УДК 681.3.06
DOI: 10.24160/1993-6982-2017-5-111-116

Быстродействующие реализации метода Крускала построения минимального остова графа

В.С. Зубов

Задача построения минимального остова взвешенного графа относится к числу фундаментальных. Применение минимального остова графа обычно связывают с выполнением оптимальной по стоимости (весу) коммуникации узлов дискретных систем. Два базовых метода построения минимального остова графа были известны до появления современных компьютеров. Основным направлением совершенствования методов стало уменьшение вычислительной сложности. Этому способствовали изобретения в области вычислительных структур.

В предыдущей работе авторов дан подробный анализ наиболее совершенной реализации метода Прима. В конкурирующем методе Крускала велика вычислительная сложность первого этапа — этапа упорядочения ребер графа по возрастанию веса. Она и определяет общую сложность. В настоящей статье исследован эффект построения специальных структур данных для первого этапа метода. Изменен статус метода Крускала, который становится быстрее метода Прима.

В трех предлагаемых реализациях метода Крускала использована идея частичного упорядочения набора ребер. В первой из них взята слабая пирамида. Число сравнений ребер при ее построении минимально. Среднее число ребер, извлекаемых из пирамиды для построения остова, $L \approx V(0{,}333 \log_2 V + 0{,}45)$; логарифм двоичный. Среднее время построения остова стало меньше, но лучшим по-прежнему является метод Прима.

Так называемая d-арная пирамида прежде не использовалась в методе Крускала. В специальной ее реализации сокращено не только среднее число основных действий, но и число вспомогательных. Вычислительная сложность построения пирамиды оказалась наименьшей при $d > 4$. В формуле средних затрат времени построения остова первое слагаемое соответствует построению пирамиды, второе — извлечению из нее L ребер, третье — проверке и включению ребер в остов, машинозависимые коэффициенты определяются экспериментально. Эта реализация быстрее метода Прима за исключением графов малой плотности.

На первом этапе новой реализации метода Крускала для быстрого упорядочения ребер возможно использование системы цепных списков (стеков). За один просмотр описания графа строятся списки ребер. Каждый список включает ребра равного веса. Списки используются в порядке возрастания веса ребер, пока не будет построен остов. Эта реализация значительно быстрее метода Прима. Испытания новых реализаций метода проведены на графах, случайных (псевдослучайных) по составу и весу ребер. В статье также представлены коды реализаций.

Ключевые слова: граф, минимальный остов, алгоритмы Крускала и Прима, вычислительная сложность, время поиска.

Для цитирования: Зубов В.С. Быстродействующие реализации метода Крускала построения минимального остова графа // Вестник МЭИ. 2017. № 5. С. 111—116. DOI: 10.24160/1993-6982-2017-5-111-116.

Fast Implementations of the Kruskal Method for Constructing the Minimum Spanning Tree of a Graph

V.S. Zubov

Constructing the minimum spanning tree of a weighted graph falls in the category of fundamental problems in the graph theory. The minimum spanning tree of a graph is generally applied for setting up communication between the nodes of discrete systems that is optimal in terms of cost (weight). Two basic methods for constructing the minimum spanning tree were known before the advent of modern computers. Reduction of computational complexity has become the main line of improving the methods. Inventions made in the field of computational structures facilitated these efforts.

The previously published paper by V.S. Zubov and V.V. Kraskov gave a detailed analysis of the most sophisticated implementation of the Prim method. The competing Kruskal's method features a high computational complexity of its first stage, at which the graph edges are sorted in ascending order of their weights. It is exactly this stage that determines the overall complexity of the method. The article analyzes the effect from constructing special data structures for the first stage of the method. As a result, the status of the Kruskal method has been changed, which becomes faster than the Prim method.

Three implementations of the Kruskal method are proposed, in which an idea of partially ordering the set of edges is used. In the first of them, the weak pyramid is used. The number of edge comparison operations in the course of its construction is minimal. The average number of edges extracted from the pyramid for constructing the minimum spanning tree is $L \approx V(0.333 \log_2 V + 0.45)$. The average time taken to construct the minimum spanning tree has become shorter, but the Prim method still remains more advantageous.

A so-called d^{th} pyramid was not previously used in the Kruskal method. Its special implementation contains a fewer average number of not only the main operations, but also a fewer number of the auxiliary operations. The computational complexity of the pyramid construction stage was found to be minimal at $d > 4$. A formula for estimating the average time taken to construct the minimum spanning tree is given, in which the first, second, and third terms correspond, respectively, to the time taken to construct the pyramid, to extract L edges from it, and to check the edges and

include them into the minimum spanning tree; the formula also contains machine-dependent coefficients which are determined experimentally. This implementation is faster than the Prim method, except for low-density graphs.
The system of chained lists (stacks) can be used to speed up the edge ordering process at the first stage of the new implementation of the Kruskal method. Edge lists are drawn up within a single graph description viewing cycle. Each list includes edges of equal weight. The lists are used in ascending order of edge weight, until the minimum spanning tree construction process is completed. This implementation is significantly faster than the Prim method.
The new implementations of the method were tested on graphs random (pseudorandom) in terms of edge content and weights. References to the implementation codes are also given in the article.

Key words: graph, minimum spanning tree, Kruskal's algorithm, Prim's algorithm, computational complexity, search time.

For citation: Zubov V.S. Fast Implementations of the Kruskal Method for Constructing the Minimum Spanning Tree of a Graph. MPEI Vestnik. 2017;5: 111—116. (in Russian). DOI: 10.24160/1993-6982-2017-5-111-116.

Задача построения минимального остова (МО) взвешенного графа относится к числу фундаментальных. Ее размерами являются числа вершин V и ребер E графа. Базовые алгоритмы решения хорошо известны и исследованы [1]. Плотность или насыщенность графа — отношение E/V. Графы малой насыщенности называют разреженными, большой — плотными. Хорды — это ребра графа, не входящие в МО.

Асимптотические оценки вычислительной сложности (ВС) базовых алгоритмов даны в [1]. Детальный анализ ВС модифицированного алгоритма Прима описан в [2], и сделан вывод о том, что он гораздо быстрее конкурирующих алгоритмов построения МО.

Начиная с конца 40-х гг. XX в. данные алгоритмы совершенствуются, что связано с изобретениями в структурах данных и алгоритмов. Предложены новый подход в реализации метода Крускала, меняющий его статус и делающий его значительно быстрее (в настоящем случае трудоемкость используется как синоним ВС).

Результат метода представлен массивом *KR* ребер, образующих МО. В методе Крускала моделируется лес деревьев (частей МО), соединение которых приводит к получению МО. Представления деревьев назовем ядрами. Исходные ядра — вершины графа. Система ядер необходима для проверки ребер, предотвращающей включение хорд в массив *KR*. Если вершины, инцидентные проверяемому ребру, принадлежат одному и тому же ядру, значит, они уже связаны. Добавление ребра образует цикл, а в остове нет (и не может быть) циклов.

Ядро — это построенное на подмножестве вершин сильно ветвящееся корневое дерево. Приведем записанную на языке Pascal функцию Poisk поиска корня того ядра, которому принадлежит вершина z. Функция должна быть включена в указанные далее программы. Совокупность ядер представлена массивом p курсоров. Курсор $p[i]$ — это номер вершины, следующей за вершиной i на пути к корню ядра, хотя бы и номер корня.

```
Function Poisk (z: integer): integer;
  begin
    If z <> p[z] then p[z]:= Poisk (p[z]);
    // значение z равно p[z] только в корне ядра
    Poisk:= p[z];
  end;
```

С помощью функции Poisk реализуется один из способов решения задачи связности, рассмотренных в [1, § 1.3]. Для проверки ребра $\{u, v\}$ нужно дважды обратиться к функции с аргументами u, v; если она возвращает один и тот же номер корня, то ребро $\{u, v\}$ является хордой. Достаточно сложно найти корень $O(1)$, поскольку большая часть вершин находится на первом ярусе дерева (ядра). Заметим, что функция заменяет все ссылки на пути поиска корня ссылкой на найденный корень, и дерево растет не «ввысь», а «вширь». Этому способствует и способ слияния двух ядер. Одно из них назовем поглощаемым, другое — поглощающим. При слиянии корень поглощаемого ядра ссылается на корень поглощающего. Если высота этих деревьев различна, то поглощается ядро меньшей высоты. Вычислительная сложность слияния двух ядер равна $O(1)$. Число слияний ядер ограничено сверху значением V, общая ВС равна $O(V)$.

Выбор ребер для анализа и включения в МО должен проводиться в порядке неубывания их веса. Требуется либо сортировка, либо построение структуры, реализующей подобный выбор. Этой структурой служит один из вариантов приоритетной очереди.

Если в графе имеются хорды малого веса, то из очереди извлекается для проверки больше, чем $(V - 1)$ ребер. Пусть эти ребра будут кандидатами на включение в МО. Среднее число кандидатов L вычисляется по приближенной формуле

$$L \approx V(0{,}333\log_2 V + 0{,}45), \qquad (1)$$

представленной для связного взвешенного графа со случайным набором ребер, имеющих случайный вес. Например, если в случайном графе 1000 вершин и 400 000 ребер, следовательно, в среднем анализируется 3730 ребер малого веса, из них включаются в остов лишь 999 ребер. Формула (1) не применима к разреженным графам из-за ограниченной области выбора ребер. В графах средней и большой насыщенности $L << E$, причем значение L слабо зависит от E.

Построение пирамиды — трудоемкая первая фаза алгоритма. Построение слабой пирамиды требует $(E - 1)$ сравнений весов ребер [3, 4]. Это минимум на множестве вариантов пирамид, поэтому в одной из предлагаемых реализаций метода Крускала используется слабая пирамида. С помощью нее выбирается

L ребер-кандидатов. Взятие кандидата сопровождается восстановлением пирамиды, ВС которого равна $\log_2 E$. Суммируя ВС построения пирамиды, поиска корней, слияния ядер и упорядоченного выбора K ребер, для данной реализации метода Крускала получим формулу средних затрат времени

$$k_1 E + k_2 L \log_2 E + k_3 V, \qquad (2)$$

где k_1 — k_3 — машинозависимые коэффициенты, определяемые экспериментально.

Аналогичная формула в [2] получена для модифицированного алгоритма Прима, где ребра графа выступают в роли кандидатов на включение в приоритетную очередь. Общее среднее число L кандидатов зависит от V и средней степени $2E/V$ вершин графа и вычисляется по асимптотической формуле

$$L \approx 0{,}938V(\ln(2E/V) + 0{,}577) \qquad (3)$$

с эмпирическим множителем 0,938. В насыщенных графах $L << E$. Так, если в графе 1000 вершин и 400 000 ребер, то в среднем в приоритетную очередь будут включаться лишь 6810 ребер из 400 000, из них будет выбрано 999 ребер.

Характер действий в методах Прима и Крускала различен, поэтому необходимо экспериментальное исследование производительности реализаций методов.

Ниже дан код быстродействующей реализации метода Крускала, записанный на языке Pascal как функция Krus, возвращающая суммарный вес МО. Параметр *KR* — массив ребер графа, образующих МО (результат); *G* — описание графа (множество ребер графа); метка rav[*c*] представляет высоту дерева-ядра, корнем которого является вершина *c*.

```
// Типы, используемые в программах:
Type Tip = integer;     // тип веса ребер
     TEdge = Record u, v: integer; w: Tip end;
   // w – вес ребра, u, v — инцидентны ребру
Function Krus (Var G, KR: Array of TEdge;
                              V, E: integer): Tip;
Var a, b, i, j, h, k, t: integer; q:TEdge; s: Tip;
    p: Array of integer; r, rav: Array of byte;
begin
  h:= E–1; SetLength (r, E);
  SetLength (p, V); SetLength (rav, E);
  For j:=0 to h do r[j]:=0;
  For j:= h downto 1 do
  begin k:= j; Repeat t:= k; k:= k div 2 Until Odd (t);
   If G[k].w > G[j].w then
    begin q:= G[k]; G[k]:= G[j]; G[j]:= q; r[j]:= 1 end
  end; // Конец цикла построения пирамиды
  For i:= 0 to V–1 do begin p[i]:= i; rav[i]:= 1 end;
  j:= 0; s:= 0;
   Repeat // Начало цикла выбора и проверки ребер
   If j = 0 then i:= 0 Else
    begin k:=1;
      While k+k+r[k] < h do k:= k+k+r[k];
      While k > 0 do
      begin
       If G[h].w > G[k].w then
         begin q:= G[k]; G[k]:= G[h]; G[h]:= q;
          r[k]:= 1 – r[k] end;
        k:= k div 2
      end;
     i:= h; Dec(h) // Выбрано для проверки ребро G[i]
    end;
  a:= Poisk (G[i].u); b:= Poisk (G[i].v);
  If a<>b then              // a и b — корни различных ядер
    begin If rav[a]> rav[b] then // Начало слияния ядер
          p[b]:= a // Ядро «a» поглощает ядро «b»
         Else p[a]:= b; // Ядро «b» поглощает ядро «a»
    // При равных метках увеличивается высота «b»
      If rav[a] = rav[b] then Inc (rav[b]);
      KR[j]:= G[i]; Inc (s, G[i].w); Inc (j)
    end                 // Ребро G[i] включено в остов
   Until j = V–1;
   Krus:= s           // s — суммарный вес ребер остова
end;
```

Функция Krus выполняется заметно быстрее функции, использующей в качестве приоритетной очереди традиционную бинарную пирамиду.

Благодаря простой логике метода Крускала, его легко модифицировать. Согласно экспериментальным данным [1], использование сортировки на первой фазе метода (взамен очереди) оправдано в случае разреженных графов. В [1] предложено применять алгоритм QuickSort, содержащий простые циклы, экономный по числу перемещений сортируемых записей. Он не оптимален по числу сравнений, а время сортировки в худшем случае равно $O(E^2)$. Найдем замену.

Так называемая *d*-арная пирамида работает как очередь в версии Джонсона алгоритма Прима. Ее внедрение в метод Крускала до сих пор не рассматривалось, хотя именно в нем существенно уменьшается ВС. С увеличением d среднее число перемещений записей при построении и использовании пирамиды уменьшается. Если $d > 4$, высота дерева пирамиды в два с лишним раза меньше высоты бинарной пирамиды. Соответственно сокращаются и трассы перемещений элементов.

Функция Krus5, где очередь представлена 5-арной пирамидой, представляет новую реализацию метода Крускала. В ней на ускоренной первой фазе метода используется экономный способ построения дерева пирамиды. Для этой реализации метода средние затраты времени

$$k_4 E + k_5 L \log_5 E + k_6 V, \qquad (4)$$

где L взято из формулы (1); k_4 — k_6 — машинозависимые коэффициенты, определяемые экспериментально.

```
Function Krus5 (Var G, car: Array of TEdge; V, E: integer): integer;
Var a, b, i, j, k, m, h, q, s, t: integer; z:TEdge;
    p: Array of integer; r: Array of byte;
begin
  h:= E–1; SetLength(p, V); SetLength(r, V);
  t:= h div 5 –1; q:= (t–1) div 5; k:= 5*q + 6;
  For j:= q+1 to t do
  begin If G[k].w < G[k+1].w then i:=k else i:=k+1;
```

```
      For m:= k+2 to k+4 do If G[m].w < G[i].w then i:= m;
      If G[i].w < G[j].w then begin z:= G[j]; G[j]:= G[i]; G[i]:= z end;
      k:=k+5
  end;
  For j:= q downto 0 do
  begin z:= G[j]; k:=5*j+1; t:= j;
    While k < h-3 do
    begin If G[k].w < G[k+1].w then i:=k else i:=k+1;
     For m:= k+2 to k+4 do If G[m].w < G[i].w then i:= m;
     If G[i].w < z.w then G[t]:= G[i] Else Break;
     t:= i; k:=5*i+1
    end;   If t <> j then G[t]:= z;
  end;
  For i:= 1 to h mod 5 do
    begin k:= h – i +1; Repeat j:= (k–1) div 5;
                        If G[j].w <= G[k].w then Break;
                        z:= G[k]; G[k]:= G[j]; G[j]:= z; k:= j
                        Until k = 0
    end;
  for i:=0 to V–1 do begin p[i]:= i; r[i]:= 1 end;
  k:=0; s:=0;
  Repeat
      a:= Poisk (G[0].u);  b:= Poisk (G[0].v);
      if a<>b then
      begin If r[a]> r[b] then p[b]:=a else p[a]:=b;
      If r[a] = r[b] then Inc(r[b]);
      car[k]:= G[0]; Inc(s, G[0].w); Inc(k) end;
  z:= G[h]; G[h]:= G[0]; Dec(h);
  j:=0; i:= 1;
     While h – i > 3 do
    begin If G[i+2].w < G[i+3].w then t:= i +2 else t:= i +3;
     If G[i+4].w < G[t].w then t:= i+4;   If G[i+1].w < G[i].w
     then Inc(i);
     If G[t].w < G[i].w then i:= t;  If G[i].w >= z.w then begin
     i:=h+1; Break end;
      G[j]:= G[i];  j:= i;  i:= 5*i + 1
    end;
    If i <= h then begin For t:= i+1 to h do If G[t].w < G[i].w
    then i:= t;
                       If G[i].w < z.w then begin G[j]:= G[i]; j:=i end
                       end;
     G[j]:= z
    Until k = V–1;   Krus5:= s
  end;
```

Оценим ВС реализации Krus5 на примере использования плотного графа, где $V = 1000$, $E = 400\,000$. При построении 5-арной пирамиды в среднем выполняется около 224 000 перемещений записей, а при извлечении из нее L записей — 36 000 перемещений. При сортировке E записей алгоритмом QuickSort в среднем выполняется около 1 719 000 обменов записей, т. е. 5 157 000 перемещений. По числу сравнений QuickSort также проигрывает. Даже если пойти на усложнение алгоритма, реализуя частичную сортировку, до начала выбора ребер QuickSort выполняет примерно $1{,}5E = 600\,000$ перемещений записей.

Время поиска МО алгоритмом Krus5 сокращается примерно в 1,5 раза по сравнению с лучшими реализациями метода Крускала. Только в разреженных графах при $E < 4V$ быстрее становится вариант метода Крускала, использующий на первой фазе алгоритм QuickSort. В [1] утверждается, что разработка алгоритмов с гарантированным линейным временем выполнения на разреженных графах все еще остается недостижимой целью. Для графов с целочисленными весами ребер, а это весьма распространенный случай, алгоритм с линейной оценкой ВС построен, его идея проста.

Назовем доменом множество потенциальных значений веса ребер. При порядковом типе веса ребер используем конечный домен, например [25, 10 000]. Мощность D домена может быть значительной. За один просмотр массива ребер построим D ассоциаций ребер с одинаковым значением веса. Возможны пустые (нет ребер какого-либо веса) и тривиальные ассоциации, содержащие одно ребро.

Найдем непустую ассоциацию с наименьшим значением веса ребер и по методу Крускала используем все ее элементы (ребра), аналогично найдем и используем следующую (по возрастанию веса ребер) непустую ассоциацию и т. д., вплоть до выбора в остов $(V – 1)$ ребер.

Сортировка является возможным, но медленным способом получения ассоциаций. Отказавшись от нее, представим ассоциации стеками. Указатели верхушек стеков — элементы массива C. Ребро веса key будет включаться в стек, указатель верхушки которого — C[key]. За один просмотр массива ребер получим все непустые стеки. Такой способ упорядочения в [3] назван отобразительным в отличие от сопоставительного, использующего сравнения ключей.

Ниже приведен код новой реализации метода Крускала, записанный как функция KrusNew на языке Pascal. Ее параметры те же, что и у функции Krus. Домен представлен парой ($k1$, $k2$), где $k1$, $k2$ — наименьшее и наибольшее возможные значения веса ребер.

Асимптотическая ВС данной реализации равна $O(E + V + D)$. Поскольку используется связный граф, $E \geq V – 1$, оценка упрощается: $O(E + D)$.

```
Function KrusNew (Var G, car: Array of TEdge;
                                  V, E: integer): integer;
Const k1=25; k2=10000;
Var a, b, i, j, k, s: integer; rav: Array of byte;
    cep, p: Array of integer; c: Array[k1..k2] of integer;
begin
  SetLength (cep, E); SetLength (p, V); SetLength (rav, V);
  For j:= k1 to k2 do c[j]:= –1; // Пустые головы списков
  For j:= 0 to E–1 do   // Цикл заполнения списков
  begin k:= G[j].w; cep[j]:= c[k]; c[k]:=j end;
  For i:= 0 to V–1 do begin p[i]:= i; rav[i]:= 1 end;
   s:= 0; j:= 0; k:= k1; i:= c[k1];
  Repeat
    While i < 0 do begin Inc(k); i:=c[k] end;
    a:= Poisk (G[i].u); b:= Poisk (G[i].v);
    If a<>b then  // a и b — корни различных ядер
     begin If rav[a]> rav[b] then // Начало слияния ядер
          p[b]:= a // Ядро «a» поглощает ядро «b»
        Else p[a]:= b; //Ядро «b» поглощает ядро «a»
     If rav[a] = rav[b] then Inc (rav[b]);
```

```
    car[j]:= G[i]; Inc (s, G[i].w); Inc (j)
   end;
  i:= cep[i]            // Продвижение по цепному списку
 Until j = V–1;
 KrusNew:= s
end;
```

Отметим особенности данной реализации метода. Она допускает отрицательные веса ребер, которые могут иметь совпадающие веса. Программный код невелик и может быть сокращен, если веса ребер уникальны. Исходный массив ребер не подвержен изменениям. Главная особенность заключается в рекордном быстродействии, так как за один просмотр массива ребер они распределяются по цепным спискам и готовы к использованию.

Код функции KrusNew предполагает высокую емкостную сложность алгоритма в случае домена большой мощности. Разработана и испытана функция, свободная от этого недостатка. Упорядочение ребер в ней частичное. Вначале строится M цепных списков ребер малого веса. Если после их использования МО еще не получен, строится и используется следующее (по весу ребер) число M списков, вплоть до построения МО. Величина M — это число элементов массива C. Массив C голов списков используется заново для каждой группы.

Значение M влияет на быстродействие программы. Пусть D = 1 000 000, а M = 1200, тогда если веса ребер случайны, первых M списков достаточно для построения МО, но даже если требуется $2M$ или $3M$ списков, быстродействие по-прежнему остается рекордным.

Например, при уменьшении M до 1000, вероятнее всего, потребуются $2M$ списков. Дальнейшее уменьшение M для экономии памяти вряд ли имеет практический смысл.

Новые реализации метода Крускала являются примером снижения ВС за счет подбора структур. Оценить их влияние можно по данным эмпирического исследования.

Назовем плотностью графа дробь $100E/E_{max}$. В таблице для псевдослучайного графа различной плотности, содержащего 10 000 вершин, дано в условных единицах среднее время построения МО по методам Прима (t_{Prim})и Крускала в сопоставительном (t_{Krus}, t_{Krus5}) и отобразительном ($t_{KrusNew}$) вариантах. В эксперименте использовались функции Krus, Krus5, KrusNew и функция ModKK, реализующая метод Прима с приоритетной очередью [3].

Содержание строк таблицы свидетельствует о практически линейной зависимости времени построения МО от параметра E. В результате усовершенствований лидером по производительности при разных значениях V, E оказался метод Крускала (функции Krus5 и KrusNew). Итоги данного эксперимента отличаются от выводов, сделанных в [1].

Эксперимент начинался с отбора нетрудоемких реализаций методов из множества реализаций, указанных в [1]. Так, было установлено, что усложняющее программу использование в методе Прима приоритетной очереди в виде d-арной пирамиды весомых преимуществ не дает. Приводимые в [1] итоги испытаний алгоритма Борувки свидетельствуют о его бесперспективности.

Объективной причиной повторения экспериментального исследования и пересмотра оценки методов становятся усовершенствования пирамидальных структур и появление отобразительного варианта метода Крускала (функция KrusNew).

Требует уточнения утверждение, что при больших значениях насыщенности лучшей остается классическая реализация алгоритма Прима (КРП) [1].

Рассмотрим пример. В полном графе, содержащем 10 000 вершин, t_{Prim} примерно на 1 % больше времени $t_{КРП}$ выполнения классической реализации, но КРП значительно проигрывает при плотности графа 98 % и меньшей. Дело в том, что время $t_{КРП}$ практически не зависит от плотности графа. Заметный выигрыш КРП наблюдается только в насыщенных графах с малым числом вершин.

В отличие от метода Крускала метод Прима использует традиционное представление графа и удобное для применений представление МО. Их преобразования могут увеличить трудоемкость метода Крускала, поэтому реализация метода Прима с приоритетной очередью рассматривается как конкурентноспособная.

Литература

1. **Седжвик Р.** Алгоритмы на C++. СПб: Вильямс, 2016.

2. **Зубов В.С., Красков В.В.** Быстродействующий алгоритм построения кратчайшего каркаса взвешенного графа // Вестник МЭИ. 2009. № 6. С. 172—178.

Среднее время построения минимального остова

Плотность графа, %	1	10	30	50	70	80	90	100
$t_{KrusNew}$	8,5	42	109	174,5	245	276	310	347
t_{Krus}	30,1	128	324,5	512,5	689,3	792	869	971
t_{Prim}	16,7	95,5	241,4	386,8	532	604,5	676	752
t_{Krus5}	20,2	82	195	315	446	492	554	607

3. **Зубов В.С., Шевченко И.В.** Структуры и методы обработки данных: практикум в среде Delphi. М.: Филинъ, 2004.

4. **Dutton R.D.** Weak-heap Sort // BIT. 1993. V. 33. Pp. 372—381.

References

1. **Sedzhvik R.** Algoritmy na C++. SPb: Vil'yams, 2016. (in Russian).

2. **Zubov V.S., Kraskov V.V.** Bystrodeystvuyushchiy Algoritm Postroeniya Kratchayshego Karkasa Vzveshennogo Grafa. Vestnik MPEI. 2009;6:172—178. (in Russian).

3. **Zubov V.S., Shevchenko I.V.** Struktury i metody Obrabotki Dannyh: Praktikum v Srede Delphi. M.: Filin, 2004. (in Russian).

4. **Dutton R.D.** Weak-heap Sort. BIT. 1993;33: 372—381.

Сведения об авторе

Зубов Валерий Сергеевич — кандидат технических наук, доцент кафедры математического моделирования НИУ «МЭИ», e-mail: zubovvs@mpei.ru

Information about author

Zubov Valeriy S. — Ph.D. (Techn.), Assistant Professor of Mathematical Modeling Dept., NRU MPEI, e-mail: zubovvs@mpei.ru

Статья поступила в редакцию 02.03.2017