Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Auxiliar 10 - "Dominios Discretos, Algoritmos Online y Aproximados"

Profesor: Pablo Barceló
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

9 de Junio del 2017

P1. Autómatas!

a) Dado un patrón P de largo m y un texto T de largo n, ambos sobre el alfabeto binario, podemos encontrar una ocurrencia de P en T en tiempo $\mathcal{O}(n+m)$. Considere ahora que P puede contener *comodines*. Un caracter *comodin* puede calzar con una cadena de largo arbitrario (incluyendo cero). Resuelva este problema en tiempo $\mathcal{O}(n+m)$.

b) Dada una expresión regular R sobre un alfabeto finito Σ construya su autómata $\mathcal{A}(R)$ equivalente en tiempo $\mathcal{O}(|R|)$.

P2. k servidores, online

Considere el escenario donde tiene k puntos (*servidores*) en un espacio *métrico* y una secuencia de puntos (*peticiones*) que debe atender. Cada vez que llega una petición, un servidor debe moverse hacia esa posición para atenderla.
El problema *online* consiste en minimizar la distancia recorrida por todos los servidores luego de n peticiones, sin saber la secuencia de puntos a atender.

a) Muestre que para cualquier algoritmo su radio competitivo es al menos k.

b) Consideremos ahora el problema de los k servidores y sus peticiones en una línea. Muestre que el algoritmo de enviar al servidor más cercano no es competitivo.

P3. Problema del Vendedor Viajero

De una 3/2-aproximación para el problema del vendedor viajero *métrico*.
Hint: Puede considerar que encontrar el "Emparejamiento Perfecto" de costo mínimo es polinomial.

Soluciones

P1. a) La clave del problema consiste en notar que solo se nos pide encontrar una ocurrencia cualquiera del patrón en T (no todas ellas). Para esto dividiremos P en partes delimitadas por los comodines $*$ que contiene,

$$P = P_1 * P_2 * \ldots * P_r$$

, luego podemos obtener el autómata del algoritmo de búsqueda en texto[1] para cada uno de estos P_i, todo esto en $\mathcal{O}(|P_1| + |P_2| + \ldots + |P_r|) = \mathcal{O}(|P|) = \mathcal{O}(m)$.
Finalmente construimos el autómata de búsqueda de P "fusionando" el estado final de P_i con el inicial de P_{i+1} (conservando las transiciones del inicial). Con esto tenemos un autómata que busca P_1, luego P_2, ... y finalmente P_r y por lo tanto encontrando la primera aparición de P en T. Como el algoritmo consiste en correr el autómata, toma $\mathcal{O}(n)$ teniendo un coste total de $\mathcal{O}(n + m)$.

b) La construcción de Thompson que se puede encontrar en la sección 2.4 del apunte https://www.dcc.uchile.cl/∼gnavarro/apunte.pdf, da un algoritmo recursivo que transforma una expresión regular en un autómata $\mathcal{A}(R)$ equivalente. Como la recursión en cada paso se deshace de un caracter, el número de llamados recursivos es $\mathcal{O}(|R|)$, finalmente como el trabajo no recursivo realizado por la función es $\mathcal{O}(1)$ el algoritmo es de tiempo $\mathcal{O}(|R|)$.

P2. a) Sea k y A un algoritmo que resuelve el problema de los servidores. Considere el siguiente input: un espacio métrico con $k+1$ puntos (uno más que el número de servidores) y los servidores parten ubicados en los primeros k. Las peticiones $\sigma = \sigma_1 \ldots \sigma_r$ serán aquellos puntos que A no tiene cubierto en ese momento 😈(por palomar siempre existe uno de estos puntos) empezando por $k+1$.

Con esto construiremos k algoritmos que lo hacen "mejor" que A, $B_1, \ldots, B_k$. De modo que :

- Antes de la primera petición B_i manda al servidor que está en i a $k+1$.
- Queremos mantener el invariante que en cada petición todos los B_i tienen cubierto a σ_i.
- Para lograr lo anterior, cuando se pida σ_{i-1} y A lo cubra con un servidor que se encontraba en σ_i (esto es así pues las peticiones fueron construidas de esta forma), existirá un único[2] B_j que no tiene cubierto σ_i y este (luego de responder la petición $i-1$) lo cubrirá con el servidor que tiene en σ_{i-1}.

[1] Con el algoritmo de construcción lineal pedido en la tarea 2.
[2] Notar que este es otro invariante que mantienen los algoritmos

- Con esto tenemos que la suma de los costos de los B_i corresponden a los movimientos que se hacen antes de la primera petición, más los movimientos que hace exactamente uno de los B_i luego de cada petición. Sin embargo, este último costo es el mismo que realiza A (cuando A mueve uno de sus servidores de σ_i a σ_{i-1} alguno de los B_i ya lo hizo desde σ_{i-1} a σ_i). Es decir,

$$\begin{aligned}\sum_{i=1}^{k} C_{B_i}(\sigma) &= C_A + \sum_{i=1}^{k} dist(i,k+1)\\ &\leq C_A + k\max_i(dist(i,k+1))\end{aligned}$$

Finalmente debe existir uno de estos algoritmos que cumpla:

$$OPT \leq C_{B_j}(\sigma) \leq \frac{1}{k}C_A + \max_i dist(i,k+1)$$

,por lo que A es al menos k- competitivo.

b) Consideremos el input con los puntos de la recta numérica, los servidores inicialmente ubicados en 2 y 4 y la secuencia de peticiones $10, 1, 2, 1, 2, 1, 2, \ldots, 1, 2$ (repitiendo $1, 2$ n veces). El algoritmo para la primera petición no se moverá y luego para las peticiones siguientes estará moviendo el servidor de la izquierda entre 1 y 2 teniendo un costo total de $2n$. Sin embargo el óptimo consiste en responder la primera petición y luego mover el servidor de la izquierda a 1 a costo total 3. Finalmente, como n es un valor arbitrario, el algoritmo puede ser tan malo (respecto al óptimo) como queramos, por lo que no es competitivo.

P3. En clases vimos que se puede obtener una 2-aproximación del problema encontrando un árbol generador de costo mínimo y luego haciendo un recorrido DFS (limpiando los vértices repetidos en el recorrido) en el árbol. Para alcanzar el 3/2 encontraremos un "mejor" camino a partir del árbol generador de costo mínimo T^*.

Notemos que en T^* hay un número par de vértices de grado impar (esto se cumple en todo grafo, por el lema del apretón de manos 🤓). Queremos agregarle exactamente una arista a todos estos nodos de manera que después de esto todos los vértices sean de grado par.
Lo anterior lo podemos hacer encontrando un "Emparejamiento Perfecto"[3] de costo mínimo sobre estos nodos, llamémoslo M^*.

Entonces en $T^* \cup M^*$ todos los vértices son de grado par y por lo tanto (🤓) tiene un circuito euleriano (es aquel que pasa por todas las aristas) C', finalmente nuestra aproximación será el ciclo C^* obtenido luego de eliminar las repeticiones de C'. Luego:

$$costo(C^*) \leq costo(C') \leq costo(T^*) + costo(M^*)$$

[3]Conjunto de aristas que cubre todos los vértices considerados y que son disjuntas entre ellas.

Por un lado tenemos que $costo(T^*) \leq costo(T) \leq OPT$, siendo T algún árbol generador, puesto que podemos sacarle una arista al ciclo óptimo y formar con esto un árbol generador.

Por otro lado, consideremos un ciclo hamiltoniano óptimo de costo OPT' sobre los vértices de grado impar de T^*, de este ciclo, se pueden extraer dos "Emparejamientos perfectos" de estos vértices M_1 y M_2, pero como M^* es de costo mínimo tenemos que $costo(M^*) \leq \frac{costo(M_1)+costo(M_2)}{2} = \frac{OPT'}{2} \leq \frac{OPT}{2}$ (donde la última desigualdad es válida gracias a la desigualdad triangular).

Juntando estas observaciones obtenemos que:

$$costo(C^*) \leq \frac{3}{2}OPT$$