emparejamiento actual, y el otro es al menos indiferente.
   - **Super-estable:** si no existe un par (b, r) no unido, que prefieren estar emparejados o a ambos les resulta indiferente.

   Suponiendo que el algoritmo soportara un mismo orden de preferencia para varias bandas/ organizadores, y que el mecanismo de desempate sea arrojar una moneda entonces no alcanzaría con preguntar si se prefiere más a uno u a otro (recital/banda) ya que también podría preferirse a ambos en un mismo grado, y en ese caso se arrojaría la moneda para resolver la preferencia. Es decir:

```
S = {}

Mientras haya recitales que todavía puedan contratar bandas y no se
haya comunicado con todas las bandas:

        Seleccionar recital r de los que cumplen la condición anterior

        Sea b la banda que mejor rankea en la lista de preferencias de r, y b
        no fue contactada previamente por r

        Si b participa en  k recitales, con 0<k < Y entonces:

                se agrega (b, r)  a S

                se aumenta la cantidad de bandas que contrató el recital r

                se aumenta la cantidad de recitales que aceptó la banda b

        Si b ya participa en  Y recitales, entonces:

                Sea ~r el recital menos preferido de b

                Por cada recital r' previamente aceptado por b

                        Si b prefiere a r y r' en un mismo grado

                                Arrojo moneda, determino a quién a quién prefiere
                                más y evalúo contra las condiciones (a) y (b)
```

Si **b** prefiere más al recital **r'** que al nuevo **r**, continúo (a)

Si **b** prefiere más el nuevo recital **r** que al existente **r'** (b)

Si **~r** no está seteado o si **b** prefiere más a **~r** que a **r'** -> **~r** = **r'**

Si existe **~r** entonces:

b rechaza a **~r** -> quita **(b,~r)** de **S**

~r se encola para ser procesada posteriormente
se disminuye la cantidad de bandas que tiene contratadas **~r**

b acepta a **r** -> se agrega **(b, r)** a **S**

se aumenta la cantidad de bandas que contrató el recital **r**

retorna **S**

5. **Comparación de ejecuciones**

Table 1: Comparación entre tiempos de ejecución

N	M	X	Y	Ejecución (ms)
10	5	1	1	89
10	5	2	1	113
10	5	2	2	116
10	10	1	1	138
10	10	2	1	145
10	10	2	2	153

Al ejecutar las pruebas se observa que el tiempo de ejecución empeora a medida que crece el tamaño de las entradas, y el tamaño de las restricciones impuestas a las bandas/recitales. Se observa que las mejores ejecuciones son las obtenidas para las restricciones triviales (X,Y = 1) que corresponderían a un Gale-Shapley tradicional, y luego para igual tamaño de entrada, los tiempos empeoran a medida que aumentan dichas restricciones.

6. **Comparación entre complejidad teórica y real**

De las ejecuciones realizadas se pudo observar que la cantidad de iteraciones reales realizadas por el algoritmo propuesto son mayores a la complejidad propuesta O(NxMxY). Creemos que esto se puede deber a que al momento de comenzar el ciclo que completa el cupo de bandas permitidas por recital, se valida que todavía queden cupos disponibles, pero no validamos que todavía haya bandas pendientes de ser contactadas. Si bien la validación se hace una vez dentro del ciclo, no alcanza para impedir que se realicen iteraciones innecesarias. Por lo anterior consideramos que la complejidad final de nuestra solución propuesta es de O(NxMxXxY).

2 Complejidad algorítmica

Eratóstenes de Cirene, matemático, astrónomo y geógrafo griego propuso un algoritmo para calcular los números primos menores a un valor “N”.

Iniciaba escribiendo todos los numeros de 1 a N. Luego en forma creciente partiendo desde el 2, iba tomando los números y tachando a sus múltiplos (menores o iguales a n). Al repetir el procedimiento el primer valor no tachado corresponde a un número primo. Se conoce a este procedimiento como “criba de Eratóstenes”.

1. Describa en pseudocódigo el algoritmo (procure realizar la solución más eficiente posible. Investigue!)
2. Analice y justifique su complejidad.
3. Describa en pseudocódigo el algoritmo de fuerza bruta para calcular los números primos y analice y justifique su complejidad.
4. Programe ambas soluciones (punto 1 y 3).
5. Grafique los tiempos de ejecución de ambos algoritmos para los siguientes valores de N: 100, 1.000, 10.000, 100.000, 1.000.000, 10.000.000
6. Analice los resultados obtenidos en base a la complejidad teórica de los mismos.

Información adicional:

- El algoritmo debe recibir por parámetro:
  - número “N”.
  - F (fuerza bruta) o E (Erastóstenes)
- Debe devolver:
  - Lista de los números primos desde 2 hasta “N”.
  - Tiempo total de ejecución

1. **Pseudocódigo solución eficiente**

   Sea N un natural > 1, X un arreglo de tamaño N con todos valores True.

```
para i = 2, 3, 4, ... no más de N^1/2:
  si X[i] es True:
    para j = i^2, i^2+i, i^2+2i,... no más de N:
      X[j] <- False
devolver lista de todos los i desde 2 hasta N tal X[i] es True
```

2. **Análisis de complejidad**

   Siendo que modificar X[j] es O(1), notamos que dado un i primo, la cantidad de valores que toma j está acotado por N/i, y el mismo se ejecuta sólo para los primos p desde 2 hasta $N^{1/2}$. Luego, se obtiene que el algoritmo es $O(\sum_{p \leq N^{1/2}} \frac{N}{p})$. Reescribiendo, se obtiene que es $O(N \sum_{p \leq N^{1/2}} \frac{1}{p})$. El segundo teorema de Mertens enuncia que $\lim_{n \to \infty}(\sum_{p \leq n} \frac{1}{p} - ln(ln(n)) - M) = 0$ con M una constante positiva. De aquí se deduce que $\sum_{p \leq n} \frac{1}{p} = O(log(log(n)))$, por lo que se obtiene que nuestro algoritmo es $O(Nlog(log(N)))$.

3. **Pseudocódigo solución fuerza bruta y análisis de complejidad**

   Sea N un natural > 1, L una lista vacía:

```
para i = 2, 3, 4, ... N:
  es_primo <- True
  para j = 2, ... no más de i^1/2:
    si i % j es 0:
      es_primo <- False
      break
  si es_primo es True:
    agregar i a L
devolver L
```

   Suponiendo que agregar un elemento a L y obtener el resto de una división son operaciones O(1), como cada loop de j se completa en O($N^{1/2}$) y se realizan N de los mismos, este algoritmo es O($N^{3/2}$).

4. **Gráfico tiempos de ejecución**

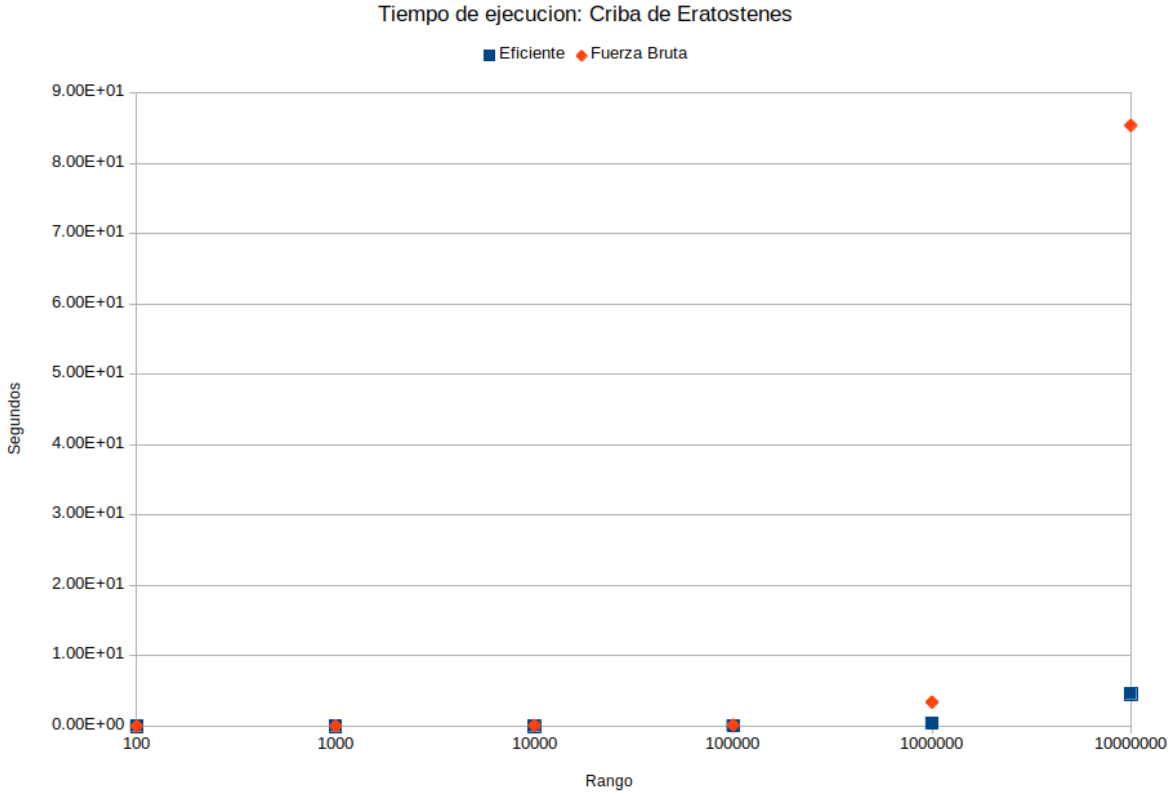

Table 2: Resultado de las ejecuciones

N	Eficiente	Fuerza Bruta
100	3.02E-05	7.58E-05
1000	0.0003	0.001
10000	0.003	0.092
100000	0.034	0.144
1000000	0.403	3.406
10000000	4.583	85.384

5. **Análisis de resultados**

Table 3: Comparación entre tiempos reales y teóricos

N	Efic. Real	Efic. Teorico	F.b. Real	F.b. Teorico
100	3.02E-05	3.013	7.58E-05	1000
1000	0.0003	15.087	0.001	3.16E+05
10000	0.003	60.205	0.092	1.00E+06
100000	0.034	221.033	0.144	3.16E+07
1000000	0.403	778.151	3.406	1.00E+09
10000000	4.583	2672.434	85.384	3.16E+10

El primer resultado que salta a la vista es la "poca" diferencia que hay entre las ejecuciones eficientes y de fuerza bruta del algoritmo para el caso de N=10.000.000. Por más que hay una diferencia de 80 segundos entre ambos no es tan grande como sugiere la complejidad teórica. El orden del tiempo teórico para la implementación por fuerza bruta es varias veces mayor al de la implementación eficiente (3e10 vs 2e3).

Esto se da también para los demás N. A pesar de que hay una clara diferencia real entre ambos, la misma no es tan marcada como puede sugerir la complejidad teórica. A fines prácticos, la diferencia es casi imperceptible para el orden de los cientos de miles (0.1 segundos) pero se vuelve realmente notoria en el orden de los millones (3 segundos de diferencia). Cuanto más se agrande el número mayor será la diferencia.

3 Requisitos y ejecución

Requisitos:

Python3

Ejecución:

-Gale-Shapley: Desde la terminal, estando en el directorio con el código fuente, ejecutar el comando "python3 gale_shapley.py N M X Y" donde N,M,X,Y son el número de bandas, el número de recitales, la cantidad de bandas por recital y la cantidad de recitales por banda respectívamente.

-Criba de Eratóstenes: Desde la terminal, estando en el directorio con el código fuente, ejecutar el comando "python3 eratostenes.py X N" donde X es el modo a usar (E/F) y N es el número máximo.