1 Поиск в глубину

1.1 Понятия и алгоритм

1.1.1 Поиск (обход) графа

Обход графа — перечисление его вершин и рёбер в некотором порядке, и алгоритм, который это перечисление строит.

1.1.2 Основной алгоритм

Поиск (обход) в глубину определяется следующим алгоритмом:

1. Рассмотреть необработанную вершину v
2. Запомнить, что v обработана
3. Найти все рёбра $e : v \to u$, исходящие из v, и запустить обход в глубину из этой вершины

Несложно заметить, что обработанными будут все вершины, до которых есть путь из вершины, откуда был запущен обход.

Чаще всего указанный алгоритм запускают последовательно для всех необработанных вершин графа, таким образом строится обход всего графа вне зависимости от его связности.

1.1.3 Метки посещения

Для каждой вершины алгоритм требует помнить, обработана ли она. Чаще всего это можно запомнить в булевом массиве: для вершины v будет храниться логическое значение $visited[v]$, равное $false$ исходно. Как мы увидим дальше, для каждой вершины полезно будет хранить более подробные отметки.

1.1.4 Дерево обхода

Заметим, что в каждую вершины алгоритм зайдёт не больше одного раза, потому что как только это произойдёт, вершина будет помечена как посещённая. Это значит, что все посещённые вершины образуют дерево с корнем в той вершине, из которой обход был начат.

Чтобы построить дерево явным образом, можно запоминать для каждой вершины её родителя в момент, когда производится рекурсивный вызов.

1.1.5 Классификация рёбер

В процессе обхода в глубину рёбра можно классифицировать.

1. *Рёбра дерева* — это рёбра, соединяющие каждую вершину с её родителем в дереве обхода. Иными словами, это те рёбра, по которым алгоритм проходил в новые, ещё не посещённые вершины.
2. *Обратные рёбра* — это рёбра, ведущие из вершины в её предков в дереве обхода.
3. *Прямые рёбра* — это рёбра, ведущие из вершины в её потомков в дереве обхода.
4. *Перекрёстные рёбра* — это рёбра, не попавшие ни в один из предыдущих классов.

- Если граф является ориентированным, можно заметить, что каждое обратное ребро соответствует простому циклу.
- Если граф является неориентированным, то можно заметить несколько вещей:
  1. Во-первых, перекрёстных рёбер нет (докажите).
  2. Каждое прямое ребро $u \to v$ является также обратным $v \to u$
  3. Каждое обратное ребро $u \to v$ является также прямым $v \to u$, либо ребром дерева $v \to u$.

1.1.6 Время входа и выхода

В алгоритм можно добавить отметки «времени»: моменты, когда рекурсивная функция начинала обрабатывать вершину, и когда заканчивала это делать. Обычно для этого можно использовать последовательные целые числа. Обозначим за t_v^{in} время входа, а за t_v^{out} время выхода из вершины.

Заметим, что эти времена образуют правильную скобочную последовательность:

- $t_v^{in} < t_u^{in} < t_u^{out} < t_v^{out}$, если вершина v является предком вершины u,
- $t_u^{in} < t_v^{in} < t_v^{out} < t_u^{out}$, если вершина v является потомком вершины u,
- $t_v^{in} < t_v^{out} < t_u^{in} < t_u^{out}$ или $t_u^{in} < t_u^{out} < t_v^{in} < t_v^{out}$ иначе.

1.1.7 Метки трёх цветов

В процессе работы алгоритма можно использовать не только разделение на обработанные и необработанные вершины, но и более сложную структуру. Часто это называют раскраской вершин в чёрный, серый и белый цвета.

Исходно все вершины помечаются белым: они не начали обработку. В момент входа в вершину она помечается серым: обработка начата. В момент выхода из вершины она помечается чёрным: обработка закончена.

Это хороший способ обнаружить обратные рёбра: они и только они ведут из серой вершины в серую. Рёбра дерева при этом ведут из серой в белую, а остальные — из серой в чёрную.

1.1.8 * Особенности реализации для дерева

Если обход в глубину запускается на дереве (чаще всего, из корня), единственными рёбрами, не входящими в дерево обхода, будут обратные к ним. Разумеется, это верно только для неориентированных деревьев. В частности, это позволяет не хранить массив пометок. Зацикливания можно избежать, запретив переход из вершины в её родителя.

1.2 Применения

1.2.1 Проверка на предка

Запустив на дереве обход в глубину и вычислив времена входа и выхода, достаточно проверить, вкладываются ли эти пары друг в друга.

1.2.2 Компоненты связности

Для неориентированного графа обход в глубину помечает все вершины, лежащие в одной компоненте связности с данной, и только их.

1.2.3 Поиск циклов

Любое обратное ребро соответствует циклу. Для неориентированных графов, обратные рёбра, совпадающие с рёбрами дерева, являются единственным исключением.

1.2.4 Топологическая сортировка

Если в графе нет циклов, можно расположить вершины в порядке, обратном времени выхода. Этот порядок и будет топологической сортировкой: все рёбра будут вести «вперёд».

1.2.5 Раскраска графа в два цвета

Вершина должна быть раскрашена в цвет, противоположный цвету её предка в дереве обхода. Все рёбра, кроме рёбер дерева, достаточно проверить на соответствие.

1.2.6 Компоненты сильной связности

Запустим поиск в глубину и запомним времена выхода из вершин. Инвертируем все рёбра и запустим поиск в глубину ещё раз в порядке убывания запомненного времени выхода t_v^{out}. Каждый очередной запуск поиска в глубину из ещё не помеченной вершины будет проходить очередную компоненту сильной связности.

Идеи доказательства:

- Пусть вершина u помечена при очередном запуске поиска по инвертированному графу из вершины v. Значит, из u в исходном графе достижима v.
- Известно, что $t_v^{out} > t_u^{out}$. Значит, либо $t_v^{in} < t_u^{in} < t_u^{out} < t_v^{out}$, либо $t_u^{in} < t_u^{out} < t_v^{in} < t_v^{out}$. Второй вариант невозможен, так как существует путь из u в v, а из первого варианта следует, что из v достижима u.

Каждую компоненту сильной связности можно «сжать» в одну вершину. Получится граф, который называется *конденсацией* исходного графа.

Докажите:

- Конденсация ациклична
- Описанный алгоритм перечисляет вершины конденсации в порядке топологической сортировки

1.2.7 Мосты

Вычислим следующую функцию: для всех вершин в поддереве v минимальное время входа в вершину, достижимое по одному обратному ребру из вершин этого поддерева. Обозначим эту функцию как l_v.

1. Если ребро не является ребром дерева, оно не является мостом. Очевидно: ведь тогда его можно удалить, и связность по дереву не изменится.

2. Ребро дерева $p \to v$ является мостом тогда и только тогда, когда $l_v > t_p^{in}$. Предположим, что неравенство выполняется и удалим данное ребро. Заметим, что путь из любой вершины данного поддерева наружу невозможен: все обратные рёбра будут вести ниже. Импликация в обратную сторону очевидна.

1.2.8 Точки сочленения

Аналогичным образом вычислим функцию l_v.

Посмотрим, потеряется ли связность при удалении вершины. Для этого поддерево одного из её детей не должно иметь возможности выйти «выше».

Таким образом, вершина v с отцом p будет точкой сочленения в том и только том случае, если среди её детей есть вершина $u : l_u > t_p^{in}$ (докажите).

Ситуация с корнем несколько отличается: его удаление не приведёт к пропаданию связности с предками. Корень является точкой сочленения в том и только том случае, если у него хотя бы два ребёнка в дереве обхода (докажите).

1.2.9 Компоненты двусвязности

Для компонент как вершинной, так и рёберной двусвязности, можно применить схожие алгоритмы. Во-первых, нужно находить, соответственно, точки сочленения или мосты. Во-вторых, в процессе обхода нужно добавлять текущие рёбра или вершины в стек, а в момент обнаружения компоненты снимать все её элементы со стека.