TP1 - Algoritmos y Estructuras de Datos III

Catalina Gonzalo Juarros

2017-08-23

Índice

1. Problema a resolver

1.1. Descripción

Dado un conjunto de i agentes, queremos determinar la **mayor cantidad de agentes confiables** en base a una secuencia de a preguntas respondidas por ellos. Cuando un agente responde una pregunta, dice si otro agente -que puede ser él mismo- es confiable. Para cierto subconjunto de agentes, decimos que todos son confiables si y sólo si:

- Ningún agente del conjunto dice que algún agente del conjunto no es confiable: si Ricardo dice que Rubén no es confiable pero tanto Ricardo como Rubén están en el conjunto, este no es una solución válida.
- Ningún agente del conjunto dice que un agente que está fuera del conjunto es confiable: si Ricardo está en el conjunto y dice que Rubén es confiable, obligatoriamente debemos agregar a Rubén.

Cada agente se caracteriza por un número $1 \leq n \leq i$ y cada pregunta respondida se representa con un par $(x, y) : 1 \leq x, y \leq i$, donde x es el agente que respondió la pregunta, y es el agente sobre el que x respondió y el signo de y indica si x dijo que y es o no confiable (positivo es sí, negativo es no). Por ejemplo, el par $(1, 2)$ se lee como "1 dijo que 2 es confiable". Siempre hay al menos un agente, pero puede no haber preguntas respondidas. Puede deducirse que en ese caso todos los agentes son confiables.

Llamaremos a un conjunto de agentes del mayor tamaño posible una **solución óptima**. En la sección que sigue veremos ejemplos claros de soluciones óptimas y casos en los que hay más de una.

1.2. Ejemplos

1.2.1. Caso 1

Analicemos las soluciones cuando tenemos 4 agentes y la secuencia de preguntas respondidas es $E =< (1, 2), (1, -4), (2, -3), (3, 1), (3, -4) >$:

- $< 1, 2 >$ es solución, ya que 1 dice que 2 es confiable. Observemos que $< 1 >$, entonces, no podría ser una solución. Observemos también que no podemos extender nuestra solución, ya que 1 dice que 4 no es confiable y 2 dice que 3 no es confiable.
- $< 2, 4 >$ también es solución, porque a pesar de que 2 no dijo nada sobre 4, este subconjunto no rompe ninguna de las dos condiciones necesarias para ser una solución válida. Tampoco podemos extenderla, por la misma razón que la anterior.
- $< 2 >$ y $< 4 >$ son soluciones, pero obviamente no son óptimas pues ya encontramos soluciones de 2 agentes.

Entonces, concluimos que la máxima cantidad de agentes confiables es 2. En este ejemplo se ve claramente que la solución óptima **no necesariamente es única**.

1.2.2. Caso 2

Veamos ahora qué ocurre cuando tenemos un solo agente y la secuencia es $E =< (1, -1) >$:

- Observemos que una solución válida **nunca** puede contener a 1, puesto que él mismo se considera no confiable.
- Pero 1 es el único agente que tenemos, por lo que la única solución válida es el conjunto vacío.

En este caso hay una sola solución óptima y la máxima cantidad de agentes es 0.

2. Resolución

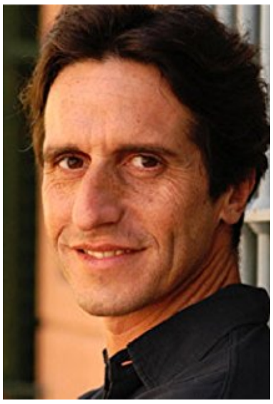

Figura 1: Diego Peretti.

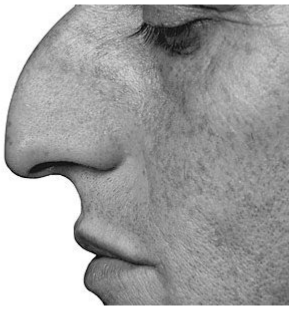

Figura 2: La nariz de Diego Peretti.

2.1. Idea

2.2. Pseudocódigo

3. Complejidad

3.1. Caracterización del peor caso

El algoritmo, como vimos en la sección 2, consiste en probar subconjuntos de agentes hasta encontrar la máxima cantidad de informantes que pueden agregarse a la solución sin que uno contradiga a otro. Como es requisito que el arreglo que representa a cada subconjunto esté ordenado, sólo vamos a probar con **una** representación de cada subconjunto, por lo que la cantidad de soluciones posibles se corresponde con la cantidad de subconjuntos distintos de $\{1, ..., i\}$ (es decir, el *cardinal del conjunto de partes* de $\{1, ..., i\}$). Este número es 2^i. La justificación la voy a escribir cuando aprenda a hacer footnotes.
En el peor caso, el algoritmo tiene que probar **todos** los subconjuntos, o sea 2^i soluciones candidatas. Lo voy a justificar cuando efectivamente haya hecho el algoritmo.

3.2. Cálculo de complejidad

La complejidad de este algoritmo, en el peor caso, es

$$T(n) \in \mathcal{O}(2^i \times i^2 \times \log i \times a)$$

Justificación Dado que el algoritmo debe probar

4. Código fuente

5. Experimentación