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

Integrante	LU	Correo electrónico

Reservado para la cdra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contents

1 Problema 1: Cruzando el puente

1.1 Inrtoducción

En este problema, un Grupo formado por arqueologos y caníbales debe cruzar un puente. El mismo solo puede ser atravezado por dos personas a la vez, y como el grupo solo dispone de una linterna, siempre debe volver alguien con la linterna. Además en ningun lugar puede haber mas canibales que arqueologos. Cada individuo consta de la propiedad "velocidad" un numero natural que indica cuanto tarda en atravezar el puente. Sí dos personas atraviezan el puente juntas, lo hacen a la velocidad minima de los dos.

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 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.

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. 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¿)

objXpesos ← Hipercubo() incializado en -1 (en las posiciones donde no puede haber objetos incializo en 0)

sol= ValorOptimo()

ValorOptimo(objXpeso:hipercubo, cofre:vector¡tesoro¿, objeto:int, peso1: int, peso2:int, peso3:int)

int pesoObj ← objeto.peso int pesoVal ← objeto.valor

if(peso1 ¡ 0 ∨ peso2 ¡ 0 ∨ peso3 ¡ 0) return -1

if(objetoxPesos[objeto][peso1][peso2][peso3] ≠ -1) return objetoxPesos[objeto][peso1][peso2][peso3]

if(objeto = 0) int val ← 0 if(peso1 pesoObj ∨ peso2 pesoObj ∨ peso3 pesoObj) val ← valorObj; objetoxPesos[objeto][peso1][peso2][peso3] ← val return val else PosiblesSolus ← vector¡int¿ int sinObj ← ValorOptimo(objetoxPesos, cofre, objeto -1, peso1, peso2, peso3) PosiblesSolus.Agregar(sinObj) if(peso1-pesoObj 0) int objen1 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1 - pesoObj, peso2, peso3) PosiblesSolus.Agrergar(objen1)

if(peso2 - pesoObj 0) int objen2 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1, peso2 - pesoObj, peso3) PosiblesSolus.Agregar(objen2)

if(peso3-pesoObj 0) int objen3 = valorObj + ValorOptimo(objetoxPesos, cofre, objeto - 1, peso1, peso2, peso3 - pesoObj) PosiblesSolus.Agregar(objen3) int valor=Max(PsiblesSolus) objetoxPesos[objeto][peso1][peso2][peso3] = valor return valor

3.2.2 Demostración de Correctitud

3.2.3 Demostración de Complejidad

3.3 Experimentación

3.3.1 Resultados

3.3.2 Análisis