добавлено: 10 Jun 2008 19:27
редактировано: 9 Jul 2009 18:24

Поиск компонент сильной связности, построение конденсации графа

Содержание [скрыть]

Определения, постановка задачи

Дан ориентированный граф G, множество вершин которого V и множество рёбер — E. Петли и кратные рёбра допускаются. Обозначим через n количество вершин графа, через m — количество рёбер.

Компонентой сильной связности (strongly connected component) называется такое (максимальное по включению) подмножество вершин C, что любые две вершины этого подмножества достижимы друг из друга, т.е. для $\forall u, v \in C$:

$$u \mapsto v, v \mapsto u$$

где символом $\mapsto$ здесь и далее мы будем обозначать достижимость, т.е. существование пути из первой вершины во вторую.

Понятно, что компоненты сильной связности для данного графа не пересекаются, т.е. фактически это разбиение всех вершин графа. Отсюда логично определение **конденсации** G^{SCC} как графа, получаемого из данного графа сжатием каждой компоненты сильной связности в одну вершину. Каждой вершине графа конденсации соответствует компонента сильной связности графа G, а ориентированное ребро между двумя вершинами C_i и C_j графа конденсации проводится, если найдётся пара вершин $u \in C_i, v \in C_j$, между которыми существовало ребро в исходном графе, т.е. $(u, v) \in E$.

Важнейшим свойством графа конденсации является то, что он **ацикличен**. Действительно, предположим, что $C \mapsto C'$, докажем, что $C' \not\mapsto C$. Из определения конденсации получаем, что найдутся две вершины $u \in C$ и $v \in C'$, что $u \mapsto v$. Доказывать будем от противного, т.е. предположим, что $C' \mapsto C$, тогда найдутся две вершины $u' \in C$ и $v' \in C'$, что $v' \mapsto u'$. Но т.к. u и u' находятся в одной компоненте сильной связности, то между ними есть путь; аналогично для v и v'. В итоге, объединяя пути, получаем, что $v \mapsto u$, и одновременно $u \mapsto v$. Следовательно, u и v должны принадлежать одной компоненте сильной связности, т.е. получили противоречие, что и требовалось доказать.

Описываемый ниже алгоритм выделяет в данном графе все компоненты сильной связности. Построить по ним граф конденсации не составит труда.

Алгоритм

Описываемый здесь алгоритм был предложен независимо Косараю (Kosaraju) и Шариром (Sharir) в 1979 г. Это очень простой в реализации алгоритм, основанный на двух сериях поисков в глубину, и потому работающий за время $O(n+m)$.

На первом шаге алгоритма выполняется серия обходов в глубину, посещающая весь граф. Для этого мы проходимся по всем вершинам графа и из каждой ещё не посещённой вершины вызываем обход в глубину. При этом для каждой вершины v запомним **время выхода** $\mathrm{tout}[v]$. Эти времена выхода играют ключевую роль в алгоритме, и эта роль выражена в приведённой ниже теореме.

Сначала введём обозначение: время выхода $\mathrm{tout}[C]$ из компоненты C сильной связности определим как максимум из значений $\mathrm{tout}[v]$ для всех $v \in C$. Кроме того, в доказательстве теоремы будут упоминаться и времена входа в каждую вершину $\mathrm{tin}[v]$, и аналогично определим времена входа $\mathrm{tin}[C]$ для каждой компоненты сильной связности как минимум из величин $\mathrm{tin}[v]$ для всех $v \in C$.

Теорема. Пусть C и C' — две различные компоненты сильной связности, и пусть в графе конденсации между ними есть ребро (C, C'). Тогда $\mathrm{tout}[C] > \mathrm{tout}[C']$.

При доказательстве возникает два принципиально различных случая в зависимости от того, в какую из компонент первой зайдёт обход в глубину, т.е. в зависимости от соотношения между $\mathrm{tin}[C]$ и $\mathrm{tin}[C']$:

- Первой была достигнута компонента C. Это означает, что в какой-то момент времени обход в глубину заходит в некоторую вершину v компоненты C, при этом все остальные вершины компонент C и C' ещё не посещены. Но, т.к. по условию в графе конденсаций есть ребро (C, C'), то из вершины v будет достижима не только вся компонента C, но и вся компонента C'. Это означает, что при запуске из вершины v обход в глубину пройдёт по всем вершинам компонент C и C', а, значит, они станут потомками по отношению к v в дереве обхода в глубину, т.е. для любой вершины $u \in C \cup C', u \neq v$ будет выполнено $\mathrm{tout}[v] > \mathrm{tout}[u]$, ч.т.д.
- Первой была достигнута компонента C'. Опять же, в какой-то момент времени обход в глубину заходит в некоторую вершину $v \in C'$, причём все остальные вершины компонент C и C' не посещены. Поскольку по условию в графе конденсаций существовало ребро (C, C'), то, вследствие ацикличности графа конденсаций, не существует обратного пути $C' \not\to C$, т.е. обход в глубину из вершины v не достигнет вершин C. Это означает, что они будут посещены обходом в глубину позже, откуда и следует $\mathrm{tout}[C] > \mathrm{tout}[C']$, ч.т.д.

Доказанная теорема является **основой алгоритма** поиска компонент сильной связности. Из неё следует, что любое ребро (C, C') в графе конденсаций идёт из компоненты с большей величиной tout в компоненту с меньшей величиной.

Если мы отсортируем все вершины $v \in V$ в порядке убывания времени выхода $\mathrm{tout}[v]$, то первой окажется некоторая вершина u, принадлежащая "корневой" компоненте сильной связности, т.е. в которую не входит ни одно ребро в графе конденсаций. Теперь нам хотелось бы запустить такой обход из этой вершины u, который бы посетил только эту компоненту сильной связности и не зашёл ни в какую другую; научившись это делать, мы сможем постепенно выделить все компоненты сильной связности: удалив из графа вершины первой выделенной компоненты, мы снова найдём среди оставшихся вершину с наибольшей величиной tout, снова запустим из неё этот обход, и т.д.

Чтобы научиться делать такой обход, рассмотрим **транспонированный граф** G^T, т.е. граф, полученный из G изменением направления каждого ребра на противоположное. Нетрудно понять, что в этом графе будут те же компоненты сильной связности, что и в исходном графе. Более того, граф конденсации $(G^T)^{SCC}$ для него будет равен транспонированному графу конденсации исходного графа G^{SCC}. Это означает, что теперь из рассматриваемой нами "корневой" компоненты уже не будут выходить рёбра в другие компоненты.

Таким образом, чтобы обойти всю "корневую" компоненту сильной связности, содержащую некоторую вершину v, достаточно запустить обход из вершины v в графе G^T. Этот обход посетит все вершины этой компоненты сильной связности и только их. Как уже говорилось, дальше мы можем мысленно удалить эти вершины из графа, находить очередную вершину с максимальным значением $\text{tout}[v]$ и запускать обход на транспонированном графе из неё, и т.д.

Итак, мы построили следующий **алгоритм** выделения компонент сильной связности:

1 шаг. Запустить серию обходов в глубину графа G, которая возвращает вершины в порядке увеличения времени выхода tout, т.е. некоторый список order.

2 шаг. Построить транспонированный граф G^T. Запустить серию обходов в глубину/ширину этого графа в порядке, определяемом списком order (а именно, в обратном порядке, т.е. в порядке уменьшения времени выхода). Каждое множество вершин, достигнутое в результате очередного запуска обхода, и будет очередной компонентой сильной связности.

Асимптотика алгоритма, очевидно, равна $O(n+m)$, поскольку он представляет собой всего лишь два обхода в глубину/ширину.

Наконец, уместно отметить связь с понятием **топологической сортировки**. Во-первых, шаг 1 алгоритма представляет собой не что иное, как топологическую сортировку графа G (фактически именно это и означает сортировка вершин по времени выхода). Во-вторых, сама схема алгоритма такова, что и компоненты сильной связности он генерирует в порядке уменьшения их времён выхода, таким образом, он генерирует компоненты - вершины графа конденсации в порядке топологической сортировки.

Реализация

```
vector < vector<int> > g, gr;
vector<char> used;
vector<int> order, component;

void dfs1 (int v) {
        used[v] = true;
        for (size_t i=0; i<g[v].size(); ++i)
                if (!used[ g[v][i] ])
                        dfs1 (g[v][i]);
        order.push_back (v);
}

void dfs2 (int v) {
        used[v] = true;
        component.push_back (v);
```

```
        for (size_t i=0; i<gr[v].size(); ++i)
                if (!used[ gr[v][i] ])
                        dfs2 (gr[v][i]);
}

int main() {
        int n;
        ... чтение n ...
        for (;;) {
                int a, b;
                ... чтение очередного ребра (a,b) ...
                g[a].push_back (b);
                gr[b].push_back (a);
        }

        used.assign (n, false);
        for (int i=0; i<n; ++i)
                if (!used[i])
                        dfs1 (i);
        used.assign (n, false);
        for (int i=0; i<n; ++i) {
                int v = order[n-1-i];
                if (!used[v]) {
                        dfs2 (v);
                        ... вывод очередной component ...
                        component.clear();
                }
        }
}
```

Здесь в g хранится сам граф, а gr — транспонированный граф. Функция $dfs1$ выполняет обход в глубину на графе G, функция $dfs2$ — на транспонированном G^T. Функция $dfs1$ заполняет список $order$ вершинами в порядке увеличения времени выхода (фактически, делает топологическую сортировку). Функция $dfs2$ сохраняет все достигнутые вершины в списке $component$, который после каждого запуска будет содержать очередную компоненту сильной связности.

Литература

- Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. **Алгоритмы: Построение и анализ** [2005]
- M. Sharir. **A strong-connectivity algorithm and its applications in data-flow analysis** [1979]