Tarea 1

Trinidad Hernández Norma Verónica
Vargas Muñoz José de Jesús
Vilchis Domínguez Miguel Alonso

22 de agosto de 2015

1. Ejercicios

1. Explicar la complejidad de los problemas tipo NP, NP-Completos y NP-Duros para panaderos.

   Reducciones
   A un panadero se le encomienda la tarea de cocinar galletas con forma de cuadrado, pero sabe que si le da dicha forma a las galletas, estas no se cocinarán del centro. Por el contrario, sabe que si cocina galletas de forma circular estas se cocinarán de una manera uniforme. ¿Cómo entrega el panadero galletas cuadradas y cocidas del centro? Haciendo uso de una reducción.

   Definición 1.1. Sean P y Q dos problemas. Supóngase que un ejemplar arbitrario E de P puede solucionarse convirtiendo al ejemplar E en un ejemplar E' de Q, resolviendo para E' y reduciendo la solución S' en la solución S para E. Si existen algoritmos C_{Ej}, para convertir un ejemplar E en otro E', y C_{Sol} para convertir la solución S' en S, se dice entonces que existe una reducción de P en Q, la cual se denota como $P \propto Q$.

   El panadero toma la masa de galletas (E) y utiliza un cortador (C_{Ej}) para darles forma circular (E'). Cocina esta masa con formas redondas en el horno y obtiene galletas perfectas (S). Cuando las galletas se enfrían, el panadero utiliza un cuchillo para cortarlas (C_{Sol}) y obtener ejemplares cuadrados (S'). Nuestro panadero acaba de aplicar el *Algoritmo de Reducción (o de Simplificación)* para resolver su problema. Tal algoritmo es usado cuando P es muy difícil de resolver de manera directa y Q es más fácil.

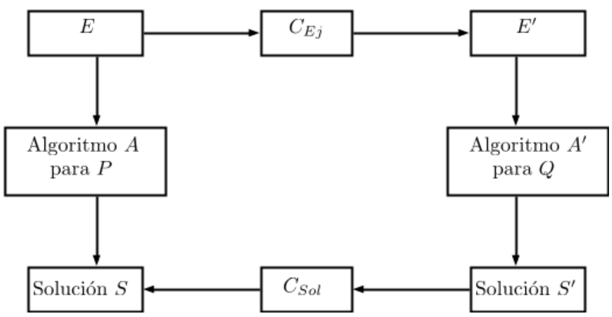

Figura 1: Reducción de P en Q: $P \propto Q$

   Supongamos que el panadero tarda 2 segundos en cortar una porcin de masa en forma de una galleta circular, y supongamos que le toma 4 segundos cortar con cuchillo una galleta circular para obtener una cuadrada. Cortar n galletas circulares le tomará al panadero $T(n) = 2n$ segundos, y cortarlas después en galletas cuadradas le tomará $T(n) = 4n$ segundos. Se dice que ambas transformaciones son *polinomiales*, ya que el tiempo que toma convertir el ejemplar de un problema a otro ($T(n) = 2n$), así como convertir la solución de un problema en la solución de otro ($T(n) = 4n$), es polinomial. Por desempeño computacional de orden polinomial nos referimos a que es posible expresar el tiempo de transformación como un polinomio que está en función del tamaño de la entrada. En este caso, del número de galletas.

   Nuestro panadero puede no saber cómo cocinar galletas cuadradas en un horno, pero sí cómo cocinarlas circulares y después cortarlas. El panadero le ocultará a los demás que desconoce la manera de cocinar

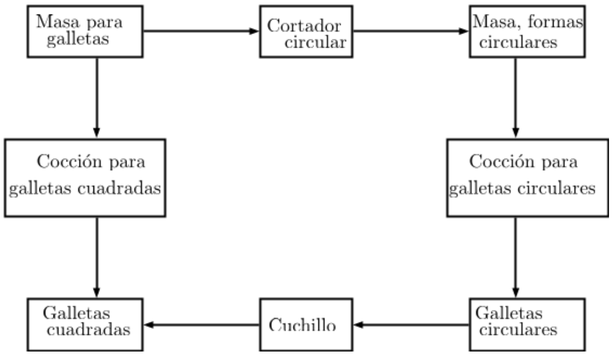

Figura 2: Reducción de P en Q para panaderos

galletas cuadradas en el horno y esto no le angustia: el panadero puede obtener galletas cuadradas de otra forma, en tiempo polinomial, y eso es lo importante. Nuestra idea se resume en el siguiente teorema. **Teorema.** Si L_1 es polinomialmente reducible a L_2 y existe un algoritmo polinomial para L_2, entonces existe un algoritmo polinomial para L_1. $\square$

El No Determinismo

Nuestro panadero estricto sigue recetas al pie de la letra para sus galletas, lo cual le garantiza exactamente las mismas galletas cada vez, siempre y cuando use los mismos ingredientes. Este panadero sigue algorimos deterministas cada vez que cocina. En términos de computación, decimos que un algoritmo es *determinista* si para una entrada particular, siempre produce la misma salida, con la máquina subyacente siempre pasando por la misma secuencia de estados. Volviendo a la cocina, el algoritmo determinista del panadero es la receta que sigue, la entrada particular son los ingredientes de sus galletas, la salida es el producto final, la máquina subyacente está representada por sus manos y los utensilios de los que se vale, y la secuencia de estados es la forma que va tomando la masa.

Ahora bien, supongamos que a nuestro panadero se le encarga crear una galleta que sea crujiente en su exterior y suave en su interior. El panadero sabe que este requerimiento particular para la galleta depende de la manera de batir la masa, pero no sabe en específico que método al aplicarse resultará en la galleta deseada. El panadero modifica la receta como sigue: comienza la preparación de la galleta como sabe hacerlo y cuando llega el momento de batir tira un dado, el cual le indicará qué técnica de batido usar.

Un *algoritmo no determinista* tiene todas las operaciones cláiscas de uno determinista, con la adición de una primitiva no determinista llamada *nd-choice*, la cual es usada para manipular selecciones.

Nuestro panadero usó una primitiva nd-choice en su receta (algoritmo): el dado. Entonces, dependiendo del resultado del dado se obtiene una galleta distinta. Él prueba la galleta una vez salida del horno; esto para determinar si resultó según los requerimientos. Nuestro panadero acaba de crear, sin saberlo, un algoritmo no determinista de tiempo polinomial. El algoritmo es el siguiente:

a) Aplicar cierto número de pasos de la receta como suele hacerlo.

b) Utilizar una primitiva *nd-choice* para hacer un cambio aleatorio a la receta.

c) Seguir con los pasos de la receta.

d) **Verificar** si el producto obtenido es como se deseaba. De ser así, reportar **sí**, y **no** en caso contrario.

Es importante notar que esta estructura de algoritmo no determinista tiene desempeño polinomial porque tanto la aplicación de la primitiva *nd-choice* como la verificación toman tiempo polinomial.

Definición 1.2. La clase de problemas para los cuales existe un algoritmo no determinista cuyo desempeño computacional es polinomial (en función del tamaño de la entrada) es llamada **Clase NP**.

Supongamos para efectos de nuestro ejemplo que la Clase NP se restringe a la panadería de nuestro personaje.Esto es, la Clase NP se compone de cierto número de galletas. Digamos que el panadero elige una galleta X y descubre que todas las demás recetas de galletas donde usa su dado (los demás problemas de la clase NP) pueden reducirse a la galleta X. Decimos entonces que la galleta es **NP-Dura.**

Definición 1.3. Un problema X es llamado *Problema NP-Duro*, *NP-Hard*, si cada problema en NP es polinomialmente reducible en X.

Ahora supongamos que nuestro panadero conoce una receta de galletas X que es de la clase NP, es decir, utiliza un dado en su preparación. Supongamos además que la galleta es NP-Dura, es decir, todas las demás recetas galletas de la Clase NP pueden reducirse en tiempo polinomial a la receta de la galleta X. Entonces decimos que nuestra galleta es NP-Dura.

Definición 1.4. Un problema X es llamado *Problema NP-Completo*, *NP-Complete*, si

a) el problema X pertenece a NP y

b) el problema X es NP-Duro