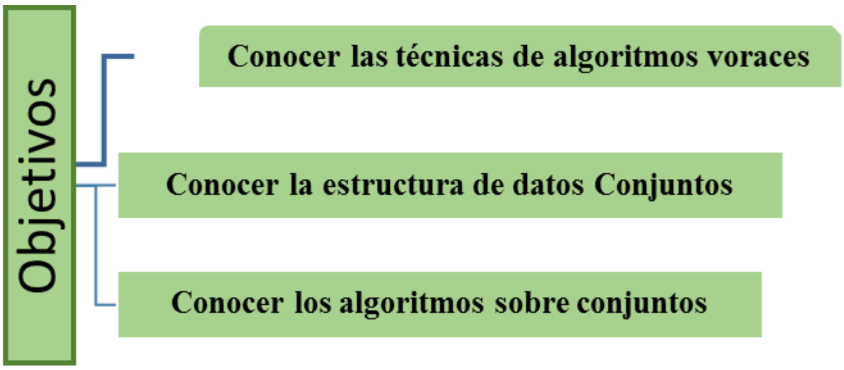
GUIA 5 Algoritmos voraces y Conjuntos

Valor.. a pesar de todas las debilidades del cuerpo, el espíritu debe triunfar

Beethoven

Avram Noam Chomsky (nacido el 7 de diciembre de 1928 en Filadelfia, EEUU) es profesor emérito de Lingüística en el MIT y una de las figuras más destacadas de la lingüista del siglo XX. Creó la gramática generativa, disciplina que situó la sintaxis en el centro de la investigación lingüística y con la que cambió por completo la perspectiva y los programas y métodos de investigación en el estudio del lenguaje, actividad que elevó definitivamente a la categoría de ciencia moderna. Desde el año 1959 en que Chomsky dio su clasificación de las gramáticas, se han publicado muchos trabajos sobre lenguajes y gramáticas en el aspecto formal. La perspectiva lógico-matemática del distribucionalismo harrisiano, es reconocida y adoptada por Chomsky, como explícitamente se manifiesta en su libro The logical structure of linguistic theory, redactado entre 1951 y 1955, pero sin publicarse íntegramente.
Chomsky es quien introduce estructuras sintácticas en el estudio de la complejidad del sistema lingüístico, Chomsky considera adecuada la d términos de niveles de representación de las oraciones. Adicionalmente, l frente a los estudiosos anteriores es la capacidad de conjugar orgánic tradicional con saberes de naturaleza lógico-matemática para tratar de manera clara, formal y explicita los procesos recursivos del lenguaje humano. Como lingüista Chomsky ha sacudido de manera despiadada al estructuralismo lingüístico en sus manifestaciones estadounidenses y europeas, lo que suscito grandes discusiones y controversias en torno a su modelo generativo transformacional. Pero sus reflexiones no son solo de naturaleza lingüística, sino que tiene perfiles filosóficos, pues despierta controversias entre el empirismo y el racionalismo.



Introducción

Los algoritmos voraces suelen ser sencillos. Se emplean sobre todo para resolver problemas de optimización, como por ejemplo, encontrar la secuencia óptima para procesar un conjunto de tareas por un computador, hallar el camino mínimo de un grafo, etc. Habitualmente, los elementos que intervienen son:

- un conjunto o lista de ***candidatos*** (tareas a procesar, vértices del grafo, etc);

- un conjunto de ***decisiones*** ya tomadas (candidatos ya escogidos);
- una ***función*** que determina si un conjunto de candidatos es una ***solución*** al problema (aunque no tiene por qué ser la óptima);
- una ***función*** que determina si un conjunto es ***completable***, es decir, si añadiendo a este conjunto nuevos candidatos es posible alcanzar una solución al problema, suponiendo que esta exista;
- una ***función*** de selección que escoge el candidato aún no seleccionado que es más ***prometedor***;
- una ***función objetivo*** que da el valor/coste de una solución (tiempo total del proceso, la

- longitud del camino, etc) y que es la que se pretende maximizar o minimizar; Para

  resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos. Inicialmente el conjunto de candidatos es vacío. A continuación, en cada paso, se intenta añadir al conjunto el mejor candidato de los aún no escogidos, utilizando la función de selección. Si el conjunto resultante no es completable, se rechaza el candidato y no se le vuelve a considerar en el futuro. En caso contrario, se incorpora al conjunto de candidatos escogidos y permanece siempre en él. Tras cada incorporación se comprueba si el conjunto resultante es una solución del problema. Un algoritmo voraz es correcto si la solución así encontrada es siempre óptima. El esquema genérico del algoritmo voraz es:

El nombre voraz proviene de que, en cada paso, el algoritmo escoge el mejor "pedazo" que es capaz de "comer" sin preocuparse del futuro. Nunca deshace una decisión ya tomada

Ejemplo 1: Se quiere cambiar una cantidad N haciendo uso del menor número de monedas

```
Función DevolverCambio(N) : número de monedas.
Inicio
        C        = { 100, 25, 10, 5, 1}
        S        = { },   SUM     = 0
        Mientras  ( SUM < N )
                X =  mayor elemento de C  tal que
                        SUM + X <= N
                Si no existe  ese elemento
                        Retornar “No existe solucion”
                Sino
                        S = S U{ una moneda de valor X}
                        SUM = SUM + X
                FinSi
        FinMientras

        Devolver S
Fin
```

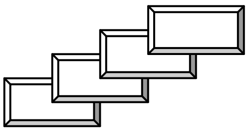
N : cantidad a cambiar
S : conjunto de monedas
SUM: cantidad acumulada

Supongamos que se disponen monedas de 1, 4 y 6 unidades cambiar 8 implicaría 1x 6x 2x 1 = 8 Sin embargo sabemos que la mejor solución es 2 x 4

Ejemplo 2:

Problema de la mochila

Sea N objetos y una mochila M



Oi i= 1..n

Oi tiene un peso W_i , un valor V_i

M puede llevar un peso que no sobre pase ***W***

```
funcion voraz(C:conjunto):conjunto
        { C es el conjunto de todos los candidatos }
        S <= vacio    { S es el conjunto en el que se construye la solucion}
        mientras  ¬solucion(S) y C <> vacio hacer
                x <= el elemento de C que maximiza seleccionar(x)
                C <= C \ {x}
                si completable(S U {x}) entonces S <= S U {x}
                si solucion(S)
                        entonces devolver S
                si no devolver no hay solucion
        Fin Mientras
Fin
```

Por cada objeto Oi se puede llevar una fracción Xi de el
Se quiere llenar la mochila de tal forma que se maximice el valor de los objetos
Formalmente

$$\text{Max} \quad \{ \quad \sum_{i=1} X_i V_i \qquad \text{tq.} \quad \sum_{i=1} X_i W_i \leq \boldsymbol{W}$$

Precondicion. $V_i > 0$ $W_i > 0$ $0 \leq X_i \leq 1$, $\forall\ 1 \leq i \leq n$

Algoritmos voraz
Utilizaremos un vector X: $(X_1, X_2, \ldots\ldots X_n)$ en donde guardaremos la fraccion X_i del objeto Oi hay que incluir.

N= 5 $\boldsymbol{W} = 100$

	1	2	3	4	5
W	10	20	30	40	50
V	20	30	66	40	60
V/W	2.0	1.5	2.2	1.0	1.2

```
Función mochila (W[1...n], V[1....n], W ) : matriz [1....n]
Inicio
            Para i de 1 a n
                  X[i] =0
            FinPara
            Peso = 0
            Mientras (Peso < W)
                  i = elegir el mejor objeto restante
                  Si ( Peso  + W [i] ≤ W
                              X[i] = 1
                              Peso = Peso + W [i]
                  Sino
                        X[i] = (W - Peso) / W [i]
                        Peso = W
                  FinSi.
            FinMientras
            Retornar X
Fin
```

Para elegir el mejor objeto restante, se pueden utilizar tres criterios descritos mediante las siguientes funciones

Función de selección **$\varnothing_1, \varnothing_2, \varnothing_3$**

∅1 Seleccionar el objeto mas valioso
incrementa el valor de la carga del modo más rápido posible.

∅2 Seleccionar el objeto más pequeño restante. La capacidad se agota de la forma más lenta posible.

∅3 Seleccionar el objeto cuyo valor porcentual de peso sea el mayor posible.

Si seleccionamos los objetos por orden decreciente de valores seleccionamos.

Objeto	0_3 ,	0_5	, $\frac{1}{2}$	0_4	total
Valor	66	60	40/2		146
Peso	30	50	40/2		100

Si seleccionamos los objetos en orden de peso creciente

Objeto	0_1 ,	0_2 ,	0_3 ,	0_4	total
Valor	20	30	66	40	156
Peso	10	20	30	40	100

Si seleccionamos los objetos por orden decreciente de V/W

Objeto	0_3 ,	0_1 ,	0_2 ,	0_5	total
Valor	20	30	66	4/5 (60)	164
Peso	10	20	30	4/5 (50)	100

Conjuntos

Los conjuntos son una de las estructuras básicas de las matemáticas, y por tanto de la informática. No se va a entrar en la definición de conjuntos ni en sus propiedades. Se supondrá que el lector conoce algo de teoría de conjuntos. Con lo más básico es suficiente.

En realidad las estructuras de datos que se han implementado hasta ahora no son más que elementos diferentes entre sí (en general) en los que se ha definido una relación. Por ejemplo, en las listas ordenadas o los árboles binarios de búsqueda se tiene una serie de elementos que están ordenados entre sí. Obviando las propiedades de las estructuras, se ve que forman un conjunto, y su cardinal es el número de elementos que contenga la estructura. En los conjuntos no existen elementos repetidos, y esto se respeta en las implementaciones que se ofrecen a continuación.

Definiremos unas implementaciones que permitan aplicar el álgebra de conjuntos, ya sea unión, intersección, pertenencia, etc.

Definición

Un conjunto es una estructura básica fundamental de las matemáticas. Se utiliza como base de una gran cantidad de TDA. Se han desarrollado muchas técnicas para la implantación de TDA basada en conjuntos.

Características

Sea C un conjunto

- Los elementos de C no necesariamente están ordenados
- C no tiene elementos repetidos

Representación de listas enlazadas

Mediante Arreglos

Mediante apuntadores

OPERACIONES

Unión(A, B) toma los argumentos A y B cuyos valores son conjuntos y asigna el resultado AUB a la variable C que es también un conjunto

Intersección(A, B) toma los argumentos A y B cuyos valores son conjuntos y asigna el resultado A∩B a la variable C que es también un conjunto

Diferencia(A, B) toma los argumentos A y B cuyos valores son conjuntos y asigna el resultado A- B a la variable C que es también un conjunto

Combinacion(A, B) toma los argumentos A y B cuyos valores son conjuntos disjuntos y asigna el resultado AUB a la variable C que es también un conjunto (parecido a la unión, la diferencia es que se requiere que los conjuntos A y B sean disjuntos)

Asigna(A, B) hace que el valor de la variable A de conjuntos, sea igual al valor de la variable B también de conjuntos

Miembro(x, A) toma el conjunto A y x del tipo de los elementos de A y devuelve un valor booleano: verdad si x esta en A, y falso si no lo esta

Mínimo(A) toma como argumento A y devuelve su menor elemento

Máximo(A) toma como argumento A y devuelve su mayor elemento

Igual(A, B) toma como argumentos los conjuntos A y B y devuelve un valor booleano: si A y B tienen los mismos elementos, y falso en otro caso

La implementación es directa, si todos los bits de A y B se corresponden entonces son iguales:

```
int iguales(tconjunto A, tconjunto B)
{ return (A == B); }
```

Inserta(x, A) toma el conjunto A y x del tipo de los elementos de A y lo hace elementos de A. Es decir A = A U {x }

Subconjuntos:

Si un conjunto A es subconjunto (considerando que un conjunto cualquiera es subconjunto de si mismo) de otro B entonces verifica esta relación: A intersección B = A. Notar que A es subconjunto de A, pues A intersección A = A.

Ejemplo:
A = {1,2,3,4}, B = {0,1,2,3}
C = A intersección B = {1,2,3}; C es distinto de A.

Representación de conjuntos mediante arreglos de bits

Un bit solo permite representar dos estados diferentes. Por supuesto, pueden representar atributos muy variados, por ejemplo, ser hombre o mujer, adulto o niño, Windows o Linux, etc. También sirve para indicar si un elemento está o no dentro de un conjunto.

El array se utiliza para representar un conjunto de números naturales (u otro tipo de datos cuyos elementos se identifiquen por un número natural único mediante una correspondencia) entre 0 y N, siendo N la capacidad del array unidimensional (es decir, un vector); almacenará valores booleanos, esto es, 1 ó 0.

Por ejemplo, suponer el conjunto universal formado por los enteros entre 0 y 4: U = {0, 1, 2, 3, 4}, y el conjunto C = {1, 2}. Se representará de esta manera:

0	**1**	**2**	**3**	**4**
0	1	1	0	0

1 : indica que el elemento pertenece al conjunto.
0 : indica que el elemento no pertenece al conjunto.

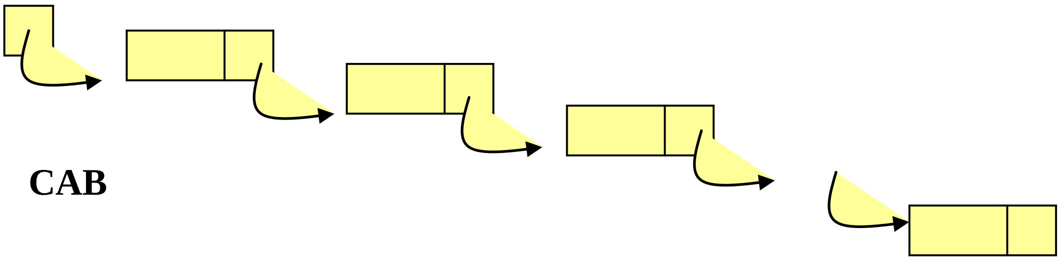
Ahora bien, se ha dicho que se va a emplear un array de bits. ¿Qué se quiere decir con esto? Que no se va a emplear un array o vector como tal, sino un tipo de datos definido por el lenguaje de programación, que suele ocupar entre 8 y 64 bits, y por tanto podrá incluir hasta 64 elementos en el conjunto. Por ejemplo, en C o Pascal se define un tipo que ocupa 8 bits:

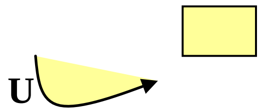
Representación de conjuntos mediante vectores

Un conjunto puede representarse como una secuencia de elementos en un vector, en donde cada casillero contiene un elemento del conjunto. Los elementos pueden disponerse en el vector de izquierda a derecha o viceversa en el orden que llegan.

Representación de conjuntos mediante listas enlazadas

Un conjunto puede representarse como una lista enlazada, en donde cada nodo contiene unos elementos del conjunto
Para las demás adiciones, CAB apunta al primer elemento y U apunta al último elemento. U es utilizado para que cuando se adicione nuevos elementos sean enlazados al último nodo U.





Acción Vacío (cab)
Inicio
    Si cab = NULL
        Retornar verdad
    Sino
        Retornar falso
    Fin-si
Fin

sta esta vacía o no. La lista este caso retorna el valor verdad, en otro caso retorna el valor falso

Acción Adiciona (cab, Val ,u)
  Inicio
    n ← nuevo nodo
    n-> valor   ← Val
    n-> sig   ← Null
    Si (Vacío(cab))
        cab ← n
            Sino
        u->sig = n
    Fin_Si
    u=n
Fin

Combinacion(A, B) toma los argumentos A y B cuyos valores son conjuntos y asigna el resultado AUB a la variable C que es también un conjunto

Sea C1 y C2 conjunto representados mediante listas con cabecera Cab1, Ult1 y Cab2, Ult2 que apuntan al primer y ultimo nodo de cada lista respectivamente. Inicialmente Cab3 y Ult3 son nulos

```
// Recorre la lista 1 y copia sus elementos en la lista 3, luego
hace lo mismo con la lista 2

Acción Combinacion (Cab1, Cab2, Cab3 )
Inicio
   p=cab1
   Si Vacío(cab1 )
        Cab3 = Cab2
   Sino
        Mientras (p ≠ Null )
                Adiciona(Cab3, p->val , Ult3)
                p ← p->sig
        FinMientras
        p=cab2
        Mientras (p ≠ Null )
                Adiciona(Cab3, p->val , Ult3)
                p ← p->sig
        FinMientras
   FinSi
Fin
```

Intersección(A, B) toma los argumentos A y B cuyos valores son conjuntos y asigna el resultado A∩B a la variable C que es también un conjunto

Sea C1 y C2 conjunto representados mediante listas con cabecera Cab1, Ult1 y Cab2, Ult2 que apuntan al primer y ultimo nodo de cada lista respectivamente. Se crea una lista L3 con Cab3 y Ult3 inicialmente son nulos

Se presentan aquí dos casos

- Si L1 y L2 estan ordenados, se avanza en ambas listas simultáneamente, copiando los que están en ambas listas en la lista 3, esto se repite hasta que se llegue al final de una de las listas. Lo que queda de la otra lista, se desecha. Esto nos da un algoritmo lineal O(N)

- Si L1 y L2 no están estan ordenados, se avanza en la primera lista. Por cada nodo de la lista 1 se recorere toda la lista 2. Esto nos da un algoritmo lineal $O(N^2)$

Caso 1

// Recorre la lista 1 y la lista 2 simultaneamente, compara uno a uno sus elementos copiando los que están en ambas listas en la lista L3 inicialmente vacia

```
Acción Interseccion (Cab1, Cab2, Cab3 )
Inicio
    p1=cab1
    p2=cab2
    Mientras (p1  ≠ Null y p2  ≠ Null  )
          Si(p1->val = p1->val)
                    Adiciona(Cab3, p1->val , Ult3)
          Sino
                    Si(p1->val < p2->val)
                              p1 ← p1->sig
                    Sino
                              p2 ← p2->sig
                    FinSi
          FinSi
    FinMientras
Fin
```

Caso 2

- // Si L1 y L2 no están estan ordenados, se avanza en la primera lista. Por cada nodo de la lista 1 se recorre toda la lista 2.

```
Acción Interseccion (Cab1, Cab2, Cab3 )
Inicio
    p1=cab1
    encontró = falso
    Mientras (p1  ≠ Null   )
          P2=cab2
          Mientras (p2  ≠ Null y no encontro   )
                    Si(p1->val = p2->val)
                              Adiciona(Cab3, p1->val , Ult3)
                              Encontro = verdad
                    Sino
                              p2 ← p2->sig
                    FinSi
          FinMientras
          p1 ← p1->sig
    FinMientras
Fin
```

Laboratorio

Implementar una aplicación para

a) Crear dos con juntos A y B
b) Adicionar un elemento a un conjunto
c) Intersectar dos conjuntos
d) Unir dos conjuntos
e) Hallar la diferencia de dos con juntos
f) Salvar los conjuntos
g) Recuperar los conjuntos

Ejercicios propuestos

1 Una aplicación de los algoritmos voraces lo constituye el algoritmo de Kruskal y algoritmo de Prim. Investigue que trata cada uno de los algoritmos. De una especificación e implementación da cada uno. Proporcione un ejemplo de su funcionalidad.

2 Considere una competición mundial en la cual hay dos equipos A y B, que juegan un máximo de 2n -1 partidas, y en donde el ganador es el primer equipo que consiga n victorias. Suponemos que no hay posibilidad de empate y que los resultados de todos los partidos son independientes y que para cualquier partida dada hay una probabilidad constante **P** de que A gane, y por tanto una probabilidad constante **Q** = 1-**P** de que gane B. Sea P(i,j) la probabilidad de que el equipo A gane el campeonato, cuando todavía le falta i victorias mas para conseguirlo, mientras que el equipo B necesita j victorias para ganar. Por ejemplo, antes del primer partido del campeonato, la probabilidad de que gane el equipo A es P(n , n): ambos equipos necesitan todavía n victorias para ganar el campeonato. Si A necesita cero victorias mas, entonces lo cierto es que ya ha ganado el campeonato y por tanto P(0, i) = 1 con 1 <= i <=n. De manera similar, si el equipo B necesita cero victorias mas, entonces ya ha ganado el campeonato y por tanto P(i,0) = 0, con 1 <= i <=n .

Como no se puede dar una situación en la que A y B ganen todas las partidas que necesitan P(0,0) = 0. Carece de

significado dado que A gana cualquier partida con una probabilidad P y pierde con una probabilidad Q

P(i,j) = P **P(i-1,j) + Q P(i,j-1) para i >=1 y j >=1**

Construya un algoritmo voraz para hallar P(n,n)

3 Construya los algoritmos para manejar TADs conjuntos, con representación de arreglos :
- **a) Miembro(A,a)**
- **b) Union(A,B)**
- **c) Interseccion(A,B)**
- **d) Diferencia(A,B)**
- **e Complemento(A,B)**

4 Construya los algoritmos para manejar TADs conjuntos, con representación de listas :
- **a) Miembro(A,a)**
- **b) Union (A, B)**
- **c) Interseccion(A,B)**
- **d) Diferencia(A,B)**
- **e Complemento(A,B)**

5 Una aplicación de algoritmos voraces lo constituye el codigo de Huffman. Investigue de que se trata.

6 Implemente el problema de la Mochila

REFERENCIAS BIBLIOGRAFICAS

1) [AHO 1988], Alfred V.
“Estructura de Datos y algoritmos” Addison Wesley. 1988.
2) [ALMEIDA 2008], Francisco Almeida “Introducción a la programación paralela” Paraninfo 2008 España ISBN 978-84-9732-674-2.

3) [ANDERSON 2008] Anderson James “Redes Neuronales” Edit AlfaOmega México ISBN 978-970-15-1265-4

4) [BRASSARD 2001], G. / BRATLEY, T.
“Fundamentos de Algoritmia”. Prentice Hall. 2001

5) [CORTEZ 2010] Cortez Vásquez Augusto. “Algoritmia”. Edit EsVega Lima Perú ISBN 978-612-00-0257-5

6) [CORTEZ 2012] Cortez Vásquez Augusto. “Algoritmia, Técnicas algorítmicas” Edit San Marcos Lima Peru ISBN 978-612-00-0964-2

7) [LEE 2005] R.Lee. “Introducción al diseño y análisis de algoritmos” Edit Mc Graw Hill Mexico ISBN 978-970-10-6124-4

8) [LEIJA 2009] Lorenzo Leija “Metodos de procesamiento avanzado” Edit Reverte Mexico 209 ISBN 978-607-7815-01-3

9) [WEISS 2001] ALLEN WEISS, Mark “Lenguaje de programación Java”. Addison Wesley. Madrid 2001.

10) [WEISS 2003] ALEN WEISS, Mark “Estructura de Datos en Java”. Addison Wesley. 2003