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

2.1. Idea

2.1.1. Algoritmo

El algoritmo propuesto se basa en la técnica de *backtracking*, que consiste básicamente en:

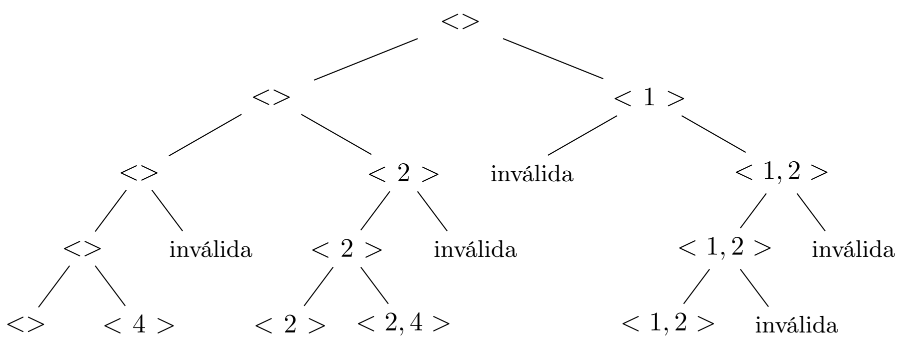
- Construir una *solución parcial* que pueda extenderse a **cualquier** solución candidata del problema en un número finito de pasos. Por ejemplo, si se intentara resolver un Sudoku, la solución parcial inicial podría ser “dejar el tablero vacío”, y una extensión consistiría en llenar un casillero más. En el problema presentado en este trabajo, la solución parcial inicial es el conjunto vacío, que se extenderá a subconjuntos del conjunto total de agentes. [1]

- Recorrer **todo** el espacio de soluciones válidas o que podrían extenderse a soluciones válidas, a menudo mediante llamadas recursivas al mismo algoritmo. A partir de este esquema, puede modelarse el espacio de soluciones como un **árbol** donde cada nodo representa una solución parcial: la raíz es la solución inicial y los hijos de cada nodo son las soluciones que pueden construirse directamente a partir de él. En este caso, el árbol es **binario**, la raíz es el conjunto vacío y, para cada ($1 \leq j \leq i$), el subárbol izquierdo de un nodo de nivel $j-1$ representa los conjuntos que **no contienen al agente** $\boldsymbol{j}$ y el derecho representa a los que **sí**. Por lo tanto, dado un nodo de nivel $j-1$ con una solución S, sus hijos representan decisiones a tomar: agregar a j a S y no agregar a j a S.

- Elegir la solución óptima entre las recorridas.

Es importante notar que un árbol de backtracking **no debe contener** soluciones candidatas que no puedan extenderse a soluciones válidas. En el ejemplo del apartado 1, donde el conjunto de preguntas es $E =< (1,2), (1,-4), (2,-3), (3,1), (3,-4) >$, $< 1 >$ puede estar en el árbol, puesto que es posible extenderla a la solución válida $< 1, 2 >$, pero $< 1, 3 >$ no, ya que no puede agregarse ningún agente que la haga válida.

El árbol de backtracking para este caso puede visualizarse así:



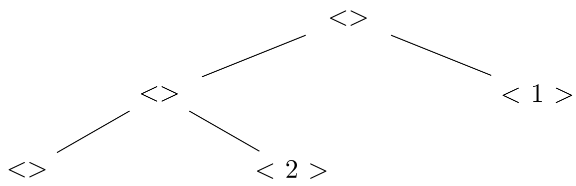
Lo recorremos de izquierda a derecha: desde la raíz, bajamos por la izquierda hasta llegar a la hoja $<>$, guardamos el valor 0 como la mayor cantidad de agentes encontrada y subimos al nivel 4. Como quedan más hojas por revisar, miramos la hoja derecha $< 4 >$, y como su longitud es mayor a 0, actualizamos el máximo valor encontrado. Ahora, como ya revisamos todas las hojas del nodo $<>$ en el nivel 4, subimos al 3 y vemos si hay un subárbol derecho por explorar; no hay, y por lo tanto volvemos al 2. Repetimos esta lógica hasta llegar a la raíz y hacemos lo mismo con el subárbol derecho. Entonces habremos chequeado todas las soluciones y tendremos la óptima. Esta idea de **subir al nodo padre una vez que exploramos los subárboles** es que lo que le da su nombre al *backtracking* (en inglés, “seguir un mismo camino hacia atrás”).

[1] Observación: una solución parcial **no es** necesariamente una solución al problema. En el ejemplo del Sudoku, el tablero vacío no es una solución válida, pero es posible llegar a una a partir de él mediante operaciones sencillas como llenar un casillero. Por el contrario, en el problema de los agentes, la solución parcial inicial **siempre** es válida, aunque en muchos casos no será la óptima.

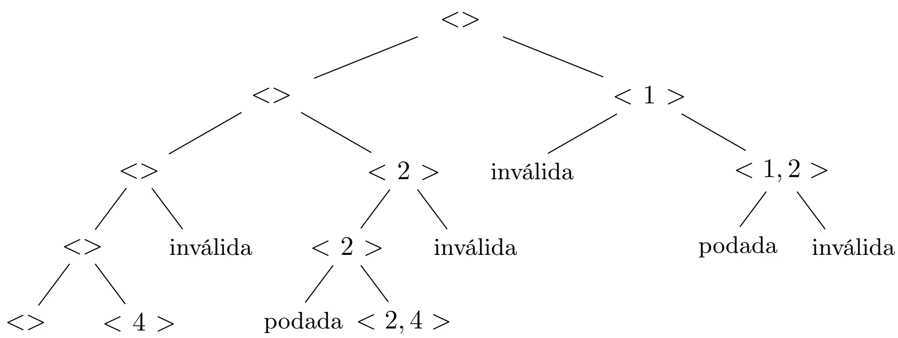
Además de estructurar el espacio de soluciones, el modelado de éste como un árbol nos da una gran ventaja: es posible definir *podas* que nos ayudan a recorrerlo más inteligentemente y, en muchos casos, ahorrar llamadas recursivas y mejorar el rendimiento del algoritmo. Podemos imaginar una poda como “cortar” un determinado subárbol del espacio de soluciones cuando sabemos que no contiene **ninguna solución mejor que la encontrada hasta el momento**, por más que todos sus nodos sean soluciones parciales válidas.

2.1.2. Poda 1

Supongamos que tenemos 4 agentes y el conjunto de encuestas es $E =< (1, 2), (1, -3) >$. El árbol de backtracking tiene este aspecto:



2.1.3. Poda 2



2.2. Pseudocódigo

2.3. Correctitud

Veamos que el algoritmo presentado en la sección anterior efectivamente revisa **todo** el espacio de soluciones y se queda con la de mayor valor.

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