DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Guia 4

Algoritmos y Estructuras de Datos II

Integrante	LU	Correo electrónico
Castro Russo, Matias Nahuel	203/19	castronahuel14@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Ordenamiento

Mi algoritmo en palabras:

- Voy a usar una especie de bucket sort, donde voy a crear K buckets de listas vacías. Asigno los conjuntos a los buckets tomando como criterio el tamaño de cada conjunto. Un conjunto de tamaño i, será agregado a la lista del bucket en la posición i.
  Luego de asignar todos los conjuntos a su bucket correspondiente, ya voy a tenerlos ordenados, pues mis buckets lo están entre sí. Después paso todos los conjuntos del bucket ,de manera ordenada, al arreglo que tenía originalmente.
  Finalmente ya quedó ordenado.

- ¿Por qué ordeno así?

  - Consigna resumida: Si un conjunto está incluido en otro, debe aparecer antes en el arreglo ordenado.
  - Sean a, b conjuntos, decir que $a \subseteq b$, $\longrightarrow$ que la cardinalidad de a sea menor o igual al de b.
    Por lo tanto, ordenando los conjuntos por cardinalidad, ya basta para garantizar lo pedido.
  - Mi criterio será:
    - Sea A un arreglo de de N conjuntos $A[1], ..., A[N]$
      $\#A[i] \leq \#A[j] \longrightarrow i \leq j \ \forall \ i, j \in [1, N)$

- Pasos importantes del algoritmo y sus complejidades:

  - Busco el K (cardinal del conjunto con mayor cantidad de elementos) $\triangleright \ \mathcal{O}(N \cdot K)$
  - Crear un arreglo "buckets" con k buckets de listas vacías $\triangleright \ \mathcal{O}(K)$
  - Recorrer el arreglo A original e ir agregando los conjuntos a la lista del bucket correspondiente $\triangleright \ \mathcal{O}(N \cdot K)$
  - Recorrer los buckets, ya ordenados, ir sacando los conjuntos del bucket, e ir poniendolos en el arreglo original $\triangleright \ \mathcal{O}(N \cdot K)$
  - Complejidad total de todo el algoritmo de ordenamiento : $\mathcal{O}(N \cdot K)$

Aclaración:

- Voy a decir conjN como abreviatura de Conjunto de Naturales

---

```
1   void ordenamiento (inout: arreglo<ConjN> A):      # O(N*K)
2       k       <-- 0                                 # O(1)
3       itK     <-- crearIt(A)                        # O(1)
4
5       #busco el K
6       for c in A:                                   #O(N*K)
7           kAux  <-- 0
8           while haySiguiente(itM):                  # O(K)
9               k++                                   # O(1)
10              avanzar(itM)                          # O(1)
11          end while
12          M  <-- max(k,kAux)                        # O(1)
13      end for
14
15      #Creo un Arreglo con K posiciones de listas vacias
16      buckets     <-- crearArregloDeListaVacias(k)              # O(k)
17
18      #recorro el arreglo original y voy agregando al bucket que corresponda
19      #mando al bucket i, aquellos conjuntos de cardinal = i
20      for i = 0 to tam(A) - 1 :                         # O(N*K)
```

```
21              tamConjunto <-- 0                           # O(1)
22              itC         <-- crearIt(A[i])               # O(1)
23              while haySiguiente(itC):                    # O(K)
24                  tamConjunto++                           # O(1)
25                  avanzar(itC)                            # O(1)
26              end while
27              #agrego el conjunto a la lista del bucket correspondiente
28              agregarAtras(buckets[tamConjunto],A[i])     #O(k)
29          end for
30          #Ahora ya tengo cada conjunto en su bucket
31          iA      <-- 0                                   # O(1)
32          for i = 0 to tam(buckets) -1:                   # O(k + N*K), VER (ACLARACION *1)
33              while longitud(buckets[i]) != 0:
34                  A[iA]  <-- buckets[i][0]
35                  fin(buckets[i])                         # O(1)
36                  iA ++
37              end while
38          end for
39      end
40
41      #Complejidad total = O(3 N*K + K) = O(4 N*K) = O(N*K)
```

Aclaración *1:

- Desde la línea 31 , hasta la 38, estoy en la parte de mi algoritmo donde tengo mis buckets ordenados. Por lo tanto solo me resta ir recorriendo mis buckets y "volcar"los conjuntos que ya tengo ordenados, en el arreglo A original.
- Se que en mi arreglo "buckets", tengo K buckets, N conjuntos con un máximo de K elementos en cada conjunto. Mi algoritmo para pasar los conjuntos ya ordenados es el siguiente:
  - Recorro los k buckets, en aquellos que la lista no sea vacía, voy a recorrerla e ir pisando los conjuntos del arreglo A con una copia de dicho conjunto.
    Una vez copiado, borro ese conjunto de mi lista y en caso de seguir teniendo más conjuntos en el mismo bucket, voy poniéndolos de igual manera en A (como mi bucket está en orden, voy a indexar en A de izquierda a derecha). El copiar un conjunto tiene complejidad O(largo del conjunto) $\in$ O(K)
- Voy a recorrer en total, N elementos que me sale O(K) cada copia $\longrightarrow \mathcal{O}(N \cdot K) + \mathcal{O}(K)$. El "$+k$" viene de que al recorrer los buckets, puede pasar que tengo una porción de K buckets con listas vacías, por lo tanto no hay nada que recorrer, ni copiar, en dicho bucket.
  - Por ejemplo: En el caso donde de mis N conjuntos, todos tienen exactamente K elementos, en mi for de la línea 32, voy a hacer K iteraciones en las cuales hago algo O(1), y solo en la última iteración voy a tener los N elementos. $\longrightarrow$ O((K-1) $\cdot 1 + 1 \cdot (N \cdot k)$)
    Este es solo un ejemplo, pero en conclusión, siempre voy a hacer exactamente N veces esa copia de conjuntos, en todo el for que recorro buckets.
- Por lo tanto: la complejidad de esta aclaración, es $\mathcal{O}(k + N \cdot K)$

---

2. Divide & Conquer

Mi algoritmo en palabras:

- Consigna resumida: Determinar di todos los conjuntos son disjuntos dos a dos, es decir si $\forall \ \ i \neq j$ en el rango $1..N$ se tiene que $C[i] \bigcap C[j] = \emptyset$
- Básicamente: Divido el arreglo en dos, hago recursión sobre ambas partes, y luego combino la información que tengo de ambas (la mitad izquierda y derecha)
- Caso Base:
  - Este sucede al llamar a la función con un solo conjunto. Al tengo un solo conjunto, trivialmente cumple que sea disjunto dos a dos.
    Además voy a crear un arreglo de N posiciones, en el cual voy a meter todos los elementos del conjunto. Como el iterador del Conjunto de Naturales recorre todos los elementos de manera ordenada, al insertarlos en el arreglo, lo hago de manera ordenada.
    Finalmente devuelvo una tupla $< True, arreglo < Nat >>$
- Casos generales:
  - Divido por la mitad, llamo recursivamente a su mitad izquierda y derecha.
  - Me guardo la tupla que me devuelve cada mitad, izq y der respectivamente.
    - Cada tupla va a tener en π_1 un bool informando si esa tupla cumple con la disjuntividad dos a dos.
    - π_2 tendrá un arreglo con todos los elementos de los conjuntos de la mitad correspondiente (izq o der). π_2 esta ordenado.
  - Finalmente llega la parte de combinar esas dos mitades y analizar disjuntividad entre ambas, teniendo en cuenta por supuesto, los datos que tengo provenientes de los llamados recursivos:
    - En caso de de que una mitad no sea disjunta, es decir $\pi_1 = False$, entonces ya no vale la pena seguir analizando la disjuntividad. Por lo tanto devuelvo $< False, arregloVacio >$.
    - Si las dos mitades cumplen lo pedido, entonces creo un nuevo arreglo que contenga a los elementos de ambas. Es decir, mergeo de manera ordenada los arreglos de las mitades, y por ultimo analizo si cumplen lo pedido o no.
      Devuelvo < Si cumple o no , arreglo Mergeado >.
- Mi algoritmo para ver si dos conjuntos cumplen dicha disjuntividad, es meter ambos conjuntos en un arreglo de manera ordenada. Con complejidad lineal, mirar los pares de dos elementos contiguos, en caso de encuentrar un par que ambos elementos sean iguales, quiere decir que sus conjuntos no son disjuntos. En caso contrario, son disjuntos entre si.

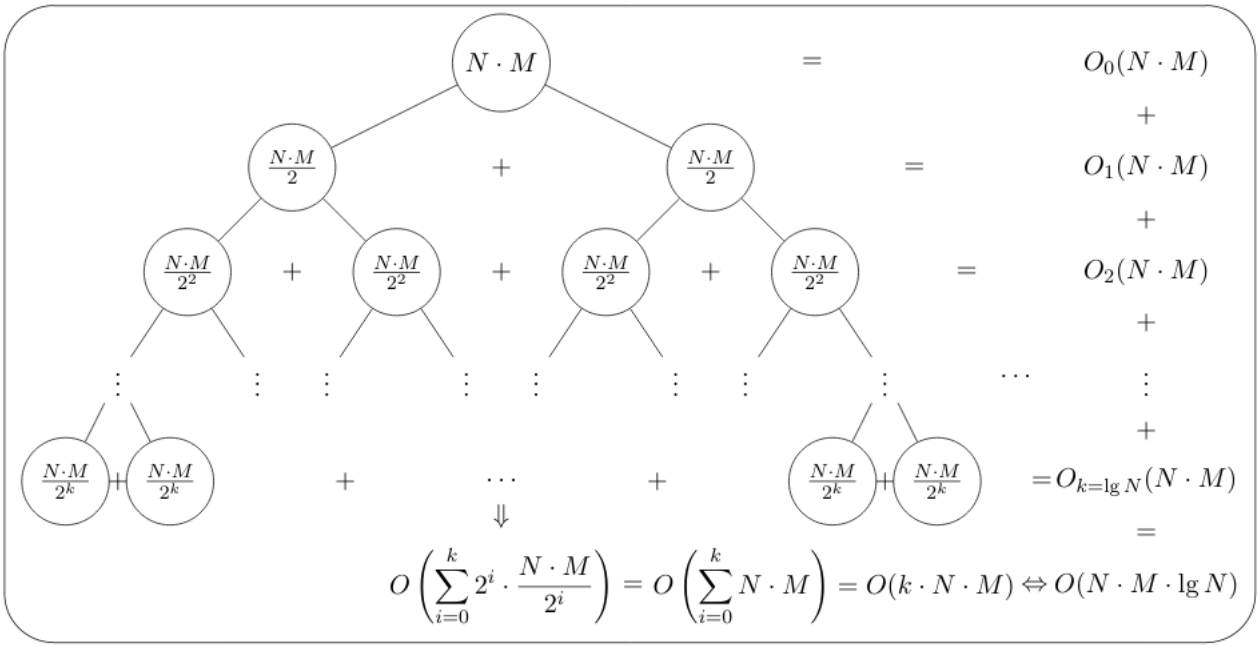
```
1   bool disjuntos(in: arreglo<ConjN> A):
2       return P1_1( todosDisjuntos(A,0,tam(A)) )  # T(N) = O(M*N log N)
3   end
4
5   #todos Distintos tiene complejidad O(M*N log N)
6   <bool,arreglo<Nat>> todosDisjuntos(in: arreglo<ConjN> A, in: Nat desde,
7   in: Nat hasta):
8
9       #Caso Base
10      if desde == hasta:                          # O(1)
11          #calculo cual es la longitud M
12          M       <-- 0                           # O(1)
13          itM     <-- crearIt(A[desde])           # O(1)
14          while haySiguiente(itM):                # O(M)
15              M++                                 # O(1)
```

```
16                  avanzar(itM)                                # O(1)
17              end while
18
19              arrCopia <-- crearArreglo(M)                    # O(M)
20              it       <-- crearIt(A[desde])                  # O(1)
21              i        <-- 0                                  # O(1)
22
23              #copio el conjunto de manera ordenada en el nuevo arreglo
24              while haySiguiente(it):                         # O(M)
25                  arrCopia[i]    <-- siguiente[it]    # O(1)
26                  i++                                         # O(1)
27                  avanzar(it)                                 # O(1)
28              end while
29              return <True, arrCopia>                         # O(M)
30          end if
31
32          #divide
33          mitad    <-- (desde + hasta) /2                     # O(1)
34
35          #conquer
36          izq    <-- todosDisjuntos(A,0,mitad)                # T(n/2)
37          der    <-- todosDisjuntos(A,mitad,hasta)            # T(n/2)
38
39          #combine
40
41          # si la izq o der es False, entonce no tiene sentido seguir
42          # calculando, por transitividad, la funcion disjuntos ser´a false
43          sinRepetidos <-- Pi_1(izq) && Pi_1(der)
44          if (!sinRepetidos):                                                 # O(1)
45              arregloVacio    <-- crearArregloVacio()                         # O(1)
46              return (<False,arregloVacio>)                                   # O(1)
47          end if
48
49          #me guardo el largo de los arreglos de izq + der
50          tamArrays    <-- tam(Pi_2(izq)) + tam(Pi_2(izq))                    # O(1)
51          mergeado      <-- crearArreglo(tamArrays)                           # O(N*M)
52
53          #ambos arreglos estan ordenados, entonces mergeo de manera ordenada
54          # mrge toma los parametros por referencia
55          merge(Pi_2(izq), Pi_2(der), mergeado)                               # O(N*M)
56
57          # veo si hay o no repetidos
58          for i = 1 to  tam(mergeado) - 1:                                    # O(N*M)
59              if mergeado[i-1] == mergeado[i]:                                # O(1)
60                  sinRepetidos  <-- False                                     # O(1)
61              end if
62          end for
63
64          return(<sinRepetidos,mergeado>)                                     # O(M*N)
65      end
```

```
66
67
68
69          # los tres parametros los tomaria por referencia
70          #  Complejidad de funcion merge O(N*M)
71          void merge(inout: arreglo<ConjN> A, inout: arreglo<ConjN> B,
72          inout: arreglo<ConjN> mergeado):
73              i <-- 0                                  # O(1)
74              j <-- 0                                  # O(1)
75              for m = 0 to tam(mergeado) - 1:          # O(N*M)
76                  if( j >= tam(B) || ( i < tam(A) && A[i] < B[j] )): # O(1)
77                      mergeado[m] = A[i];              # O(1)
78                      i++;                             # O(1)
79                  else:
80                      mergeado[m] = B[j];              # O(1)
81                      j++;                             # O(1)
82                  end if                               # O(1)
83              end for                                  # O(1)
84          end
```

Complejidad:

$$T(N) = \begin{cases} \mathcal{O}(M) & N = 1 \\ 2.T(\frac{N}{2}) + \mathcal{O}(N \cdot M) & N > 1 \end{cases}$$


$N \cdot M$
$= O_0(N \cdot M)$
$\frac{N \cdot M}{2}$
$+$
$\frac{N \cdot M}{2}$
$= O_1(N \cdot M)$
$\frac{N \cdot M}{2^2}$
$\frac{N \cdot M}{2^2}$
$\frac{N \cdot M}{2^2}$
$\frac{N \cdot M}{2^2}$
$= O_2(N \cdot M)$
$\frac{N \cdot M}{2^k}$
$\frac{N \cdot M}{2^k}$
$\frac{N \cdot M}{2^k}$
$\frac{N \cdot M}{2^k}$
$= O_{k=\lg N}(N \cdot M)$
$\Downarrow$
$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{N \cdot M}{2^i}\right) = O\left(\sum_{i=0}^{k} N \cdot M\right) = O(k \cdot N \cdot M) \Leftrightarrow O(N \cdot M \cdot \lg N)$