Algoritmia y Complejidad

Ejercicio 24 de Sipser

Juan Carlos Alcausa Luque

Universidad de Málaga

Índice

1. Enunciado

Una 2cnf-fórmula es un AND de cláusulas, donde cada cláusula es un OR de a lo sumo dos literales. Dado $2SAT = \{\langle\phi\rangle \mid \phi \text{ es una 2cnf-fórmula}\}$ se pide demostrar que $2SAT \in P$.

2. Satisfacibilidad de 2-CNF

La cláusula $(x \vee y)$ es lógicamente equivalente a cada una de las expresiones $(\neg x \rightarrow y)$ y $(\neg y \rightarrow x)$. Representamos la fórmula 2-CNF ϕ sobre las variables $x_1, \ldots, x_m$ mediante un grafo dirigido G con $2m$ nodos etiquetados con los literales sobre estas variables. Para cada cláusula en ϕ, colocamos dos aristas en el grafo correspondientes a las dos implicaciones mencionadas. Adicionalmente, colocamos una arista de $\neg x$ a x si (x) es una cláusula unitaria y la arista inversa si $(\neg x)$ es una cláusula.

Teorema 1. *ϕ es satisfacible si y solo si G no contiene un ciclo que contenga tanto x_i como $\neg x_i$ para algún i.*

Llamaremos a tal ciclo un *ciclo de inconsistencia*. Verificar si G contiene un ciclo de inconsistencia se puede realizar fácilmente en tiempo polinómico aplicando el algoritmo de Kosaraju y comprobando que cada variable y su complementaria no pertenecen a la misma SCC (*Componente Fuertemente Conectada*).

Demostración. Demostramos que G contiene un ciclo de inconsistencia si y solo si no existe una asignación satisfactoria.

Comenzamos suponiendo, por reducción al absurdo, que G contiene un ciclo de inconsistencia. Como la secuencia de implicaciones en cualquier ciclo de inconsistencia produce la equivalencia lógica $x_i \leftrightarrow \neg x_i$ para algún i, llegamos a contradicción y por tanto, si G contiene un ciclo de inconsistencia, ϕ debe ser insatisfacible.

A continuación, demostramos el recíproco. Escribimos $x \xrightarrow{*} y$ si G contiene un camino del nodo x al nodo y. Debido a que G contiene las dos aristas designadas para cada cláusula en ϕ, tenemos $x \xrightarrow{*} y$ si y solo si $\neg y \xrightarrow{*} \neg x$.

Si G no contiene un ciclo de inconsistencia, construimos una asignación satisfactoria para ϕ de la siguiente manera:

1. Escogemos cualquier variable x_i. No podemos tener tanto $x_i \xrightarrow{*} \neg x_i$ como $\neg x_i \xrightarrow{*} x_i$ porque G no contiene un ciclo de inconsistencia.

2. Seleccionamos el literal x_i si $x_i \xrightarrow{*} \neg x_i$ es falso, y en otro caso seleccionamos $\neg x_i$.

3. Asignamos el literal seleccionado y todos los literales implicados (aquellos alcanzables a lo largo de caminos desde el nodo seleccionado) como Verdadero.

4. Nótese que nunca asignamos tanto x_j como $\neg x_j$ a Verdadero porque si $x_i \xrightarrow{*} x_j$ y $x_i \xrightarrow{*} \neg x_j$ entonces $\neg x_j \xrightarrow{*} \neg x_i$ y por lo tanto $x_i \xrightarrow{*} \neg x_i$, y no habríamos seleccionado el literal x_i (similarmente para $\neg x_i$).

5. Luego, eliminamos todos los nodos etiquetados con literales asignados o sus complementos de G, y repetimos este procedimiento hasta que todas las variables sean asignadas.

La asignación resultante satisface cada implicación y por lo tanto cada cláusula en ϕ. Así, ϕ es satisfacible.

□

3. Algoritmo de Kosaraju

El algoritmo de Kosaraju permite encontrar las **componentes fuertemente conexas (SCC)** de un grafo dirigido $G = (V, E)$. Una componente fuertemente conexa es un conjunto de vértices tal que para cualquier par de vértices u y v dentro del conjunto, existe un camino de u a v y de v a u.

Descripción del algoritmo de Kosaraju

El algoritmo de Kosaraju permite encontrar las componentes fuertemente conexas (SCCs) de un grafo dirigido en tiempo polinómico. Su funcionamiento se basa en dos recorridos en profundidad (DFS), con un paso intermedio en el que se invierten todas las aristas del grafo. A continuación, se describen los pasos:

1. **Primera búsqueda en profundidad (DFS)**: Se realiza un recorrido en profundidad sobre el grafo original. A medida que se completan las llamadas recursivas de cada nodo, se almacenan los nodos en una pila (o lista) en el orden en que finaliza su recorrido. Este orden representa un *orden de salida* o *postorden inverso* del grafo.

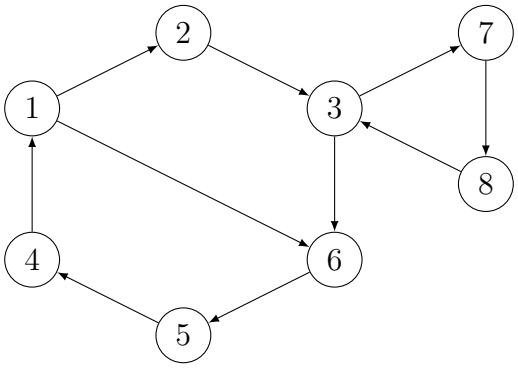
2. **Inversión de aristas (grafo transpuesto)**: Se construye el grafo transpuesto G^T, es decir, se invierte la dirección de todas las aristas del grafo original. Si había una arista de u a v, ahora habrá una arista de v a u.

3. **Segunda búsqueda en profundidad (DFS en el grafo transpuesto)**: Se procesan los nodos en el orden de salida obtenido en el primer DFS (es decir, desde el último finalizado hasta el primero), realizando una nueva búsqueda en profundidad en el grafo transpuesto. Cada DFS completo en este paso identifica una componente fuertemente conexa del grafo original.

El algoritmo es eficiente ya que cada paso tiene una complejidad lineal respecto al número de vértices y aristas, es decir, $\mathcal{O}(V + E)$.

Ejemplo

Consideremos el siguiente grafo dirigido G con 8 vértices:

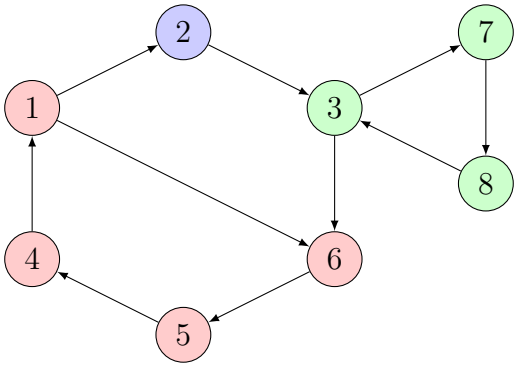


Observamos que en este grafo:

- Los nodos 1, 4, 5 y 6 forman un ciclo: $1 \to 6 \to 5 \to 4 \to 1$
- El nodo 2 está aislado en términos de ciclos
- Los nodos 3, 7 y 8 forman otro ciclo: $3 \to 7 \to 8 \to 3$

Componentes Fuertemente Conexas

Tras aplicar el algoritmo de Kosaraju, las componentes fuertemente conexas identificadas son:



Componentes identificadas:

- $\mathbf{SCC}_1 = \{1, 4, 5, 6\}$ (rojo): Forma un ciclo completo
- $\mathbf{SCC}_2 = \{2\}$ (azul): Nodo aislado en términos de ciclos
- $\mathbf{SCC}_3 = \{3, 7, 8\}$ (verde): Forma otro ciclo completo

4. Algoritmo para 2SAT

Algorithm 1 2SAT (vars, cláusulas)

```
1: grafo ← construir_grafo_de_implicación(vars, cláusulas)
2: mapa_scc ← encontrar_SCCs(grafo)
3: for cada x ∈ vars do
4:     if mapa_scc[x] = mapa_scc[-x] then
5:         return falso
6:     end if
7: end for
8: return verdadero
```

Complejidad temporal

$$T(V,E) = O(|V| + |E| + |V| + |E| + \frac{|V|}{2} * 2) = O(|V| + |E|)$$

Esta complejidad se justifica analizando cada paso del algoritmo:

1. **Construcción del grafo de implicación**: $|V| + |E|$
2. **Encontrar las componentes fuertemente conexas**: $O(|V| + |E|)$
   - La función `encontrar_SCCs(grafo)` implementa el algoritmo de Kosaraju
   - Este algoritmo tiene una complejidad temporal de $O(|V| + |E|)$, donde $|V|$ es el número de vértices y $|E|$ es el número de aristas
   - **Esta es la parte dominante de la complejidad total**
3. **Verificación de contradicciones**: $\frac{|V|}{2} * 2$
   - El bucle `for` itera sobre cada variable, habiendo $\frac{|V|}{2}$ variables
   - La comprobación de si una variable y su negación están en la misma SCC es una operación de tiempo constante (2 para cada iteración). Simplemente consultamos a qué SCC pertenece cada nodo y verificar si coinciden gracias al mapa que nos devuelve el algoritmo.

El término $O(|V| + |E|)$ del algoritmo para encontrar componentes fuertemente conexas es el que domina la complejidad total, ya que engloba a los demás términos.

Por tanto, la complejidad total del algoritmo 2SAT es $O(|V| + |E|)$, expresado en término de las variables y cláusulas de 2SAT nos quedaría $O(|n| + |m|)$, ya que $|V| = 2n$ y $|E| = 2m$, donde n es el número de variables y m el número de cláusulas. Es decir, toma tiempo polinómico y podemos asegurar que $2SAT \in P$.

5. Conclusión

Este algoritmo permite verificar la satisfacibilidad de cualquier fórmula 2-CNF en tiempo polinómico, transformando el problema en la detección de ciclos de inconsistencia en un grafo dirigido.