**О. А. Лакийчук**

# АЛГОРИТМЫ ПОИСКА ДОМИНАТОРОВ В УПРАВЛЯЮЩЕМ ГРАФЕ*

## 1. ВВЕДЕНИЕ

Теория графов применяется в программировании с самого начала возникновения ЭВМ. Очень удобно выражать задачи обработки информации на теоретико–графовом языке. В заметке Р.Карпа (1960 г., на русском — 1962 г.) была введена в практику теоретико-графовая модель программы в виде управляющего графа. Эта модель стала к настоящему времени классической для решения задач трансляции и конструирования программ.

Современное состояние программирования нельзя представить себе без теоретико-графовых алгоритмов. В литературе можно найти описания огромного количества таких алгоритмов. Достаточно хотя бы рассмотреть книги “Теория графов : алгоритмы обработки деревьев” (1994) и “Сводимые графы и граф-модели в программировании” (1999). Авторы — В.А. Евстигнеев и В.Н. Касьянов. Для представления алгоритмов в этих книгах используется подход, базирующийся на языке высокого уровня (ВУ-язык) и модели машины с произвольным доступом к памяти (РАМ) [1, 2]. Мы также будем использовать этот подход, поскольку он позволяет формулировать алгоритмы работы с графами в естественной и наглядной форме, допускающей прямой анализ их корректности и сложности, а также довольно простой перенос на традиционные языки программирования.

В данной работе рассматриваются алгоритмы поиска обязательных предшественников (доминаторов) в управляющем графе. В разд. 2 описаны общие теоретические сведения, в разд. 3 представлены существующие алгоритмы поиска доминаторов. Разд. 4 посвящён новому алгоритму поиска доминаторов.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

## 2. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАБОТЫ

Дальнейшее изложение будет строиться на понятии управляющего графа [1–3].

Напомним некоторые сведения о нём. Транслируемая программа с точки зрения многих алгоритмов оптимизирующей трансляции рассматривается и обрабатывается как управляющий граф (также называют уграф, граф переходов, граф потока управления) — ориентированный граф, вершинами которого являются операторы программы, а дуги отражают возможность передачи управления между операторами при её исполнении.

Список задач анализа управляющих графов достаточно широк. Это задачи по определению структурных свойств графов (например, интервальная сводимость), достижению нужных структурных свойств графов (преобразование графа в сводимый, разрыв контуров и другие), задачи по декомпозиции графа на подграфы специального вида (гамаки, зоны). Здесь же рассматриваются задачи определения отношений доминирования и постдоминирования на уграфе.

Дальше под уграфом будем понимать ориентированный граф $G =(V,E,r)$ с выделенной начальной вершиной $r$ *(вход)*, в которую не входит ни одна дуга. Каждая вершина $G$ достижима из $r$. Вершина $x$ *доминирует над* вершиной y *(x — обязательный предшественник* вершины y), если любой путь в $G$ из $s$ в $y$ проходит через $x$. *Непосредственным доминатором* вершины $w$ называется вершина v такая, что v доминирует над w и любой доминатор вершины w доминирует над v (обозначаем idom(w) = v). Отношение доминирования задаёт частичный порядок на $V$, а результирующее частично упорядоченное множество представляет собой ордерево, называемое *доминаторным деревом*.

## 3. ОПИСАНИЕ АЛГОРИТМОВ

### 3.1. Прямой алгоритм поиска доминаторов

Получается из определения доминируемости. Для каждой вершины $v \neq r$ выполняем следующие действия.

*Основной шаг.* Ищем множество всех вершин $P$, достижимых из $r$ на путях, не содержащих вершину $v$. Множество $V\backslash\{v\}\backslash S$ состоит в точности из тех вершин уграфа, которые доминируются вершиной v.

Этот алгоритм имеет временную сложность $O(nm)$, где n — число вершин, m — число дуг.

### 3.2. Алгоритм Ленгауэра—Тарьяна

Более подробное описание алгоритма можно найти в [4].

Сначала введём некоторые понятия.

Пусть M(v) — M-номер вершины v и T — дерево поиска в глубину, тогда справедливо следующее свойство путей.

**1.** Пусть v,w — вершины. Если $M(v) \le M(w)$, то любой путь от v до w должен проходить через одного из общих предшественников вершин v и w в T.

Каждой вершине v сопоставляется семидоминатор *sdom(v)*, определяемый следующим образом:

$$Sdom(w) = \min\{M(v) \mid \text{существует такой путь } (v_0 = v, v_1, \ldots, v_k = w), \text{ что } M(v_i) > M(w) \text{ для всех I, } 1 <= I < k\}.$$

Дальше при формулировании свойств отождествляем имена вершин с их M-номерами. Используем обозначения $x \xrightarrow{*} y$, если x — предшественник y в T, а также $x \xrightarrow{+} y$, если $x \xrightarrow{*} y$ и $x \neq y$.

#### *3.2.1. Свойства семидоминаторов*

**2**. Для любой вершины $w \neq r$, $idom(w) \xrightarrow{+} w$, $sdom(w) \xrightarrow{+} w$, $idom(w) \xrightarrow{+} sdom(w)$.

**3**. Пусть v, w такие 2 вершины, что $v \xrightarrow{*} w$, тогда $v \xrightarrow{*} idom(w)$ или $idom(w) \xrightarrow{*} idom(v)$.

**4.** Пусть $w \neq r$ и каждая вершина v, для которой $sdom(w) \xrightarrow{+} v \xrightarrow{*} w$, удовлетворяет условию $sdom(w) \le sdom(v)$, тогда $idom(w) = sdom(w)$.

**5.** Пусть $w \neq r$ и v — вершина, которая имеет минимального семидоминатора sdom(v) среди вершин v таких, что $sdom(w) \xrightarrow{+} v \xrightarrow{*} w$, тогда $sdom(v) \le sdom(w)$ и $idom(v) = idom(w)$.

**6.** Пусть $w \neq r$ и v — вершина, которая имеет минимального семидоминатора sdom(v) среди вершин v таких, что $sdom(w) \xrightarrow{+} v \xrightarrow{*} w$, тогда

$$idom(w) = \begin{cases} sdom(w), & \text{если } sdom(w)=sdom(v) \\ idom(w), & \text{иначе} \end{cases}$$

**7.** Для любой вершины $w \neq r$ $sdom(w) = \min(\{v: (v, w) \in E, v < w\} \cup \{sdom(z) : z > w,$ существует $(v,w) \in E$ такая, что $z \xrightarrow{*} v\})$.

Доказательства этих свойств можно найти в [4].

С помощью рассмотренных свойств можно предложить быстрый алгоритм нахождения доминаторов. Он состоит из четырёх этапов.

1. Выполняется поиск в глубину. Вершинам приписываются их M-номера от 0 до n – 1, начиная с r. Проводится инициализация переменных.

2. Вычисляются семидоминаторы вершин уграфа по свойству 7. Обработка ведётся в порядке убывания M номеров.

3. По свойству 6 неявно вычисляются непосредственные доминаторы вершин.

4. Явно вычисляются idom(v) для каждой вершины v.

Каждой вершине сопоставим следующие атрибуты:

1) parent(v) — отец вершины v в дереве поиска в глубину T;
2) semi(v) — число, принимающее следующие значения:
   a) semi(v) = 0, пока v не получила своего M-номера,
   b) semi(v) = M(v) после получения вершиной v M-номера, но до вычисления idom(v),
   c) semi(v) = M(idom(v)) после вычисления idom(v);
3) bucket(v) — множество вершин, для которых v — доминатор;
4) dom(v) — вершина, определяемая следующим образом:
   a) после шага 3, если sdom(v) = idom(v), то dom(v) = idom(v), иначе dom(v) = x, где M(x) < M(v) и idom(x) = idom(v),
   b) после шага 4 dom(v) = idom(v).

Шаги 2 и 3 алгоритм выполняет одновременно. В процессе обработки вершин поддерживается временная структура данных, которая представляет собой лес, содержащийся в дереве поиска в глубину. Он образован множеством вершин V и дуг {(parent(w), w): w — уже обработана}. Для работы с этой структурой алгоритм использует 2 процедуры:

LINK(v,w) — добавляет в лес дугу (v,w);

EVAL(v) — возвращает v, если v — корень дерева в лесу. Иначе, пусть x — корень дерева в лесу, содержащего v. Eval возвращает любую вершину $u \neq r$ с минимумом semi(v) на пути $x \xrightarrow{*} v$.

При обработке вершины v алгоритм вычисляет семидоминатор вершины v, используя свойство 7; semi(v) = M(sdom(v)). После вычисления semi(v) алгоритм добавляет вершину v к множеству bucket($M^{-1}$(semi(v))) и добавляет дугу в лес, используя вызов процедуры LINK(parent(v),v). Дальше выполняется шаг 3 для каждой вершины из множества bucket(parent(w)). Неявно вычисляется idom(v) по свойству 6. На шаге 4 алгоритм вычисляет явно idom(v), просматривая вершины в порядке возрастания их M-номеров.

Нетрудно доказать корректность алгоритма, используя свойства 6 и 7 при условии, что операции LINK и EVAL работают правильно.

### *3.2.2. Реализация операций LINK и EVAL*

**1. Простой способ** — использует сжатие пути при выполнении операции EVAL. Для представления леса используются два атрибута вершин уграфа: ancestor и label. Сначала ancestor(v) = 0 и label(v) = v для любой вершины v. В общем, ancestor(v) = 0, если v — корень дерева в лесу, иначе ancestor(v) содержит отца v в лесу.

Значения атрибута label должны удовлетворять следующему свойству.

Пусть v — вершина, r — корень дерева в лесу, содержащего v, и $v = v_k, v_{k-1}, \ldots, v_0 = r$ — последовательность вершин такая, что ancestor($v_i$) = $v_{i-1}$ для всех i. Пусть x такая вершина, что semi(x) принимает минимальное значение среди вершин $x \in \{label(v_i): 1 \le i \le k\}$. Тогда semi(x) должно принимать минимальное значение среди вершин x, удовлетворяющих условию $r \xrightarrow{+} x \xrightarrow{*} v$.

При выполнении LINK(v,w) алгоритм присваивает ancestor(w) = v. Чтобы осуществить EVAL(v), алгоритм следует по ссылкам на отцов вершин для определения последовательности $v = v_k, v_{k-1}, \ldots, v_0 = r$; ancestor($v_i$) = $v_{i-1}$ для всех i. Если v = r, то EVAL(v) возвращает v. Иначе алгоритм осуществляет сжатие пути, присваивая ancestor($v_i$) = r для всех i от 2 до k, одновременно изменяя атрибуты label по правилу: если semi(label($v_{i-1}$)) < semi(label($v_i$)), то label($v_i$) = label($v_{i-1}$). Затем EVAL(v) возвращает v.

2. **Сложный способ** — использует сжатие пути при выполнении EVAL, но реализует операцию LINK так, чтобы сжатие пути выполнялось только на сбалансированных деревьях.

### *3.2.3. Оценка временной сложности*

Нетрудно показать, что временная сложность алгоритма Ленгауэра—Тарьяна составляет O(m + n) плюс время, необходимое для выполнения n – 1 операции LINK и m + n – 1 операции EVAL. Простая реализация этих

операций требует O(m * log n) времени. Такая же сложность и у всего алгоритма. В сложном случае время, требуемое для n – 1 операции LINK и m + n – 1 операции EVAL, составляет O(m * l(m, n)), где l — функция, обратная функции Аккермана.

## 4. НОВЫЙ АЛГОРИТМ

Проблема уменьшения временной сложности алгоритма поиска доминаторов — интересная задача. Этой проблемой занимались многие учёные. В 1985 г. Харел предложил почти линейный алгоритм [5], потом в 1997 г. Алструп усовершенствовал его [6]. Тарьян впервые использовал в своём алгоритме разбиение дерева поиска на микродеревья. Ниже описан новый алгоритм поиска доминаторов, использующий микродеревья и простые структуры данных. Он базируется на материалах работы [7].

Основные понятия такие же, как в алгоритме Ленгауэра—Тарьяна. При рассмотрении свойств вершины отождествляются со своими M-номерами.

### 4.1. Микродеревья

Рассмотрим следующую процедуру, помечающую вершины в дереве T. Параметр g задан, и все вершины изначально не помечены.

```
For x in D in reverse DFS order do
    S (x): = Σ (y child of x) S(y)
    If S (x)>g then
        Mark all children of x
    Endive
Done
Mark root (T)
```

Для каждой вершины v положим бпп(v) — ближайший помеченный предшественник. Функция бпп разбивает T на микродеревья следующим образом. Пусть v — помечена ; D(v) = {x|бпп(v) = v} — микродерево. root(D(v)) = v — корень микродерева D(v), micro(x) — микродерево, содержащее x (см. рис. 1).

Мы называем микродерево *нетривиальным*, если оно содержит хотя бы один лист дерева T. Все остальные микродеревья состоят ровно из одной

вершины и называются *тривиальными*. Вершина v, составляющая тривиальное микродерево, называется *специальной*, если каждый потомок v — корень нетривиального микродерева. На рис.1 c и e — такие вершины.

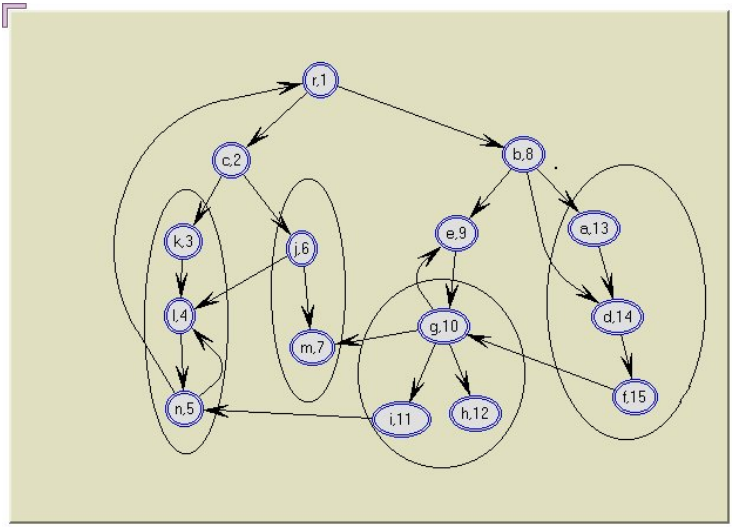


*Рис. 1.* Разбиение T на микродеревья величины не больше 3.
Корни — вершины k, j, g, a

### 4.2. Определение путей

Пусть P = (u = $x_0$, $x_1$, …, $x_{k-1}$, $x_k$ = v) — путь в G. P — *внешний доминаторный путь* (XDOM путь), если P — Sdom путь и $x_0$, …, $x_{k-1} \notin$ micro(v).

*Внешний доминатор* вершины v:

$$xdom(v) = \min\{\{v\} \cup \{u| \text{ существует XDOM путь из u в v}\}\}.$$

Если v совпадает с тривиальным микродеревом, то xdom(v) = semi(v).
P — PXDOM путь, если $x_i \geq$ root(micro(v)), $1 \leq I \leq k–1$.

$$pxdom(v) = \min\{ u| \text{ есть PXDOM путь из u в v}\}.$$

**Пример.** Путь P = (c, j, l) — XDOM путь от c к l. Больше таких путей нет. idom(l) = 2. P = (r, b, e, g, I, n, l) — PXDOM путь. pxdom(l) = 1.

Справедливы следующие свойства.

**2.1.** Для каждой вершины, составляющей тривиальное микродерево, pxdom(v) = semi(v).

**2.2.** Если idom(v) $\notin$ micro(v), то idom(v) $\xrightarrow{*}$ pxdom(v).

### 4.3. Вычисление внутренних доминаторов

Для нетривиального микродерева введём понятие *расширенного графа* aug(D). Пусть G(D) — подграф уграфа G, построенный на вершинах микродерева D. aug(D) — граф G(D) плюс следующие дуги и вершины:

1) вершина t, называемая *корнем* aug(D) или root(aug(D));
2) дуга (t, v) для каждой вершины v $\in$ D такой, что есть дуга (u, v) $\in$ E для некоторой u $\notin$ D. Назовём эти дуги *синими* дугами.

Определяем внутреннего непосредственного предшественника вершины x — iidom(x) как idom(x) в aug(micro(x)).

Справедливы следующие свойства.

**3.1**. Если iidom(x) $\neq$ root(aug(micro(x))), то idom(x) = iidom(x).

**3.2.** Если iidom(x) = root(aug(micro(x))), то idom(x) $\notin$ micro(x).

### 4.4. Вычисление pxdom

Используем атрибут вершины label(v) в лесу (как в алгоритме Ленгауэра—Тарьяна), label(v) = v изначально.

Пусть EN(v) = {x | (x, v) $\in$ E, x $\notin$ micro(v)} — внешние соседи вершины v. Процедура работает в порядке убывания M-номеров.

1. Для v$\in$D:
   a) B = {label(x) | x $\in$ EN(v)};
   b) C = {label(eval(parent(root(micro(x))))) | x $\in$ EN(v), x не $\xrightarrow{*}$ v};
   c) Label(v) = min({v} $\cup$ B$\cup$C).
2. Пусть v$\in$D. Y(v) — множество вершин u из D таких, что есть путь из u в v, состоящий только из вершин G(D). Положим label(v) = min{label(y) | y $\in$ Y(v)}.
3. Если D — тривиальное микродерево, то выполняем link(v).

Справедливы следующие свойства.

**4.1**. После шага (1) label(x) = xdom(x) для всех x $\in$ D.

**4.2.** После шага (2) label(x) = pxdom(x) для всех x $\in$ D.

### 4.5 Вычисление доминаторов

Справедливы следующие свойства.

**5.1.** Для любой вершины v существует такая вершина $w \in micro(v)$, что

1) $w \xrightarrow{*} v$;
2) pxdom(v) = pxdom(w);
3) pxdom(w) = sdom(w);
4) iidom(w) = root(aug(micro(w))).

**5.2.** Пусть v и w — вершины в микродереве D такие, что

1) $w \xrightarrow{*} v$;
2) pxdom(v) = pxdom(w);
3) iidom(v) = iidom(w) = root(aug(T));

тогда idom(v) = idom(w).

Теперь представим сам алгоритм.

```
Algorithm IDOM
        For v ∈ D в порядке убывания M-номеров do
                Process(v)
        Done
    For u ∈ D, {u} — тривиальное микродерево, в порядке убывания M-
    номеров do
                Process(bucket(u))
                Link(u)
    Done
End Algorithm

Process(v)
    If iidom(v) ∉ micro(v) then
                Idom(v) = iidom(v)
    Else
                Add v to bucket(pxdom(v)) endif
End Process

Process(bucket(u))
    For v ∉ bucket(u) do
                If u = parent(root(micro(v))) then z = v
                Else z = eval(parent(root(micro(v))) endif
                If pxdom(z) = u then idom(v) = u
                Else idom(v) = idom(z) endif
  Done
End Process
```

**Теорема.** Алгоритм IDOM корректно вычисляет непосредственных предшественников вершин уграфа.

**Доказательство.** Свойство 3.1 показывает, что присваивание idom(v) = iidom(v) корректно, если iidom(v) $\in$ micro(v). Пусть теперь iidom(v) $\notin$ micro(v). Тогда idom(v) $\notin$ micro(v) по свойству 3.2.

Рассмотрим обработку вершины v из bucket(u). Пусть сначала pxdom(v) = sdom(v) = u. Пусть u1 — сын u на пути $u \xrightarrow{+} v$. Мы берем z — вершину на пути $u1 \xrightarrow{*} v$ с минимумом sdom и pxdom(z) = sdom(z). Это требование нужно, так как, если pxdom(z) = u, то по свойству 4 (алгоритм Л—Т) idom(v) = sdom(v) = u, и если pxdom(z) < u, то по свойству 5(Л—Т) idom(v) = idom(z).

Заметим, что для каждой w $\in$ micro(v) такой, что

$$w \xrightarrow{*} v,\ sdom(v) = pxdom(v) \le pxdom(w) \le sdom(w).$$

Таким образом, если u = parent(root(micro(v))), то u1 = root(micro(v)), z = v и требование выполняется.

С другой стороны, если

$$u \xrightarrow{+} parent(root(micro(v))),$$

то z = eval(parent(roor(micro(v)))) — вершина на пути

P = $u1 \xrightarrow{*}$ parent(root(micro(v))) с минимумом pxdom. Требование выполняется, так как (1) pxdom(u1) $\le$u = pxdom(v) и (2) pxdom(y) = sdom(y) для всех y $\in$ P (свойство 2.1).

Рассмотрим оставшийся случай, когда pxdom(v) $\ne$ sdom(v). По свойству 5.1 находим w. По предположению iidom(v) = root(aug(micro(x))), и idom(v) = idom(w) по свойству 5.2. Так как pxdom(v) = pxdom(w) и micro(v) = micro(w), то v и w лежат в одном bucket. Поэтому, исходя из предыдущих рассуждений, алгоритм правильно вычисляет значение idom(w), что и требовалось доказать**.**

### *4.6. Анализ временной сложности*

Простой анализ показывает, что при g = $O(\log^{1/3} n)$ вычисление iidom требует порядка O(n + m) времени. Вычисление pxdom требует также O(m + n) времени плюс время на выполнение операций link и eval. Нетрудно показать, что тогда общее время алгоритма $O(m * l(m, n/\log^{1/3} n) + n)$. Известно, что в этом случае $l(m, n/\log^{1/3} n) = O(1)$, а значит алгоритм работает за линейное время O(m + n).

## 5. ЗАКЛЮЧЕНИЕ

В статье рассмотрены некоторые существующие алгоритмы поиска доминаторов в управляющем графе и представлен новый линейный алгоритм.

## СПИСОК ЛИТЕРАТУРЫ


1. **Евстигнеев В.А., Касьянов В.Н.** Теория графов: алгоритмы обработки деревьев. — Новосибирск: Наука, 1994.
2. **Евстигнеев В.А., Касьянов В.Н**. Сводимые графы и граф — модели в программировании. — Новосибирск: ИДМИ, 1999.
3. **Ахо А., Ульман Дж.** Теория синтаксического анализа, перевода и компиляции. Т. 1, 2. — М.: Мир, 1978.
4. **Lengauer T., Tarjan R.E.** A fast algorithm for finding dominators in a flow graph // ACM Trans. Program. Lang. Syst. — 1979. —Vol. 1, N 1. — P.121–141.
5. **Harel D.** A linear time algorithm for finding dominators in flow graphs and related problems // Proc. of the 17th ACM Sympos. on Theory of Computing. –ACM Press, N.-Y., 1985. — P. 185–194
6. **Alstrup S., Harel D.** Dominators in linear time. — 1997. `ftp://ftp.diku.dk/pub/diku/users/stephen/dom.ps`
7. **Buchsbaum A. L., Kaplan H., Rogers A., Westbrook J. R.** A new, simpler linear-time dominators algorithm // ACM Trans. Progr. Lang. and Systems. — 1998. — Vol. 20, N 6. — P. 1265–1296.