75.29 - Teoría de Algoritmos I: Trabajo Práctico n. 2

Equipo Q:
Lavandeira, Lucas (#98042)

Rozanec, Matias (#97404)
rozanecm@gmail.com

Sbruzzi, José (#97452)

14.mayo.2018

Facultad de Ingeniería, Universidad de Buenos Aires

Índice

Parte I
Resolución

1. Parte 1: Spy vs Spy

2. Parte 2

2.1. Ejercicio 1

El problema a resolver es uno de detección de rotaciones cíclicas en dos strings (cadenas de caracteres). Se pide resolverlo usando distintos algoritmos y comparar sus órdenes de tiempo de ejecución.

La primera solución a implementar es por fuerza bruta. Si asumimos que la comparación de strings es de orden lineal, ir haciendo la rotación de uno de los dos strings y comparando hasta llegar a una igualdad da como resultado un algoritmo de orden cuadrático en el peor caso, cuando se prueban todas las rotaciones y ninguna comparación da verdadera.

La segunda solución es implementando el algoritmo KMP de *string matching*. El algoritmo se puede resumir en una forma optimizada de decidir si una palabra *word* está contenida dentro de un texto fuente *text* en orden lineal, realizando un precómputo de una tabla de valores que asisten al algoritmo durante la ejecución. El caso de uso común de KMP es para ubicar una única palabra en un texto grande, mientras que nuestro problema es diferente, queremos decir si un string es una rotación cíclica de otro. Lo que hacemos es ejecutar el algoritmo bajo un ciclo *for*, rotando uno de los strings entre cada iteración. El algoritmo KMP es de orden lineal de la longitud del texto, que para nuestro caso es el mismo que el de la palabra.

Esta aplicación se puede ver como una *reducción* de un problema a otro. Estamos reduciendo el problema de detección de rotaciones cíclicas al de string matching de KMP, usándolo como una çaja negra"para ayudarnos a resolver el original. El ciclo que ejecuta esta caja negra es en el peor caso de N iteraciones, para poder comprobar que ninguna rotación es igual al texto fuente. El orden de complejidad resultante es, entonces, cuadrático.

Este uso de KMP es lejos de ser óptimo, no obtuvimos una complejidad mejor a la fuerza bruta, y en ejecuciones reales los cálculos adicionales que hace KMP para lograr un orden lineal realentizan la ejecución suficiente para que sea un orden entero más lento que una implementación por fuerza bruta. Una mejora sustancial al algoritmo es ejecutar KMP tomando como palabra a ubicar el texto rotado, y como fuente **una copia duplicada** del texto original, es decir si el texto es `ABC`, tomaremos como fuente `ABCABC`. Es fácil de ver que cualquier rotación posible del texto original está completamente contenida dentro de el texto duplicado, tendrá una parte dentro de la primera mitad, y el resto contiguo en la segunda. Con esta estrategia, una única ejecución de KMP es suficiente para poder decidir si un string es rotación de la otra.

Esta estrategia también es una reducción del problema a uno de string matching resolvible por KMP. Aquí, se debe aplicar una transformación de los datos de entrada para que .[en]cajeçon el uso que queremos darle a KMP, a diferencia del caso anterior que se pasaban el string original y la potencial rotación como palabra y texto de KMP. El orden mejora sustancialmente al solo ser una única ejecución de KMP, es lineal al ser la duplicación del string fuente también lineal, al costo de duplicar el espacio de memoria utilizado (aunque el orden espacial tampoco cambie, sigue siendo lineal sobre el tamaño del texto).

2.2. Ejercicio 2

Se demostrará a continuación que el problema es difícil.

Se mostrará que *Ciclo Hamiltoniano* $\leq_p$ *TSP*.

Dado un grafo dirigido $G = (V, E)$, se define la siguiente instancia del TSP. Se tiene una ciudad v'_i para cada nodo v_i del grafo G. Se define $d(v'_i, v'_j)$ igual a 2 de haber una arista (v_i, v_j) en G, y

se define igual a 1 en caso contrario.

G tiene un ciclo Hamiltoniano si y solo si hay un tour de longitud como mínimo $2n$ en el TSP estudiado. Si G tiene un ciclo Hamiltoniano, entonces este ordenamiento de las correspondientes ciudades define un tour de longitud $2n$. A la inversa, suponga que hay un tour de como mínimo $2n$. La expresión para la longitud de este tour es la suma de n términos, cada uno de los cuales es como máximo 2, por lo que debe ser el caso que todos los términos son iguales a 2. Por lo tanto cada par de nodos en G que corresponde a ciudades consecutivas en el tour debe estar conectado por una aristañ se sigue que el ordenamiento de esos correspondientes nodos debe formar un ciclo Hamiltoniano.

2.3. Ejercicio 3

Como primer paso, demostraremos que, dada una cantidad n de cursos, si llamamos m a la cantidad máxima de cursos superpuestos en cualquier instante de tiempo, la cantidad de aulas necesarias para realizar la asignación es m.

Demostración

Para demostrar esto, se plantearán las cotas superior e inferior, y se demostrará que ambas coinciden, siendo el valor de ambas: m.

Cota inferior

De haber m cursos que se dictan al mismo tiempo, como no se pueden dictar cursos distintos en una misma aula en simultáneo, queda probado que como mínimo debe haber tantas aulas como cursos se dictan en simultáneo. Por lo tanto, es válido afirmar que

$$Aulas\ necesarias \geq m$$

Cota superior

Para todo instante t de tiempo habrá, según lo demostrado anteriormente, como mínimo m aulas disponibles. Se puede comprobar rápidamente que para ningún t se necesitarán más que m aulas, dado que si dos cursos no se superponen temporalmente, no hay razón por la que no puedan compartir una misma aula. Además se está tratando el caso en que todas las aulas son iguales, lo que evita complicaciones más allá del análisis presentado.

Llegamos entonces a que

$$Aulas\ necesarias \leq m$$

Queda así demostrado que $Aulas\ necesarias = m$

2.3.1. Pseudocódigo

Algorithm 1: Pseudo código que resuelve el problema.

Data: Horarios de inicio y finalización de cada uno de los n cursos: $T_{inicio,j}$ y $T_{fin,j}$ denotan los tiempos de inicio y finalización del curso $j-$ésimo. Conjunto de todos los tiempos: $T_i, i \in (1, 2n)$

Result: Menor cantidad de aulas necesarias para acomodar todos los cursos suponiendo que todas las aulas son iguales.

```
min time slice ← ∞;
min time ← ∞;
max time ← 0;
foreach T_i do
    if (current min slice = |T_i − T_j|) < min_aulas, i ≠ j then
        min time slice := current min;
    if T_i < min time then
        min time := T_i;
    if T_i > max time then
        max time := T_i;
current time := min time;
max superpositions := 0;
while min time < max time do
    current num of superpositions := 0;
    foreach T_inicio,j, T_fin,j do
        if T_inicio,j ≥ current time slice AND > T_fin,j then
            ++current num of superpositions;
    if current num of superpositions > max superpositions then
        max superpositions := current num of superpositions;
    current time += min time slice;
```

2.3.2. Reducción a problema de coloreo de grafos

Considérese cada una de las materias como un nodo. Dados dos nodos, se establece una conexión entre ellos mediante una arista únicamente en caso de superposición horaria entre los cursos que representan dichos nodos. De esta forma, si se encuentra la forma de pintar todos los vértices de forma que no queden dos adyacentes con el mismo color, utilizando la cantidad mínima de colores, se habrá resuelto el problema propuesto.

2.4. ¿P=NP?

No.