ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ ПРОБЛЕМИ
ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ПІДТРИМКИ
ПРИЙНЯТТЯ РІШЕНЬ

УДК 519.168

ВРЕМЯ РАБОТЫ АЛГОРИТМА КРАСКАЛА С ДРЕВОВИДНОЙ И СПИСОЧНОЙ СТРУКТУРОЙ ДАННЫХ

А.Н. ТРОФИМЧУК, В.А. ВАСЯНИН

Путем численных экспериментов выполнено сравнение двух реализаций алгоритма Краскала, основанных на списочной (предложенный алгоритм) и древовидной (алгоритм Тарьяна) структуре данных и алгоритма Прима. Результаты сравнения позволяют утверждать, что для решения практических задач нахождение минимального или максимального остовного дерева (леса) алгоритмы со списочной структурой данных работают не хуже, а в большинстве случаев быстрее, чем алгоритмы с древовидной структурой. Показана практическая оценка сложности предложенного алгоритма, которая для связных графов составляет $O(e)$, где e — число ребер графа. Экспериментально доказано, что время работы алгоритма на связных разреженных графах сравнимо со временем «карманной» сортировки ребер (bucket sort). Выявлено, что предложенный алгоритм работает быстрее алгоритма Прима для графов с числом ребер не больше, чем $0{,}27v^2$, где v — число вершин графа. Экспериментальное исследование алгоритма на графах, содержащих от 499500 до 71994000 ребер, показало его высокую вычислительную эффективность, и он может быть рекомендован для решения практических задач на разреженных графах или сетях большой размерности.

ВВЕДЕНИЕ

В настоящее время известно достаточно много алгоритмов определения минимального остовного дерева на взвешенном графе (MST-дерева). Наиболее известными из них являются алгоритмы Борувки [1,2], Ярника-Прима-Дейкстры [3–5] и Краскала-Лобермана-Вайнберга [6,7]. Алгоритм Борувки был много раз переоткрыт другими авторами (Шоке, Флорек, Лукасевич, Перкал, Штейнгауз, Зубжицкий) и известен также как алгоритм Соллина [8]. Обширный библиографический обзор по истории развития MST-алгоритмов можно найти в работах Тарьяна [9], Грэхэма и Хелла [10].

Для построения современных алгоритмов используются АТД (абстрактные типы данных — abstract data type), которые включают различные комбинации традиционных данных, таких как переменные и массивы и специальных (указателей, списков, стеков, очередей, деревьев и т.д.). Для заданных классов АТД определяются функциональные операторы (процедуры) обработки внутренних структур данных, которые реализуются средствами используемого языка программирования. В книгах [11–16] можно найти

подробнейшее описание АТД, применяемых в алгоритмах построения остовных деревьев, оценки временной сложности алгоритмов для их различных реализаций, а также исчерпывающий обзор и исторические справки по алгоритмам нахождения остовных деревьев. Следует также отметить, что в современных объектно-ориентированных языках программирования, таких как C++, Java и др. имеются библиотеки со стандартными АТД и программами для решения задач нахождения оптимальных остовных деревьев [17–19].

Цель работы — экспериментально показать, что алгоритм Краскала, реализованный на списочных АТД [20] во многих практических случаях работает быстрее, чем алгоритм Тарьяна [21], основанный на древовидных АТД и сравнить время работы этих алгоритмов и алгоритма Прима на связных и несвязных, плотных и разреженных графах большой размерности.

ПОСТАНОВКА ЗАДАЧИ

Пусть задан простой (не содержащий петель и параллельных ребер) неориентированный граф $G(V,E)$ с множеством вершин V, $v=|V|$ и множеством ребер E, $e=|E|$, где v и e соответственно число вершин и ребер графа, а $|\cdot|$ — знак мощности множества. Граф может быть неполным и несвязным. Для графа задан вектор весов ребер $C=\| c_i \|$, $i=\overline{1,e}$, где $c_i \in Z^+$, Z^+ — множество неотрицательных целых чисел.

Требуется найти ациклический подграф (лес) графа G с минимальным или максимальным общим весом. В литературе принято называть такой ациклический подграф в случае связного графа MST-деревом, а в случае несвязного графа — MTS-лесом. В дальнейшем для упрощения изложения будем использовать для связных и несвязных графов общий термин MST (minimum/maximum spanning trees — минимум/максимум остовных деревьев).

АЛГОРИТМЫ ПОСТРОЕНИЯ MST-ДЕРЕВЬЕВ

В настоящее время наилучшими по быстродействию для разреженных графов являются алгоритмы Тарьяна [21] (без учета сортировки ребер), Габова, Галила, Спенсера и Тарьяна [22] и Чазелла [23, 24]. Дальнейшее развитие алгоритмов возможно за счет усложнения АТД и уменьшения времени выполнения отдельных операций, используемых при работе с их внутренними структурами данных [14, 15]. Уменьшение времени работы алгоритмов может быть достигнуто также за счет распараллеливания процесса построения MST, особенно в алгоритмах Борувки-Соллина.

Несмотря на существование достаточно эффективных MST-алгоритмов, интерес к разработке еще более быстрых алгоритмов не ослабевает [25, 26], (рассматривается современное положение дел в поиске линейного алгоритма построения MST). Это объясняется тем, что одновременно с постоянным ростом быстродействия и оперативной памяти современных ПК, появляется возможность решать более сложные оптимизационные задачи для графов

или сетей большой размерности (десятки тысяч вершин, сотни тысяч и миллионы ребер), в которых в качестве подзадач нужно многократно находить MST-деревья. Тогда алгоритм определения MST-деревьев с минимальной сложностью может значительно сократить время решения общей задачи.

Приведем реализацию «жадного» алгоритма Краскала, который находит остовное дерево с минимальным или максимальным весом за время $O(e)$ [20]. Введем обозначения: $\wedge$, $\vee$ — знаки логического «и», логического «или»; «$\leftarrow$» — знак операции присваивания, «!» — знак комментария, а $\{i | i = \overline{1,e}\}$ — означает «для всех i от единицы до e с шагом 1».

Для сортировки ребер будем применять алгоритм со сложностью порядка $O(e)$, основанный на идеях «карманной» сортировки или сортировки «вычерпыванием» и сортировки подсчетом. Такой тривиальный подход к сортировке ребер оправдан низкой временной сложностью и тем, что для большинства графов (сетей), характеризующих реальные объекты, разброс значений весов ребер, как правило, незначителен.

Для описания алгоритма сортировки определим следующие переменные и структуры данных. Обозначим: v, e — число вершин и неориентированных ребер в графе; $MINVAL$, $MAXVAL$ — минимальный и максимальный вес ребра; $IS(e)$, $ST(e)$, $C(e)$ — массивы, содержащие соответственно номера левых и правых вершин ребер и веса ребер. Для удобства эти массивы могут быть упорядочены так, чтобы номер левой вершины был меньше номера правой вершины, а номера левых и правых вершин увеличивались от начала к концу массивов; $U(MMAX,2)$ — массив, содержащий ссылки на списки номеров ребер, упорядоченных по неубыванию весов в массиве $E(e)$; $E(e)$ — массив, содержащий списки номеров ребер, упорядоченных по неубыванию весов.

Алгоритм сортировки не перемещает исходные значения в массивах IS, ST, C. Сортируются только ссылки на номера ребер в исходных массивах в порядке неубывания весов ребер. Для этого строится массив $E(e)$ списков номеров ребер. Поскольку может быть несколько ребер с одинаковым весом, массив содержит списки для ребер с одинаковым весом. В массиве $U(MMAX,2)$ элемент $U(i,1)$ содержит голову списка — номер первого ребра k в массиве $E(e)$ с весом i, а элемент $U(i,2)$ указывает на номер последнего ребра j в массиве $E(e)$ с весом i. При этом элемент $E(k)$ указывает на следующее ребро в $E(e)$ с таким же весом и т.д. Для последнего ребра j значение $E(j) = 0$ и означает конец списка ребер с одинаковым весом. Для того чтобы в начале массива U не было пустых элементов, выполняется масштабирование весов ребер. Поэтому значение $MMAX = MAXVAL - MINVAL + 1$ равно максимальному весу ребра после масштабирования плюс 1, а минимальный вес ребра будет всегда равен единице. Если веса ребер являются дробными неотрицательными числами, то с учетом необходимой точности вычислений, можно ввести соответствующие масштабные коэффициенты (например, умножить веса всех ребер на 1000, 10000 и т.д.) для перевода весов ребер в целые числа, а затем определить значение $MMAX$.

Алгоритм *SORT*. Сортировка ребер по неубыванию весов с временной сложностью $O(e)$.

1. $U \leftarrow 0$; $E \leftarrow 0$.
2. Для $\{i \mid i = \overline{1,e}\}$ выполнить шаги 3–6.
3. $l \leftarrow C(i) - MINVAL + 1$.
4. Если $U(l,1) = 0$, то $U(l,1) \leftarrow i$; иначе $E(U(l,2)) \leftarrow i$.
5. $U(l,2) \leftarrow i$.
6. Перейти к шагу 2. ! Конец цикла по i.
7. Конец.

Очевидно, что асимптотическая оценка временной сложности алгоритма *SORT* составляет $O(e)$ операций, а реальное время выполнения сортировки ребер будет зависеть от конкретного языка программирования алгоритма, компилятора, операционной системы и мощности компьютера на котором работает конкретная программа, реализующая алгоритм.

Рассмотрим алгоритм построения MST. Пусть MK — максимальное число независимых компонент графа, $MK = v/2$; NK — номер очередной порожденной компоненты; KK — количество связных компонент; NY — счетчик количества вершин в строимом минимальном остовном лесе; SV — признак наличия изолированных вершин в графе; $W(MK)$ — массив весов порожденных компонент. Определим следующие массивы: $A(v)$ — массив, содержащий номера связных компонент, в которые входят вершины i, $i = \overline{1,v}$; $H(MK,3)$ — массив, содержащий ссылки на списки номеров вершин компоненты MK в массиве $G(v)$; $G(v)$ — массив, содержащий списки номеров вершин связных компонент графа; $EK(MK,2)$ — массив, содержащий ссылки на списки номеров ребер компоненты MK в массиве $B(e)$; $B(e)$ — массив, содержащий списки номеров ребер связных компонент графа. Массивы $H(MK,3)$, $G(v)$ и $EK(MK,2)$, $B(e)$ устроены аналогично массивам $U(MMAX,2)$, $E(e)$. Элемент $H(i,3)$ содержит число вершин в i-й компоненте графа (i-ом поддереве). Введенные структуры данных позволяют за две операции сравнения определять, к какой компоненте принадлежат вершины ребра, включаемого в остовный лес на очередном шаге алгоритма, а также за одну операцию присваивания сливать две независимые компоненты в одну. При этом всегда компонента с меньшим числом вершин или ребер сливается в большую компоненту.

Алгоритм *LINK*. Построение остовного леса (дерева) с минимальным весом с использованием списочных структур данных.

1. $A \leftarrow 0$; $NK \leftarrow 1$; $KK \leftarrow 0$; $NY \leftarrow 0$; $G \leftarrow 0$; $B \leftarrow 0$; $W \leftarrow 0$; $SV \leftarrow 0$.
2. $KNOV \leftarrow 0$; $KPRI \leftarrow 0$; $KSLI \leftarrow 0$; $KOST \leftarrow 0$; $KPUS \leftarrow 0$; $KSUZ \leftarrow 0$.
3. Для $\{i \mid i = \overline{1,MMAX}\}$ выполнить шаги 4–31.
4. $ND \leftarrow U(i,1)$. ! Выбрать номер первого ребра из списка ребер с одинаковым весом.

5. Пока $ND \neq 0$ выполнить шаги 6–30.

6. $l \leftarrow IS(ND)$; $k \leftarrow ST(ND)$; $l1 \leftarrow A(l)$; $k1 \leftarrow A(k)$. ! Выбрать левую и правую вершины ребра и номера компонент, в которые эти вершины входят.

7. Если $l1 \neq 0 \wedge k1 \neq 0 \wedge l1 = k1$, то $KPUS \leftarrow KPUS + 1$; перейти к шагу 28. ! Обнаружено циклическое ребро, перейти на выбор нового ребра.

8. Если $l1 = 0 \wedge k1 \neq 0$, то перейти к шагу 15. ! Присоединить к существующей компоненте новую вершину и новое ребро, перейти к присоединению к компоненте.

9. Если $l1 \neq 0 \wedge k1 = 0$, то $ll \leftarrow l$; $l11 \leftarrow l1$; $l \leftarrow k$; $l1 \leftarrow k1$; $k \leftarrow ll$; $k1 \leftarrow l11$; перейти к шагу 15. ! Присоединить к существующей компоненте новую вершину и новое ребро, перейти к присоединению к компоненте.

10. Если $l1 = 0 \wedge k1 = 0$, то перейти к шагу 12. ! Образовать новую компоненту, добавить две новые вершины и одно новое ребро, перейти на образование компоненты.

11. Перейти к шагу 18. ! Слить две независимые компоненты и добавить новое ребро, перейти на слияние компонент.

12. $A(l) \leftarrow NK$; $A(k) \leftarrow NK$; $H(NK,1) \leftarrow l$; $G(l) \leftarrow k$; $H(NK,2) \leftarrow k$. ! Образование новой компоненты.

13. $H(NK,3) \leftarrow 2$; $EK(NK,1) \leftarrow ND$; $EK(NK,2) \leftarrow ND$; $W(NK) \leftarrow \leftarrow C(ND)$. !

14. $KK \leftarrow KK + 1$; $NK \leftarrow NK + 1$; $NY \leftarrow NY + 2$; $KNOV \leftarrow KNOV + 1$; $KOST \leftarrow KOST + 1$; перейти к шагу 28.

15. $A(l) \leftarrow k1$; $G(H(k1,2)) \leftarrow l$; $H(k1,2) \leftarrow l$; $H(k1,3) \leftarrow H(k1,3) + 1$. ! Присоединение к компоненте

16. $B(EK(k1,2)) \leftarrow ND$; $EK(k1,2) \leftarrow ND$; $W(k1) \leftarrow W(k1) + C(ND)$.

17. $NY \leftarrow NY + 1$; $KPRI \leftarrow KPRI + 1$; $KOST \leftarrow KOST + 1$; перейти к шагу 28.

18. Если $H(l1,3) > H(k1,3)$, то $lp \leftarrow l1$; $l1 \leftarrow k1$; $k1 \leftarrow lp$. ! Слияние компонент.

19. $W(k1) \leftarrow W(k1) + W(l1)$; $W(l1) \leftarrow 0$.

20. $KSLI \leftarrow KSLI + 1$; $KOST \leftarrow KOST + 1$; $KSUZ \leftarrow KSUZ + H(l1,3)$.

21. $G(H(k1,2)) \leftarrow H(l1,1)$; $H(k1,2) \leftarrow H(l1,2)$; $H(k1,3) \leftarrow H(k1,3) + H(l1,3)$.

22. $B(EK(k1,2)) \leftarrow EK(l1,1)$; $EK(k1,2) \leftarrow EK(l1,2)$; $B(EK(k1,2)) \leftarrow ND$.

23. $EK(k1,2) \leftarrow ND$; $W(k1) \leftarrow W(k1) + C(ND)$.

24. $j \leftarrow H(l1,1)$.

25. Пока $j \neq 0$ выполнить шаг 26. ! Обновить массив вхождения вершин в компоненты после слияния двух компонент.

26. $A(j) \leftarrow k1$; $j \leftarrow G(j)$.

27. $KK \leftarrow KK - 1$. ! Уменьшить число компонент на единицу.

28. Если $KK = 1 \wedge NY = v$, то перейти к шагу 33. ! Граф связный и все узлы включены в остовный лес.

29. $ND \leftarrow E(ND)$. ! Выбрать номер очередного ребра.

30.Перейти к шагу 5. ! Конец цикла по ND.

31.Перейти к шагу 3. ! Конец цикла по i.

32.Если $NY < v$, то $SV \leftarrow SV + 1$.

33.Конец.

Приведенный алгоритм может быть использован для построения остовного леса (дерева) с максимальным весом, если в цикле по i ребра просматривать в обратном порядке, т.е. в шаге 3 записать: для $\{i \mid i = \overline{MMAX,1,-1}\}$ выполнить шаги 4–31.

В записи алгоритма в шаге 2 введены переменные $KNOV$, $KPRI$, $KSLI$, в которых соответственно накапливается: число образований новых компонент; число присоединений к существующей компоненте вершин (ребер); число слияний двух независимых компонент. В переменных $KOST$, $KPUS$, $KSUZ$ соответственно подсчитывается число ребер в остовном лесе, число пустых проходов (пропуск ребер, образующих циклы), суммарное число слитых вершин. Все эти переменные необходимы для проведения дальнейшего экспериментального анализа сложности алгоритма. Значения $KPUS$, $KPRI$, $KNOV$, $KSLI$ и $KSUZ$ зависят от структуры и размерности графа, их трудно выразить через основные параметры графа, поэтому в дальнейшем изложении для их определения будем использовать эмпирические методы. Значение $KOST = KPRI + KNOV + KSLI$, а $KOST + KPUS = e'$ или $KOST + KPUS = e$ — числу просмотренных ребер графа для построения остовного дерева или леса соответственно для связного и несвязного графа.

Рассмотрим алгоритм Тарьяна [21] построения MST-деревьев с применением древовидной структуры для представления наращиваемого остовного леса с операциями слияния по рангу и сжатия путей. Будем считать, что ребра графа предварительно отсортированы алгоритмом *SORT*.

Определим следующие массивы: $A(i)$, $i = \overline{1,v}$ — массив указателей на вхождение вершин i в независимые компоненты, первоначально каждая вершина указывает сама на себя; $H(i)$, $i = \overline{1,v}$ — массив, содержащий ранги вершин i; $B(i)$, $i = \overline{1,v-1}$ — массив, содержащий номера ребер MST графа.

Алгоритм *TREE*. Построение остовного леса (дерева) с минимальным весом с использованием древовидных структур данных.

1. $H \leftarrow 0$; $B \leftarrow 0$; $SUM \leftarrow 0$; $LP \leftarrow 0$; $KSLI \leftarrow 0$; $KOST \leftarrow 0$; $KPUS \leftarrow 0$; $KSUZ \leftarrow 0$.

2. Для $\{i \mid i = \overline{1,v}\}$ выполнить $A(i) \leftarrow i$.

3. Для $\{i \mid i = \overline{1,MMAX}\}$ выполнить шаги 4–14.

4. $ND \leftarrow U(i,1)$. ! Выбрать номер первого ребра из списка ребер с одинаковым весом.

5. Пока $ND \neq 0$ выполнить шаги 6–13.

6. $l \leftarrow IS(ND)$; $k \leftarrow ST(ND)$.

7. $KSLI \leftarrow KSLI + 1$.

8. Вызвать $FND(l)$; Вызвать $FND(k)$.

9. Если $l \neq k$, то перейти к шагу 10, иначе $KPUS \leftarrow KPUS + 1$, перейти к шагу 11.

10. Вызвать $UN(k,l)$; $KOST \leftarrow KOST + 1$; $LP \leftarrow LP + 1$; $B(LP) \leftarrow ND$; $SUM \leftarrow SUM + C(ND)$.

11. Если $KOST = v - 1$, то перейти к шагу 15.

12. $ND \leftarrow E(ND)$.

13. Перейти к шагу 5. ! конец цикла по ND.

14. Перейти к шагу 3. ! конец цикла по i.

15. Конец.

1. Процедура $FND(x)$. ! x, y, z — целые.

2. $z \leftarrow x$. ! Ищем корень для x.

3. Пока $A(x) \neq x$ выполнить $x \leftarrow A(x)$. ! x — корень, первый проход.

4. Пока $A(z) \neq x$ выполнить шаг 5.

5. $y \leftarrow z$; $z \leftarrow A(z)$; $A(y) \leftarrow x$; $KSUZ \leftarrow KSUZ + 2$. ! Обновляем указатели на корень, второй проход.

6. Возвратить x в точку вызова. ! Конец процедуры $FND(x)$.

1. Процедура $UN(x, y)$. ! x, y — целые.

2. Если $H(x) < H(y)$, то $A(x) \leftarrow y$, перейти к шагу 5, иначе перейти к шагу 3.

3. Если $H(x) > H(y)$, то $A(y) \leftarrow x$, перейти к шагу 5, иначе перейти к шагу 4.

4. $A(x) \leftarrow y$; $H(y) \leftarrow H(y) + 1$.

5. Возврат в точку вызова. ! Конец процедуры $UN(x, y)$.

В алгоритме *TREE* переменные $KOST$ и $KPUS$ имеют то же назначение, что и в алгоритме *LINK*, а в переменных $KSLI$, $KSUZ$ и SUM накапливается соответственно число просмотренных ребер, число циклических проходов для нахождения корней деревьев и обновления указателей и суммарный вес остовного дерева или леса. Значение $KSLI = KOST = KPUS$ равно e' или e для связного и несвязного графа. Чтобы не усложнять алгоритм Тарьяна в нем не формируются массивы подобные $H(MK,3)$, $G(v)$, $EK(MK,2)$, $B(e)$ и $W(MK)$ для хранения вершин, ребер и весов для компонент несвязного графа. Поэтому при дальнейшем экспериментальном сравнении быстродействия алгоритмов *LINK* и *TREE* из алгоритма *LINK* были убраны операторы формирования этих массивов и введены операторы $LP \leftarrow LP + 1$, $B(LP) \leftarrow ND$, $SUM \leftarrow SUM + C(ND)$ как в строке 10 алгоритма *TREE*.

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМОВ

Для проведения эксперимента были составлены главный модуль программы и четыре модуля (подпрограммы), реализующие алгоритм сортировки

SORT, алгоритм *LINK*, алгоритм *TREE* и алгоритм Прима *PRIM* с представлением графа в виде матрицы смежности. Все тестовые программы написаны на языке Digital Visual Fortran. Вычислительный эксперимент проводился на ПЭВМ с процессором Intel Core 2 Duo с тактовой частотой 2,66 ГГц и оперативной памятью 2 Гб под управлением операционной системы Windows Vista.

В главном модуле программы в режиме диалога вводились: число вершин графа — v; число исходящих из каждой вершины ребер (степень или валентность вершины) — VAL; границы изменения значений весов ребер от $MINVAL$ до $MAXVAL$; параметр, управляющий выводом входных и выходных данных. Далее с помощью датчика псевдослучайных чисел (встроенной функции языка $RAND()$) генерировались веса ребер от $MINVAL = 80$ до $MAXVAL = 800$, формировались массивы IS, ST, C. Вся оперативная память, необходимая для работы алгоритмов выделялась и освобождалась динамически в главном модуле программы. Время работы алгоритмов фиксировалось встроенным модулем $cpu\_time(T)$ непосредственно до входа и после выхода из модулей сортировки и построения минимального остовного дерева. Работа алгоритмов проверялась на неориентированных полных графах с числом вершин v от 1000 (499500 ребер) до 12000 (71994000 ребер). Результаты эксперимента приведены в таблице и на рис. 1. Где e' — число ребер, просмотренных для полного построения остовного дерева. Для $KOST$, $KPUS$, $KPRI$, $KNOV$, $KSLI$ проценты указаны по отношению к e', а для e' проценты указаны по отношению к e. Для $KSUZ$ в алгоритме *LINK* (*KSUZL* в таблице) указано суммарное число слитых вершин (число выполнений оператора $A(J) \leftarrow K1$ в цикле в шаге 25) и среднее число слитых вершин за одну итерацию. Для алгоритма *TREE* параметр $KSUZ$ обозначен как $KSUZT$. Кроме того в таблице приняты следующие обозначения: $SORT$ — время сортировки ребер алгоритмом *SORT*; $LINK$, $TREE$ и $LSUM$, $TSUM$ — соответственно время работы предложенного алгоритма *LINK*, алгоритма Тарьяна *TREE* и общее время построения MST-дерева, которое определяется суммой времени сортировки ребер и времени работы алгоритма *LINK* или алгоритма *TREE*; $PRIM$ — время работы алгоритма Прима; MST — вес MST.

Из таблицы видно, что процесс построения минимального остовного дерева для связных графов заканчивается при просмотре в среднем не более 0,25% ребер графа, а средние значения в процентах для $KPUS$, $KPRI$, $KNOV$, $KSLI$ приблизительно равны 81%, 15%, 2%, 2%. При этом среднее число слитых вершин за одну операцию слияния равно 6. Время работы алгоритмов *LINK* и *TREE* практически совпадает, растет очень медленно и для полного графа с 12 тыс. вершин не превысило 0,03 с (рис. 1). Общее время построения MST-дерева главным образом определяется временем сортировки ребер в алгоритме *SORT*. Поэтому для связных графов можно практически принять, что общая сложность алгоритмов построения MST-дерева соизмерима со сложностью алгоритма *SORT* для сортировки ребер $O(e)$. Из таблицы и графиков также видно, что на полных графах лучшим остается алгоритм Прима.

Таблица. Время построения MST-дерева полного графа в зависимости от числа вершин в секундах

№	1	2	3	4	5	6	7	8	9	10	11	12	Среднее значение
v	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12000	
e (млн.)	0,4995	1,999	4,4985	7,998	12,4975	17,997	24,4965	31,996	40,4955	49,995	60,4945	71,994	
e'	3993	8276	15637	24805	23820	38117	39228	38898	48523	53627	65813	77612	
%	0,80	0,41	0,35	0,31	0,19	0,21	0,16	0,12	0,12	0,11	0,11	0,11	0,25
KOST	999	1999	2999	3999	4999	5999	6999	7999	8999	9999	10999	11999	
%	25,02	24,15	19,18	16,12	20,99	15,74	17,84	20,56	18,55	18,65	16,71	15,46	19,08
KPUS	2994	6277	12638	20806	18821	32118	32229	30899	39524	43628	54814	65613	
%	74,98	75,85	80,82	83,88	79,01	84,27	82,16	79,44	81,45	81,35	83,29	84,54	80,92
KPRI	576	1322	2228	3246	4198	5174	6222	7228	8224	9174	10164	11192	
%	14,43	15,97	14,25	13,09	17,63	13,57	15,86	18,58	16,95	17,11	15,44	14,42	15,61
KNOV	212	339	386	377	401	413	389	386	388	413	418	404	
%	5,31	4,10	2,47	1,52	1,68	1,08	0,99	0,99	0,80	0,77	0,64	0,52	1,74
KSLI	211	338	385	376	400	412	388	385	387	412	417	403	
%	5,28	4,08	2,46	1,51	1,68	1,08	0,99	0,99	0,80	0,77	0,63	0,52	1,73
KSUZL	1040	1656	1880	2332	2364	2597	2800	3068	3050	3004	2873	2754	
Число слитых вершин за одну итерацию	5	5	5	6	6	6	7	8	8	7	7	7	6
KSUZT	1288	2240	2642	3180	3346	3636	3818	4158	4164	4396	4324	4236	
SORT	0,02	0,02	0,03	0,06	0,09	0,09	0,16	0,19	0,22	0,28	0,34	0,37	
LINK	0,00	0,00	0,00	0,00	0,00	0,02	0,00	0,02	0,02	0,00	0,00	0,02	
TREE	0,00	0,00	0,02	0,00	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,03	
LSUM	0,02	0,02	0,03	0,06	0,09	0,11	0,16	0,21	0,24	0,28	0, 34	0,39	
TSUM	0,02	0,02	0,05	0,06	0,11	0,11	0,18	0,21	0,24	0,30	0,36	0,40	
PRIM	0,00	0,00	0,02	0,03	0,06	0,06	0,09	0,12	0,16	0,19	0,22	0,23	
MST	80351	160083	240013	319950	399930	479925	559921	639920	719920	799920	879920	959920	

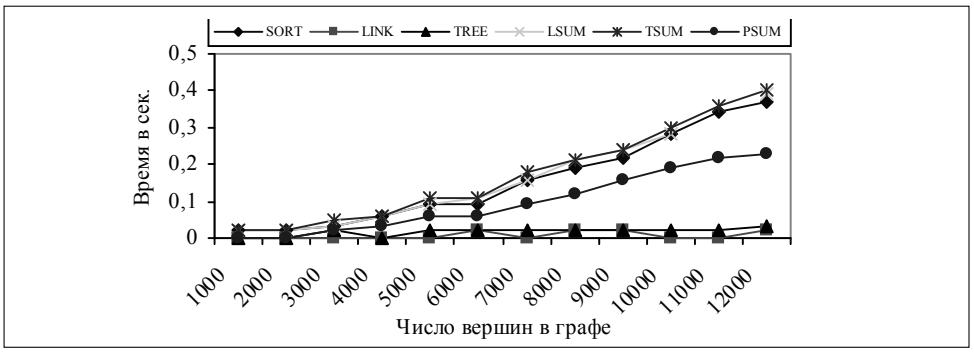

Рис. 1. Сравнение времени работы алгоритмов на полных графах в зависимости от числа вершин в графе

При тестировании алгоритмов *LINK* и *TREE* генерировались также разреженные связные и несвязные графы с различным числом вершин и ребер, и результаты вычислений сравнивались с результатами, полученными для этих графов алгоритмом Прима. Результаты эксперимента для связного

и несвязного графа содержащего 11000 вершин при изменении степени вершин (параметра VAL) показаны на рис. 2.

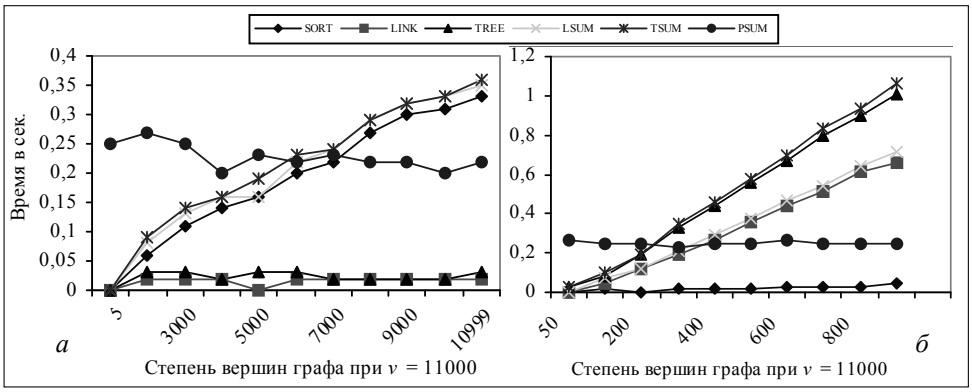

Рис. 2. Сравнение времени работы алгоритмов на разреженных связных (*а*) и несвязных (*б*) графах при изменении степени вершин и постоянном числе вершин $v = 11000$

Из приведенных данных видно, что алгоритмы *LINK* (*LSUM*) и *TREE* (*TSUM*) с учетом времени на сортировку ребер на разреженных связных графах работают быстрее алгоритма Прима (*PRIM*) при $VAL \le 6000$ и числе ребер $e = 0{,}5 \cdot VAL \cdot v$ не больше, чем $0{,}27v^2$. Полученная оценка уточнена и лучше, чем показанная в работе [20]. Для сильно разреженных графов быстродействие алгоритмов *LINK* и *TREE* с учетом времени сортировки может значительно превышать быстродействие алгоритма Прима — в 15 и более раз. Для несвязных разреженных графов алгоритмы *LINK* (*LSUM*) и *TREE* (*TSUM*) с учетом времени сортировки лучше алгоритма Прима (*PRIM*) при $VAL \le 300$, $e \le 0{,}014v^2$ и $VAL \le 200$, $e \le 0{,}009v^2$ соответственно.

На рис. 3 показаны результаты сравнения алгоритмов на несвязных графах, содержащих полные несвязные подграфы или изолированные вершины. Из рис. 2*б* и 3 видно, что на несвязных графах алгоритм *LINK* (*LSUM*, *LINK*) при увеличении степени вершин начинает заметно опережать алгоритм *TREE* (*TSUM*, *TREE*). Из графиков на рис. 2*б* и 3 следует, что временная сложность построения MST для несвязных графов больше сложности сортировки ребер алгоритмом *SORT*.

Проведем асимптотическую оценку сложности алгоритмов *LINK* и *TREE*. Ясно (см. таблицу), что

$$e' = KOST + KPUS = KPRI + KNOV + KSLI + KPUS.$$

Все операции $KPRI, KNOV, KSLI, KPUS$ выполняются в основном цикле по ND при выборе ребер для анализа. Поскольку сложность каждой из этих операций в алгоритме *LINK* составляет $O(1)$, а всего просматривается e' или e ребер соответственно для связных и несвязных графов, то их общая сложность составит $O(e')$ или $O(e)$. Если учесть, что сложность операций инициализации массива A вхождения вершин в компоненты графа (операции make set) составляет $O(v)$, получим время порядка $O(e'' + v)$ или

$O(e+v)$. Осталось оценить сложность операций обновления указателей в массиве A при выполнении операций слияния компонент с меньшим числом вершин в компоненты с большим числом вершин. В работах [14, 15] показано, что сложность таких операций в худшем случае составляет $O(v\log v)$. Однако, если проанализировать таблицу, то очевидно, что число вершин, для которых обновляются указатели не соизмеряется с ростом функции $v\log v$. Значения $KSUZL$ в таблице для числа вершин v от 1000 до 12000 изменяются неравномерно от 1040 до 2754, а среднее число обновлений указателей за одну операцию слияния $\rceil KSUZL/KSLI\lfloor$ изменяется в пределах от 5 до 8, где $\rceil\cdot\lfloor$ — знак округления числа до ближайшего целого. Выразить функцию для числа обновлений указателей через параметры исходного графа v и e затруднительно, поэтому запишем ее в виде некоторой медленно растущей функции вида $\beta(KSUZL/KSLI,v)$. Ожидаемую полную сложность алгоритма *LINK* без сложности сортировки e ребер запишем в виде $O(e'+v+KSLI\beta(KSUZL/KSLI,v))$ для связных и $O(e+v+$ $+KSLI\beta(KSUZL/KSLI,v))$ для несвязных графов, причем $KSLI \cong 0{,}02\,e' \cong$ $\cong 0{,}02\cdot 0{,}0025\,e \cong 0{,}00005\,e$ является малой величиной по сравнению с e.

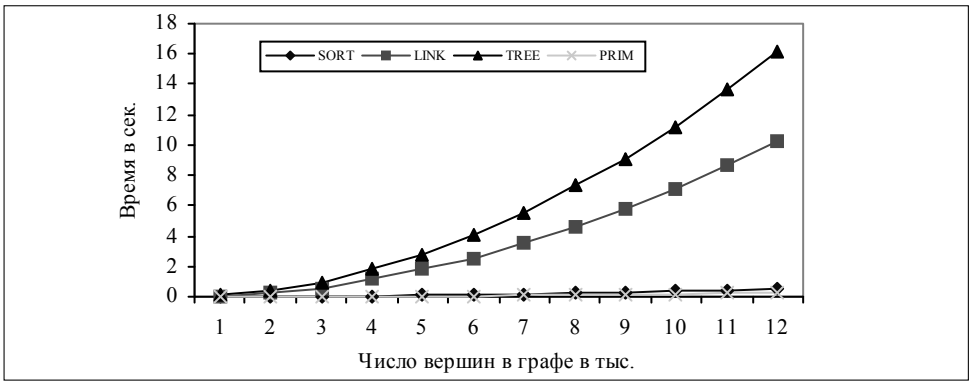

Рис. 3. Сравнение времени работы алгоритмов на несвязных графах с полными изолированными подграфами в зависимости от числа вершин в графе

В древовидной реализации MST-алгоритма Краскала (алгоритм *TREE*) каждая из операций $KPUS, KPRI, KNOV, KSLI$ имеет сложность $O(1)$ и выполняется в процедуре merge/union. Общая сложность этих операций составит $O(e')$ и $O(e)$ для связных и несвязных графов соответственно. Сложность операций make set составляет $O(v)$. Операции find set выполняются для каждого ребра дважды, т.е. имеем $2e'$ или $2e$ входов в процедуру find set. Сложность обновления указателей в древовидной структуре с операциями объединения по рангу и сжатия пути (для сжатия пути применяется двойной проход от листа до корня дерева) определяется как $\alpha(e',v)$ или $\alpha(e,v)$, где $\alpha(\cdot,\cdot)$ — очень медленно растущая инверсная функция Аккермана. Поэтому сложность древовидной реализации MST-алгоритма *TREE* без сложности сортировки ребер составит $O(e'+v+e'\alpha(e',v))$ или $O(e+v+e\alpha(e,v))$. В работах Тарьяна и ван Леувена [27, 28] были рассмотрены также однопроходные варианты эвристик со сжатием пути. Однако, по

нашему мнению, сложность однопроходных процедур find set на практике не будет сильно отличаться от двухпроходных, так как в оценках однопроходных процедур скрыты дополнительные константы накладных расходов на их реализацию.

Определенный интерес представляет скорость роста функции $KSLI\beta(KSUZL/KSLI,v)$ и в алгоритме Краскала. Тарьян показал [28], что полученная им оценка $O(m\alpha(m,v))$, где $m=e'$ или $m=e$ асимптотически неулучшаема.

В настоящей работе мы экспериментально показали, что в практических случаях алгоритм *LINK*, реализованный на списочных структурах с использованием массивов работает не хуже чем алгоритм, основанный на древовидных АТД с использованием операций сжатия пути и объединения по рангу.

На рис. 4 показано число операций производимых для обновления указателей алгоритмами *LINK* (*KSUZL*) и *TREE* (*KSUZT*) для данных из таблицы.

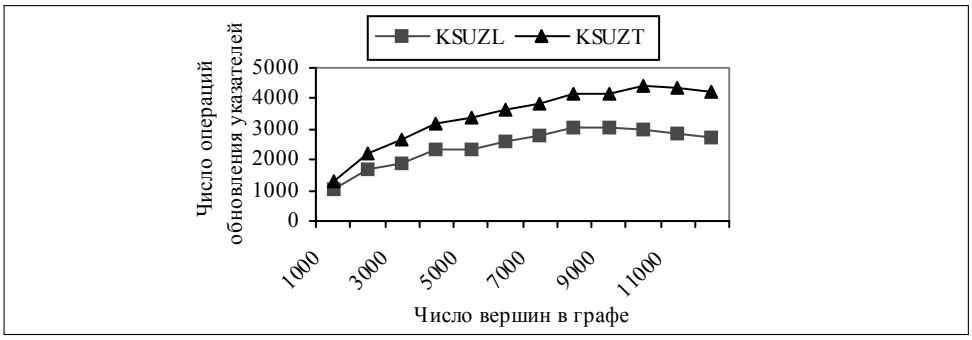

Рис. 4. Сравнение алгоритмов *LINK* (*KSUZL*) и *TREE* (*KSUZT*) по числу операций обновления указателей при построении MST-дерева на полных связных графах

ВЫВОДЫ

Подводя итог по оценке сложности алгоритмов *SORT*, *LINK* и *TREE* можно сделать следующие выводы:

1. Для связных графов практическая временная сложность обоих алгоритмов построения MST-дерева с «карманной» сортировкой (bucket sort) весов ребер составляет $O(e)$ операций, так как значения $KSLI\beta(KSUZL/KSLI,v)$, $e'\alpha(e',v)$, $(e'+v)$ нуль сравнимы со значением e, где e' — число просмотренных отсортированных ребер графа для построения MST-дерева.

2. Для несвязных графов сложность построения MST-леса с «карманной» сортировкой и использованием алгоритма *LINK* — $O(e+v+e+$ $+KSLI\beta(KSUZL/KSLI,v))$, использованием алгоритма *TREE* — $O(e+v+$ $+e+e\alpha(e,v))$, но $KSLI\beta(KSUZL/KSLI,v)$ на практике растет медленнее, чем $e\alpha(e,v)$.

3. Во всех прогонах алгоритм *LINK* оказывался немного быстрее алгоритма *TREE*.

4. Сложность предложенного алгоритма построения MST-дерева для связных графов с использованием алгоритмов *SORT* и *LINK* порядка $O(e)$ получена для практических случаев. В то же время до сих пор не известно о существовании алгоритма построения MST, для которого строго доказано, что его временная сложность составляет $O(e)$ в среднем или в худшем случае для разреженных графов. Поэтому разработка алгоритмов с гарантированной линейной оценкой для разреженных графов все еще остается недостижимой целью [14]. Интенсивному изучению подверглись различные варианты алгоритма Борувки как базиса алгоритмов для вычисления MST на сильно разреженных графах за почти линейное время, а также рандомизированные алгоритмы поиска MST, математическое ожидание времени работы которых сравнимо с $O(e+v)$ [15]. Такие исследования вселяют надежду на успех и будоражат энтузиазм разработчиков более быстрых алгоритмов построения MST.

5. В целом эксперимент показал высокую вычислительную эффективность алгоритмов, которые могут с успехом применяться при решении практических задач определения оптимальных остовных деревьев (лесов) для графов и сетей большой размерности. В частности, предложенные алгоритмы могут быть включены в состав типового инструментария разработчика программ и использоваться при решении различных задач анализа и проектирования коммуникационных сетей.

ЛИТЕРАТУРА

1. *Boruvka O.* O jistém problému minimálním // Pràca Moravské Prirodovedecké Spolecnosti. — 1926. — **3**. — P. 37–58.
2. *Nešetřil J.E.*, *Milková H.*, *Nešetřilová.* Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history // Discrete Mathematics. — 2001. — **233** (1–3). — P. 3–36.
3. *Jarník V.* O jistém problému minimálním // Práce Moravské Prirodovedecké Spolecnosti. — 1930. — **6**. — P. 57–63.
4. *Prim R.C.* Shortest Connection Networks and Some Generalizations // Bell Syst. Tech. J. — 1957. — **36**. — P. 1389–1401.
5. *Dijkstra E.* A note on two problems in connexion with graphs // Num. Math. — 1959. — **1**. — P. 269–271.
6. *Kruskal J.B.* On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem // Proc. Amer. Math. Soc. — 1965. — **7**. — P. 48–50.
7. *Loberman H.*, *Weinberger A.* Formal procedures for connecting terminals with a minimum total wire length // Journal of ACM. — 1957. — **4**. — P. 428–437.
8. *Sollin M.* Le tracé de canalisation // Programming, Games, and Transportation Networks (in French). — 1965.
9. *Tarjan R.E.* Data Structures and Network Algorithms. — Philadelphia: Society for Industrial and Applied Mathematics, 1983.
10. *Graham R.L.*, *Hell P.* On the History of the Minimum Spanning Tree Problem // Annals of the History of Computing. — 1985. — 7(l). — P. 43–57.
11. *Кнут Д.* Искусство программирования для ЭВМ. Т.3 (Сортировка и поиск). — М.: Мир, 1978. — 800 с.
12. *Ahuja R.K.*, *Orlin J.B.*, *Magnanti T.L.* Network flows: theory, algorithms, and applications. — New Jersey: Prentice-Hail, Inc. Upper Saddle River, 1993. — 846 p.

13. *Ахо А.*, *Дж. Хопкрофт*, *Дж. Ульман*. Структуры данных и алгоритмы: Пер. с англ.: Уч. пос. — М.: Издательский дом «Вильяме», 2000. — 384 с.
14. *Седжвик Р.* Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ. — СПб: ООО «ДиаСофтЮП», 2002. — 496 с.
15. *Кормен Т.*, *Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ, 2-е издание: Пер. с англ. — М.: Издательский дом «Вильямс», 2005. — 1296 с.
16. *Mehlhorn K., Sanders P.* Algorithms and Data Structures. The Basic Toolbox. — Springer-Verlag Berlin Heidelberg, 2008. — 305 p.
17. *Boost.org.* Boost C++ Libraries. — www.boost.org.
18. LEDA (Library of Efficient Data Types and Algorithms). — www.algorithmic-solutions.com.
19. *Goodrich M.T.*, *Tamassia R*. JDSL — the data structures library in Java. — http://www.jdsl.org/.
20. *Васянин В.А.* О вычислительной эффективности одного алгоритма для нахождения основного леса графа с минимальным (максимальным) весом // Екологічна безпека та природокористування: Зб. наук. праць. — Київ, 2009. — Вип. 4. — С. 155–169.
21. *Tarjan R.E.* Efficiency of a good but not linear set union algorithm // Journal of the ACM. — 1975. — 22, № 2. — P. 215–225.
22. *Gabow H.N., Galil Z., Spencer T., Tarian R.E.* Efficient algorithms for finding minimum spanning trees in undirected and directed graphs // Combinatorica. — 1986. — **6**. — P. 109–122.
23. *Chazelle B.* The Soft Heap: An Approximate Priority Queue with Optimal Error Rate // Journal of the ACM. — 2000. — **47**. — P. 1012–1027.
24. *Chazelle B.* A minimum spanning tree algorithm with inverse-Ackermann type complexity // Journal of the ACM. — 2000. — **47**. — P. 1028–1047.
25. *Pettie S.*, *Ramachandran V*. An optimal minimum spanning tree algorithm // In 7th International Colloquium on Automata, Languages and Programming: of Lecture Notes in Computer Science, Springer, 2000. — **1853**. — P. 49–60.
26. *Pettie S.*, *Ramachandran V*. Randomized minimum spanning tree algorithms using exponentially fewer random bits // ACM Transactions on Algorithms. — 2008. — **4**, № 1. — P. 1–27.
27. *Tarjan R.E., Leeuwen J.V*. Worst-Case Analysis of Set Union Algorithms // J. of the ACM. — 1984. — **31**, № 2. — P. 245–281.
28. *Tarjan R.E.* Amortized computational complexity // SIAM J. on Algebraic and Discrete Methods. — 1985. — **6**, № 2. — P. 306–318.

Надійшла 15.06.2014