УДК 519.1

# О ВЫЧИСЛИТЕЛЬНОЙ ЭФФЕКТИВНОСТИ ОДНОГО АЛГОРИТМА ДЛЯ НАХОЖДЕНИЯ ОСТОВНОГО ЛЕСА ГРАФА С МИНИМАЛЬНЫМ (МАКСИМАЛЬНЫМ ) ВЕСОМ

*В. А. Васянин, канд. техн. наук*

*В работе рассматривается эффективная реализация «жадного» алгоритма нахождения остовного леса (дерева) с минимальным (максимальным) весом на неориентированном взвешенном графе. Получена асимптотическая оценка трудоемкости алгоритма, которая составляет 0(M), где M число ребер графа. Показано, что предложенный алгоритм асимптотически лучше алгоритма Прима для графов с числом ребер меньше, чем $N^2/6$, где N 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.*

*У роботі розглядається ефективна реалізація «жадібного» алгоритму знаходження остовного лісу (дерева) із мінімальною (максимальною) вагою на неорієнтованому зваженому графі. Отримана асимптотична оцінка трудомісткості алгоритму, що складає 0(M), де M — число ребер графа. Показано, що запропонований алгоритм асимптотично краще алгоритму Прима для графів із числом ребер менше, ніж $N^2/6$, де N — число вершин графа. Експериментальне дослідження алгоритму на графах, що містять від 499500 до 71994000 ребер, показало його високу обчислювальну ефективність і він може бути рекомендований для рішення практичних задач на розріджених графах чи мережах великої розмірності.*

*In work effective realization of «greedy» algorithm for finding minimum (maximum) spanning woods (trees) of an undirected weighed graph is considered. Is given the rating of the expected computing time of algorithm is 0 (M), where M — number of edges in a graph. Is shown, that the offered algorithm is better than a Prim's algorithm for graphs with number of edges less, than $N^2/6$, where N — number of vertices in a graph. The experimental research of algorithm on the graphs, containing from 499500 up to 71994000 edges, has shown its high*

*computing efficiency and his can be recommended for the decision of practical problems on rarefied graphs or networks of the big dimension.*

Во многих случаях геопространственные данные топографических объектов могут быть представлены в виде неориентированного взвешенного графа или сети. При этом в качестве весов ребер графа или дуг сети могут выступать различные числовые характеристики (параметры) геоинформационных объектов: рельефа, гидрографической сети, населенных пунктов, дорог, различных инженерных коммуникаций и т. п. При проведении всестороннего анализа таких геоинформационных объектов может потребоваться нахождение минимального или максимального стягивающего (остовного) леса (дерева). В этой связи оказывается актуальным наличие эффективных алгоритмов для решения подобных задач.

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 |·| — знак мощности множества. Граф может быть неполным и несвязным. Для графа задан вектор весов ребер $C = \| c_i \|$, $i = 1,e$, где $c_i \in Z^+$, $Z^+$ — множество неотрицательных целых чисел.

Требуется найти ациклический подграф (лес) графа G с минимальным или максимальным общим весом. Известно достаточно много алгоритмов определения минимального или максимального остовного леса (дерева) на взвешенном графе. Среди них можно выделить группу «жадных» алгоритмов, основанных на предварительной сортировке ребер графа по весу и последующем последовательном выборе ребер минимального или максимального веса для наращивания остовного леса (дерева) [1]. Алгоритмы этой группы имеют асимптотическую трудоемкость порядка 0(eloge).Трудоемкость этих алгоритмов зависит от выбора метода сортировки ребер и методов определения и объединения (слияния) независимых компонент графа (поддеревьев) на каждой итерации алгоритма при добавлении нового ребра. Другая группа алгоритмов основана на методе «ближайшего соседа», который не требует предварительной сортировки ребер и проверки на циклич-

ность при наращивании поддеревьев на каждой итерации [2, 3]. Трудоемкость этих алгоритмов составляет $0(v^2)$ и они асимптотически оптимальны, т. е. для полных графов асимптотически неулучшаемы. Для разреженных графов известны алгоритмы со сложностью 0(elogv), 0(eloglogv) [46]. Эти алгоритмы асимптотически лучше алгоритма Прима для графов, имеющих меньше чем $v^2/logv$ è $v^2/loglogv$ ребер соответственно. Описание алгоритмов обеих групп можно также найти, например, в работах [7, 8].

Несмотря на существование достаточно эффективных алгоритмов нахождения минимальных и максимальных остовных лесов, интерес к разработке еще более быстрых алгоритмов не ослабевает. Это объясняется тем, что одновременно с постоянным ростом быстродействия и оперативной памяти современных ПЭВМ, 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 минимальные или максимальные остовные леса (деревья). Тогда алгоритм определения остовных лесов с минимальной трудоемкостью может значительно сократить время решения общей задачи. Рассмотрим возможную реализацию «жадного» алгоритма, который находит остовный лес с минимальным или максимальным весом за время 0(e). Введем обозначения: $\forall$, $\exists$ — кванторы всеобщности и существования; $\cup$, $\cap$, $\setminus$, $\supseteq$ — знаки объединения, пересечения, вычитания и включения множеств; $\in$, $\notin$ — знаки принадлежности и не принадлежности к множеству; $\varnothing$ — пустое множество; $\{\cdot\}$ — множество подмножеств или элементов множества; &, ! — знаки логического «и», логического «или»; i = 1,k означает «для всех i от единицы до k. Для построения алгоритма докажем следующую теорему.

**Теорема 1.** Пусть $\{(V_0,E_0),(V_1,E_1),(V_2,E_2),\ldots,(V_k,E_k)\}$ — частично построенный остовный лес графа G = (V,E), где $(V_0,E_0) = (\varnothing,\varnothing),(V_i,E_i)$ связные независимые компоненты графа (поддеревья), Ç$\{(V_i,E_i)\}$ = $\varnothing$ , i = 0,k и выполняется условие

$$|\cup V_i| < v \;\&\; E\setminus \cup E_i \;/ = \varnothing,\ i = 0,k\ . \qquad (1)$$

Тогда для $\forall\ e_{l\,r} \in E \setminus \cup E_i$, i = 0,k , включаемого в остовный лес, пока выполняется условие (1), справедливо:

1) если $l \notin V_i$ & $r \notin V_i$ , i = 1,k, то ребро $e_{l\,r}$ образует новую k+1 компоненту ($\{l,r\},\{e_{l\,r}\}$), à $\{(V_i,E_i), (\{l,r\},\{e_{l\,r}\})\}$, i = 0,k — частичный остовный лес графа G ;

2) если ( $l \in V_j$, $j \in \{1,2,\ldots,k\}$) & ($r \notin V_i$, i = 1,k) ! ($l \notin V_i$, i = 1,k) & ($r \in V_j$, $j \in \{1,2,\ldots,k\}$) , то ребро $e_{l\,r}$ соответственно добавляется к компонентам ($V_j \cup r, E_j \cup e_{l\,r}$) или ($V_j \cup l$, $E_j \cup e_{l\,r}$), à $\{(V_i,E_i),(V_j,E_j)\}$, i = 0,k, i /= j — частичный остовный лес графа G;

3) если ( $l \in V_j$, $j \in \{1,2,\ldots,k\}$) & ($r \in V_m$, $m \in \{1,2,\ldots,k\}$) & j /= m , то ребро $e_{l\,r}$ объединяет (сливает) компоненты ($V_j$,$E_j$) è ($V_m$,$E_m$) , a $\{(V_i,E_i), (V_j \cup V_m, E_j \cup E_m \cup \cup e_{l\,r})\}$, i = 0,k, i /= j, i /= m частичный остовный лес графа G ;

4) если $l \in V_j$ & $r \in V_j$ , $j \in \{1,2,\ldots,k\}$, то ребро $e_{l\,r}$ образует цикл в компоненте ($V_j$,$E_j$).

Если условие (1) не выполняется, то при $|\cup V_i| = v$, i = 0,k остовный лес построен и k число его компонент. Если $|\cup V_i| < v$ , а $E \setminus \cup E_i = \varnothing$, i = 0,k , то остовный лес построен, k число его компонент и в графе G = (V,E) имеется $V \setminus \cup V_i$ , i = 0,k изолированных вершин.

Доказательство. Покажем по индукции, что добавление к остовному лесу любых ребер из $E \setminus \cup E_i$, i = 0,k при выполнении условия (1) и пунктов 1), 2), 3) приводит к образованию нового леса на множестве выбранных вершин. Очевидно, что для $\{(V_0,E_0)\}$ добавление любого ребра приводит к образованию первой компоненты с двумя вершинами и одним ребром между ними. Предположим далее, что после выбора j ребер получен $\{(V_0,E_0),(V_1,E_1),(V_2,E_2),\ldots,(V_k,E_k)\}$ частично построенный k компонентный остовный лес графа G на множестве выбранных к данному моменту вершин (нулевая компонента пустая и введена для исключения вырожденного случая, когда $\{(V_i,E_i)\}= \varnothing$). Рассмотрим выполнение j+1 шага, т. е. выбор j+1 ребра. Пусть (1) выполняется, тогда при выборе j+1 ребра и выполнении условий 1), 2), 3) будет соответственно образована новая компонента ($V_{k+1}$,$E_{k+1}$), добавлена одна новая вершина и одно новое ребро к существующему лесу , или две независимые компоненты леса сольются вместе с добавлением одного нового ребра.

Таким образом, обновленный лес не может содержать цикла ни в одной из своих компонент и остается остовным. Если при выборе j+1 ребра не будет выполняться условие (1), то очевидно, что на этом шаге при $|\cup V_i| = v$, i = 0,k будет окончательно построен k — компонентный остовный лес (остовное дерево при k = 1), содержащий v – k ребер, а при $|\cup V_i| < v$ è $E\setminus \cup E_i = \varnothing$, i = 0,k остовный лес будет содержать $V\setminus\cup V_i$ , i = 0,k изолированных вершин. Доказательство 4) немедленно следует из того, что каждая компонента $(V_j,E_j)$ является поддеревом и две любые вершины этого поддерева связаны единственным путем. Поэтому добавление нового ребра, соединяющего две любые вершины из $(V_j,E_j)$ приводит к образованию цикла в $(V_j,E_j)$.

Ясно, что для любого графа с использованием теоремы 1 можно найти множество различных остовных лесов, если определенным образом задавать выбор ребер для построения остовного леса. Так, например, для полного графа с v = | V | вершинами существует $v^{v-2}$ различных остовных деревьев.

Пусть множество ребер $E = \{e_i\}$, i = 1,e графа G = (V,E) упорядочено по неубыванию или по невозрастанию их весов $\{c_i\}$, i = 1,e , $c_i <= c_{i+1}$ или $c_i >= c_{i+1}$, i = 1,e-1.

**Теорема 2.** Для нахождения остовного леса графа G = (V,E) с минимальным или максимальным весом необходимо и достаточно при построении остовного леса с использованием условий теоремы 1, последовательно выбирать ребра в порядке неубывания или невозрастания их весов .

Доказательство. Пусть $\{(V_j,E_j)\}$, j = 0,k частично построенный остовный лес графа G=(V,E), $T=\cup E_j$ , j= 0,k è $e_i$ — ребро с наименьшим весом, соединяющее вершину из $\{V_j\}$, j = 1,k с вершиной из $V\setminus\cup V_j$ , j = 0,k. Покажем, что существует остовный лес $T' = T\cup\{e_i\}$, такой, что для любого остовного леса $T''\supseteq T$, справедливо $C(T')<=C(T'')$, где: $C(T') = \Sigma c(e_j)$, для $\forall e_j\in T$; $C(T'')=\Sigma c(e_j)$, для $\forall e_j\in T''$; $c(e_j)$ — вес ребра $e_j$.

Предположим, что $T''\supseteq T$, не включает $e_i$ и имеет наименьший вес среди всех остовных лесов, содержащих T. Добавим ребро $e_i$ ê T″ и получим единственный цикл, содержащий ребро $e_i$ и некоторое другое ребро x, соединяющее вершину из $\{V_j\}$, j=1,k с вершиной из $V\setminus\cup V_j$ , j=0,k. Согласно предположению $c(e_i) <= c(x)$ и $x \notin T$. Удалим ребро x из T″ и получим новый

остовный ëåñ T′ = T″ \{x}∪{$e_i$}, содержащий T, ребро $e_i$ и имеющий вес не больше, чем вес T″ , ò. å. C(T′) <= C(T″).

Очевидно, что все сказанное справедливо для каждого ребра, выбираемого из {$e_i$}, i = 1,e при построении остовного леса с использованием условий теоремы 1. Это доказывает необходимость условия теоремы 2. Достаточность условия следует из того, что оснований для выбора нового ребра на каждом шаге построения остовного леса достаточно, чтобы гарантировать получение остовного леса с минимальным весом, содержащего выбранное ребро.

Аналогичные рассуждения можно провести для доказательства теоремы для случая построения остовного леса с максимальным весом.

Другое доказательство теоремы 2 непосредственно вытекает из теоремы Радо-Эдмондса [9, 10], суть которой заключается в следующем. Рассмотрим систему подмножеств $M_G$ = (E, F), где F множество различных остовных лесов, являющихся независимыми подмножествами E. Система $M_G$ образует графический матроид и задачу определения минимального или максимального остовного леса на взвешенном графе G = (V, E) можно рассматривать как комбинаторную задачу оптимизации для $M_G$ с использованием «жадного» алгоритма.

Пусть найден остовный лес графа с максимальным весом и множество ребер остовного леса S = {$e_i$}, i = 1, v-k упорядочено в лексикографическом порядке по невозрастанию весов, т. е. $c(e_1)>=c(e_2)>=\ldots>=c(e_{v-k})$. Рассмотрим произвольное независимое множество ребер T = {$e'_i$}, i = 1,m, m <= v-k è $c(e'_1) >= c(e'_2) >= \ldots >= c(e'_m)$. Докажем, что S лексикографически максимальная база матроида $M_G$ , ò. å. C(S) >= C(T) или, что для произвольного i <= m справедливо $c(e_i) >= c(e'_i)$ . Предположим, что $c(e'_i) > c(e_i)$ и рассмотрим независимые множества $S_{i-1}$ = {$e_1$ , …, $e_{i-1}$ }, $T_i$ = { $e'_1$ ,…, $e'_{i-1}$, $e'_i$ }. Согласно аксиомам независимости матроидов существует элемент $e'_j$, j <= i , такой, что множество S′ = { $e_1$ ,…, $e_{i-1}$ , $e'_j$ } — независимое. Но S′ лексикографически больше S, поскольку $c(e'_j) >= c(e'_i) >= c(e_i)$, что противоречит лексикографической максимальности S и доказывает, что $c(e_i) >= c(e'_i)$, i = 1,m.

Доказательство теоремы 2 для построения остовного леса с минимальным весом для матроида $M_G$ проводится аналогично.

Рассмотрим алгоритм нахождения остовного леса графа G = (V,E) с минимальным или максимальным весом. Прежде всего, необходимо выбрать наиболее быстрый алгоритм сортировки ребер графа, поскольку время сортировки будет существенно влиять на вычислительную эффективность алгоритма в целом. Обширный библиографический обзор по методам и алгоритмам сортировки можно найти в широко известной книге Кнута [11]. В книге проведено тщательное сравнение различных алгоритмов сортировки и изложена история развития методов сортировки. Сразу отметим, что любой алгоритм, основанный на сравнении пар элементов, имеет среднюю трудоемкость и трудоемкость в худшем случае порядка Î(|E|log|E|). Поэтому совершенно очевидно, что использовать такой алгоритм сортировки для решения задачи нахождения минимального или максимального остовного дерева с трудоемкостью ниже, чем Î(|E|log|E|) бессмысленно. В связи с этим, для сортировки ребер будем применять алгоритм со сложностью порядка Î(|E|), основанный на методе преобразования ключа (веса ребра) в адрес оперативной памяти ЭВМ. Такой тривиальный подход к сортировке ребер оправдан низкой трудоемкостью и тем, что для большинства графов (сетей), описывающих реальные объекты, разброс значений весов ребер, как правило, незначителен.

Для описания алгоритма сортировки определим следующие переменные и структуры данных. Обозначим: N, M — число вершин и неориентированных ребер в графе; MINVAL, MAXVAL — минимальный и максимальный вес ребра; IS(M), ST(M), C(M) 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 упорядоченных по неубыванию ребер в массиве E(M); E(M) 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(M) 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(M) с весом I, а элемент U(I,2) указывает на номер последнего ребра J в массиве E(M) с весом I. При этом элемент E(K) указывает на следующее ребро в E(M) 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 1. 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 1.** Сортировка ребер по неубыванию весов с трудоемкостью O(M).

1. U ← 0; E ← 0 .
2. Для I = 1,M выполнить 3 6 .
3. L ← C( I ) MINVAL+1 .
4. Если U(L,1) = 0, то U(L,1) ← I; иначе E(U(L,2)) ← I.
5. U(L,2) ← I.
6. Конец цикла по I.
7. Стоп.

Где « ← » знак операции присваивания, а I = 1,M означает «для всех I от единицы до M с шагом 1» .

Оценим трудоемкость алгоритма. Пусть a, b, c время на операции присваивания, сравнения и перехода соответственно. Для определения приблизительной оценки, не будем отдельно различать время выполнения простой операции присваивания от операций сложения и операций присваивания со сложным индексом. Тогда трудоемкость шага 1 составит a*(MMAX*2) + a*M ; шага 2 a*M + b*M + c*M ; шага 3 a*M ; шага 4 b*M + a*M/2 + a*M/2 ; шага 5 a*M . Общее время сортировки $T_{сорт.}$ = M*(5a+2b+c) + MMAX*2a . Ясно, что реальная вычислительная трудоемкость сортировки будет зависеть от конкретного языка кодирования

алгоритма, компилятора и операционной системы, а скорость выполнения конкретной программы сортировки от мощности компьютера. Так как для реальных задач величина MMAX нульсравнима по отношению к M, то асимптотическая оценка трудоемкости алгоритма 1 всегда составляет O(M) операций.

Пусть MK максимальное число независимых компонент графа, MK = N/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(N) — массив, содержащий номера связных компонент, в которые входят вершины I, I = 1, N; H(MK,3) — массив, содержащий ссылки на списки номеров вершин компоненты MK в массиве G(N); G(N) — массив, содержащий списки номеров вершин связных компонент графа; EK(MK,2) — массив, содержащий ссылки на списки номеров ребер компоненты MK в массиве B(M); B(M) — массив, содержащий списки номеров ребер связных компонент графа. Массивы H(MK,3), G(N) и EK(MK,2), B(M) устроены аналогично массивам U(MMAX,2),E(M). Элемент H(I,3) содержит число вершин в I 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 2.** Построение остовного леса (дерева) с минимальным весом с трудоемкостью O(M).

1. A ← 0; NK ← 1; KK ← 0; NY ← 0; G ← 0; B ← 0; W ← 0; SV ← 0.
2. KNOV ← 0; KPRI ← 0; KSLI ← 0; KOST ← 0; KPUS ← 0; KSUZ ← 0.
3. Для I = 1, MMAX выполнить 4—31.
4. ND ¬ U(I,1).! Выбрать номер первого ребра из списка ребер с одинаковым весом
5. Пока ND /=0 выполнить 6—30.
6. L ← IS(ND); K ← ST(ND); L1 ← A(L); K1 ← A(K) . ! Выбрать левую и правую вершины ребра и номера компонент, в которые эти вершины входят

7. Если L1 /= 0 & K1 /= 0 & L1 = K1), òî KPUS ←KPUS+1; перейти к 28 . ! Обнаружено циклическое ребро, перейти на выбор нового ребра

8. Если L1 = 0 & K1 /= 0, то перейти к 15 . ! Присоединить к существующей компоненте новую вершину и новое ребро, перейти к присоединению к компоненте

9. Если L1 /= 0 & K1 = 0, то LL ← L; L11 ← L1; L ← K; L1 ← K1; K ← LL; K1 ← L11; перейти к 15 . ! Присоединить к существующей компоненте новую вершину и новое ребро, перейти к присоединению к компоненте

10. Если L1 = 0 & K1 = 0, то перейти к 12 . ! Образовать новую компоненту, добавить две новые вершины и одно новое ребро, перейти на образование компоненты

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

12. A(L) ← NK; A(K) ← NK; H(NK,1) ← L; G(L) ← K; H(NK,2) ← K . ! Образование новой компоненты

13. H(NK,3) ← 2; EK(NK,1) ← ND; EK(NK,2) ← ND; W(NK) ← C(ND) . ! KK ← KK+1; NK ← NK+1; NY ← NY+2; KNOV ← KNOV+1; KOST ← KOST+1; перейти к 28 .

14. A(L) ← K1; G(H(K1,2)) ← L; H(K1,2) ← L; H(K1,3) ← H(K1,3)+1 . ! Присоединение к компоненте

15. B(EK(K1,2)) ← ND; EK(K1,2) ← ND; W(K1) ← W(K1)+C(ND) . !

16. NY ← NY+1; KPRI  KPRI+1; KOST ← KOST+1; ïåðåéòè ê 28 .

17. Если H(L1,3) > H(K1,3), òî LP ← L1; L1 ← K1; K1 ← LP . ! Слияние компонент

18. W(K1) ← W(K1)+W(L1); W(L1) ← 0 .

19. KSLI ← KSLI+1; KOST ← KOST+1; KSUZ ← KSUZ+H(L1,3) .

20. G(H(K1,2)) ← H(L1,1); H(K1,2) ← H(L1,2); H(K1,3) H(K1,3)+H(L1,3).

21. B(EK(K1,2)) ← EK(L1,1); EK(K1,2) ← EK(L1,2); B(EK(K1,2)) ← ND.

22. EK(K1,2) ← ND; W(K1) ← W(K1)+C(ND).

23. J ← H(L1,1).

24. Пока J /= 0 выполнить 26 . ! Обновить массив вхождения вершин в компоненты

25. A(J) ← K1; J ← G(J) . ! после слияния двух компонент

26. KK ← KK-1 . ! уменьшить число компонент на 1

27. Если KK = 1 & NY = N), то перейти к 33 . ! Граф связный и все узлы включены в остовный лес
28. ND ← E(ND) . ! Выбрать номер очередного ребра
29. Конец цикла по ND.
30. Конец цикла по I.
31. Если NY < N, то SV ← SV+1.
32. Стоп.

Очевидно, что 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 = MMAX,1,-1 выполнить 4—31.

В записи алгоритма 2 в шаге 2 введены переменные KNOV, KPRI, KSLI, в которых накапливается соответственно число выполнений операторов: «перейти к 12» образование новой компоненты (добавление двух вершин и одного ребра); «перейти к 15» присоединение к существующей компоненте новой вершины и нового ребра; «перейти к 18» слияние двух независимых компонент и добавление нового ребра. В переменных KOST, KPUS, KSUZ соответственно подсчитывается число ребер в остовном лесе, число пустых проходов (число выполнений оператора «перейти к 28» пропуск ребер, образующих циклы), суммарное число слитых вершин (число выполнений оператора A(J) ¬ K1 в цикле в шаге 25). Все эти переменные введены для проведения дальнейшего экспериментального анализа трудоемкости алгоритма 2.

Приведем приблизительную оценку алгоритма по количеству выполняемых операций. Определим трудоемкость отдельных шагов:

шаг 1 a*M + 2a*N + a*MK + 4a;

шаги 3, 4, 31 2a*MMAX + b*MMAX + c*MMAX = (2a+b+c)*MMAX. В шаге 3 организован цикл по I = 1, MMAX для просмотра всех списков ребер с одинаковым весом, который может повторяться MMAX раз в худшем случае, когда граф является несвязным или содержит изолированные вершины;

шаги 5, 6, 28, 30 a*M + b*M + c*M + 4a*M + b*M = (5a+2b+c)*M. В шаге 5 организован цикл по ND для просмотра всех ребер с одинаковым весом в каждом из списков. Также как и в шаге 3 он может повторяться M раз в худшем случае. Циклы по I

и ND независимые, поэтому их трудоемкость складывается;

шаг 7 (3b + c)*KPUS;

шаги 8 , 9, (15-17) [(4b+c + 6b+6a+c)/2+(8a+c)] * KPRI = (11a+5b+2c)* KPRI ;

шаги 10, (12-14) (7b+c + 12a+c)* KNOV = (12a+7b+2c)* KNOV ;

шаги 11, (18-27) [(8b+c + b+3a/2+11a +(2a+b+c)*KSUZ/KSLI + a]*KSLI = [13,5a+9b+c+ +(2a+b+c)*KSUZ/KSLI ]*KSLI .

Значения KPUS, KPRI, KNOV, KSLI и KSUZ зависят от структуры и размерности графа, поэтому для их определения был проведен вычислительный эксперимент (использовалась ПЭВМ IP-IV с тактовой частотой 1,5 Ггц и оперативной памятью 2 Гб под управлением операционной системы Windows XP). Для проведения эксперимента была составлена тестовая программа на языке Digital Visual Fortran (DVF) компании Digital Equipment Corporation в среде Microsoft (MS) Developer Visual Studio (VS) DVF 6.1. Во внешней программе в режиме диалога вводились: число вершин графа N; число исходящих из каждой вершины ребер; границы изменения значений весов ребер от MINVAL до MAXVAL; параметр, управляющий выводом входных и выходных данных. Далее с помощью датчика случайных чисел (встроенной функции языка RAND( ) ) генерировались веса ребер от MINVAL до MAXVAL, формировались массивы IS, ST, C. Вся оперативная память, необходимая для работы алгоритмов 1 и 2 выделялась и освобождалась динамически во внешней программе. Время работы алгоритмов ($t_c$, $t_м$, $t_o$) фиксировалось встроенной подпрограммой cpu_time(T) непосредственно до входа и после выхода из алгоритмов сортировки и построения остовного леса. Работа алгоритмов проверялась на полных графах с числом вершин N от 1000 (499500 ребер) до 12000 (71994000 ребер), так как для меньших значений N подпрограмма cpu_time(T) не могла фиксировать время выполнения (подпрограмма возвращает время в секундах с точностью до двух знаков после запятой). Результаты эксперимента приведены в таблице 1. Где M′ – общее число ребер, просмотренных при построении остовного леса. Для KPUS, KPRI, KNOV, KSLI проценты указаны по отношению к M′, а для Mў проценты указаны по отношению к M. Для KSUZ указано суммарное число слитых вершин (число выполнений оператора A(J) ← K1 в цикле в шаге 25) и среднее число слитых

*Таблиця 1*

| № | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | Ср. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M(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 | |
| N | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 | 11000 | 12000 | |
| M′ | 3993 | 8276 | 15637 | 24805 | 23820 | 38117 | 39228 | 38888 | 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 |
| KPUS | 2994 | 6277 | 12638 | 20806 | 18821 | 32118 | 32229 | 30889 | 39524 | 43628 | 54814 | 65613 | |
| % | 74,98 | 75,85 | 80,82 | 83,88 | 79,01 | 84,27 | 82,16 | 79,43 | 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 | 18,59 | 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 |
| KSUZ | 1040 | 1656 | 1880 | 2332 | 2364 | 2597 | 2800 | 3068 | 3050 | 3004 | 2873 | 2754 | |
| | 5 | 5 | 5 | 6 | 6 | 6 | 7 | 8 | 8 | 7 | 7 | 7 | 6 |
| $t_c$,сек. | 0,00 | 0,05 | 0,11 | 0,22 | 0,33 | 0,49 | 0,71 | 0,94 | 1,19 | 1,45 | 1,83 | 2,28 | |
| $t_м$,сек. | 0,00 | 0,00 | 0,00 | 0,00 | 0,06 | 0,06 | 0,06 | 0,07 | 0,07 | 0,07 | 0,08 | 0,08 | |
| $t_o$,сек. | 0,00 | 0,05 | 0,11 | 0,22 | 0,39 | 0,55 | 0,77 | 1,01 | 1,26 | 1,52 | 1,91 | 2,36 | |

вершин за одну итерацию. Примем для средних значений в процентах для KPUS, KPRI, KNOV, KSLI соответственно значения: 81%, 15%, 2%, 2%. Тогда общая трудоемкость алгоритма 2 в худшем случае (для несвязного графа или графа с изолированными вершинами) составит

$T_{ìîä}$ = a*M + 2a*N + a*MK + 4a + (2a+b+c) * MMAX + (5a+2b+c) * M + (3b+c) * M* *0,81+ (11a+5b+2c) * M * 0,15 + (12a+7b+2c) * M * 0,02 + [13,5a+9b+c+(2a+b+c) *6 ] * *M * 0,02 = (8,4a+5,62b+2,29c) * M + 2,5a * N + (2a+b+c) * MMAX + 4a.

Для связного графа $T_{ìîä}$ = a*M + 2,5a*N + (2a+b+c)*MMAX + 4a + +(7,4a+5,62b+2,29c)* M * 0,0025 = (1,019a+0,014b+0,006c) * M+ 2,5a * N + (2a+b+c) * *MMAX + 4a.

Ясно, что асимптотическая сложность алгоритма 2 и сложность решения задачи в целом составит Î (Ì ). При тестировании программы генерировались также неполные связные графы с различным числом вершин и ребер и результаты вычислений сравнивались с результатами, полученными для этих графов программой, реализующей алгоритм Прима. При этом при увеличении числа вершин и ребер процентные значения KPUS, KPRI, KNOV, KSLI приближались к средним: 81%, 15%, 2%, 2%, а среднее отношение (M′/M)*100% к 0,25%. Результаты эксперимента показали, что приведенный алгоритм асимптотически лучше алгоритма Прима для графов с числом ребер меньше, чем $N^2/6$. При практическом использовании алгоритмов их быстродействие можно улучшить, если выполнение операций U ← 0, E ← 0, A ← 0, G ← 0, B ← 0, W ← 0 перенести во внешнюю программу.

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

В заключение отметим, что программа, реализующая приведенные алгоритмы многократно использовалась в пакете программ для решения задач оптимизации распределения дискретных потоков в многопродуктовых сетях большой размерности [12]

при проведении анализа различных проектируемых коммуникационных *сетей*.

* * *

1. Kruskal J. B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem / J. B. Kruskal, Proc. Amer. Math. Soc., 7 (1965), 48—50.

2. Prim R. C. Shortest Connection Networks and Some Generalizations / R. C. Prim, Bell Syst. Tech. J., 36 (1957), 1389—1401.

3. Dijkstra E. Two Problems in Connexion with Graphs / E. Dijkstra, Num. Math., 1, (1959), 269—271.

4. Berge C. Programming, Games and Transportation Networks / C. Berge, A. Ghouilla-Hourr, N.Y.: John Wiley & Sons, Inc., 1965.

5. Yao A.C.C. An 0(|E|loglog|V|) Algorithm for Finding Minimum Spanning Trees, Info. Proc. Let., 4 (1975), 21—25.

6. Cheriton D. Finding Minimum Spanning Trees / D. Cheriton, R.E. Tarjan, J. SIAM Comp., 5 (1976), 724—742.

7. Пападимитриу Х. Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц // Пер. с англ. — М.: Мир, 1985. — 512 с.

8. Рейнгольд Э. Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део // Пер. с англ. — М.: Мир, 1980. — 476 с.

9. Rado R. Note on Independence Function / R. Rado. Proc. London Math. Soc., 7, (1957), 300—320.

10. Edmonds J. Matroids and the Greedy Algorithm / J. Edmonds, Math. Programming, 1, (1971), 127—136.

11. Кнут Д. Искусство прграммирования для ЭВМ: Т.3 (Сортировка и поиск) / Д. Кнут. — М.: Мир, 1978.

12. Васянин В. А. Пакет программ для решения задач оптимизации распределения дискретных потоков в многопродуктовых сетях большой размерности / В. А. Васянин, А. И. Савенков // Интеллектуальные системы принятия решений и проблемы вычислительного интеллекта: материалы международной научной конференции, Евпатория, Украина. — 2008. — Том 2 (часть 1). — С. 41—44.

*Отримано: 5.10.2009 р.*