Teoría de Algoritmos I

75.29 / 95.06

Trabajo Práctico 3

Amigo, Nicolás	105832
Nicolini, Franco	105632
Serrano, Maria Candela	105287
Singer, Joaquín	105854
Primerano Lomba, Franco Alejandro	106004

Profesores:
Víctor Podberezski
Lucas Ludueño
Kevin Untrojb
Jose Sbruzzi
Ernesto Alvarez

21 de Junio de 2021

Índice general

Parte 1: La mudanza

1.1. Presentación del problema

Un grupo de amigos que conviven están mudándose a un departamento nuevo. Han juntado sus pertenencias en cajas de diferentes volúmenes que recolectaron en supermercados y tiendas.

Dados r recipientes de capacidad V a tu disposición como máximo por viaje, una lista de objetos con volumen y una cantidad de viajes k, se buscará saber si los objetos podrán ser transportados en menos de k viajes. Para esto se podrán llenar los r recipientes a su totalidad o menos con los objetos, optando cuando se quiera en viajar y descargar los recipientes, volviendo a estar vacíos y aumentando en 1 la cantidad de viajes realizados.

1.2. Que necesitamos probar para demostrar que es un problema NP-Completo

Para que un problema de decisión $L \subseteq (Si, No)$ sea NP-Completo se deberá cumplir:[1]

1. $L \in NP$
2. $L' \leq_p L$, para cualquier $L' \in NP$.

Donde $L' \leq_p L$ significa que L' es como mucho igual de difícil que L.

Para probar que el problema es NP-Completo necesitaremos en primer lugar probar que el problema es NP, demostrando que existe un algoritmo que en tiempo polinomial certifique una solución a una instancia del mismo. Luego necesitaremos utilizar una reducción polinomial de un problema NP-Completo a una instancia de este problema que queremos demostrar.

1.3. Prueba de que el problema es NP

Para probar que el problema es NP bastará con probar que existe un algoritmo que, dado un certificado con una posible solución a una instancia, compruebe en tiempo polinomial si la misma es correcta.

Si se diera un certificado diciendo los grupos de objetos(dando su volumen) que se repartiría en cada día se podría verificar que todos los objetos de la instancia original se encuentran en el certificado. Que la cantidad de días es menor a k. Y por último que cada grupo de objetos entra en los r recipientes de capacidad V. En este caso cada una de las acciones descritas se puede realizar en tiempo polinomial, por lo que el algoritmo de certificación demuestra que es un problema NP.

```
Verificador(r,v,k,objetos, certificado):
    #Graba en la matriz grupos para cada día, para cada recipiente una tupla
                                                    con los volúmenes almacenados.
    Grupos[k][r], días = Recibir Certificado(certificado)
    Si días >= k
        Devolver Falso
    Total = Unir(Grupos) #O(n)
    Si Comparar Conjuntos (Total, objetos) == Distintos #O(n*m) siendo m la
                                            cantidad de objetos dados por el certificado.
        Devolver Falso
    Por cada grupo[k] #Representa a los k-1 viajes
        Verificar si el volumen de cada r excede a la capacidad V:
            Devolver Falso
    Devolver Verdadero
```

Recibir Certificado debería devolver una matriz donde cada fila representa un día y cada columna uno de los recipientes utilizado ese día. Aquí presentamos el resto del pseudocódigo:

```
    Recibir Certificado(certificado)
        dias = certificado.dias
        grupos = certificado.matriz
        Devolver grupos, dias

    Unir(Grupos)
        Conjunto = Lista vacia
        Para cada Fila:
            Para cada Columna:
                Conjunto -> agregar(grupos[fila][columna].elementos)
        Devolver Conjunto

    Verificar Volumen(Grupo[r], V)
        Para cada elemento:
            suma = sumar(elemento)
            si suma > V
                Devolver Excede
        Devolver Cumple

    Comparar Conjuntos(Total, objetos)
        Para cada elemento del Total:
            Encontro = Búsqueda lineal en objetos
            Si Encontro es Falso
                Devolver 'Distintos'
        Devolver 'Iguales'
```

1.4. Prueba de que cualquier problema NP se puede reducir polinomicamente al problema del transporte

Como cualquier problema NP se puede reducir polinomicamente a uno NP-Completo, si reducimos cualquier problema NP-Completo a nuestro problema, habremos demostrado por transitividad lo propuesto.

Para ello utilizaremos el problema NP-Completo conocido como 2-Partition problem (nombrado como el problema NP-Hard más fácil) asumiendo y citando las fuentes que confirman que lo es. [2]

La reducción polinomial consistiría en lo siguiente:

Se iteran los números y se calcula su suma, si es impar ya se sabe que el problema no tiene solución. Se asignará un solo recipiente de capacidad $\frac{suma}{2}$ y se pedirá que el transporte sea menor a 3 viajes. El volumen de los objetos a transportar será igual a los elementos del subset suma. Si se puede realizar en 2 viajes significará que el subset se puede dividir en 2 subsets de igual tamaño.

```
Reductor(subset):
    total = sumar(subset) #O(n)
    Si total % 2 != 0:
        Devolver False
    Devolver Problema_Transporte(r=1, k=3, V=total/2, subset)
```

Habiendo demostrado que es posible esta reducción polinomial se demostró finalmente que el problema del transporte es NP-Completo.

1.5. Análisis de la complejidad de la transformación propuesta

Recibir certificado solo accedería a los campos del certificado, lo cual se realizaría en $O(1)$. Unir grupos recorrería k filas y k columnas, luego adjuntaría una lista en la lista principal. Teniendo una complejidad de $O(k \cdot r)$. Comparar los conjuntos recorrerá para cada elemento del total en una lista de objetos, que debe ser finita. Por lo tanto la complejidad temporal para n objetos totales será $O(n \cdot m)$ donde m es la cantidad de elementos recibidos en el certificado.

La complejidad del verificador será entonces:

La suma de, $O(1)$ por Recibir Certificado. $O(k \cdot r)$ por unir los objetos. $O(n \cdot m)$ por comparar los conjuntos. Finalmente para cada casillero de grupos se verificará que no se exceda la cantidad, para ello se sumaran todos los elementos de un casillero y se comparará con el máximo. En el peor de los casos recibiremos los m objetos en un solo casillero, así que la complejidad será $O(k \cdot r \cdot m)$. Habiendo analizado todo esto podremos concluir que la complejidad final será $O(k \cdot r \cdot m + n \cdot m)$. La cual es polinomial.

Parte 2: La personalización del yate

2.1. Presentación del problema

Una constructora de barcos está preparando para la venta un nuevo yate. Para atraer posibles compradores han planteado que el cliente pueda personalizar seleccionando entre "n" diferentes packs de accesorios y modificaciones. Por otro lado existen "m" regulaciones que se deben cumplir para la navegación segura. Se realizó dos propuestas. En la primera cada regulación está vinculada a cuatro packs diferentes. En la segunda por dos packs diferentes.

1. Pruebe que en caso de usar 4 pack por regulación el problema pertenece a NP-C.

2. Pruebe que en caso de usar 2 pack por regulación el problema pertenece a P

2.2. Prueba de que con 4 packs es un problema NP-Completo

En primer lugar probaremos que este problema es NP. Demostraremos que es NP dando un ejemplo que nos lleve a una resolución en tiempo polinomial. Para ello se podría utilizar un algoritmo de certificación que en base a si un pack se utilice o no, revise regulación por regulación a ver si se cumplen. Esto se puede realizar de la siguiente manera: Dada una solución posible de combinación de packs, verificar si todas las regulaciones se están cumpliendo. Esto se realizaría en tiempo polinomial ya que bastará con recorrer linealmente las regulaciones.

Por otra parte, se puede demostrar que cualquier problema NP-Hard puede ser reducido a una instancia de este problema. Para ello utilizaremos el conocido problema NP-Completo 3SAT, demostrando que si este problema puede ser reducido a una instancia del problema de packs, cualquier problema NP-Hard podría ser reducido a éste(sea o no a través del 3SAT). En primer lugar asignaríamos las variables del problema 3SAT y las utilizaríamos como parte de una regulación, duplicando la última de cada clausula para alcanzar los 4 packs en cada regulación. Esto se podría realizar en tiempo polinomial ya que solo tendríamos que recorrer linealmente cada clausula. Una vez obtenido el resultado, se informará si se puede o no combinar los packs de forma que se cumplan las regulaciones, es decir, no tendríamos que transformar esa respuesta ya que es la solución del problema 3SAT.

```
Transformación(clausulas)
    Regulaciones = Crear lista vacía
    Para cada clausula: #O(n)
        Regulaciones -> Agregar(clausula[0], clausula[1], clausula[2], clausula[2])
    Devolver Regulaciones

Resolver Problema 3-SAT(clausulas)
    Regulaciones = Transformación(clausulas)
```

```
Se Puede = Resolver Problema Yate(Regulaciones)
Devolver Se Puede
```

2.3. Análisis de la transformación

La transformación contiene un bucle que recorre un vector de 3-tuplas que representan las clausulas del problema 3-SAT, donde cada valor de dicha tupla representa uno de los operandos de una clausula. Para n tuplas lo realizará en $O(n)$. Esta será la complejidad temporal final, por lo que podemos afirmar que es polinómica. En el caso de la complejidad espacial se utilizará una lista de regulaciones, representadas como 4-tuplas. Esta lista contendrá una complejidad de $O(n)$, también polinomial, siendo lineal con respecto a las clausulas del problema original. Siendo ésta última la complejidad espacial final de la transformación.

2.4. Prueba de que con 2 packs es un problema P

2.4.1. Posible solución del problema

Con 2 packs por regulación el problema se podría representar en un grafo cuyos vértices corresponden a variables y sus negaciones, y los ejes corresponden a conexiones entre variables. A cada clausula $(p1 \vee p2)$ corresponderán dos pares de vértices $(\neg p1 \Rightarrow p2)$ y $(\neg p2 \Rightarrow p1)$. Esto significa que si p1 no es utilizado, p2 tendrá que ser utilizado, y viceversa.

Por ejemplo, dadas las restricciones:

$(p1 \vee \neg p2) \wedge (p3 \vee \neg p2) \wedge (\neg p1 \vee p3) \wedge (p4 \vee \neg p1) \wedge (p3 \vee p4)$

El grafo representado dichas restricciones quedaría:

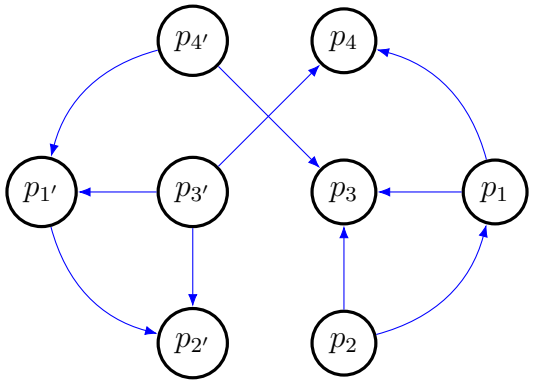

Figura 2.1: Grafo generado desde la expresión lógica

La estructura del grafo determinará si la elección de packs será posible de asignar o no. Para que sea posible una variable p_i y su negación $\neg p_i$ no deberán pertenecer al mismo componente fuertemente conexo. Si se diera que pertenezcan a un mismo componente, es decir hay un camino

de p_i hacia $\neg p_i$ y viceversa, ambos deberían ser verdaderos(usarse y no usarse) y por lo tanto no podría ser posible.

La posible solución del uso de una selección de packs que respete todas las regulaciones estará determinada por un recorrido topológico en sentido inverso del grafo de condensación, el algoritmo utilizado será el **Tarjan's strongly connected components algorithm**[3]. El mismo devolverá un grafo en el que cada vértice representará una componente fuertemente conexa. A partir de eso, cada componente(representada por un nodo) se recorrerá en el orden topológico inverso y se irá eligiendo las variables de acuerdo al nodo que actualmente se encuentre siendo recorrido solo si esa variable ya no ha sido asignada antes.

Por ejemplo, en este caso el orden topológico podría haber dado como solución(no hay componentes de más de 1 vértice):

$P2\|P1\|\neg P4\|P3\|\neg P3\|\neg P1\|\neg P2\|P4$

Por lo que en sentido inverso se hubiese asignado $\{P4, \neg P2, \neg P1, \neg P3\}$ Habiendo encontrado una posible solución.

2.5. Prueba de que la transformación del problema de 2 packs es polinomial

El grafo utilizado se representará como una lista de aristas cuya complejidad espacial será $O(E)$. Para crear el mismo se utilizará el siguiente algoritmo:

```
Negar(variable)
    Devuelve ¬variable

Crear Grafo(regulaciones)
    Crear Grafo Vacío #O(1)
    Para cada regulación:
        Agregar Arista(negar(regulación[0]), regulación[1])#O(1)
        Agregar Arista(regulación[0], negar(regulación[1]))#O(1)
    Devolver Grafo
```

Agregar al final de la lista de aristas se realizará en $O(1)$, por lo que la creación del grafo tendrá una complejidad temporal de $O(n)$ por el bucle que recorre n regulaciones. La complejidad espacial será $O(E) = O(2n) = O(n)$ ya que contendrá 2 aristas por cada regulación.

2.6. Análisis de la complejidad de la solución

Para saber la cantidad de vértices se tendrá en cuenta el peor de los casos, donde para cada regulación habrá 2 variables nuevas y sus negadas, siendo entonces la cantidad máxima posible de vértices $(4n)$. La complejidad resultante de resolver este problema viene dada por la creación del grafo, la verificación de la condición, el algoritmo de Tarjan y el recorrido del grafo de componentes. La complejidad de cada cálculo es:

- Creación del grafo: $O(E) = O(n)$
- Verificación de la condición: $O(V^2 + V \cdot E) = O(16n^2 + 4n \cdot n) = O(n^2)$
- Algoritmo de Tarjan: $O(V + E) = O(4n + n) = O(n)$
- Recorrido del grafo de componentes:$O(V + E) = O(4n + n) = O(n)$

Por lo tanto la complejidad resultante es $O(2V + V^2 + V \cdot E + 3E) = O(V(E + V)) = O(4n(n + 4n)) = O(n^2)$, la cual es polinómica.

Parte 3: Un poco de teoría

3.1. Reducción polinomial

Dado una instancia 'y' de un problema Y, realizamos una transformación a una instancia "x" del problema X, resolvemos "x"mediante un proceso B obteniendo el resultado solución(x) que transformaremos en solución(y), ese proceso es llamado reducción polinomial, es decir que un problema puede ser reducido polinomialmente a otro. Se concluye que el segundo es al menos tan difícil (misma complejidad temporal para resolverlo) que el primero. Es por ello que las reducciones se pueden utilizar para probar que un problema es $NP - HARD$ reduciendo un problema que previamente se conocía como $NP - C$ a una instancia del problema a demostrar.

3.2. Relación entre los problemas P y NP

La relación entre los problemas P y NP es un problema sin resolver dentro de la ciencia de la computación. Este problema esta dado por la pregunta ¿$P = NP$?, es decir si un problema puede certificarse en tiempo polinómico también puede resolverse en tiempo polinómico en todos los casos.

En esta teoría de relación, la clase P consiste en los problemas de decisión que pueden ser resueltos en un periodo de tiempo polinomial en relación con los datos de entrada, la clase NP consiste en aquellos problemas cuyas soluciones se verifican en tiempo polinomial. Entonces P es una clase que es resoluble en tiempo polinomial y NP certificable en tiempo polinomial.

Muchos descartaron la idea de que sean iguales, ya que seria ilógico que la resolución de un algoritmo tenga la misma complejidad (en términos de polinomial o no) que la de su verificación para todos los casos. De todas maneras, se sabe que los problemas P están estrictamente incluidos en los problemas NP.

3.3. Algunos casos de aplicación

Tenemos un problema A, un problema B y una caja negra N_A y N_B que resuelven el problema A y B respectivamente. Sabiendo que B es P

1. ¿Qué podemos decir de A si utilizamos N_A para resolver el problema B? (asumimos que la reducción realizada para adaptar el problema B al problema A es polinomial)

   Se puede decir que $B \leq_p A$. Esto quiere decir que A es al menos igual de difícil que B, es decir, el problema A tiene una complejidad mayor o igual a la complejidad que resulta de resolver B. Con esto se puede concluir que B estará incluido en el grupo donde este incluido A, en este caso no sabemos cual es y tampoco podemos concluir nada sobre A ya que podría ser tranquilamente un problema por fuera del conjunto NP (Como por ejemplo el ajedrez).

2. ¿Qué podemos decir de A si utilizamos N_B para resolver el problema A? (asumimos que la reducción realizada para adaptar el problema A al problema B es polinomial)

   Esto nos dice que B es al menos tan difícil como A, es decir, $A \leq_p B$. Como sabemos que B es P, entonces se puede decir que A también será P.

3. ¿Qué pasa con los puntos anteriores si no conocemos la complejidad de B, pero sabemos que A es NP-C?

   En el primer caso habríamos demostrado que B tiene una complejidad menor o igual a la de A, es decir, es a lo sumo NP-Hard. $B \leq_p A$

   En el segundo caso podríamos haber dicho que B tiene una complejidad mayor o igual a la complejidad de A, es decir que B es al menos tan difícil como A y por lo tanto será al menos NP-Hard. $A \leq_p B$.

Bibliografía

[1] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms (tercera edición), MIT Press (2009).

Capítulo 34, **NP-Completeness**

[2] The easiest hard problem

Brian Hayes

http://bit-player.org/wp-content/extras/bph-publications/AmSci-2002-03-Hayes-NPP.pdf

[3] On finding the strongly connected components in a directed graph

https://www.sciencedirect.com/science/article/abs/pii/0020019094900477?via%3Dihub

[4] The complexity of theorem-proving procedures

https://dl.acm.org/doi/10.1145/800157.805047