УДК 519.172

А.Ш. НЕПОМНЯЩАЯ

АССОЦИАТИВНАЯ ВЕРСИЯ АЛГОРИТМА РАМАЛИНГАМА ДЛЯ ДИНАМИЧЕСКОЙ ОБРАБОТКИ ПОДГРАФА КРАТЧАЙШИХ ПУТЕЙ ПОСЛЕ ДОБАВЛЕНИЯ К ГРАФУ НОВОЙ ДУГИ

Ключевые слова: *ориентированный взвешенный граф, подграф кратчайших путей, матрица смежности, инкрементальный алгоритм, ассоциативный параллельный процессор.*

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 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 при изменении графа без его полного перевычисления после каждого локального изменения в нем. Типичными операциями для этого вида преобразований являются добавление или удаление одной дуги либо изменение ее веса. Если граф представляет коммуникационную или транспортную сеть, то добавление или удаление дуги отражает такие 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 ее веса, и декрементальным, если происходит только удаление дуги или увеличение ее веса.

В работах [1, 2] для графов с произвольными вещественными весами дуг построены полностью динамические алгоритмы для обработки кратчайших путей из одной вершины. При этом предполагается, что граф не имеет циклов отрицательной длины до и после обработки входной информации. Авторы используют модель, учитывающую сложность выходной информации после модификации входа. Сложность динамического алгоритма оценивается суммой числа аффектных вершин, которые модифицируют свое кратчайшее расстояние от корня s после преобразования входной информации, и числа дуг, у которых по крайней мере одна из вершин является аффектной.

В [3] построены полудинамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей (sp-tree) как в ориентированном, так и в неориентированном графах с положительными вещественными весами ребер. Декрементальные алгоритмы применяются для планарных графов, а инкрементальные — для произвольных. Временная сложность алгоритмов оценивается в терминах амортизированного времени.

В статье [4] построены полностью динамические алгоритмы для обработки кратчайших расстояний и дерева кратчайших путей как в ориентированном, так и в неориентированном графах с положительными действительными весами ребер после выполнения произвольной последовательности преобразований ребер. Стоимость операции обработки задается как функция от числа преобразований выходных данных с использованием понятия k-ограниченной вычисляющей функции (k-bounded accounting function).

В [5] построен полностью динамический алгоритм для обработки кратчайших путей из корня s в ориентированном графе с произвольными весами дуг. Используются два новых алгоритма: для удаления дуги и увеличения ее веса (который явно работает с циклами нулевой длины), а также для добавления дуги и уменьшения ее веса (в котором явно используются циклы отрицательной длины). Сложность операции обработки задается как функция от структуры графа и числа обработок выходной информации. Все упомянутые выше алгоритмы используют динамическую версию алгоритма Дейкстры [6].

В работе [7] приведены два инкрементальных метода, которые позволяют преобразовать хорошо известные статические алгоритмы Дейкстры, Беллмана–Форда и Исопо–Пейпа в новые динамические алгоритмы для обработки дерева кратчайших путей после изменения веса одной дуги.

В настоящей статье рассмотрены ориентированный взвешенный граф G с выделенным стоком и подграф кратчайших путей $SP(G)$, состоящий из всех кратчайших путей из каждой вершины в сток. Построена ассоциативная версия алгоритма Рамалингама [1] для динамической обработки $SP(G)$ после добавления одной дуги к графу G. В качестве модели вычисления использована STAR-машина [8], которая моделирует работу ассоциативных (контекстно-адресуемых) параллельных систем типа SIMD с последовательно-поразрядной (вертикальной) обработкой информации и простыми однобитовыми процессорными элементами. Такая архитектура наилучшим образом приспособлена к решению задач нечисловой обработки, к которым относятся реляционные базы данных, базы знаний, теория графов, экспертные системы и обработка сейсмических данных. Предложена простая и естественная структура данных, позволяющая построить на STAR-машине эффективную параллельную реализацию инкрементального алгоритма Рамалингама. Ассоциативная версия этого алгоритма описана в виде процедуры InsertNewArc, корректность которой доказывается. Процедура выполняется на STAR-машине с трудоемкостью $O(hk)$, где h — число битов для кодирования бесконечности, а k — число вершин, для которых в $SP(G)$ изменяются кратчайшие расстояния до стока после добавления к графу G новой дуги. При этом предполагается, что каждая элементарная операция STAR-машины выполняется за единицу времени. Приведены основные достоинства ассоциативной версии инкрементального алгоритма Рамалингама.

МОДЕЛЬ АССОЦИАТИВНОГО ПАРАЛЛЕЛЬНОГО ПРОЦЕССОРА

Опишем кратко используемую модель (подробное описание и сравнение ее с другими моделями ассоциативной обработки приведены в [9]). Модель определяется как абстрактная STAR-машина типа SIMD с вертикальной обработкой информации, основными компонентами которой являются:

- последовательное устройство управления, где 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 в таблицах нумеруются сверху вниз, а столбцы — слева направо. В матричную память можно загружать несколько таблиц.

Устройство ассоциативной обработки представляет собой совокупность h вертикальных одноразрядных регистров длиной p, каждый из которых можно

представить как массив, состоящий из одного столбца. Обработка информации происходит следующим образом. Из данной таблицы в определенном порядке извлекаются 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**, которую далее будем называть слайсом, моделируется доступ к таблице по столбцам, а с помощью переменной типа **word** — доступ по строкам. С каждой переменной типа **table** ассоциируется матрица из n строк и k столбцов, где $n \le p$. С каждым слайсом ассоциируется последовательность из p компонентов, принадлежащих множеству $\{0,1\}$.

Пусть X и Y — слайсы, 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-й компонент в слайсе Y $(1 \le i \le p)$;
- FND(Y) выдает порядковый номер i позиции первой (или старшей) единицы в слайсе Y, где $i \ge 0$;
- 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) применяются 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 — конъюнкция, X or Y — дизъюнкция, not X — отрицание, X xor Y — сравнение по модулю два. Также используется предикат SOME(Y), который возвращает значение **true** тогда и только тогда, когда в слайсе Y имеется хотя бы один компонент «1». Условимся, что $Y \ne \varnothing$ означает, что предикат SOME(Y) возвращает значение **true**.

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

В настоящей статье использована новая версия языка STAR. В ней расширен список предикатов и операций для переменных типа **word**. Следуя Поттеру [10], для переменных u и v типа **word** общепринятым способом введены предикаты: LESS (u, v), EQ(u, v), NOTEQ(u, v) и GREAT(u, v).

Для переменных u и v типа **word** введем операцию побитового сложения ADD(u, v), которая может использоваться только в правой части оператора присваивания.

Для переменной u типа **word** введем операцию REP(i, j, u_0, u), которая в заданной строке u заменяет ее подстроку $u(i)u(i+1)\dots u(j)$ на строку u_0 длиной $j-i+1$ и записывает результат замены в строку u.

Пусть T — переменная типа **table**. Используем операции ROW(i, T) и COL(i, T), которые в матрице T выделяют i-ю строку и i-й столбец соответственно.

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

Приведем группу базовых процедур [11, 12], которые выполняются с трудоемкостью $O(k)$, где k — число битовых столбцов в соответствующей матрице, а также укажем с помощью глобального слайса X позиции анализируемых строк в соответствующих процедурах:

• MATCH(T, X, v, Z) одновременно определяет позиции строк в заданной матрице T, совпадающие с заданной строкой v, и возвращает слайс Z, в котором $Z(i) =$'1', тогда и только тогда, когда ROW$(i, T) = v$ и $X(i) =$'1';

• MIN(T, X, Z) одновременно определяет позиции строк в заданной матрице T, содержащих минимальный элемент, и возвращает слайс Z, в котором $Z(i) =$'1', тогда и только тогда, когда ROW(i, T) — минимальный элемент и $X(i) =$'1';

• SETMIN(T, F, X, Z) одновременно определяет позиции строк матрицы T, которые меньше соответствующих строк матрицы F, и возвращает слайс Z, в котором $Z(i) =$'1', тогда и только тогда, когда ROW$(i, T) <$ ROW(i, F) и $X(i) =$'1';

• HIT(T, F, X, Z) одновременно определяет позиции идентичных строк матриц T и F, используя слайс X, и возвращает слайс Z, в котором $Z(i) =$'1', тогда и только тогда, когда ROW$(i, T) =$ ROW(i, F) и $X(i) =$'1';

• TMERGE(T, X, F) записывает строки матрицы T, отмеченные '1' в слайсе X, в соответствующие строки матрицы F, при этом остальные строки матрицы F не меняются;

• TCOPY1(T, j, h, F) записывает h столбцов из матрицы T, начиная с $(1 + (j-1)h)$-го столбца, в результирующую матрицу F, где $j \geq 1$;

• CLEAR(h, T) записывает нули в каждый i-й столбец заданной матрицы T, где $1 \leq i \leq h$;

• 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 табл. 5.1 из [13];

• ADDC(T, X, v, F) одновременно добавляет двоичное слово v к строкам матрицы T, отмеченных '1' в слайсе X, и записывает результат в соответствующие строки матрицы F, в остальные строки матрицы F записываются нули;

• SUBTC(T, X, v, F) одновременно вычитает двоичное слово v из строк матрицы T, отмеченных '1' в слайсе X, и записывает результат в соответствующие строки матрицы F, в остальные строки матрицы F записываются нули. Этот алгоритм использует табл. 5.2 из [13].

ОСНОВНЫЕ ПОНЯТИЯ

Пусть $G = (V, E, wt)$ — ориентированный взвешенный граф, в котором $V = \{1, 2, \ldots, n\}$ — множество вершин, $E \subseteq V \times V$ — множество ориентированных ребер (дуг) и wt — функция веса. Полагаем, что $|V| = n$ и $|E| = m$.

Рассмотрим графы с выделенной вершиной s, называемой стоком.

Обозначим $e = (i, j)$ дугу, которая ориентирована от вершины i до вершины j. При этом вершина i называется головой дуги e (или отцом), а вершина j — ее хвостом (или сыном). Если $(i, j) \in E$, то вершина j называется смежной с вершиной i.

Матрицей смежности назовем квадратную матрицу $A = [a_{ij}]$ размера $n \times n$, элементы которой определяются следующим образом: $a_{ij} = 1$, если вершина i смежна с вершиной j, и $a_{ij} = 0$ — в противном случае.

Рассмотрим графы, дуги которых имеют положительный вес. Будем считать, что $wt(i, j) = \infty$, 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 графе G. Обозначим h число битов для кодирования этой суммы.

Путем из вершины v_1 к вершине s в графе G назовем последовательность вершин $v_1, v_2, \ldots, v_k = s$, в которой $(v_i, v_{i+1}) \in E$ для $1 \le i < k-1$ и $k > 1$. Кратчайшим путем из вершины i к стоку назовем путь с минимальной суммой весов его дуг.

Обозначим $dist(l)$ длину кратчайшего пути из вершины l к стоку. Пусть $SP(G)$ — подграф кратчайших путей из всех вершин графа G к стоку.

Следуя Рамалингаму [1], введем следующие обозначения и понятия.

Обозначим $outdegree_{SP(G)}(l)$ число дуг, которые выходят из вершины l в $SP(G)$.

Пусть $Succ(u) = \{z / u \to z \in E\}$ и $Pred(u) = \{x / x \to u \in E\}$.

Пусть дуга (i, j) добавляется в подграф кратчайших путей $SP(G)$. Вершина u называется аффектной в SP(G) тогда и только тогда, когда $dist(u, i) + wt(i, j) + dist_{\text{old}}(j) < dist_{\text{old}}(u)$, где $dist(u, i)$ — длина кратчайшего пути из вершины u к вершине i в новом графе, $wt(i, j)$ — вес дуги (i, j) и $dist_{\text{old}}(j)$ (соответственно $dist_{\text{old}}(u)$) — длина кратчайшего пути из вершины j (соответственно u) к стоку перед добавлением дуги (i, j) к графу G.

ИНКРЕМЕНТАЛЬНЫЙ АЛГОРИТМ РАМАЛИНГАМА ДЛЯ ОБРАБОТКИ ПОДГРАФА КРАТЧАЙШИХ ПУТЕЙ С ОДНИМ СТОКОМ

Алгоритм Рамалингама для динамической обработки подграфа кратчайших путей после добавления дуги (i, j) к графу G использует структуру данных кучу PriorityQueue (очередь с приоритетами), в которой каждой аффектной вершине будет присвоен ключ. Ключом для аффектной вершины u будет кратчайшее расстояние от нее до вершины i. Первоначально PriorityQueue $= \varnothing$.

Алгоритм работает следующим образом.

Вначале дуга (i, j) добавляется к графу G. Если $wt(i, j) + dist(j) = dist(i)$, то дуга (i, j) добавляется к $SP(G)$ и $outdegree_{SP}(i)$ увеличивается на единицу.

Если $wt(i, j) + dist(j) < dist(i)$, то $dist(i) := wt(i, j) + dist(j)$ и вершина i включается в кучу PriorityQueue с ключом, равным нулю.

Каждая аффектная вершина обрабатывается следующим образом. На каждой итерации в текущей куче выбирается аффектная вершина, скажем u, с минимальным ключом.

Затем из $SP(G)$ удаляются все дуги вида (u, x) и $outdegree_{SP}(u) := 0$.

Далее для каждой вершины $x \in Succ(u)$ проверяется справедливость соотношения $wt(u, x) + dist(x) = dist(u)$. Если оно выполняется, то в подграф кратчайших путей $SP(G)$ добавляется дуга (u, x) и $outdegree_{SP}(u)$ увеличивается на единицу.

Для каждой вершины $y \in Pred(u)$ проверяется справедливость соотношения $wt(y, u) + dist(u) = dist(y)$. Если оно выполняется, то в $SP(G)$ добавляется дуга (y, u) и $outdegree_{SP}(y)$ увеличивается на единицу. В противном случае проверяется справедливость неравенства $wt(y, u) + dist(u) < dist(y)$. Если оно выполняется, то $dist(y) := wt(y, u) + dist(u)$ и вершина y включается в кучу PriorityQueue с ключом, равным кратчайшему расстоянию от вершины y до вершины i. Если $y \in$ PriorityQueue, то предыдущий ключ для y заменяется на меньшее значение.

Алгоритм завершится, когда будут обработаны все аффектные вершины.

Приведем временную сложность инкрементального алгоритма Рамалингама. Пусть k — число аффектных вершин в подграфе кратчайших путей после добавления к графу новой дуги, а r — число аффектных вершин и аффектных дуг, у которых по крайней мере одна из вершин является аффектной. Тогда алгоритм Рамалингама выполняет $O((k + r)\log(k + r))$ арифметических операций.

Нетрудно заметить, что в процессе выполнения инкрементального алгоритма Рамалингама строится дерево с корнем i, состоящее из аффектных вершин, причем из каждой аффектной вершины r существует единственный путь к вершине i. Первоначально дерево состоит из корня i. На каждой итерации в строя-

щееся дерево добавляется та аффектная вершина l, для которой $dist(l) - dist(i)$ имеет минимальное значение.

Обозначим T_i дерево, которое будет построено в процессе динамической обработки подграфа кратчайших путей после добавления дуги (i, j) к графу G.

АССОЦИАТИВНАЯ ВЕРСИЯ ИНКРЕМЕНТАЛЬНОГО АЛГОРИТМА РАМАЛИНГАМА

Чтобы построить ассоциативную версию инкрементального алгоритма Рамалингама для динамической обработки подграфа кратчайших путей, используем следующую структуру данных:

• матрица смежности G размера $n \times n$, каждый i-й столбец которой хранит с помощью '1' всех сыновей вершины i;

• матрица смежности SP размера $n \times n$, каждый i-й столбец которой хранит с помощью '1' сыновей вершины i, принадлежащих подграфу кратчайших путей;

• матрица *Weight* размера $n \times hn$, 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* размера $n \times hn$, 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* размера $n \times h$, каждая i-я строка которой хранит кратчайшее расстояние от вершины i до стока.

Заметим, что каждое i-е поле матрицы *Weight* хранит веса дуг, выходящих из вершины i, тогда как i-е поле матрицы *Cost* хранит веса дуг, заходящих в вершину i.

Приведем следующее свойство матриц G и SP.

Свойство 1. В каждой i -й строке матриц G и SP отмечены '1' головы дуг, заходящих в вершину i.

Пусть дуга (i, j) добавлена к графу G.

Ассоциативный параллельный алгоритм (скажем, алгоритм **A**) для обработки дуг, выходящих из аффектной вершины k, выполняет следующие шаги.

Шаг 1. С помощью слайса (скажем, Z) сохранить позиции всех дуг, которые в графе G выходят из вершины k.

Шаг 2. Одновременно определить длину различных путей, ведущих из вершины k к стоку s и включающих дугу, отмеченную '1' в слайсе Z.

Шаг 3. С помощью слайса (скажем, Y) сохранить позиции тех дуг (k, l), для которых $dist(k) = wt(k, l) + dist(l)$.

Шаг 4. Включить в матрицу SP позиции дуг, отмеченных '1' в слайсе Y.

На STAR-машине этот алгоритм реализован в виде процедуры UpdateOutgoingArcs.

Ассоциативный параллельный алгоритм (скажем, алгоритм **B**) для обработки дуг, заходящих в аффектную вершину k, выполняет следующие шаги.

Шаг 1. С помощью слайса (скажем, Z) сохранить головы дуг, которые в графе G заходят в аффектную вершину k.

Шаг 2. Для всех вершин l, отмеченных '1' в слайсе Z, одновременно определить новые кратчайшие расстояния от l до стока в каждом пути, начинающимся дугой (l, k).

Шаг 3. Добавить в матрицу SP позиции тех дуг вида (r, k), для которых $dist(r) = wt(r, k) + dist(k)$.

Шаг 4. С помощью слайса (скажем, Y) сохранить позиции тех вершин r, отмеченных '1' в слайсе Z, для которых $dist_{\text{new}}(r) < dist_{\text{old}}(r)$. Затем записать $dist_{\text{new}}(r)$ в соответствующие строки матрицы *Dist*.

На STAR-машине этот алгоритм реализован в виде процедуры UpdateIncomingArcs.

Рассмотрим ассоциативный параллельный алгоритм для динамической обработки подграфа кратчайших путей после добавления к графу G новой дуги (i, j) с весом $v0$. Пусть переменная $v2$ хранит кратчайшее расстояние от вершины i до стока в исходном $SP(G)$. Алгоритм использует слайс $Z1$, который хранит позиции текущих аффектных вершин, и матрицу *Queue* размера $n \times h$, каждая l-я строка которой хранит кратчайшее расстояние от вершины l до вершины i в текущем подграфе кратчайших путей.

Алгоритм выполняет следующие шаги.

Шаг 1. Добавить позицию дуги (i, j) в матрицу G, а ее вес $v0$ добавить в матрицы *Weight* и *Cost*.

Шаг 2. Определить расстояние (скажем, $v3$) от вершины i до стока, когда путь проходит через вершину j.

Шаг 3. Если $v2 = v3$, то добавить позицию дуги (i, j) в матрицу *SP*. Затем перейти на выход.

Шаг 4. Если $v3 < v2$, то записать значение $v3$ в i-ю строку матрицы *Dist*, установить для аффектной вершины i максимальный приоритет в матрице *Queue* и включить вершину i в слайс (скажем, $Z1$).

Шаг 5. Пока $Z1 \neq \varnothing$, обработать аффектные вершины с учетом их новых расстояний до стока следующим образом:

- выделить в матрице *Queue* вершину с максимальным приоритетом (скажем, k) и удалить ее из слайса $Z1$;
- одновременно удалить из матрицы *SP* все дуги, выходящие из вершины k;
- с помощью алгоритма **A** найти позиции дуг вида (k, l), для которых $dist(k) = wt(k, l) + dist(l)$, и добавить их в матрицу *SP*;
- с помощью алгоритма **B** сначала одновременно найти позиции тех дуг вида (r, k), для которых $dist(r) = wt(r, k) + dist(k)$, и добавить их в матрицу *SP*. Затем одновременно найти позиции новых аффектных вершин l и записать $dist_{\text{new}}(l)$ в соответствующие строки матрицы *Dist*;
- для новых аффектных вершин l одновременно вычислить значения $dist(l)$ — $dist(i)$ и записать их в соответствующие строки матрицы *Queue*;
- включить новые аффектные вершины в слайс $Z1$.

На STAR-машине этот алгоритм реализован в виде процедуры InsertNewArc.

ПРЕДСТАВЛЕНИЕ НА STAR-МАШИНЕ АССОЦИАТИВНОЙ ВЕРСИИ ИНКРЕМЕНТАЛЬНОГО АЛГОРИТМА РАМАЛИНГАМА

Рассмотрим две вспомогательные процедуры, реализующие алгоритмы **A** и **B**, а затем основную процедуру для динамической обработки подграфа кратчайших путей после добавления к исходному графу новой дуги.

Вспомогательная процедура UpdateOutgoingArcs. Зная текущую обрабатываемую вершину k, число битов h для кодирования бесконечности и текущие матрицы G, *SP*, *Weight* и *Dist*, процедура возвращает обработанную матрицу *SP*.

```
procedure UpdateOutgoingArcs(h,k: integer; G: table; Weight: table;
Dist: table; var SP: table);
/* Здесь h — число битов для кодирования бесконечности, k — номер
   обрабатываемой вершины. */
var W1,W2: table; v: word(Dist); Y, Z: slice(G);
1. Begin Z := COL(k,G);
2.       TCOPY1(Weight,h,k,W1);
3.       ADDV(W1,Dist,Z,W2);
```

```
/* Матрица W2 хранит различные расстояния от вершины k до стока. */
4.      v:=ROW(k,Dist);
/* Строка v хранит кратчайшее расстояние от вершины k до стока. */
5.      MATCH(W2,Z,v,Y);
/* В слайсе Y отмечены '1' те вершины l графа G,
        для которых dist(k) = wt(k,l) + dist(l). */
6.      COL(k,SP):=Y;
/* Дуги вида (k,l) добавляются в SP. */
7.  End;
```

Утверждение 1. Пусть заданы число битов h для кодирования бесконечности и номер обрабатываемой вершины k, а также текущие матрицы G, SP, $Weight$ и $Dist$. Тогда после выполнения процедуры UpdateOutgoingArcs в матрицу SP добавятся те дуги вида (k,l), для которых выполнится соотношение

$$dist(k) = wt(k,l) + dist(l). \qquad (1)$$

Доказательство. От противного. Пусть в графе G есть дуга (k,r), которая удовлетворяет соотношению (1), однако после завершения процедуры UpdateOutgoingArcs эта дуга не будет включена в матрицу SP. Докажем, что это противоречит выполнению нашей процедуры.

Действительно, так как $(k,r) \in G$, то после выполнения строки 1 $Z(r)$ ='1'. После выполнения строк 2 и 3 в r-й строке матрицы $W2$ будет записана длина пути из вершины k к стоку, включающего дугу (k,r). По предположению дуга (k,r) удовлетворяет соотношению (1). Поэтому после выполнения процедуры MATCH получим $Y(r)$ ='1'. Тогда после выполнения строки 6 дуга (k,r) будет добавлена в матрицу SP. Это противоречит нашему допущению.

Вспомогательная процедура UpdateIncomingArcs. Зная текущую обрабатываемую вершину k, число битов h для кодирования бесконечности и текущие матрицы G, SP, $Cost$ и $Dist$, процедура возвращает слайс Y и обработанные матрицы $Dist$ и SP.

```
procedure UpdateIncomingArcs(h,k: integer; G: table; Cost: table;
var Y: slice(G); var Dist: table; var SP: table);
var X,Z: slice(G);
    v: word(G);
    v1: word(Dist);
    W,W1: table;
1. Begin v := ROW(k,G); Z := CONVERT(v);
/* Слайс Z хранит головы дуг, которые заходят в вершину k. */
2.      v1 := ROW(k,Dist);
/* Строка v1 хранит кратчайшее расстояние от k до s. */
3.      TCOPY1(Cost,k,h,W1);
/* Матрица W1 хранит k-е поле матрицы Cost. */
4.      ADDC(W1,Z,v1,W);
/* В каждой l-й строке матрицы W, отмеченной '1' в слайсе Z, записано
        новое расстояние от l до стока. */
5.      HIT(W,Dist,Z,X);
/* В слайсе X отмечены '1' позиции тех строк матрицы Dist, в которых
        dist(l) = wt(l,k) + dist(k). */
6.      v := CONVERT(X);
7.      ROW(k,SP) := v;
/* В матрицу SP добавляются дуги вида (r,k), для которых
        dist(r) = wt(r,k) + dist(k). */
8.      SETMIN(W,Dist,Z,Y);
/* В слайсе Y отмечены '1' позиции тех вершин, у которых новое
        расстояние до стока уменьшилось. */
```

```
9.      TMERGE(W,Y,Dist);
/* В l-ю строку матрицы Dist записывается новое значение dist(l) тогда и
      только тогда, когда Y(l)='1'. */
10. End;
```

Утверждение 2. Пусть заданы число битов h для кодирования бесконечности и номер текущей обрабатываемой вершины k, а также заданы текущие матрицы G, SP, $Cost$ и $Dist$. Тогда процедура UpdateIncomingArcs возвращает слайс Y и измененные матрицы $Dist$ и SP. При этом в матрицу SP добавляются дуги вида (l,k), для которых $dist(l) = wt(l,k) + dist(k)$, слайс Y хранит головы тех дуг (r,k), для которых $dist_{\text{new}}(r) < dist_{\text{old}}(r)$, и новые расстояния $dist_{\text{new}}(r)$ записываются в матрицу $Dist$.

Доказательство. Будем доказывать от противного. Пусть вначале $(l,k) \in G$ и $dist(l) = wt(l,k) + dist(k)$. Однако после завершения процедуры UpdateIncomingArcs дуга (l,k) не будет добавлена в матрицу SP. Докажем, что это противоречит выполнению нашей процедуры.

Действительно, согласно свойству 1 в k-й строке матрицы G отмечены '1' те вершины, из которых выходит дуга в вершину k. Поэтому в результате выполнения строки 1 слайс Z хранит головы дуг, которые заходят в вершину k. Так как $(l,k) \in G$, то $Z(l) = $'1'. После выполнения строки 2 переменная $v1$ хранит кратчайшее расстояние от вершины k до стока. После выполнения строки 3 в l-й строке матрицы $W1$ будет записан вес дуги (l,k). Непосредственно проверяется, что после выполнения строки 4 в l-й строке матрицы W, отмеченной '1' в слайсе Z, будет записано новое кратчайшее расстояние от вершины l до стока, причем кратчайший путь начинается дугой (l,k). По предположению $dist(l) = wt(l,k) + dist(k)$. Поэтому после выполнения строк 5 и 6 получаем $v(l) = $'1'. Тогда после выполнения строки 7 l-й бит k-й строки матрицы SP будет равен '1'. Это означает, что дуга (l,k) добавлена в матрицу SP. Это противоречит нашему допущению.

Пусть теперь $(r,k) \in G$ и $dist_{\text{new}}(r) < dist_{\text{old}}(r)$. Однако после завершения процедуры UpdateIncomingArcs r-я строка в матрице $Dist$ не изменилась. Докажем, что это противоречит выполнению этой процедуры.

Действительно, поскольку $dist_{\text{new}}(r) < dist_{\text{old}}(r)$, то после выполнения базовой процедуры $\text{SETMIN}(W, Dist, Z, Y)$ (строка 8) получаем $Y(r) = $'1'. Тогда после выполнения строки 9 получаем $\text{ROW}(r, Dist) = dist_{\text{new}}(r)$. Это противоречит нашему допущению.

Утверждение 2 доказано.

Основная процедура InsertNewArc. Приведем динамическую обработку подграфа кратчайших путей после добавления к исходному графу новой дуги.

```
procedure InsertNewArc(h,i,j: integer; v0: word(Dist);
var Weight,Cost: table; var G, SP: table; var Dist: table);
/* Дуга (i,j) весом v0 добавляется в граф G, h — число битов для
      кодирования бесконечности. */
var v1,v2,v3: word(Dist);
      v4: word(Weight);
      k: integer;
      X,Y,Y1,Z1,Z2: slice(G);
      W1,W2,Queue: table;
1. Begin CLR(Z1); CLR(Y1);
2.      CLEAR(h,Queue);
3.      X := COL(i,G); X(j) := '1'; COL(i,G) := X;
/* Дуга (i,j) добавляется в граф G. */
4.      v4:=ROW(j,Weight);
5.      REP(1+(i –1)h,ih,v0,v4);
6.      ROW(j,Weight) := v4;
```

```
/* Вес дуги (i, j) добавляется в матрицу Weight. */
7.      v4 := ROW(i,Cost);
8.      REP(1+(j –1)h,jh,v0,v4);
9.      ROW(i,Cost) := v4;
/* Вес дуги (i, j) добавляется в матрицу Cost. */
10.     v1 := ROW(j,Dist); v2 := ROW(i,Dist);
/* Здесь v1 хранит dist_old(j), а v2 хранит dist_old(i).*/
11.     v3 := ADD(v0,v1);
/* Здесь v3 — расстояние от вершины i до стока, когда путь проходит
       через вершину j. */
12.     if EQ(v3,v2) then
13.         begin X := COL(i,SP); X(j) := '1';
14.             COL(i,SP) := X;
/* Дуга (i, j) добавляется в матрицу SP. */
15.         end;
16.     if LESS(v3,v2) then
17.         begin ROW(i,Dist) := v3; Z1(i) := '1';
/* Для вершины i в матрице Queue устанавливается максимальный
       приоритет. */
18.         end;
19.     while SOME (Z1) do
20.         begin MIN(Queue,Z1,Z2);
21.             k := FND(Z2); Z1(k) := '0';
22.             COL(k,SP) := Y1;
/* Из матрицы SP удаляются все дуги, выходящие из k. */
23.             UpdateOutgoingArcs(h,k,G,Weight,Dist,SP);
/* В матрицу SP добавляются те дуги вида (k, l),
       для которых dist(k) = wt(k, l) + dist(l). */
24.             UpdateIncomingArcs(h,k,G,Cost,Y,Dist,SP);
25.             v2:=ROW(i,Dist);
/* Строка v2 хранит текущее кратчайшее расстояние от вершины i до стока. */
26.             SUBTC(Dist,Y,v2,W2);
/* В каждой l-й строке матрицы W2, отмеченной '1' в слайсе Y, хранится
       величина dist(l) – dist(i). */
27.             TMERGE(W2,Y,Queue);
/* В матрицу Queue записываются приоритеты для новых аффектных
       вершин. */
28.             Z1 := Z1 or Y;
/* Новые аффектные вершины добавляются в слайс Z1. */
29.         end;
30. End;
```

Теорема. Пусть ориентированный взвешенный граф задан в виде матрицы смежности G и матрицы весов *Weight*. Пусть также заданы матрицы *Cost*, *SP* и *Dist* и число битов h для кодирования бесконечности. Пусть дуга (i, j) добавляется к заданному графу. Тогда после выполнения процедуры InsertNewArc дуга (i, j) будет добавлена к матрице *SP*, а ее вес будет включен в матрицы *Weight* и *Cost*. Кроме того, матрицы *Dist* и *SP* будут обработаны согласно алгоритмам **A** и **B**.

Доказательство. Будем доказывать индукцией по числу аффектных вершин l, которые включаются в строящееся дерево с корнем i.

Базис индукции проверяем для случая $l \le 1$, т.е. в графе G никакая дуга не заходит в вершину i.

Непосредственно проверяется, что после выполнения строк 1–9 дуга (i, j) добавляется в матрицу G, а ее вес — в матрицы *Weight* и *Cost*. В результате выполнения строк 10 и 11 переменная $v3$ будет хранить новое кратчайшее расстояние от вершины i до стока, когда этот путь проходит через вершину j.

Теперь сравним новое расстояние от вершины i до стока с тем расстоянием, которое хранится в переменной $v2$. Если $v3 > v2$, то ничего не надо пересчитывать. Поэтому рассмотрим следующие два случая.

Случай 1. Пусть $v3 = v2$. Тогда текущего дерева нет. После выполнения строк 13–15 позиция дуги (i, j) добавляется в матрицу SP. Затем переходим к выполнению строки 19. Так как после выполнения строки 1 слайс $Z1$ состоит из нулей и его содержимое не менялось до выполнения строки 19, то цикл **while** SOME(Z1) **do** (строки 19–30) не будет выполняться. Поэтому попадаем на конец процедуры InsertNewArc.

Случай 2. Пусть $v3 < v2$. Тогда дерево состоит из единственной вершины i. После выполнения строк 17 и 18 новое кратчайшее расстояние от вершины i до стока записывается в i-ю строку матрицы $Dist$ и $Z1(i) =$ '1'. После этого выполняется цикл **while** SOME($Z1$) **do** (строка 19). Непосредственно проверяется, что после выполнения строк 20 и 21 $k = i$ и слайс $Z1$ состоит из нулей. После выполнения строки 22 i-й столбец матрицы SP состоит из нулей ввиду выполнения операции CLR($Y1$) в строке 1. Поэтому все дуги, выходящие из вершины i, удаляются из матрицы SP. По утверждению 1 после выполнения вспомогательной процедуры UpdateOutgoingArcs (строка 23) в матрицу SP будут добавлены те дуги вида (i, l), для которых $dist(i) = wt(i, l) + dist(l)$. В частности, дуга (i, j) будет добавлена в матрицу SP, поскольку новое расстояние $v3$ совпадает с текущим кратчайшим расстоянием от вершины i до стока, которое записано в i-й строке матрицы $Dist$. По предположению только вершина i является аффектной. Поэтому после выполнения вспомогательной процедуры UpdateIncomingArcs (строка 24) результирующий слайс Y будет нулевым. Поэтому строки 27 и 28 не будут выполняться. Так как после выполнения строки 29 слайс $Z1$ будет нулевым, попадаем на конец процедуры InsertNewArc.

Значит, если $l = 1$ и $v3 < v2$, то вначале новое кратчайшее расстояние $v3$ от вершины i до стока записывается в i-ю строку матрицы $Dist$. Затем из матрицы SP удаляются все дуги, выходящие из вершины i. После этого в SP добавляются все дуги вида (i, p), для которых $dist(i) = wt(i, p) + dist(p)$.

Шаг индукции. Пусть теорема верна для случая, когда из текущего дерева T удаляется не более l вершин. Докажем справедливость утверждения, когда из текущего дерева будет удалено $l+1$ аффектных вершин.

Как и при доказательстве случая 2 базиса индукции, после выполнения строк 1–11 и 16–18 дуга (i, j) добавляется в матрицу G, а ее вес — в матрицы $Weight$ и $Cost$, переменная $v3$ хранит новое кратчайшее расстояние от вершины i до стока, значение $v3$ записывается в i-ю строку матрицы $Dist$ и для вершины i устанавливается максимальный приоритет в матрице $Queue$.

После выполнения строк 20–22 получаем, что $k = i$, $Z1 = \varnothing$ и из матрицы SP удаляются все дуги, выходящие из вершины i.

По утверждению 1 после выполнения вспомогательной процедуры UpdateOutgoingArcs (строка 23) в матрицу SP добавляются те дуги вида (i, p), для которых $dist(i) = wt(i, p) + dist(p)$.

По утверждению 2 после выполнения вспомогательной процедуры UpdateIncomingArcs (строка 24) слайс Y хранит головы тех дуг (r, i), для которых $dist_{\text{new}}(r) < dist_{\text{old}}(r)$, новые расстояния $dist_{\text{new}}(r)$ записываются в матрицу $Dist$ и дуги вида (r, i), для которых $dist(r) = wt(r, i) + dist(i)$, добавляются в матрицу SP.

После выполнения строк 25–29 получаем, что $Z1 = Y$ и приоритеты для новых аффектных вершин, отмеченных '1' в слайсе Y, записываются в матрицу $Queue$.

По индуктивному предположению после того, как из текущего дерева T будут удалены первые l аффектных вершин, для каждой аффектной вершины r в мат-

рицу SP будут добавлены дуги вида (p, r), для которых $dist(p) = wt(p, r) + dist(r)$, и дуги вида (r, q), для которых $dist(r) = wt(r, q) + dist(q)$, а в r-й строке матрицы $Dist$ будет записано новое кратчайшее расстояние от вершины r до стока. После того как в матрице $Queue$ будут обработаны первые l аффектных вершин, в слайсе $Z1$ останется единственный бит '1', указывающий позицию $(l+1)$-й аффектной вершины. Поскольку $Z1 \neq \varnothing$, обрабатываем эту аффектную вершину как и при доказательстве базиса индукции. Так как $Z1$ становится нулевым слайсом, переходим на конец процедуры.

Теорема доказана.

Оценим время выполнения процедуры InsertNewArc. Пусть h — число битов для кодирования бесконечности, а k — число аффектных вершин, которые появляются после добавления дуги (i, j) к графу G. Непосредственно проверяется, что вспомогательные процедуры UpdateOutgoingArcs и UpdateIncomingArcs, а также все базовые процедуры, которые используются в процедуре InsertNewArc, выполняются за время $O(h)$. Так как в процедуре InsertNewArc основной цикл **while** SOME(Z1) **do** (строки 19–30) выполняется k раз, то эта процедура выполняется с трудоемкостью $O(hk)$.

Сравним выполнение инкрементального алгоритма Рамалингама и его ассоциативной версии:

• алгоритм Рамалингама использует кучу, элементами которой являются аффектные вершины с ключом, а ассоциативная версия — матрицу $Queue$, каждая r-я строка которой хранит расстояние от вершины r до вершины i;

• в алгоритме Рамалингама для каждой аффектной вершины u из $SP(G)$ последовательно удаляются все дуги вида (u, x). В ассоциативной версии позиции таких дуг одновременно удаляются из $SP(G)$;

• в алгоритме Рамалингама для каждой вершины $x \in Succ(u)$ дуга (u, x) добавляется в $SP(G)$ тогда и только тогда, когда выполняется соотношение $wt(u, x) + dist(x) = dist(u)$. В ассоциативной версии позиции дуг, выходящих из вершины u и удовлетворяющих этому соотношению, одновременно добавляются в подграф кратчайших путей $SP(G)$;

• в алгоритме Рамалингама каждая вершина $y \in Pred(u)$ включается в кучу PriorityQueue тогда и только тогда, когда выполняется неравенство $wt(y, u) + dist(u) < dist(y)$. В ассоциативной версии одновременно определяются головы дуг, заходящих в вершину u и удовлетворяющих этому неравенству. Такие вершины одновременно включаются в множество аффектных вершин, а их приоритеты одновременно записываются в соответствующие строки матрицы $Queue$;

• в алгоритме Рамалингама для каждой вершины $y \in Pred(u)$ дуга (y, u) добавляется в $SP(G)$ тогда и только тогда, когда выполняется соотношение $wt(y, u) + dist(u) = dist(y)$. В ассоциативной версии одновременно определяются позиции дуг, заходящих в вершину u и удовлетворяющих этому соотношению. Позиции этих дуг одновременно добавляются в $SP(G)$.

ЗАКЛЮЧЕНИЕ

Предложена простая и естественная структура данных, позволяющая построить эффективную параллельную реализацию инкрементального алгоритма Рамалингама на STAR-машине, содержащей не более n процессорных элементов. Ассоциативная версия этого алгоритма описана в виде процедуры InsertNewArc, корректность которой доказана. Процедура выполняется на STAR-машине с трудоемкостью $O(hk)$, где h — число битов для кодирования бесконечности, а k — число аффектных вершин, для которых надо находить

новые кратчайшие пути к стоку после добавления к графу новой дуги. Эта оценка оптимальна, поскольку параметр h не меняется с ростом k, а обрабатывать надо все аффектные вершины. Инкрементальный алгоритм Рамалингама использует современную структуру данных, называемую кучей или очередью с приоритетами. Ассоциативная версия этого алгоритма эффективно моделирует обработку очереди с приоритетами на STAR-машине. Для построения процедуры InsertNewArc пришлось расширить язык STAR путем введения новых операций и предикатов для переменных типа **word**. Приведены основные достоинства ассоциативной версии инкрементального алгоритма Рамалингама.

Планируется построить ассоциативную версию алгоритма Фригиони и Итальяно [14] для обработки планарного графа после динамического включения или выключения некоторых его вершин.

СПИСОК ЛИТЕРАТУРЫ

1. Ramalingam G. Bounded incremental computation // Lecture Notes in Computer Sciences. — Heidelberg: Springer. — 1996. — **1089**. — 190 p.
2. Ramalingam G., Reps T. An incremental algorithm for a generalization of the shortest paths problem // J. of Algorithms, Academic Press. — 1996. — **21**. — P. 267–305.
3. Frigioni D., Marchetti-Spaccamela A., Nanni U. Semi-dynamic algorithms for maintaining single source shortest paths trees // Algorithmica. — 1998. — **25**. — P. 250–274.
4. Frigioni D., Marchetti-Spaccamela A., Nanni U. Fully dynamic algorithms for maintaining shortest paths trees // J. of Algorithms. Academic Press. — 2000. — **34**. — P. 351–381.
5. Frigioni D., Marchetti-Spaccamela A., Nanni U. Fully dynamic shortest paths in digraphs with arbitrary arc weights // J. of Algorithms. Elsevier Science. — 2003. — **49**. — P. 86–113.
6. Dijkstra E.W. A note on two problems in connection with graphs // Numerische Mathematik. — 1959. — N 1. — P. 269–271.
7. Narv'aez P., Siu K.-Y., Tzeng H.-Y. New dynamic algorithms for shortest paths tree computation // IEEE/ACM Trans. Networking. — 2000. — **8**. — P. 734–746.
8. Nepomniaschaya A.S. Language STAR for associative and parallel computation with vertical data processing // Proc. of the Intern. Conf. Parallel Computing Technologies. — Singapore: World Scientific, 1991. — P. 258–265.
9. Непомнящая А.Ш., Владыко М.А. Сравнение моделей ассоциативного вычисления // Программирование. — 1997. — № 6. — С. 41–50.
10. Potter J.L. Associative computing: a programming paradigm for massively parallel computers. Kent State University. — New York; London: Plenum Press, 1992. — 286 p.
11. Nepomniaschaya A.S. Solution of path problems using associative parallel processors // Intern. Conf. on Parallel and Distributed Systems, ICPADS'97. — New York: IEEE Computer Society Press, 1997. — P. 610–617.
12. Nepomniaschaya A.S., Dvoskina M.A. A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. — Amsterdam: IOS Press. — 2000. — **43**. — P. 227–243.
13. Фостер К. Ассоциативные параллельные процессоры. — М.: Энергоиздат / Пер. с англ., 1981. — 240 с.
14. Frigioni D., Italiano G.F. Dynamically switching vertices in planar graphs // Algorithmica. — 2000. — **28**, N 1. — P. 76–106.

Поступила 18.07.2009

После доработки 20.01.2011