Tarjan's strongly connected components algorithm

Алгоритм Таріяна - це алгоритм в теорії графів для знаходження тісно зв'язаних компонентів (SCC) спрямованого графа. Він працює в лінійному часу, відповідаючи часу, обмеженому для альтернативних методів.

Алгоритм приймає на вхід орієнтований граф і робить розбиття вершин графа на сильно зв'язані між собою компоненти. Кожна вершина графа з'являється рівно в одній з сильно зв'язаних компонент. Будь-яка вершина, яка не перебуває на направленому циклі, сама по собі утворює сильно пов'язану компоненту: наприклад, вершину, чия степінь дорівнює 0, або будь-яку вершину ациклічного графа.

Основна ідея алгоритму полягає в наступному: пошук у глиб (DFS) починається з довільного стартового вузла (і подальший пошук у глиб виконується на будь-яких ще не знайдених вузлах). Як завжди при першому пошуку в глиб, пошук відвідує кожну вершину графа рівно один раз, не переглядаючи будь-яку вершину, яка вже була переглянута. Таким чином, колекція дерев пошуку являє собою великий ліс графа. Сильно пов'язані компоненти будуть відновлені як певні піддерева цього лісу. Коріння цих піддерев називаються "root" або сильно пов'язана компонента. Будь-який вузол сильно пов'язаної компоненти може служити коренем, якщо він виявиться першим вузлом компоненти в результаті пошуку.

Інваріант стека

Вузли розміщуються на стеці в порядку їх відвідування. Коли перший рекурсивний пошук в глиб відвідує вузол v і його нащадків, не всі ці вузли обов'язково вискакують з стека при поверненні цього рекурсивного виклику. Критично важливою властивістю інваріанта є те, що вузол залишається на стеку після його відвідування, якщо і тільки якщо на вхідному графі існує шлях від нього до якогось вузла раніше на стеку. Іншими словами, це означає, що в DFS вузол буде видалений з стека тільки після того, як всі пов'язані з ним шляху будуть пройдені. Коли DFS буде виконувати зворотне відстеження, то буде видаляти вузли по одному шляху і повертатися в корінь для того, щоб почати новий шлях.

В кінці виклику, який відвідує v і її нащадки, ми будемо знати, чи є у v шлях до якого-небудь вузла раніше в стеці. Якщо так, то виклик повертається, залишаючи v на стеку для збереження інваріанта. Якщо немає, то v повинна бути коренем його сильно пов'язаного компонента, який складається з v разом з будь-якими вузлами пізніше на стеку, ніж v. Підключений компонент, корені якого сягають в v, витягується з стека і повертається назад, знову ж зберігаючи інваріант.

Кожній вершині v присвоюється унікальний цілочисельний v.index, який нумерує вершини послідовно в порядку їх знаходження. Він також підтримує значення v.lowlink, яке представляє собою найменший індекс з усіх відомих вершин, доступних з v через піддерево v DFS, включаючи й саму вершину. Тому v повинна бути залишена на стеку, якщо v.lowlink <v.index, в той час як v повинна бути видалена як корінь сильно пов'язаного компонента, якщо v.lowlink == v.index. Значення v.lowlink обчислюється під час першого глибинного пошуку з v, так як при цьому знаходять вузли, доступні з v.

The algorithm in pseudocode

**algorithm** tarjan **is**

**input:** graph *G* = (*V*, *E*)

**output:** set of strongly connected components (sets of vertices)

*index* := 0

*S* := empty stack

**for each** *v* **in** *V* **do**

**if** *v*.index is undefined **then**

strongconnect(*v*)

**end if**

**end for**

**function** strongconnect(*v*)

*// Set the depth index for v to the smallest unused index*

*v*.index := *index*

*v*.lowlink := *index*

*index* := *index* + 1

*S*.push(*v*)

*v*.onStack := true

*// Consider successors of v*

**for each** (*v*, *w*) **in** *E* **do**

**if** *w*.index is undefined **then**

*// Successor w has not yet been visited; recurse on it*

strongconnect(*w*)

*v*.lowlink := min(*v*.lowlink, *w*.lowlink)

**else if** *w*.onStack **then**

*// Successor w is in stack S and hence in the current SCC*

*// If* w *is not on stack, then (*v*,* w*) is an edge pointing to an SCC already found and must be ignored*

*// Note: The next line may look odd - but is correct.*

*// It says w.index not w.lowlink; that is deliberate and from the original paper*

*v*.lowlink := min(*v*.lowlink, *w*.index)

**end if**

**end for**

*// If v is a root node, pop the stack and generate an SCC*

**if** *v*.lowlink = *v*.index **then**

start a new strongly connected component

**repeat**

*w* := *S*.pop()

*w*.onStack := false

add *w* to current strongly connected component

**while** *w* ≠ *v*

output the current strongly connected component

**end if**

**end function**

Змінна index є лічильником номерів вузлів пошуку в глиб. S - це стек вузлів, який починається з порожнього місця і зберігає історію вузлів, досліджених, але ще не прив'язаних до сильно зв'язаних компонент.

Самий зовнішній контур виконує пошук в кожному вузлі, який ще не був відвіданий, гарантуючи, що вузли, недоступні з першого вузла, в кінцевому підсумку все одно будуть подолані. Функція strongconnect виконує єдиний глибинний пошук графа, знаходить всіх наступників з вузла v і повідомляє про всі сильно з'єднаних компонент цього подграфа.

Коли кожен вузол завершує рекурсивне рух, якщо його низька посилання все ще встановлена ​​на свій індекс, то це кореневий вузол сильно пов'язаної компоненти, утворений усіма вузлами, що перебувають над ним на стек. Алгоритм виводить стек на поточний вузол включно і представляє всі ці вузли як сильно пов'язані компоненти.

Складність

Складність в часі: Функція Tarjan викликається один раз для кожного вузла; оператор розглядає кожен фузол максимум один раз. Тому час роботи алгоритму лінійний за кількістю ребер і вузлів в графі Ο(|V| + |E|)