Литература

1. *Журавков М. А.* Математическое моделирование деформационных процессов в твердых деформируемых средах (на примере задач механики горных пород и массивов). Мн.:БГУ, 2002. 456с.
2. *Журавков М. А., Напрасникова Ю. В.* Численное исследование напряженного состояния в породе с выработкой. Материалы Международной научно-технической конференции «Наука – образованию, производству, экономике» 4–7 марта 2003г, БНТУ. Мн.: БНТУ, 2003.

ЛИНЕЙНЫЙ АЛГОРИТМ НАХОЖДЕНИЯ СВЯЗНЫХ КОМПОНЕНТ ДОПОЛНИТЕЛЬНОГО ГРАФА

А. Х. Перез-Чернов

Введение

В работе описывается алгоритм, который по списку смежности вершин графа предъявляет связные компоненты или самого графа, или его дополнения. Каждая строка списка смежности должна быть упорядоченной. Главное преимущество алгоритма заключается в том, что он позволяет за линейное $O(n+m)$ время найти связные компоненты дополнительного графа $(n=|VG|, m=|EG|)$. Эта задача может быть решена и традиционным способом, путем непосредственного перехода к дополнению. Однако, использование связки «переход к дополнению — применение волнового алгоритма» приводит к сложности $O(n^2)$. В случае, если данная проблема входит в качестве подзадачи в алгоритм, имеющий сложность меньшую, чем квадрат, необходимо использовать более быстрые алгоритмы нахождения связных компонент графа $\overline{G}$.

Существуют практические алгоритмы, в которых желательно использовать именно линейное нахождение связных компонент дополнительного графа. В сфере работ, связанных с модульной декомпозицией, данная процедура часто востребована. Например, с помощью данного алгоритма удалось получить машинно-ориентированную $O((n+m)\ Log(n+m))$ реализацию (P,S,N)-декомпозиции. Более того, алгоритм описывает программную реализацию одного из этапов модульной декомпозиции [1], которая в ряде случаев позволяет уменьшить размерность некоторых сложных задач. В некоторых классах графов применение модульной декомпозиции оказывается эффективным методом для решения определенных NP-полных задач, например, КЛИКА и ХРОМАТИЧЕСКОЕ ЧИСЛО.

На основе описанных в [1] идей разработаны машинно-ориентированные алгоритмы на структурах данных (стек дополнения). Для того, чтобы добиться линейности при работе со списком смежности, пришлось создать такую структуру данных, как *FD*-таблица смежности.

Основной алгоритм

Пусть граф задан таблицей смежности. Выпишем в список L все его вершины. Список D, в котором будем хранить связные компоненты, первоначально будет пустым.

Пусть есть некоторая стековая структура *S*, которая поддерживает следующие операции:

1. «поместить множество X в вершину стека, предварительно удалив все вхождения элементов $x_i \in X$ из стека»–обозначаемая как $push(X)$.
2. «удалить элемент (сверху) стека»–обозначаемая как $pop()$

Поместим в *S* одну вершину из списка L и воспользуемся следующей процедурой:

Процедура (*):
Пока S≠∅ выполнять {
1) v = pop() //вытолкнуть из S элемент;

2) X = neighbor(*v*) //узнать множество смежных с этим
//элементом вершин;

3) push (X) //поместить это множество сверху в S,
//удаляя все элементы множества X на-
//ходящиеся ранее в S;

4) induced (v) //удалить из списка смежности строку
//v и все вхождения вершины v
// в других строках;

5) D. add (v) // добавим к D вершину v;

6) L.delete(v) // удалим v из L;
}

Справедливы следующие леммы:

Лемма 1. Как только стек S окажется пустым, в списке D будут находиться вершины, образующие одну связную компоненту графа, и только они.

Если на шаге 3) процедуры (*) вместо операции $push(X)$ применять операцию $\overline{push}(X)$, логически (но не на уровне реализации) эквивалент-

ную с $push(V \setminus X)$, то в S каждый раз будет оказываться именно окружение вершины v в дополнительном графе. Определенную таким образом процедуру обозначим (**).

Назовем связные компоненты дополнения *косвязными компонентами*.

Лемма 2. Процедура (**) разлагает граф на связные или на косвязные компоненты

Таким образом, нам необходимо создать две структуры данных, первая из которых поддерживала бы линейное оперирование со списком смежности (операции *neighbor*() и *induced*()). Вторая должна выполнить операции $\overline{push}(X)$ за линейное время.

Структура данных: FD-таблица смежности

Например, по списку смежности (см. рис. 1) построим соответствующую FD-таблицу (см. рис. 2). Каждая ячейка FD-таблицы состоит из 5 сегментов (см. рис.3). Один из сегментов имеет номер столбца, в котором содержится, а оставшиеся—предназначены для вертикального и горизонтального «сшивания» указателями соответствующих ячеек.

Операция $induced(i)$ обращается к i-му элементу $FirstV[i]$ вершины i (i-столбцу), делает шаг по указателю и оказывается в первом содержащем вершину i столбце. Удаляет текущую ячейку (т.е «сшивает» (splices) указатели (W,E) и (N,S) соответствующих соседей (по указателям) текущей ячейки). В результате выполнения этой последовательности из цепочки указателей каждой строки FD-таблицы будет исключена вершина i.

1	1,2,4
2	1,4
3	4
4	1,2,3

Рис. 1

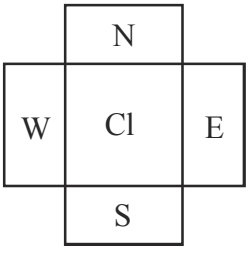

Рис.3

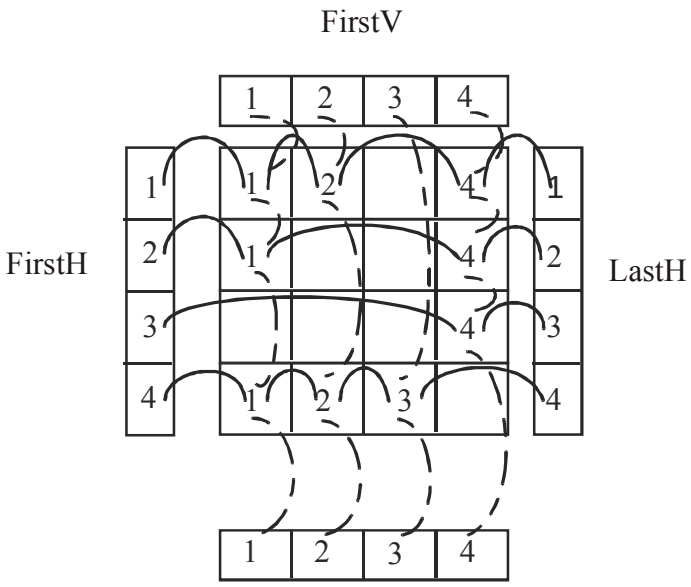

Рис. 2

Операция $neighbor(i)$ возвращает список Cl-сегментов (имен) в цепочке указателей i-ой строки.

Несложно заметить, что количество операций затрачиваемых при работе с FD-таблицей смежности равно $O(n+m)$.

Структуры данных: стек дополнения

Для эффективной работы процедуры (*) и (**) необходимо реализовать структуру, которая бы обеспечивала быструю реализацию операций pop, $\overline{push}(X)$, $push()$.

Механизм работы стека дополнения основан на двух идеях:

1. Различать вершины поступающие в стек по времени помещения. Будем хранить два структуры, в одной – вершины помещенные с помощью последнего вызова $\overline{push}(X_k)$, в другой – вершины помещенные с помощью предыдущих вызовов $\overline{push}(X_i), \tilde{a}\ddot{a}\mathring{a}\ i<k$
2. Хранить изначально в стеке все вершины графа, вместо того, чтобы помещать в стек дополнение окружения вершины с помощью операции $push(V \setminus X)$. В таком случае дополнение $V \setminus X$ окажется в стеке, если мы исключим из стека X. Мы запретим в стеке вершины из X и с помощью указателей создадим переходы через группы запрещенных вершин. С помощью временных пометок обеспечим корректное функционирование структуры данных.

Обозначим через $F(B_k)$ временную сложность процедуры $\overline{push}(X_k)$. Можно доказать, что $F(B_k) \le \tilde{N}(|X_k|+|X_{k-1}|)$, где C — некоторая константа, не зависящая от k. Далее, просуммировав все $F(X_k)$, $k=\overline{1,n}$, получим, что процедура (**) требует $O(n+m)$ операций.

Литература

1. *Dahlhaus E.* Efficient and practical modular decomposition // Technische Universität Berlin, No. 524/1996

О ЧИСЛЕННОМ РЕШЕНИИ СИНГУЛЯРНОГО ИНТЕГРАЛЬНОГО УРАВНЕНИЯ ПЕРВОГО РОДА С КРАТНЫМИ ЯДРАМИ КОШИ С ИСПОЛЬЗОВАНИЕМ МНОГОЧЛЕНОВ ЧЕБЫШЕВА

В. Ю. Пятницкий

Рассмотрим сингулярное интегральное уравнение первого рода