АССОЦИАТИВНЫЙ ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ДЛЯ ДИНАМИЧЕСКОЙ ОБРАБОТКИ ДЕРЕВА КРАТЧАЙШИХ ПУТЕЙ ПОСЛЕ УДАЛЕНИЯ ИЗ ГРАФА ОДНОЙ ДУГИ

А. Ш. Непомнящая

Институт вычислительной математики и математической геофизики СО РАН
630090 Новосибирск, просп. Лаврентьева, 6
E–mail: anep@ssd.sscc.ru

Аннотация

В работе строится эффективный ассоциативный алгоритм для динамической обработки дерева кратчайших путей после удаления одной дуги из ориентированного взвешенного графа. С этой целью мы используем STAR–машину, которая моделирует работу ассоциативных (контекстно–адресуемых) параллельных систем с простыми однобитовыми процессорными элементами и последовательно–поразрядной (вертикальной) обработкой информации. Вначале приводится STAR–машина и группа базовых процедур, которые будут использоваться в статье. Затем мы описываем основные определения и структуру данных, позволяющую выполнять параллельную обработку данных по содержимому памяти. Ассоциативный алгоритм представляется на STAR–машине в виде главной процедуры DeleteArcSPT, использующей группу вспомогательных процедур. С помощью вспомогательных процедур описываются отдельные части ассоциативного алгоритма для динамической обработки дерева кратчайших путей после удаления из графа одной дуги. В работе доказана корректность процедуры DeleteArcSPT. Мы показали, что на STAR–машине эта процедура выполняется за время $O(hk)$, где h – это число битов, которое требуется для кодирования длины максимального кратчайшего пути, а k – это число вершин, для которых вычисляются новые кратчайшие пути после удаления одной дуги из исходного графа.

Ключевые слова: Ориентированный взвешенный граф, дерево кратчайших путей, матрица смежности, декрементальный алгоритм, ассоциативный параллельный процессор

1. Введение

Задача нахождения кратчайших путей возникает в различных приложениях. Известны две версии этой проблемы: нахождение кратчайших путей из одной вершины (the single-source shortest paths problem) и нахождение кратчайших путей между каждой парой вершин (the all-pairs shortest paths problem). Динамическая версия проблемы нахождения кратчайших путей из одной вершины состоит в обработке информации

о кратчайших путях во время изменения графа без перевычисления графа целиком после каждого локального изменения в нем. Типичными операциями для этого вида преобразований являются добавление или удаление одной дуги либо изменение веса одной дуги. Если граф представляет коммуникационную или транспортную сеть, то добавление или удаление дуги отражает такие реальные изменения в сети как добавление или удаление связей во время существования сети. Алгоритм называется *полностью динамическим* (fully dynamic), если он позволяет выполнять любую последовательность упомянутых выше операций. Алгоритм называется *инкрементальным*, если он допускает только добавление дуги или уменьшение веса дуги. Если алгоритм допускает только удаление дуги или увеличение веса дуги, то он называется *декрементальным*.

В работах [1, 2] для графов с произвольными вещественными весами дуг построены полностью динамические алгоритмы для обработки дерева кратчайших путей. При этом авторы полагают, что граф не имеет циклов отрицательной длины до и после обработки входной информации. В этих работах используется модель, которая учитывает сложность выходной информации после модификации входа. Сложность динамического алгоритма оценивается суммой числа аффектных вершин, которые модифицируют свое кратчайшее расстояние после преобразования входной информации, и числа дуг, у которых по крайней мере одна из вершин является аффектной. В работе [3] построены полудинамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей как в ориентированном, так и в неориентированном графе с положительными вещественными весами ребер. Декрементальные алгоритмы строятся для планарных графов, а инкрементальные алгоритмы применяются к произвольным графам. Временная сложность алгоритмов оценивается в терминах амортизированного времени. В работе [4] построены полностью динамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей как в ориентированном, так и в неориентированном графе с положительными действительными весами ребер после выполнения произвольной последовательности преобразований ребер. Стоимость операции обработки задается как функция от числа преобразований выходных данных, используя понятие k-ограниченной вычисляющей функции. В работе [5] строится полностью динамический алгоритм для обработки дерева кратчайших путей ориентированного графа с произвольными весами дуг. Авторы строят новый алгоритм для удаления дуги и для увеличения веса дуги и новый алгоритм для добавления дуги и для уменьшения веса дуги, в которых явно используются циклы отрицательной длины. Сложность операции обработки задается как функция от структуры графа и числа обработок выходной информации. В работе [6] построена эффективная параллельная реализация алгоритма Рамалингама [1] для динамической обработки подграфа кратчайших путей после удаления одной дуги из ориентированного взвешенного графа с выделенным стоком. В качестве модели вычисления используется STAR–машина [7], которая моделирует работу ассоциативных параллельных систем типа SIMD с последовательно–поразрядной (вертикальной) обработкой информации и простыми процессорными элементами. Исходное дерево кратчайших путей строится с помощью классического алгоритма Дейкстры

[8]. Ассоциативная версия алгоритма Рамалингама представлена в виде процедуры DeleteArc, корректность которой доказана. Мы показали, что на STAR–машине эта процедура выполняется за время, которое пропорционально числу вершин, для которых изменяются кратчайшие расстояния после удаления одной дуги из графа. Следуя Фостеру [9], мы полагаем, что каждая элементарная операция STAR–машины выполняется за единицу времени. Также в работе [6] приведены основные достоинства параллельной реализации декрементального алгоритма Рамалингама.

В данной работе строится ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей ориентированного графа с неотрицательными весами дуг после удаления из графа одной дуги. Ассоциативный параллельный алгоритм использует простую структуру данных, позволяющую выполнять доступ к данным по содержимому памяти. Алгоритм представлен в виде процедуры DeleteArcSPT, корректность которой доказана. Мы показали, что эта процедура выполняется на STAR–машине с трудоемкостью $O(hk)$, где h – число битов, которое требуется для кодирования максимума кратчайших расстояний от корня, а k – число аффектных вершин, для которых строятся новые кратчайшие пути и вычисляются новые кратчайшие расстояния. Отметим, что выполнение процедуры DeleteArcSPT проще, чем выполнение процедуры DeleteArc.

2. Модель ассоциативного параллельного процессора

Приведем краткое описание используемой модели. Подробное описание модели и сравнение ее с другими моделями ассоциативной обработки приводится в работе [10].

Модель определяется как абстрактная STAR-машина типа SIMD с вертикальной обработкой информации, основными компонентами которой являются:

- последовательное устройство управления, в котором записаны программа и скалярные константы;
- устройство ассоциативной обработки, состоящее из p одноразрядных процессорных элементов;
- матричная память.

Входные данные, записанные в двоичном коде, помещаются в матричную память в виде двумерных таблиц, причем каждая единица данных хранится в отдельной строке и обрабатывается отдельным процессорным элементом. Строки каждой таблицы нумеруются слева направо, а столбцы – сверху вниз. В матричную память можно загружать несколько таблиц.

Устройство ассоциативной обработки представляется в виде совокупности h вертикальных одноразрядных регистров длины p. Вертикальный регистр можно представлять как массив, состоящий из одного столбца. Обработка информации происходит следующим образом. Из определенной таблицы в определенном порядке извлекаются ее одноразрядные столбцы и помещаются в вертикальные регистры устройства ассоциативной обработки, в котором выполняются побитовые операции. Результат

выполнения любой операции записывается либо в некоторый регистр, либо в определенный столбец обрабатываемой таблицы, либо в матричную память.

Чтобы моделировать обработку информации в матричной памяти, STAR-машина использует типы данных **slice, word** и **table**. С помощью переменной типа **slice** моделируется доступ к таблице по столбцам, а с помощью типа **word** – доступ по строкам. С каждой переменной типа **table** ассоциируется матрица из n строк и k столбцов, где $n \leq p$. С каждой переменной типа **slice** ассоциируется последовательность из p компонент, принадлежащих множеству $\{0, 1\}$. Для простоты переменную типа **slice** условимся называть *слайсом*.

Пусть X и Y – переменные типа **slice**, i, j – переменные типа **integer**.

Приведем некоторые операции и предикаты для слайсов.

SET(Y) записывает в слайс Y все единицы.

CLR(Y) записывает в слайс Y все нули.

$Y(i)$ выделяет i-ю компоненту в слайсе Y $(1 \leq i \leq p)$.

FND(Y) выдает порядковый номер i позиции первой (или старшей) единицы в слайсе Y, где $i \geq 0$.

STEP(Y) выдает такой же результат, как и операция FND(Y), и затем "гасит" (сбрасывает) старшую единицу.

CONVERT(Y) возвращает строку, каждый i-й бит которой совпадает с $Y(i)$. Эта операция применяется, когда строка одной матрицы используется как битовый столбец в другой матрице.

Операции FND(Y), STEP(Y) и CONVERT(Y) используются только в правой части оператора присваивания в то время как операция $Y(i)$ может использоваться в любой части этого оператора.

Для выполнения параллелизма по данным общепринятым способом вводятся следующие побитовые логические операции: $X\ and\ Y$, $X\ or\ Y$, $not\ X$ и $X\ xor\ Y$. Также будем использовать предикат SOME(Y), который возвращает **true** тогда и только тогда, когда в слайсе Y имеется хотя бы одна компонента $'1'$. Условимся, что запись $Y \neq \emptyset$ означает, что предикат SOME(Y) возвращает значение **true**.

Заметим, что предикат SOME(Y) и все операции для типа **slice** также используются для переменных типа **word**.

Пусть T - переменная типа **table**. Будем использовать следующие две операции:

ROW(i, T) выделяет i-ю строку в матрице T.

COL(i, T) выделяет i-й столбец в T.

Заметим, что операторы языка STAR вводятся таким же образом, как в языке Pascal.

Приведем группу базовых процедур [11, 12], которые будем использовать в данной работе. С помощью глобального слайса X будем указывать *позиции* анализируемых строк в соответствующей процедуре. В работах [11, 12] показано, что каждая базовая процедура выполняется с трудоемкостью $O(k)$, где k – это число битовых столбцов в соответствующей матрице.

Процедура MATCH(T, X, v, Z) одновременно определяет позиции строк в заданной матрице T, которые совпадают с заданной строкой v. Она возвращает слайс Z, в

котором $Z(i) =' 1'$ тогда и только тогда, когда $\mathrm{ROW}(i, T) = v$ и $X(i) =' 1'$.

Процедура $\mathrm{MIN}(T, X, Z)$ одновременно определяет позиции строк в заданной матрице T, в которых записан минимальный элемент. Она возвращает слайс Z, в котором $Z(i) =' 1'$ тогда и только тогда, когда $\mathrm{ROW}(i, T)$ - минимальный элемент и $X(i) =' 1'$.

Процедура $\mathrm{TCOPY1}(T, j, h, F)$ записывает в результирующую матрицу F h столбцов из матрицы T, начиная с $(1 + (j-1)h)$-го столбца, где $j \geq 1$.

Процедура $\mathrm{ADDV}(T, F, X, R)$ записывает в матрицу R результат одновременного сложения соответствующих строк матриц T и F, позиции которых отмечены $'1'$ в слайсе X. Этот алгоритм использует таблицу 5.1 из [9].

3. Основные понятия

Пусть $G = (V, E, w)$ – это *ориентированный взвешенный граф*, в котором $V = \{1, 2, \ldots, n\}$ – множество вершин, $E \subseteq V \times V$ – множество ориентированных ребер (*дуг*) и $w(e)$ – функция веса. Полагаем, что $|V| = n$ и $|E| = m$. Будем рассматривать графы, дуги которых имеют положительный вес. Будем считать, что $w(i, j) = \infty$, если $(i, j) \notin E$.

Для рассматриваемой задачи значение бесконечности выбирается следующим образом: $\sum_{i=1}^{n} c_i$, где c_i – максимальный вес дуг, выходящих из вершины i в G. Обозначим через h число битов для кодирования этой суммы.

Пусть $e = (i, j)$ – это дуга, которая ориентирована от вершины i до вершины j. При этом вершина j называется *головой* дуги e, а вершина i называется ее *хвостом*.

Матрицей смежности назовем квадратную матрицу $A = [a_{ij}]$ размера $n \times n$, элементы которой определяются следующим образом: $a_{ij} = 1$, если $(i, j) \in E$ и $a_{ij} = 0$, в противном случае.

Кратчайшим путем из вершины v_1 до вершины v_k в графе G назовем последовательность вершин v_1, v_2, ..., v_k, в которой $(v_i, v_{i+1}) \in E$ для $1 \leq i < k - 1$ и сумма весов его дуг минимальна. Обозначим через $dist(l)$ *длину* кратчайшего пути из вершины v_1 до вершины l.

Деревом кратчайших путей T_s с корнем s назовем связный подграф без циклов, который содержит все вершины графа G, и путь от корня до любой вершины i является *минимальным*.

Пусть дуга (i, j) удалена из графа G. Вершина y называется *аффектной* после удаления дуги (i, j) из дерева кратчайших путей T_s, если вершина y не достижима из корня s.

4. Ассоциативный декрементальный алгоритм для обработки дерева кратчайших путей

В данном разделе вначале приводится структура данных. Затем описывается группа ассоциативных параллельных алгоритмов для выполнения отдельных частей рассматриваемой задачи. После этого приводится ассоциативный параллельный алго-

ритм для динамической обработки дерева кратчайших путей после удаления из графа одной дуги. Этот алгоритм будет использовать идею, приведенную в декрементальном алгоритме Рамалингама [1]. Реализация вспомогательных алгоритмов на STAR–машине и доказательство корректности соответствующих процедур будут приведены в полной статье.

Будем использовать следующую структуру данных:

- матрица смежности G размером $n \times n$, каждый i-й столбец которой хранит с помощью $'1'$ головы дуг, выходящих из вершины i;
- матрица SPT размером $n \times n$, каждый i-й столбец которой хранит с помощью $'1'$ головы дуг, выходящих из вершины i и принадлежащих дереву кратчайших путей;
- матрица $Weight$ размером $n \times hn$, элементами которой являются веса дуг. Она состоит из n полей, причем каждое поле состоит из h битов. Вес дуги (i, j) записывается в j-ю строку i-го поля;
- матрица $Cost$ размером $n \times hn$, элементами которой являются веса дуг. Она состоит из n полей, причем каждое поле состоит из h битов. Вес дуги (i, j) записывается в i-ю строку j-го поля;
- матрица $Dist$ размером $n \times h$, каждая i-я строка которой хранит кратчайшее расстояние от корня до вершины i;
- слайс $AffectedV$, который хранит с помощью $'1'$ позиции аффектных вершин.

Будем использовать следующие два свойства:

Свойство 1. Каждое i-е поле матрицы $Weight$ хранит веса дуг, выходящих из вершины i, а i-е поле матрицы $Cost$ хранит веса дуг, заходящих в эту вершину.

Свойство 2. В каждой i-й строке матриц G и SPT отмечены $'1'$ хвосты дуг, заходящих в вершину i.

Вначале приведем ассоциативный алгоритм для нахождения аффектных вершин и аффектных дуг (скажем, *Алгоритм 1*), которые получаются после удаления дуги (i, j) из T_s. Алгоритм будет находить вершины, которые принадлежат компоненте связности, включающей вершину j. Он будет использовать матрицу SPT, слайс $AffectedV$ и вспомогательный слайс Z. Ассоциативный параллельный алгоритм выполняет следующие шаги:

Шаг 1. Включить в слайсы $AffectedV$ и Z головы дуг, выходящих из вершины j. Включить вершину j в слайс $AffectedV$. Удалить из матрицы SPT все дуги, выходящие из вершины j.

Шаг 2. Пока $Z \neq \emptyset$, выполнить следующие действия:

– удалить позицию самого верхнего бита $'1'$ (скажем r) из слайса Z;

– включить в слайсы $AffectedV$ и Z головы дуг, выходящих из вершины r;

– удалить все дуги, выходящие из вершины r в матрице SPT.

На STAR–машине этот алгоритм представляется в виде процедуры AffectedVertSPT, которая возвращает слайс $AffectedV$ и измененную матрицу SPT.

Справедлива следующая

Лемма 1. *Пусть дуга (i, j) удалена из дерева кратчайших путей T_s. Тогда процедура* AffectedVertSPT *возвращает слайс $Affected V$, в котором позиции аффектных вершин отмечены '1'. Кроме того, процедура удаляет из матрицы SPT позиции всех дуг, выходящих из аффектных вершин.*

Эта лемма доказывается индукцией по высоте поддерева T_j.

Ассоциативный параллельный алгоритм для вычисления нового расстояния от корня до каждой аффектной вершины (скажем, *Алгоритм 2*) выполняется следующим образом.

Пока $Affected V \neq \emptyset$, определять новое расстояние от корня до каждой аффектной вершины с помощью следующих шагов.

Шаг 1. Выделить позицию текущей обрабатываемой вершины k в слайсе $Affected V$ и пометить ее нулем.

Шаг 2. Вычислить в графе G веса различных путей от корня до вершины k, которые не содержат аффектных вершин.

Шаг 3. Определить минимальное расстояние от корня до вершины k и записать его в k-ю строку матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры NewDistSPT, которая возвращает измененную матрицу $Dist$.

Справедлива следующая

Лемма 2. *Пусть заданы число h, слайс $AffectedV$ и текущие матрицы G, $Cost$ и $Dist$. Тогда процедура* NewDistSPT *возвращает измененную матрицу $Dist$, которая хранит новые кратчайшие расстояния от корня до каждой аффектной вершины.*

Эта лемма доказывается индукцией по числу аффектных вершин.

Ассоциативный параллельный алгоритм для обработки дуг, выходящих из аффектной вершины k (скажем, *Алгоритм 3*) выполняет следующие шаги:

Шаг 1. Зная слайс $Affected V$ и кратчайшее расстояние до вершины k, определить в матрице G вес пути от корня до каждой аффектной вершины r, которая является головой дуги, выходящей из вершины k.

Шаг 2. С помощью слайса (скажем, Y) сохранить головы дуг, выходящих из вершины k, для которых $dist_{new}(r) < dist_{old}(r)$. Записать новые расстояния в соответствующие строки матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры LeavingArcsSPT.

Справедлива следующая

Лемма 3. *Пусть заданы число h, номер текущей обрабатываемой вершины k, слайс $AffectedV$ и текущие матрицы $Weight$ и $Dist$. Тогда процедура* LeavingArcsSPT *возвращает измененную матрицу $Dist$, где новые кратчайшие расстояния записываются для тех вершин r, которые являются головами дуг с общим хвостом k, и для которых значения $dist_{new}(r)$ уменьшаются.*

Эта лемма доказывается методом от противного.

Теперь приведем ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после удаления дуги (i, j) из графа G. Он выполняет следующие шаги.

Шаг 1. Удалить из матрицы G позицию дуги (i,j). Если $(i,j) \notin SPT$, то перейти на выход. В противном случае удалить позицию дуги (i,j) из матрицы SPT.

Шаг 2. С помощью *Алгоритма 1* построить слайс $AffectedV$ и удалить аффектные дуги из матрицы SPT.

Шаг 3. С помощью *Алгоритма 2* определить новые расстояния от корня до всех аффектных вершин и записать их в соответствующие строки матрицы $Dist$.

Шаг 4. Пока $AffectedV \neq \emptyset$, обработать все аффектные вершины следующим образом:

– определить аффектную вершину k, которая удалена от корня на минимальное расстояние и удалить вершину k из слайса $AffectedV$;

– определить хвост r дуги, заходящей в вершину k, и включить позицию дуги (r,k) в матрицу SPT;

– с помощью *Алгоритма 3* перевычислить расстояния от корня до тех аффектных вершин q, которые являются головами дуг, выходящих из вершины k в графе G и $dist_{new}(q) < dist_{old}(q)$. Записать $dist_{new}(q)$ в соответствующие строки матрицы $Dist$.

На STAR–машине этот алгоритм реализован в виде процедуры DeleteArcSPT.

5. Выполнение ассоциативного декрементального алгоритма на STAR–машине

В данном разделе приведем выполнение процедуры DeleteArcSPT. По заданной дуге (i,j), которая удаляется из дерева кратчайших путей, и по текущим матрицам G, $Weight$, $Cost$, $Dist$ и SPT, эта процедура возвращает обработанные матрицы G, SPT и $Dist$. При этом будут использоваться базовые и вспомогательные процедуры.

```
procedure DeleteArcSPT(i,j,h: integer; Weight,Cost: table;
  var G,SPT: table; var Dist: table);
/* Дуга (i,j) будет удаляться из графа G. */
var k,r: integer;
  AffectedV,X,Y,Z: slice(G);
  W1,W2: table;
  v: word(G); v1: word(Dist);
  label 1;
1. Begin X:=COL(i,G); X(j):='0';
2.  COL(i,G):=X;
/* Дуга (i,j) удаляется из G. */
3.  X:=COL(i,SPT);
4.  if X(j)='0' then goto 1;
5.  X(j):='0'; COL(i,SPT):=X;
/* Дуга (i,j) удаляется из SPT. */
6.  AffectedVertSPT(i,j,SPT,AffectedV);
/* Эта процедура возвращает матрицу SPT и слайс AffectedV. */
7.  NewDistSPT(h,G,Cost,AffectedV,Dist);
```

```
 /* Эта процедура возвращает измененную матрицу Dist. */
 8.  while SOME(AffectedV) do
 9.    begin MIN(Dist,AffectedV,Z);
 /* Слайс Z хранит позиции тех строк матрицы Dist,
   в которых записано минимальное значение. */
10.       k:=FND(Z); AffectedV(k):='0';
 /* Вершина k включается в дерево кратчайших путей. */
11.       v:=ROW(k,G); Z:=CONVERT(v);
12.       X:=Z and (not AffectedV);
 /* Слайс X хранит хвосты тех дуг, которые заходят в вершину
   k в графе G и не являются аффектными. */
13.       v1:=ROW(k,Dist);
14.       TCOPY1(Cost,k,h,W1);
15.       ADDV(Dist,W1,X,W2);
 /* Матрица W2 хранит различные расстояния от корня
   до вершины k в графе G. */
16.       MATCH(W2,X,v1,Y); r:=FND(Y);
17.       Y:=COL(r,SPT); Y(k):='1';
18.       COL(r,SPT):=Y;
 /* Дуга (r,k) включается в матрицу SPT. */
19.       LeavingArcsSPT(h,k,Weight,AffectedV,Dist);
 /* Эта процедура записывает новые расстояния в матрицу Dist для
   тех вершин q, для которых уменьшается dist_new(q) и (k,q) ∈ E. */
20.    end;
21. End;
```

Теорема. *Пусть ориентированный граф задается в виде матрицы смежности G и матрицы весов Weight. Пусть также заданы матрицы Cost, SPT и Dist и числа n и h. Пусть из графа удалена дуга (i,j). Тогда процедура* DeleteArcSPT *возвращает обработанные матрицы G, SPT и Dist.*

Доказательство. Будем доказывать индукцией по числу аффектных вершин q, которые появляются после удаления дуги (i, j) из дерева кратчайших путей.

Базис индукции доказывается для $q = 1$. Непосредственно проверяется, что после выполнения строк 1–5 позиция дуги (i, j) удаляется из матриц G и SPT. После выполнения строки 6 ввиду леммы 1 слайс $AffectedV$ хранит позицию аффектной вершины j и все дуги, выходящие из вершины j, удалены из матрицы SPT. Заметим, что j-й столбец матрицы SPT будет состоять из нулей, поскольку j – единственная аффектная вершина в T_s. После выполнения строки 7 ввиду леммы 2 получаем, что $AffectedV(j) =' 1'$ и новое кратчайшее расстояние от корня до вершины j будет записано в j-й строке матрицы $Dist$. Поскольку $AffectedV \neq \theta$, то выполняем цикл `while SOME(AffectedV) do` (строка 8). После выполнения строк 9–10 получаем, что $k = j$, $AffectedV = \theta$ и вершина j включена в дерево кратчайших путей. Чтобы найти хвост дуги, заходящей в вершину j, которую надо включить в матрицу SPT,

будем использовать метод, предложенный в нашей работе [13]. Вначале находим хвосты дуг, заходящих в вершину j в матрице G, которые не являются аффектными (строки 11–12). Затем после выполнения строк 13–15 определяем различные расстояния от корня до вершины j, которые включают выделенные дуги. После выполнения строки 16 определяем вершину r, на которой достигается минимальное расстояние от корня. Наконец после выполнения строк 17–18 новая дуга (r, j) включается в матрицу SPT.

Шаг индукции. Пусть утверждение выполняется, когда $q \geq 1$ аффектных вершин обработаны в заданном графе G. Докажем утверждение для $q+1$ аффектных вершин.

Непосредственно проверяется, что после выполнения строк 1–7 дуга (i, j) удаляется из матриц G и SPT, слайс $AffectedV$ хранит позиции $q+1$ аффектных вершин, все аффектные дуги удалены из матрицы SPT и новые расстояния от корня до всех аффектных вершин записаны в соответствующие строки матрицы $Dist$. Поскольку $AffectedV \neq \theta$, выполняем строку 8.

После выполнения строк 9–10 определяем позицию аффектной вершины k, для которой расстояние от корня имеет минимальное значение, и отмечаем ее позицию нулем в слайсе $AffectedV$. Аналогично базису после выполнения строк 11–18 вначале определяем хвост новой дуги (r, k), которая будет принадлежать новому кратчайшему пути от корня до вершины k. Затем включаем эту дугу в матрицу SPT. Ввиду леммы 3 после выполнения строки 19 в матрицу $Dist$ записываем новые расстояния от корня до тех вершин r, которые принадлежат дугам, заходящим в вершину k, и для которых уменьшаются значения $dist_{new}(r)$.

Теперь мы имеем только q аффектных вершин, позиции которых отмечены $'1'$ в слайсе $AffectedV$. По индуктивному предположению после обработки q аффектных вершин новое кратчайшее расстояние от корня до каждой аффектной вершины r записывается в r-ю строку матрицы $Dist$ и новая дуга, заходящая в вершину r, записывается в матрицу SPT. Кроме того, после включения каждой очередной вершины p в T_s мы перевычисляем в матрице $Dist$ веса путей от корня до тех вершин t, которые принадлежат дугам, выходящим из вершины p, чьи новые расстояния от корня уменьшаются. Это гарантирует нам правильный выбор нового кратчайшего пути от корня до каждой аффектной вершины, которая будет включаться в T_s.

Теорема доказана.

Оценим время выполнения процедуры DeleteArcSPT. Вначале оценим время выполнения вспомогательных процедур. Пусть h – число битов для кодирования бесконечности, а k – число аффектных вершин, которые появляются после удаления дуги (i, j) из дерева кратчайших путей. Вспомогательная процедура AffectedVertSPT выполняется с трудоемкостью $O(k)$. Вспомогательная процедура NewDistSPT выполняется с трудоемкостью $O(kh)$, поскольку она использует базовые процедуры, каждая из которых выполняется с трудоемкостью $O(h)$. Вспомогательная процедура LeavingArcsSPT выполняется с трудоемкостью $O(h)$. В процедуре DeleteArcSPT основной цикл `while SOME(AffectedV) do` (строки 8–20) выполняется с трудоемкостью $O(kh)$, поскольку внутри этого цикла каждая используемая базовая процедура и вспомогательная процедура LeavingArcsSPT выполняются с трудоемкостью $O(h)$.

6. Заключение

В данной работе построен ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после удаления из ориентированного графа одной дуги. Этот алгоритм использует простую структуру данных, которая позволяет выполнять доступ к данным по содержимому памяти. Ассоциативный алгоритм представлен на STAR–машине в виде процедуры DeleteArcSPT, корректность которой доказана. Мы показали, что эта процедура выполняется на STAR–машине за время $O(hk)$, где h – число битов для кодирования бесконечности, а k – число аффектных вершин, для которых строятся новые кратчайшие пути. Реализация этой процедуры упрощена, в частности, за счет того, что аффектные вершины образуют компоненту связности, включающую голову удаленной дуги.

Планируется построить ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления новой дуги в ориентированный граф.

Литература

[1] G. Ramalingam. Bounded Incremental Computation. Lecture Notes in Computer Science, Berlin: Springer–Verlag, Vol. 1089, 1996, pp. 30–51.

[2] G. Ramalingam, T. Reps. An incremental algorithm for a generalization of the shortest paths problem. Journal of Algorithms, Academic Press. Vol. 21, 1996, pp. 267–305.

[3] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni. Semi–dynamic algorithms for maintaining single source shortest paths trees. Algorithmica, Berlin: Springer–Verlag, Vol. 25, 1998, pp. 250-274.

[4] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms, Academic Press. Vol. 34, 2000, pp. 351-381.

[5] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. Journal of Algorithms, Elsevier Science, Vol. 49, 2003, pp. 86–113.

[6] A. S. Nepomniaschaya. Associative version of the Ramalingam decremental algorithm for dynamic updating the single-sink shortest paths subgraph. In: Proc. of the 10-th International Conference on Parallel Computing Technologies, PaCT-2009, Novosibirsk, Russia. Lecture Notes in Computer Science, Berlin: Springer-Verlag, Vol. 5698, 2009. pp. 257–268.

[7] A. S. Nepomniaschaya. Language STAR for associative and parallel computation with vertical data processing. In: Proc. of the International Conference “Parallel Computing Technologies”, World Scientific, Singapore, 1991, pp. 258–265.

[8] E. W. Dijkstra. A note on two problems in connection with graphs, Numerische Mathematik. Vol 1, 1959, pp. 269–271.

[9] C.C. Foster. Content Addressable Parallel Processors, New York: Van Nostrand Reinhold Company, 1976.

[10] А. Ш. Непомнящая, М. А. Владыко. Сравнение моделей ассоциативных параллельных вычислениний. Программирование: М: Наука, No. 6, 1997, с. 41–50.

[11] A. S. Nepomniaschaya. Solution of path problems using associative parallel processors. In: Proc. of the Intern. Conf. on Parallel and Distributed Systems, ICPADS’97, Korea, Seoul, IEEE Computer Society Press, 1997, pp. 610–617.

[12] A. S. Nepomniaschaya, M. A. Dvoskina. A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors. Fundamenta Informaticae, IOS Press, Vol. 43, 2000, pp. 227-243.

[13] Nepomniaschaya A. S.: Simultaneous Finding the Shortest Paths and Distances in Directed Graphs Using Associative Parallel Processors. In: Proc. of the Intern. Conf. "Information Visualization", IV 2003, pp. 665–670. IEEE Computer Society, Los Alamitos (2003)