2-SAT

Leopoldo Taravilse[1]

[1]Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Algoritmos y Estructuras de Datos III

Contenidos

Contenidos

Satisfability Problem

Satisfability Problem

El problema de satisficibilidad (en inglés, Satisfability Problem, o SAT como se lo conoce popularmente) consiste en encontrar, dadas variables booleanas y una fórmula, si esta es satisfacible. Toda fórmula puede ser reescrita como una o más fórmulas (cláusulas), que son OR de una o varias variables o negaciones de variables (literales), tales que la fórmula original es satisfacible sii todas las nuevas fórmulas lo son

Satisfability Problem

Satisfability Problem

El problema de satisficibilidad (en inglés, Satisfability Problem, o SAT como se lo conoce popularmente) consiste en encontrar, dadas variables booleanas y una fórmula, si esta es satisfacible. Toda fórmula puede ser reescrita como una o más fórmulas (cláusulas), que son OR de una o varias variables o negaciones de variables (literales), tales que la fórmula original es satisfacible sii todas las nuevas fórmulas lo son

$$F = F_1 \wedge F_2 \wedge ... \wedge F_n$$

$$F_i = p_{i,1} \vee p_{i,2} \vee ... \vee p_{i,m_i}$$

$$p_{i,j} = v_t | \neg v_t \text{ siendo } v_t \text{ una variable proposicional.}$$

Satisfability Problem

Satisfability Problem

El problema de satisficibilidad (en inglés, Satisfability Problem, o SAT como se lo conoce popularmente) consiste en encontrar, dadas variables booleanas y una fórmula, si esta es satisfacible. Toda fórmula puede ser reescrita como una o más fórmulas (cláusulas), que son OR de una o varias variables o negaciones de variables (literales), tales que la fórmula original es satisfacible sii todas las nuevas fórmulas lo son

$$F = F_1 \wedge F_2 \wedge ... \wedge F_n$$

$$F_i = p_{i,1} \vee p_{i,2} \vee ... \vee p_{i,m_i}$$

$$p_{i,j} = v_t | \neg v_t \text{ siendo } v_t \text{ una variable proposicional.}$$

Este problema pertenece a una clase de problemas que se denominan NP-completos y para los cuáles no se conoce solución polinomial.

Satisfability Problem

- En 1971 Cook introdujo la noción de problemas NP-completos, que verán más adelante en la materia. Estos problemas son muy difíciles, pero gracias a Cook se sabe que si uno sale en tiempo polinomial, todos salen en tiempo polinomial. El primer problema con el que Cook introdujo los problemas NP-completos fue SAT.

Satisfability Problem

- En 1971 Cook introdujo la noción de problemas NP-completos, que verán más adelante en la materia. Estos problemas son muy difíciles, pero gracias a Cook se sabe que si uno sale en tiempo polinomial, todos salen en tiempo polinomial. El primer problema con el que Cook introdujo los problemas NP-completos fue SAT.
- Muchos de estos problemas tienen casos particulares que salen en tiempo polinomial. Uno de estos casos es 2-SAT. 2-SAT es el caso de SAT en el que las clausulas son el OR de 1 o 2 literales.

Satisfability Problem

- En 1971 Cook introdujo la noción de problemas NP-completos, que verán más adelante en la materia. Estos problemas son muy difíciles, pero gracias a Cook se sabe que si uno sale en tiempo polinomial, todos salen en tiempo polinomial. El primer problema con el que Cook introdujo los problemas NP-completos fue SAT.
- Muchos de estos problemas tienen casos particulares que salen en tiempo polinomial. Uno de estos casos es 2-SAT. 2-SAT es el caso de SAT en el que las clausulas son el OR de 1 o 2 literales.

$$F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$$

$$F_i = p_{i,1} \vee p_{i,2} \text{ o bien } F_i = p_{i,1}$$

$$p_{i,j} = v_t | \neg v_t \text{ siendo } v_t \text{ una variable proposicional.}$$

Contenidos

Solución polinomial a 2-SAT

- Para resolver este problema, es necesario primero introducir el concepto de componente fuertemente conexa.

Solución polinomial a 2-SAT

- Para resolver este problema, es necesario primero introducir el concepto de componente fuertemente conexa.
- Una vez introducido este concepto, vamos a ver cómo reducir 2-SAT a un problema de encontrar dichas componentes.

Contenidos

Definición

Componente Fuertemente Conexa

Dado un grafo no dirigido una componente conexa es un conjunto maximal de vértices tales que dados dos de ellos hay un camino que los une. Si el grafo es dirigido, una componente fuertemente conexa es un conjunto de vértices tales que dados dos de ellos hay un camino que va de uno al otro, y un camino que vuelve (es decir que hay un ciclo que pasa por ambos vértices).

Definición

Componente Fuertemente Conexa

Dado un grafo no dirigido una componente conexa es un conjunto maximal de vértices tales que dados dos de ellos hay un camino que los une. Si el grafo es dirigido, una componente fuertemente conexa es un conjunto de vértices tales que dados dos de ellos hay un camino que va de uno al otro, y un camino que vuelve (es decir que hay un ciclo que pasa por ambos vértices).

No es muy difícil ver que las componentes fuertemente conexas definen clases de equivalencia ya que son por definición reflexivas (un nodo está en su componente), simétricas, y sólo queda probar la transitividad que resulta de unir los caminos.

Propiedades de las componentes fuertemente conexas

- Una componente fuertemente conexa es maximal, es decir, si se agrega otro vértice del grafo con todos los ejes que lo unen a un vértice de la componente esta deja de ser fuertemente conexa.

Propiedades de las componentes fuertemente conexas

- Una componente fuertemente conexa es maximal, es decir, si se agrega otro vértice del grafo con todos los ejes que lo unen a un vértice de la componente esta deja de ser fuertemente conexa.
- Se puede generar un grafo en el que cada nodo represente a todos los nodos de una componente, y tal que los ejes sean los mismos eliminando repeticiones. Este grafo es un grafo dirigido acíclico (conocidos por sus siglas en inglés como DAG -Directed Acyclic Graph-).

Contenidos

Algoritmos para calcular componentes fuertemente conexas

Existen dos algoritmos conocidos con complejidad $o(n + m)$ para calcular componentes fuertementes conexas. Uno de ellos es el algoritmo de Kosaraju y el otro es el algoritmo de Tarjan. El algoritmo de Kosaraju es un poco más simple de implementar y veremos sólo este algoritmo.

Algoritmos para calcular componentes fuertemente conexas

Existen dos algoritmos conocidos con complejidad $o(n + m)$ para calcular componentes fuertementes conexas. Uno de ellos es el algoritmo de Kosaraju y el otro es el algoritmo de Tarjan. El algoritmo de Kosaraju es un poco más simple de implementar y veremos sólo este algoritmo.
Vamos a dar los detalles de la implementación y la demostración de correctitud pero no el código.

Algoritmo de Kosaraju

- Empezamos con una pila vacía y recorremos todos los vértices del grafo. Para cada vértice que no está en la pila hacemos un DFS.

Algoritmo de Kosaraju

- Empezamos con una pila vacía y recorremos todos los vértices del grafo. Para cada vértice que no está en la pila hacemos un DFS.
- Cada vez que en un DFS llegamos a un vértice del cuál no nos podemos seguir expandiendo a nuevos vértices lo agregamos a la pila.

Algoritmo de Kosaraju

- Empezamos con una pila vacía y recorremos todos los vértices del grafo. Para cada vértice que no está en la pila hacemos un DFS.
- Cada vez que en un DFS llegamos a un vértice del cuál no nos podemos seguir expandiendo a nuevos vértices lo agregamos a la pila.
- Luego de agregar todos los vértices a la pila invertimos la dirección de todos los ejes.

Algoritmo de Kosaraju

- Empezamos con una pila vacía y recorremos todos los vértices del grafo. Para cada vértice que no está en la pila hacemos un DFS.
- Cada vez que en un DFS llegamos a un vértice del cuál no nos podemos seguir expandiendo a nuevos vértices lo agregamos a la pila.
- Luego de agregar todos los vértices a la pila invertimos la dirección de todos los ejes.
- El próximo paso es ir desapilando vértices de la pila, si no están hasta el momento en ninguna componente hacemos un DFS desde ese nodo con los vértices que no están en ninguna componente. Los vértices a los que llegamos con ese DFS determinan una componente fuertemente conexa.

Correctitud del Algoritmo de Kosaraju

Si en uno de los DFS de la segunda parte de nuestro algoritmo llegamos a un vértice v, partiendo desde un vértice u, ambos vértices pertenecen a la misma c.f.c. Veamos porqué:

Correctitud del Algoritmo de Kosaraju

Si en uno de los DFS de la segunda parte de nuestro algoritmo llegamos a un vértice v, partiendo desde un vértice u, ambos vértices pertenecen a la misma c.f.c. Veamos porqué:

- Como llegamos de u a v en el DFS con ejes invertidos podemos ir de v a u en el grafo original.

Algoritmo de Kosaraju

- Empezamos con una pila vacía y recorremos todos los vértices del grafo. Para cada vértice que no está en la pila hacemos un DFS.
- Cada vez que en un DFS llegamos a un vértice del cuál no nos podemos seguir expandiendo a nuevos vértices lo agregamos a la pila.
- Luego de agregar todos los vértices a la pila invertimos la dirección de todos los ejes.
- El próximo paso es ir desapilando vértices de la pila, si no están hasta el momento en ninguna componente hacemos un DFS desde ese nodo con los vértices que no están en ninguna componente. Los vértices a los que llegamos con ese DFS determinan una componente fuertemente conexa.

Correctitud del Algoritmo de Kosaraju

Si en uno de los DFS de la segunda parte de nuestro algoritmo llegamos a un vértice v, partiendo desde un vértice u, ambos vértices pertenecen a la misma c.f.c. Veamos porqué:

- Como llegamos de u a v en el DFS con ejes invertidos podemos ir de v a u en el grafo original.

Correctitud del Algoritmo de Kosaraju

Si en uno de los DFS de la segunda parte de nuestro algoritmo llegamos a un vértice v, partiendo desde un vértice u, ambos vértices pertenecen a la misma c.f.c. Veamos porqué:

- Como llegamos de u a v en el DFS con ejes invertidos podemos ir de v a u en el grafo original.
- Dado que cuando apilamos u en el primer DFS ya habíamos apilado v, hay dos opciones. Una es que desde u nos expandimos hasta v para apilar v primero, y entonces hay camino de u a v y como había camino de v a u están en una misma componente fuertemente conexa.

Correctitud del Algoritmo de Kosaraju

Si en uno de los DFS de la segunda parte de nuestro algoritmo llegamos a un vértice v, partiendo desde un vértice u, ambos vértices pertenecen a la misma c.f.c. Veamos porqué:

- Como llegamos de u a v en el DFS con ejes invertidos podemos ir de v a u en el grafo original.
- Dado que cuando apilamos u en el primer DFS ya habíamos apilado v, hay dos opciones. Una es que desde u nos expandimos hasta v para apilar v primero, y entonces hay camino de u a v y como había camino de v a u están en una misma componente fuertemente conexa.
- La otra opción es que no haya camino desde u a v, luego si apilamos v primero, no puede ser viniendo de u porque no hay camino, entonces no visitamos u antes de pasar por v porque sino lo hubiesemos apilado, pero en ese caso vamos a visitar u en el DFS de v y apilar u antes que v, lo cual sabíamos que no pasaba, luego este caso lleva a un absurdo.

Correctitud del algoritmo de Kosaraju

- Si dos nodos están en una misma componente, los tenemos que encontrar con este método ya que desde el primero que desapilamos vamos a llegar con el DFS al otro.

Correctitud del algoritmo de Kosaraju

- Si dos nodos están en una misma componente, los tenemos que encontrar con este método ya que desde el primero que desapilamos vamos a llegar con el DFS al otro.
- Esto prueba que el algoritmo de Kosaraju es correcto y su complejidad es $O(n + m)$ ya que hacemos dos DFS y recorremos todos los ejes una vez para crear el grafo con ejes invertidos.

Correctitud del algoritmo de Kosaraju

- Si dos nodos están en una misma componente, los tenemos que encontrar con este método ya que desde el primero que desapilamos vamos a llegar con el DFS al otro.
- Esto prueba que el algoritmo de Kosaraju es correcto y su complejidad es $O(n+m)$ ya que hacemos dos DFS y recorremos todos los ejes una vez para crear el grafo con ejes invertidos.
- Es importante notar que la complejidad es $O(n+m)$ y no $O(m)$ ya que el grafo puede no ser conexo y luego no hay cotas para m en función de n.

Contenidos

2-SAT

Qué es 2-SAT=?

2-SAT es un caso particular de SAT, en el que usando la notación de la definición anterior todos los m_i son 1 o 2.

2-SAT

Qué es 2-SAT=?

2-SAT es un caso particular de SAT, en el que usando la notación de la definición anterior todos los m_i son 1 o 2.

- Para este problema se conoce una solución lineal en la cantidad de variables más la cantidad de fórmulas (las F_i).

2-SAT

Qué es 2-SAT=?

2-SAT es un caso particular de SAT, en el que usando la notación de la definición anterior todos los m_i son 1 o 2.

- Para este problema se conoce una solución lineal en la cantidad de variables más la cantidad de fórmulas (las F_i).
- Para resolver este problema se genera un grafo al cuál se le calculan las componentes fuertemente conexas.

Implementación de 2-SAT

- Generamos un grafo en el que los nodos son las variables y las negaciones de las variables, y hay una arista de p_i a p_j si $p_i \Rightarrow p_j$.

Implementación de 2-SAT

- Generamos un grafo en el que los nodos son las variables y las negaciones de las variables, y hay una arista de p_i a p_j si $p_i \Rightarrow p_j$.
- Esto lo hacemos porque $(p_i \vee p_j)$ se puede reescribir como $(\neg p_i \Rightarrow p_j) \wedge (\neg p_j \Rightarrow p_i)$

Implementación de 2-SAT

- Generamos un grafo en el que los nodos son las variables y las negaciones de las variables, y hay una arista de p_i a p_j si $p_i \Rightarrow p_j$.
- Esto lo hacemos porque $(p_i \vee p_j)$ se puede reescribir como $(\neg p_i \Rightarrow p_j) \wedge (\neg p_j \Rightarrow p_i)$
- Haciendo uso de la transitividad del operador $\Rightarrow$ calculamos las componentes fuertemente conexas de este grafo. Si p_i y $\neg p_i$ pertenecen a la misma componente para algún i entonces la fórmula es insatifacible, de lo contrario, podemos satisfacer la fórmula asignandole para cada variable v_i, el valor verdadero si no hay camino de v_i a $\neg v_i$ y falso en caso contrario., ya que no hay un camino de $\neg v_i$ a v_i.

Y esto cómo lo testeo?

Este algoritmo me encantó! Lo quiero ya! Pero... ¿Cómo lo testeo?

Y esto cómo lo testeo?

Este algoritmo me encantó! Lo quiero ya! Pero... ¿Cómo lo testeo?.

Y esto cómo lo testeo?

Este algoritmo me encantó! Lo quiero ya! Pero... ¿Cómo lo testeo?..

Y esto cómo lo testeo?

Este algoritmo me encantó! Lo quiero ya! Pero... ¿Cómo lo testeo?...

Y esto cómo lo testeo?

Este algoritmo me encantó! Lo quiero ya! Pero... ¿Cómo lo testeo?... En goo.gl/iEWjAV pueden testear el código de 2-SAT tanto en C++ como en Java.

Preguntas

¿Preguntas?

Inicio de espacio publicitario

Viernes que viene (16/10) 21:00hs ComCom en la Noriega