

Algoritmos ávidos

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

06-2020

Contenido

Introducción

Ejemplos clásicos

Otros ejemplos

Estructuras ávidas

Definición

Se conocen también como *algoritmos miopes*, *golosos*, *ávidos* o *avaros*, y caracterizan por decisiones basados en la información que tienen a primera mano, sin tener en cuenta lo que pueda pasar más adelante. Además, una vez que toman una decisión nunca reconsideran otras posibilidades, lo que ocasionalmente los lleva a caer en puntos muertos o sin salida.

Los algoritmos ávidos también se caracterizan por la rapidez con la que encuentran una solución (cuando la encuentran), que casi nunca es la mejor. Normalmente son utilizados para resolver problemas en los cuales la velocidad de respuesta debe ser muy alta o el espacio de búsqueda es muy grande.

Ejemplos típicos de problemas que se pueden resolver mediante este paradigma están las búsquedas en árboles o grafos, solución de laberintos y algunos juegos entre otros. También muchos problemas que requieren obtener máximos o mínimos.

Forma general

La estrategia general de este tipo de algoritmos se basa en la construcción de una solución que comienza sin elementos, y cada vez que debe tomar algún tipo de decisión lo hace con la información que tiene en ese momento, para, de alguna manera, agregar elementos y así avanzar hacia la solución final. Cada elemento se agrega al conjunto solución, y así hasta llegar a la solución completa o a un punto en el cual el algoritmo no puede seguir avanzando, lo cual no indica que no se encontró una solución al problema.

Procedure 1 GREEDY_ALGORITHM

Input: $C : \text{Set}$

$S : \text{Set}$

while $C \neq \emptyset$ **and** $SOLUTION(S) = \text{false}$ **do**

$x \leftarrow SELECT(C)$

$S \leftarrow S + x$

$C \leftarrow C - x$

end while

if $SOLUTION(S)$ **then**

return S

else

return \emptyset

end if

Cambio de monedas

Dado un sistema monetario S con N monedas de diferentes denominaciones y una cantidad de cambio C , calcular el menor número de monedas del sistema monetario S equivalente a C .

Ejemplos:

► **Input** : $s[] = 1, 3, 4$ $c = 6$

Output : 3

Explanation : The change will be $(4 + 1 + 1) = 6$

Procedure 2 COIN_CHANGE

Input: S : Array, c : Integer

$min \leftarrow 0$

$SORT_DESC(S)$

for $i \leftarrow 1$ to $S.length$ do

$min \leftarrow min + (c/S[i])$

$c \leftarrow c \bmod S[i]$

end for

Búsqueda en profundidad

La búsqueda en profundidad (en inglés **DFS** o **Depth First Search**) es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado ¹.

¹<https://goo.gl/ZsAAzP>

Input: $start : Vertex, g : Graph$

 $x \text{ visit} : Stack$

```
x  visit.push(start)
```

```
while !x visit.empty() do
```

```
current ← x_visit.pop()
```

if $current \notin visited$ then

```
visited.add(current)
```

```
for all v in current.connections() do
```

```
x  visit.push(start)
```

end for

end if

end while

```
return visited
```

Word Transformation ²

Un rompecabezas común que se encuentra en muchos periódicos y revistas es la transformación de palabras. Al tomar una palabra de inicio y alterar sucesivamente una sola letra para formar una nueva palabra, se puede construir una secuencia de palabras que cambia la palabra original a una palabra final dada. Por ejemplo, la palabra "spice" se puede transformar en cuatro pasos a la palabra "stock" de acuerdo con la siguiente secuencia: spice, slice, slick, stick, stock. Cada palabra sucesiva difiere de la palabra anterior en una sola posición de carácter, mientras que la longitud de la palabra sigue siendo la misma. Dado un diccionario de palabras de las cuales hacer transformaciones, más una lista de palabras iniciales y finales, escribe un programa para determinar el número de pasos en la transformación más corta posible.

²<https://onlinejudge.org/external/4/429.pdf>

Entrada

La entrada tendrá dos secciones. La primera sección será el diccionario de palabras disponibles con una palabra por línea, terminada por una línea que contiene un asterisco (*) en lugar de una palabra. Puede haber hasta 200 palabras en el diccionario; todas las palabras serán alfabéticas y en minúsculas, y ninguna palabra tendrá más de diez caracteres. Las palabras pueden aparecer en el diccionario en cualquier orden. Después del diccionario hay pares de palabras, un par por línea, con las palabras en el par separadas por un solo espacio. Estos pares representan las palabras iniciales y finales en una transformación. Se garantiza que todas las parejas tendrán una transformación utilizando el diccionario dado. Las palabras iniciales y finales aparecerán en el diccionario. Dos conjuntos de entrada consecutivos se separarán por una línea en blanco.

Salida

La salida debe contener una línea por par de palabras para cada conjunto de prueba, y debe incluir la palabra inicial, la palabra final y el número de pasos en la transformación más corta posible, separados por espacios individuales.

Ejemplo de entrada

dip
lip
mad
map
maple
may
pad
pip
pod
pop
sap
sip
slice
slick
spice
stick
stock
*
spice stock
may pod

Ejemplo de salida

spice stock 4
may pod 3

Programación de actividades

Te dan N actividades con sus tiempos de inicio (S_i) y finalización (F_i).
Selecciona el número máximo de actividades que puede realizar una sola persona, asumiendo que una persona solo puede trabajar en una sola actividad a la vez.³

Ejemplo:

► **Input :**

$\text{start[]} = (1, 3, 0, 5, 8, 5)$

$\text{finish[]} = (2, 4, 6, 7, 9, 9)$

Output : 4

³<https://goo.gl/1RG25M>

Procedure 4 ACTIVITIES_SELECTION

Input: A : *Activity_Array*
 SORT_ASC_BY_END(A)
 $i \leftarrow 1$
 $S \leftarrow \emptyset + A[i]$
 for $j \leftarrow 2$ to $A.length$ do
 if $A[j].start \geq A[i].end$ then
 $S \leftarrow S + A[j]$
 $i \leftarrow j$
 end if
 end for
 return S

Reservaciones de hotel

Un gerente de hotel debe procesar N reservas anticipadas de habitaciones para la próxima temporada. Su hotel tiene K habitaciones. Las reservas contienen una fecha de llegada y una fecha de salida. Quiere saber si hay suficientes habitaciones en el hotel para satisfacer la demanda. ⁴

⁴<https://goo.gl/7e6idL>

Procedure 5 BOOKING_PROBLEM

Input: *Arrival* : Array, *Departure* : Array, *k* : Integer

SORT_ASC(*Arrival*)

SORT_ASC(*Departure*)

i \leftarrow 1

j \leftarrow 1

current \leftarrow 0

required \leftarrow 0

```
while  $i < \text{Arrival.length}$  and  $j < \text{Departure.length}$  do
  if  $\text{Arrival}[i] < \text{Departure}[j]$  then
     $\text{current} \leftarrow \text{current} + 1$ 
     $\text{required} = \text{MAX}(\text{current}, \text{required})$ 
     $i \leftarrow i + 1$ 
  else
     $\text{current} \leftarrow \text{current} - 1$ 
     $j \leftarrow j + 1$ 
  end if
end while
```

```
while  $i < n$  do
     $current \leftarrow current + 1$ 
     $required = MAX(current, required)$ 
     $i \leftarrow i + 1$ 
end while
while  $j < n$  do
     $current \leftarrow current - 1$ 
     $j \leftarrow j + 1$ 
end while
return  $k \geq required$ 
```

Dragón de Loowater ⁵

Érase una vez, en el Reino de Loowater, una molestia menor se convirtió en un problema importante. Las costas de Rellau Creek, en el centro de Loowater, siempre habían sido un excelente caldo de cultivo para los gansos. Debido a la falta de depredadores, la población de gansos estaba fuera de control. La gente de Loowater se mantuvo principalmente alejada de los gansos. Ocasionalmente, un ganso atacaría a una de las personas, y tal vez mordiera un dedo o dos, pero en general, la gente toleraba a los gansos como una molestia menor. Un día, ocurrió una mutación anormal, y uno de los gansos engendró un dragón de fuego múltiple con cabeza de fuego. Cuando el dragón creció, amenazó con quemar el Reino de Loowater hasta quebrarse. Loowater tuvo un gran problema. El rey se alarmó y pidió a sus caballeros que mataran al dragón y salvaran el reino. Los caballeros explicaron: "Para matar al dragón, debemos cortar todas sus cabezas. Cada caballero puede cortar una de las cabezas del dragón. Las cabezas del dragón son de diferentes tamaños. Para cortar una cabeza, un caballero debe ser al menos tan alto como el diámetro de la cabeza. El sindicato de caballeros exige que, para cortar una cabeza, a un caballero se le pague un salario igual a una moneda de oro por cada centímetro de la altura del caballero". ¿Habría suficientes caballeros para derrotar al dragón? El rey llamó a sus asesores para que lo ayudaran a decidir cuántos caballeros contratar. Después de haber perdido mucho dinero construyendo Mir Park, el rey quería minimizar el gasto de matar al dragón. Como uno de los asesores, su trabajo era ayudar al rey. Lo tomaste muy en serio: si fallabas, ¡tú y todo el reino quedarían quemados!

⁵<https://onlinejudge.org/external/112/11292.pdf>

Entrada

La primera línea contiene dos números enteros entre 1 y 20000 inclusive, que indican el número n de cabezas que tiene el dragón y el número m de caballeros en el reino. Las siguientes n líneas contienen un número entero y dan los diámetros de las cabezas del dragón, en centímetros. Las siguientes líneas m contienen un número entero y especifican las alturas de los caballeros de Loowater, también en centímetros.

Salida

Genera una línea que contenga la cantidad mínima de monedas de oro que el rey debe pagar para matar al dragón. Si no es posible que los caballeros de Loowater maten al dragón, muestra la línea "¡Loowater está condenado!"

Ejemplo de entrada

2 3

5

4

7

8

4

Ejemplo de salida

11

Subconjunto de producto máximo de un arreglo

Dado un arreglo A , tenemos que encontrar el producto máximo posible con el subconjunto de elementos presentes en el arreglo. El producto máximo puede ser solo uno de los elementos del arreglo. ⁶

Ejemplos:

► **Input** : $a[] = -1, -1, -2, 4, 3$

Output : 24

Explanation : Maximum product will be $(-2 * -1 * 4 * 3) = 24$

► **Input** : $a[] = -1, 0$

Output : 0

Explanation : 0 (single element) is maximum product possible

► **Input** : $a[] = 0, 0, 0$

Output : 0

⁶<https://goo.gl/spb5Ka>

Una solución simple sería generar todos los subconjuntos, encontrar el producto de cada subconjunto y regresa el máximo. Sin embargo, existe una mejor solución si tomamos en cuenta los siguiente factores:

- ▶ Si el número de elementos negativos es par, el resultado es, sencillamente, el producto de todos los elementos.
- ▶ Si el número de elementos negativos es impar, el resultado es la multiplicación de todos los elementos excepto el número negativo más grande.
- ▶ Si hay ceros, el resultado el producto de todos los números, excepto los ceros con una excepción. La excepción es cuadn hay un número negativo y todos los otros números son ceros. En este caso, el resultado es 0.

Procedure 6 MAXIMUM_PRODUCT

Input: A : Array

if $n = 1$ then

if $A[1] < 1$ then

return 0

else

return $A[1]$

end if

end if

$max_neg \leftarrow INT_MIN$

$count_neg \leftarrow 0$

$count_zero \leftarrow 0$

$product \leftarrow 1$

```
for  $i \leftarrow 1$  to  $A.length$  do
  if  $A[i] = 0$  then
     $count\_zero \leftarrow count\_zero + 1$ 
  else
    if  $A[i] < 0$  then
       $count\_neg \leftarrow count\_neg + 1$ 
       $max\_neg \leftarrow MAX(max\_neg, count\_neg)$ 
    end if
     $product \leftarrow product * A[i]$ 
  end if
end for
```

```
if  $count\_zero = n$  then
  return 0
end if
if  $count\_neg \bmod 2 = 1$  then
  if  $count\_neg = 1$  and  $count\_zero > 0$ 
    and  $(count\_neg + count\_zero) = n$  then
    return 0
  end if
   $product \leftarrow product / max\_neg$ 
end if
return  $product$ 
```

Subsecuencia lexicográficamente más grande

Dada una cadena S y un entero K . La tarea es encontrar la subsecuencia lexicográficamente más grande de S , digamos T , de modo que cada carácter en T debe aparecer al menos K veces.⁷

Entrada: $S = \text{banana}$, $K = 2$

Salida: nn

b	a	n	a	n	a
b	a	n	a	n	a
b	a	n	a	n	a
b	a	n	a	n	a

De las opciones anteriores, nn es la lexicográficamente más grande.

⁷<https://goo.gl/iwFFCA>

Procedure 7 SUBSEQUENCE

Input: $S : \text{String}, T : \text{String}, k : \text{Integer}$

$last \leftarrow 1$

$new_last \leftarrow 1$

for $ch \leftarrow 'z'$ **to** $'a'$ **do**

$count \leftarrow 0$

for $i \leftarrow last$ **to** $S.length$ **do**

if $S[i] = ch$ **then**

$count \leftarrow count + 1$

end if

end for

if $count \geq k$ **then**

NEXT SLIDE

end if

end for

return T

```
for  $i \leftarrow last$  to  $S.length$  do
  if  $S[i] = ch$  then
     $T \leftarrow T + ch$ 
     $new\_last \leftarrow i$ 
  end if
end for
 $last \leftarrow new\_last$ 
```

Problema de la mochila fraccionaria

Dado los pesos y valores de N artículos, debemos colocar estos artículos en una mochila de capacidad W para obtener el máximo valor total en la mochila. Siempre es posible tomar una parte o totalidad de cada uno de los artículos.

Ejemplos:

Input:

$arr = [[60(b), 10(w)], [100(b), 20(w)], [120(b), 30(w)]]$

$W = 50$

Output:

Maximum possible value = 220

by taking items of weight 20 and 30 kg

Input: $A : Item, W : Integer$

SORT DESC BY RATIO(A)

$$currentWeight \leftarrow 0$$
$$acum \leftarrow 0$$
for $i \leftarrow 1$ **to** $A.length$ **do**

if $currentWeight + A[i].weight \leq W$ **then**

$$currentWeight \leftarrow currentWeight + A[i].weight$$
$$acum \leftarrow acum + A[i].value$$

else

$$remain \leftarrow W - currentWeight$$
$$acum \leftarrow acum + A[i].value * (remain / A[i].weight)$$

end if

end for

```
return acum
```

Input: $G : Graph(V, E)$

for all v in V do
$$INIT_SET(v)$$

for all (u, v) ordered by $\text{weight}(u, v)$, increasing do

if $FIND_SET(u) \neq FIND_SET(v)$ then

$$A \leftarrow A + (u, v)$$
$$UNION(u, v)$$

end if

end for

end for

```

return A

```

- ▶ La estructura de datos Union-Find nos permite mantener conjuntos disjuntos. Tienen dos operaciones muy simples:
 - ▶ $\text{FIND}(p, q)$ regresa verdadero si el conjunto al cual pertenece p está en el mismo conjunto que q .
 - ▶ $\text{UNION}(p, q)$ une el conjunto al cual pertenece p con el conjunto al cual pertenece q .
- ▶ ¿Cómo podríamos implementar esta estructura de datos?

Amigos⁸

Hay una ciudad con N ciudadanos. Se sabe que algunos pares de personas son amigos. Según el famoso dicho que dice: "Los amigos de mis amigos también son mis amigos", se deduce que si A y B son amigos y B y C son amigos, entonces A y C también lo son.

Su tarea es contar cuántas personas hay en el grupo más grande de amigos.

⁸<https://onlinejudge.org/external/106/10608.pdf>

Entrada

La primera línea contiene los números N y M , donde N es el número de ciudadanos de la ciudad ($1 \leq N \leq 30000$) y M es el número de pares de personas ($0 \leq M \leq 500000$), que son conocidos por ser amigos. Cada una de las siguientes líneas M consta de dos enteros A y B ($1 \leq A \leq N$, $1 \leq B \leq N$, $A \neq B$) que describen que A y B son amigos. Podría haber repeticiones entre los pares dados. **Salida**

El resultado contiene (en una línea por sí mismo) un número que indica cuántas personas hay en el grupo más grande de amigos.

Ejemplo de entrada

10 12

1 2

3 1

3 4

5 4

3 5

4 6

5 2

2 1

7 1

1 2

9 10

8 9

Ejemplo de salida

7