

Informe 1

Grupo

Alvarez Vico Jazmín

Cortés Conde Titó Javier María

Pedraza Marcelo

Rozenberg Uriel Jonathan

6 de septiembre de 2016

Índice

1. Problema 1: Cruzando el puente	3
1.1. Inrtroducción	3
1.2. Explicación de la solución	3
2. Problema 2: Problemas en el camino	3
2.1. Introducción	3
2.2. Explicación de la solución	3
2.2.1. Pseudocódigo	3
2.2.2. Demostración de Correctitud	4
2.2.3. Demostración de Complejidad	4
2.3. Experimentación	4
2.3.1. Resultados	4
2.3.2. Análisis	4
3. Problema 3: Guardando el tesoro	4
3.1. Introducción	4
3.2. Explicación de la solución	4
3.2.1. Pseudocódigo	5
3.2.2. Demostración de Correctitud	5
3.2.3. Demostración de Complejidad	6
3.3. Experimentación	6
3.3.1. Resultados	6
3.3.2. Análisis	6

1. Problema 1: Cruzando el puente

1.1. Introducción

En este problema, un grupo formado por arqueólogos y caníbales debe cruzar un puente. El mismo sólo puede ser atravesado 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 arqueólogos. Cada individuo consta de la propiedad "velocidad", un número natural que indica cuánto tiempo tarda en atravesar el puente. Si dos personas atraviesan el puente juntas, lo hacen en el tiempo más grande entre los dos.

1.2. Explicación de la solución

2. Problema 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 izquierdo a la adición. De esta forma al agregar pesas de un lado o del otro estaríamos 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 básicamente construir el número P sumando (balanza con la llave) y restando (balanza sin la llave) números que son potencia de 3. Para resolver esto, tomamos T , que sería el valor de P en la base Ternaria, es decir, escribir cada número como

$$P = \sum_{i=0}^n a_i 3^i$$

con $0 \leq a_i \leq 2$ y $n = \log_3 P$ y entonces $T = a_1, a_2, \dots, a_n$. 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 además 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 implementación, en vez de crear el conjunto T , vamos tomando el resto de P y dividiéndolo por 3 entonces, el valor de la variable "rem" sería el equivalente al a_i (siendo la iteración)

[H] Pesas(Natural: P) + [1] + Mientras P sea mayor a 0 + + + rem = 0 + La pesa va para la balanza con la llave + rem = 2 + $P \div P+1$ + 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 iteración, dentro de la iteración, 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 número al final de una lista, cosa que en la implementación toma $O(1)$. Entonces, dentro de la iteración se hacen solamente operaciones en $O(1)$. Quedándonos así que la complejidad depende de la cantidad de iteraciones que se hagan.

La iteración tiene como condición que $P \neq 0$, siendo P inicialmente el valor de entrada. Ahora, P es editado en cada iteración: independientemente del `if` que hay adentro, a P se lo divide por 3. Si siguiésemos sin contar lo que pasa en el `if`, sabemos que se harían $\log_3(P)$ veces dividir P por 3 para que P se amenore a 0. Ahora, $t = a_1, \dots, a_n$ 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 a_i en la i -ésima iteración. Cuando $P = 1$ nos queda que en la próxima iteración P se vuelve 0, es decir a_{i+1} pasaría a valer 0 y $a_i + 1$ en vez de valer 3, valdría 0 y a_{i+2} valdría 1 y así sucesivamente hasta llegar a a_n . Si el resultado también es a_n valdría 2, entonces tendríamos una iteración más que entraría a valer 1 y a_n valdría 0. Luego no habría más casos donde a_j (para $i < j \leq n$) donde a_j sea 2, entonces no hay más iteraciones.

Entonces, la complejidad del algoritmo es de $O(\log_3(P))$. Luego, se puede probar que $\log(P)$ es $O(\sqrt{P})$.

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 óptima de llenar las mochilas para poder llevar el mayor valor posible.

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 creímos que este problema podría resolverse 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 superior a la pedida. Finalmente pudimos resolverlo mediante programación dinámica. Nuestro algoritmo principal llama a dos funciones que

utilizan la recursividad para poder 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 máximo de capacidad entre las mochilas. Asumimos en el pseudocódigo que nuestra entrada cumple esa propiedad.

3.2.1. Pseudocódigo

[H]**guardandoTesoro**(mochilas: vector[mochila], cofre: vector[tesoro]) [1] objXpesos \leftarrow Hiper-cubo() inicializado en -1 (en las posiciones donde no puede haber objetos inicializo en 0) sol= ValorOptimo(objXpesos,cofre,cofre.size-1,capacidades de las mochilas) LlenarMochilas(objXpesos, cofre, mochila1, mochila2, mochila3) return (sol, mochilas)

[H]**LlenarMochilas**(objetoXpeso: hiper-cubo, cofre:vector[tesoro], m1,m2,m3:mochilas) [1] i \leftarrow cofre.size-1 mientras i \geq 0

[H]**ValorOptimo**(objXpeso:hipercubo, cofre:vector[tesoro], objeto:int, peso1: int, peso2:int, peso3:int) [1] pesoObj \leftarrow objeto.peso (1) pesoVal \leftarrow objeto.valor (1)

peso1 < 0 \vee peso2 < 0 \vee peso3 < 0 return -1 (1)

objetoXPesos[objeto][peso1][peso2][peso3] \neq -1 return objetoXPesos[objeto][peso1][peso2][peso3]

(1) objeto = 0 val \leftarrow 0 (1) peso1 pesoObj \vee peso2 pesoObj \vee peso3 pesoObj val \leftarrow valorObj (1) objetoXPesos[objeto][peso1][peso2][peso3] \leftarrow val (1) return val (1)

PosiblesSolus \leftarrow vector[int](1) sinObj \leftarrow ValorOptimo(objetoXPesos, cofre, objeto -1, peso1, peso2, peso3) PosiblesSolus.Agregar(sinObj) (peso1-pesoObj 0) objen1 = valorObj + ValorOptimo(objetoXPesos, cofre, objeto - 1, peso1 - pesoObj, peso2, peso3) PosiblesSolus.Agregar(objen1) peso2 - pesoObj 0 objen2 = valorObj + ValorOptimo(objetoXPesos, cofre, objeto - 1, peso1, peso2 - pesoObj, peso3) PosiblesSolus.Agregar(objen2) peso3-pesoObj 0 objen3 = valorObj + ValorOptimo(objetoXPesos, cofre, objeto - 1, peso1, peso2, peso3 - pesoObj) PosiblesSolus.Agregar(objen3) valor=Max(PosiblesSolus) objetoXPesos[objeto][peso1][peso2][peso3] = valor return valor

3.2.2. Demostración de Correctitud

Presentaremos la función matemática que modela nuestro problema:

$$f(0, p_1, p_2, p_3) = 0$$

$$f(n, p_1, p_2, p_3) = \text{Max}(V_n + \text{Max}(f(n-1, p_1 - p_n, p_2, p_3), f(n-1, p_1, p_2 - p_n, p_3), f(n-1, p_1, p_2, p_3 - p_n)), f(n-1, p_1, p_2, p_3))$$

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 $f(0, p_1, p_2, p_3)$ 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 valor 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.

$$f(n, p_1, p_2, p_3) = \text{Max}(V_n + \text{Max}(f(n-1, p_1 - p_n, p_2, p_3), f(n-1, p_1, p_2 - p_n, p_3), f(n-1, p_1, p_2, p_3 - p_n)), f(n-1, p_1, p_2, p_3))$$

$p_n)), f(n-1, p_1, p_2, p_3))$

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:

$f(n, p_1, p_2, p_3) = \text{Max}(V_n + \text{Max}(V_{01}, V_{02}, V_{03}), V_{00})$

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 $P(0) \dots p(k) \forall k < n$ 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