Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Informe 1

Integrante	LU	Correo electrónico
Jazmin	333	arreglandoerroresdelatex@gmail.com

Reservado para la cdra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Problema 1: Cruzando el puente

1.1. Inrtoducción

En este problema, un grupo formado por arquéologos y caníbales debe cruzar un puente. El mismo sólo puede ser atravezado por dos personas a la vez, y tiene que ser cruzado con una linterna. Como el grupo sólo dispone de una linterna, siempre debe regresar alguno al lado del puente original. Además, en ningún lado del puente puede haber más caníbales que arquéologos. Cada individuo consta de la propiedad "velocidad", un número natural que indica cuánto tiempo tarda en atravezar el puente. Si dos personas atraviezan el puente juntas, lo hacen en el tiempo más grande entre los dos.

1.2. Explicación de la solución

2. Problemas en el camino

2.1. Introducción

En el problema enunciado, Nuestros exploradores se encuentran con una balanza de dos platos, y en el de la izquierda la llave que necesitan. Para que esta sea de utilidad, al tomarla deben mantener la posición de la balanza y para realizar esto cuentan con pesas de peso igual a las potencias de tres (una pesa por potencia). Entonces, sabiendo el peso de la llave, necesitamos saber que pesas poner en cada plato para mantener el equilibrio original. Podemos pensar que el lado derecho de la balanza equivale a la operación de sustracción y el izquiero a la adición. De esta forma al agregar pesas de un lado o del otro estariamos sumando o restando su peso Formalmente esto equivale a decir que, si tenemos un entero n queremos sumar o restar potencias de tres, sin repetirlas hasta alcanzar su valor.

2.2. Explicación de la solución

Siendo el peso de la llave al que denominaremos P, el problema pide basicamente construir el numero P sumando (balanza con la llave) y restando(balanza sin la llave) numeros que son potencia de 3. Para resolver esto, totamos T, que seria el valor de P en la base Ternaria, es decir, escribir cada numero como $P = \sum_{i=0}^{n} a_i 3^i$ con $0 \leq a_i \leq 2$ y $n = log_3 P$ y entonces $T = a_1, a_2, ..., a_n$. Ahora, para cada elemento a_i de T, en orden, si a_i es 0, no hacemos nada, si a_i es 1, en la balanza con la llave ponemos el peso de 3^i y si $a_i = 2$, ponemos una pesa de 3^i en la balanza sin la llave y ademas sumamos 1 a $a_{(i+1)}$ (y consecuentemente, se cambian los valores posteriores de T para que sigan en base ternaria)

2.2.1. Pseudocódigo

Nota: Para la implementacion, en vez de crear el conjunto T, vamos tomando el resto de P y diviendolo por 3 entonces, el valor de la variable rem"seria el equivalente al a_i (siendo i la iteracion)

Pesas(Natural: P)

```
1: Mientras P sea mayor a 0
2: if rem = 0 then
3:     La pesa va para la balanza con la llave  rem = 2
4:     P¡-P+1
5:     La pesa va para la otra balanza
```

2.2.2. Demostración de Correctitud

2.2.3. Demostración de Complejidad

El algoritmo consta de una sola iteracion, dentro de la iteracion, calcular el resto, y las comparaciones son operaciones que se hacen en O(1), mientras que enviar la pesa a cada balanza es agregar un numero al final de una lista, cosa que en la implementacion toma O(1). Entonces, dentro de la iteracion se hacen solamente operaciones en O(1). Quedandonos asi que la complejidad depende de la cantidad de iteraciones que se hagan.

La iteracion tiene como condicion que P¿0, siendo P inicialmente el valor de entrada. Ahora, P es editado en cada iteracion: independientemente del if que hay adentro, a P se lo divide por 3. Si siguiesemos sin contar lo que pasa en el if,sabemos que se harian $\log_3(P) veces dividir P por 3 para q P sea menor a 0. Ahora, t$
$a_1, ..., a_n donde$P $= \sum_{i=0}^{n} a_i 3^i$ con $0 \leq a_i \leq 2$ y $n = log_3 P$, siendo el valor de rem el valor de $a_i en la iesima iteracion. Cuando hacemos que P < -P+1 nos queda que en la proxima iteracion rem valdria una mas, e$
$1) pasaria a valer una mas del q valdria en T. Ahora, siendo estos que para todo a_k valen entre 0 y 2, si a_(i+1) en vez de vale$
$2) valdria una mas tambien, y si a_(i+2) valia 2, pasaria a valer 0 y a_(i+3) valdria una mas y asi sucesivamente hasta llega$
$1) que entraria a valer 1 y a_n valdria 0. Luego no habria mas casos donde a_j (para i < j <= n) donde a_j sea 2, entonces no h$
$1 iteraciones$

Entonces, la complejidad del algoritmo es de $(Log_3(P))$. Luego, se puede probar que Log(P) es $(\sqrt{}(P))$

2.3. Experimentación

2.3.1. Resultados

2.3.2. Análisis

3. Problema 3: Guardando el tesoro

3.1. Introducción

En este problema, los exploradores se encuentran frente a muchos tesoros que desearían poder llevarse. Para esto, cuentan con varias mochilas, cada una con una determinada capacidad de peso que puede cargar. Los tesoros son de distintos tipos, y cada tipo tiene un peso y un valor determinado. El objetivo es encontrar la manera optima de llenar las mochilas para poder llevar el mayor valor posibe.

Formalmente, tenemos un conjunto de elementos que tienen como propiedad dos naturales asociados (el peso y el valor). Entonces podemos inferir que existen dos criterios de ordenamiento asociados respectivamente a estos valores. Al mismo tiempo tenemos otro conjunto de elementos que posee como propiedad un natural asociado (capacidad). Nuestro objetivo es seleccionar una combinación de

los primeros objetos, restringida por las capacidades dadas,de modo de maximizar la sumatoria del valor de los mismos.

3.2. Explicación de la solución

Inicialmente creimos que este problema podría resolverme mediante un algoritmo de BackTracking, sin embargo observamos que nunca lograríamos conseguir la complejidad pedida puesto que este tipo de algoritmos conlleva una complejidad exponencial sperior a la pedida. Finalmente pudimos resolverlo mediante prOgramación dinámica. Nuestro algoritmo principal llama a dos funciones que utilizan la recursividad para podes obtener en un caso el valor óptimo, y en el otro "las mochilas llenas."Para facilitar el entendimiento del algoritmo vamos a introducir el concepto de "hipercubo"...

observación: nuestro algoritmo descarta los tesoros que tienen un peso mayor al maximo de capacidad entre las mochilas. Asumimos en el pseusocodigo que nuestra entrada cumple esa propiedad.

3.2.1. Pseudocódigo

guardandoTesoro(mochilas: vector<mochila>, cofre: vector<tesoro>)

```
1: objXpesos ← Hipercubo() incializado en -1   ▷ (en las posiciones donde no puede haber objetos incializo en 0) ()
2: sol= ValorOptimo(objXpesos,cofre,cofre.size-1,capacidades de las mochilas)
3: LlenarMochilas(objXpesos, cofre, mochila1, mochila2, mochila3)
4: return (sol, mochilas) =0
```

LlenarMochilas(objetoXpeso: hipercubo, cofre:vector<tesoro>, m1,m2,m3:mochilas)

```
1: desde i= cofre.size-1 hasta i=0
2: obj ← cofre[i]
3: cap1 ← m1.Capacidad
4: cap2 ← m2.Capacidad
5: cap3 ← m3.Capacidad
6: if i = 0 then
7:     Agregar obj en cualquier mochila en la que entre.
8: else
9:     valM1 ← obj.valor + ValorOptimo(objetoXPeso, cofre, i,cap1-obj.peso,cap2,cap3 )
10:    valM2 ← obj.valor + ValorOptimo(objetoXPeso, cofre,i,cap1,cap2-obj.peso,cap3 )
11:    valM3 ← obj.valor + ValorOptimo(objetoXPeso, cofre,i, cap1, cap2,cap3-obj.peso )
12:    MeterEnCorrecta(valM1,valM2,valM3,m1,m2,m3,obj)
13: end if
```

ValorOptimo(objXpeso:hipercubo, cofre:vector<tesoro>, objeto:int, peso1: int, peso2:int, peso3:int)

```
1: pesoObj ← cofre[objeto].peso ▷ (1)
2: pesoVal ← cofre[objeto].valor ▷ (1)
3: if peso1 < 0 ∨ peso2 < 0 ∨ peso3 < 0 then
4:     return −1 ▷ (1)
5: end if
6: if objetoxPesos[objeto][peso1][peso2][peso3] ≠ -1 then
7:     return objetoxPesos[objeto][peso1][peso2][peso3] ▷ (1)
8: end if
9: if objeto = 0 then
10:    val ← 0 ▷ (1)
11:    if peso1 ≥ pesoObj ∨ peso2 ≥ pesoObj ∨ peso3 ≥ pesoObj then
12:        val ← valorObj ▷ (1)
13:        objetoxPesos[objeto][peso1][peso2][peso3] ← val ▷ (1)
14:        return val ▷ (1)
15:    end if
16: else
17:    PosiblesSolus ← vector<int> ▷ (1)
18:    sinObj ← ValorOptimo(objetoxPesos, cofre, objeto -1, peso1, peso2, peso3)
19:    PosiblesSolus.Agregar(sinObj)
20:    if peso1-pesoObj ≥ 0 then
21:        objen1 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1 - pesoObj, peso2, peso3)
22:        PosiblesSolus.Agrergar(objen1)
23:    end if
24:    if peso2 - pesoObj ≥ 0 then
25:        objen2 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1, peso2 - pesoObj, peso3)
26:        PosiblesSolus.Agregar(objen2)
27:    end if
28:    if peso3-pesoObj ≥ 0 then
29:        objen3 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1, peso2, peso3 - pesoObj)
30:        PosiblesSolus.Agregar(objen3)
31:    end if
32:    valor=Max(PsiblesSolus)
33:    objetoxPesos[objeto][peso1][peso2][peso3] = valor
34:    return valor
35: end if
```

3.2.2. Demostración de Correctitud

Presentaremos la función matemática que modela nuestro problema:

$f(o, p_1, p_2, p_3) = 0$
$f(n, p_1, p_2, p_3) = Max(V_n + Max(f(n-1, p_1 - p_n, p_2, p_3), f(n-1, p_1, p_2 - p_n, p_3), f(n-1, p_1, p_2, p_3 - p_n)), f(n-1, p_1, p_2, p_3))$
Donde n representa el numero de tesoro, V_n y P_n su valor y su peso respectivamente. p_1, p_2 *y* p_3 son las capacidades de cada mochila.

Para la demostración utilizaremos inducción global en n.

Caso base: queremos ver que $f(0, p_1, p2, p_3)$ resulta ser el valor máximo que se puede obtener con 0 objetos: $f(0, p_1, p_2, p_3) = 0$ al tener 0 objetos el valor de los mismos es 0 de modo que es el máximo valos posible.

Hipotesis Inductiva: $\forall k < n$ vale que $f(k, p_1, p_2, p_3)$ da el valor máximo que se puede obtener con k objetos.

Ahora queremos ver que $f(n, p_1, p_2, p_3)$ da el valor óptimo para n objetos.

$f(n, p_1, p_2, p_3) = Max(V_n + Max(f(n-1, p_1 - p_n, p_2, p_3), f(n-1, p_1, p_2 - p_n, p_3), f(n-1, p_1, p_2, p_3 - p_n)), f(n-1, p_1, p_2, p_3))$
por hipótesis inductiva (como $n - 1 < k$ para algun k) sabemos que $f(n-1, p_1 - p_n, p_2, p_3), f(n-1, p_1, p_2 - p_n, p_3), f(n-1, p_1, p_2, p_3 - p_n) y f(n-1, p_1, p_2, p_3)$ son los valores optimos que se pueden conseguir con n-1 objetos restando (o no) el peso del objeto n de modo de luego poder guardarlo en alguna mochila (o no). utilizaremos los renombres $V_{01}, V_{02}, V_{03}, V_{00}$ respectivamente. Entonces tenemos:
$f(n, p_1, p_2, p_3) = Max(V_n + Max(V_{01}, V_{0,2}, V_{03}), V_{00})$
Podemos ver que esta función compara el valor optimo de llenar las mochilas con n-1 tesoros y el tesoro n, con el valor optimo de llenar las mochilas con n-1 tesoros sin el tesoro n (de esta forma se tiene cuenta el caso en el que el mejor valor se obtuviene de poner algun objeto de los n-1 anteriores que impide luego meter el tesoro n).
Como la función Max devuelve el mayor valor, el resultado sera el óptimo.
Como valen P(0)..p(k) $\forall k < n$ y vale p(n) vale p(n) $\forall n \in N \cup 0$

3.2.3. Demostración de Complejidad

3.3. Experimentación

3.3.1. Resultados

3.3.2. Análisis