УДК 681.3.00: 681.3.001.57: 681.324.001.57

Н.Г. АКСАК, канд. техн. наук, ***С.А. ПАРТЫКА***, аспирант**,**
Ю.Ю. ЗАВИЗИСТУП, канд. техн. наук (г. Харьков)

ИСПОЛЬЗОВАНИЕ АЛГОРИТМОВ ПОИСКА КРАТЧАЙШЕГО ПУТИ НА ГРАФАХ

У статті пропонуються алгоритми, що дозволяють знаходити найкоротшу відстань, як для орієнтованих, так і для неорієнтованих графів. Детально розглянуті аспекти практичного застосування запропонованих алгоритмів. Комп'ютерне моделювання показує високу ефективність запропонованих алгоритмів при їхньому використанні у великих мережах передачі даних.

Algorithms which allows to find the shortest path both for oriented and for non-oriented graphs is offers in the article. The application of the suggested algorithms are given in detail. Simulation shows high effectiveness of the given algorithms when using it in large data transfer nets.

Постановка проблемы в общем виде. Современное общество характеризуется бурным развитием информационных технологий во всех областях человеческой деятельности. Внедрение этих технологий невозможно без использования компьютерной техники, но с ростом числа источников и потребителей информации, как правило, объединенных в вычислительные сети, возникают проблемы эффективного обмена информацией внутри сетей.

Появилась одна очень важная тенденция, затрагивающая в равной степени как локальные, так и глобальные сети. В них стала обрабатываться несвойственная ранее вычислительным сетям мультимедийная информация, чувствительная к задержкам при передаче данных между абонентами. Задержки обычно приводят к искажению такой информации. При этом задачи маршрутизации и выбора оптимальных потоков в сети являются одними из наиболее важных и сложных. Это относится в первую очередь к системам с очень сложной внутренней структурой, особенно если характеристики данной структуры меняются во времени [1 – 3]. Из-за того, что состояние больших вычислительных сетей подвержено частым изменениям, приходится постоянно решать задачи, связанные с эффективностью передачи данных.

Одним из требований, предъявляемых к решению задач маршрутизации, является минимизация времени вычисления кратчайших путей. Алгоритм должен находить оптимальное или близкое к оптимальному решение за минимальное время, что в итоге повышает эффективность обмена информацией в сети. И как следствие, происходит рост производительности и надежности сети, т.е. повышение качества обслуживания пользователей и снижение стоимости эксплуатации всей системы.

Обзор существующих методов решения. При решении задач, связанных с поиском кратчайших путей на графах был реализован вариант алгоритма Литтла для вычислений с использованием TCP/IP сети. Система основана на концепции клиент-сервер. Приложение-сервер осуществляет самостоятельный перебор решений на основе алгоритма Литтла. Приложения-клиенты, подключаясь к серверу, сообщают ему о готовности начать перебор. Сервер выделяет в дереве перебора поддерево достаточной глубины и пересылает его клиенту, а затем получает от клиента результат перебора этого поддерева. Оценка, полученная клиентом, используется сервером для отсечения в ходе собственного перебора. Объединенные усилия машин сети позволили достичь весьма высокой производительности**.**

Задачей всех алгоритмов маршрутизации является минимизация стоимости пути к адресату, по каким-либо критериям. Алгоритмы, определяющие маршрут на основании наименьшего числа промежуточных узлов, относятся к простейшим. Более сложные алгоритмы могут производить минимизацию стоимости пути по различным критериям, например, по задержке при передаче пакетов, по пропускной способности каналов связи или денежной стоимости передачи по данной линии связи. Наиболее известными алгоритмами поиска кратчайших путей на графах являются алгоритм Дейкстры и алгоритм Беллмана-Форда.

В основу алгоритма, предложенного Дейкстра, положен метод, при котором вершинам приписываются временные пометки, причем пометка вершины дает верхнюю границу пути к этой вершине. Построение кратчайших путей осуществляется поэтапно, начиная с некоторого узла ко всем остальным узлам. Величины пометок постепенно уменьшаются с помощью некоторой итерационной процедуры, и на каждом шаге итерации точно одна из временных пометок становится постоянной. Последнее указывает на то, что пометка уже не является верхней границей, а дает точную длину кратчайшего пути. Работа алгоритма заканчивается, когда все вершины имеют постоянные пометки.

Алгоритм Дейкстры применим только для графов с положительными весами дуг. В общих чертах принцип работы алгоритма Дейкстры выглядит следующим образом [4].

Пусть дан граф $G(X, \Gamma)$, где $\{x_1,\ldots,x_n\} = X$ – множество вершин графа. Соответствие Γ показывает взаимосвязи вершин. Дугам графа приписаны веса $c(x_i, x_j) \geq 0$, задаваемые матрицей C размерности $n \times n$.

Через $x_s \in X$ обозначена начальная вершина, относительно которой ищутся все кратчайшие пути.

Пусть $L(x_i)$ – пометка вершины x_i, т.е. длина пути от начальной вершины x_s к вершине x_i.

Присвоение начальных значений.

Шаг 1. Положить $L(x_s) = 0$ и считать эту пометку постоянной. $L(x_i) = \infty$ для всех $x_i \neq x_s$ и считать эти пометки временными. Положить $p = s$.

Обновление пометок.

Шаг 2. Для всех $x_i \in \Gamma(x_p)$, пометки которых временные, изменить пометки в соответствии со следующим выражением:

$$L(x_i) \leftarrow \min[L(x_i), L(x_p) + c(x_p, x_i)].$$

Превращение пометки в постоянную.

Шаг 3. Среди всех вершин с временными пометками найти такую, для которой $L(x_i^*) = \min[L(x_i)]$.

Шаг 4. Считать пометку вершины x_i^* постоянной и положить $x_p = x_i^*$.

Шаг 5. Если все вершины помечены как постоянные, то пометки дают длины кратчайших путей. Останов. Если некоторые пометки являются временными, то перейти на шаг 2.

Если найдены все длины кратчайших путей, то сами пути можно получить, используя метод описанный в [4] при помощи следующей формулы:

$$L(x_i^*) + c(x_i^*, x_i) = L(x_i),$$

где x_i^* – вершина, предшествующая x_i в кратчайшем пути от x_s к x_i.

Алгоритм Дейкстры имеет квадратичную вычислительную сложность.

В отличие от алгоритма Дейкстры, алгоритм Беллмана-Форда [4] применим для графов с общей матрицей весов. Он также является итерационным, но никакая пометка не рассматривается как постоянная. Этот алгоритм применим для графов с общей матрицей весов (т.е. в графе могут присутствовать дуги с отрицательными весами). Алгоритм Беллмана-Форда имеет кубическую вычислительную сложность для полного связного графа.

Основной недостаток приведенных выше алгоритмов заключается в необходимости полного перебора всех вершин графа при поиске решения, что приводит в итоге к нерациональным вычислительным затратам, либо для реализации этих алгоритмов требуется очень мощная аппаратная база [5].

Постановка задачи. Для графа как ориентированного, так и неориентированного, у которого элементы матрицы весов несуществующих дуг равны бесконечности и неориентированные ребра представляются в виде пары направленных дуг, построить такой алгоритм поиска кратчайших путей на графе, который обладает следующими свойствами:

– вычислительная сложность алгоритма близка к квадратичной;

– элементы матрицы весов графа могут быть положительными, отрицательными или равными нулю.

Описание алгоритма. Пусть дан граф $G(X,\Gamma)$, где $\{x_1,\ldots,x_n\} = X$ – множество вершин графа, дугам которого приписаны веса, задаваемые матрицей

$$M = \begin{vmatrix} m(x_1,x_1) & \cdots & m(x_1,x_n) \\ \vdots & \vdots & \vdots \\ m(x_n,x_1) & \cdots & m(x_n,x_n) \end{vmatrix}$$

размерности $n \times n$, $x_s \in X$ – начальная вершина, относительно которой ищутся все кратчайшие пути. Элементы $m(x_i, x_j)$ – матрицы весов M могут быть как положительными, так и отрицательными или равны нулю. Если в графе G дуга (x_i, x_j) отсутствует, то принимаем ее вес $m(x_i, x_j) = \infty$.

Пусть $A^{t+1} = \begin{bmatrix} A^t \\ x_k \end{bmatrix}$ – вектор размерности k представляет собой очередь активных вершин, где вектор A^t – очередь сформированная из $t = k-1$ поступивших ранее вершин, а x_k – последняя поступившая активная вершина. На стадии инициализации $A^0 = [x_s]$.

Введем понятие активности вершины. Будем считать, что изменять текущие длины путей к смежным вершинам может только активная вершина. Обозначим через a_i – активность вершины x_i:

$a_i = 1$ означает, что вершина x_i является компонентом вектора A^t, но еще не была рассмотрена, т.е. является активной;

$a_i = 0$ означает, что вершина x_i еще не была активна (т.е. не является компонентом вектора A^t);

$a_i = 2$ означает, что вершина x_i была рассмотрена в векторе A^t.

Введем вектор $D = [d_{x_1},\ldots,d_{x_n}]$ размерности $n \times 1$, где компонент вектора d_{x_i} указывает вершину, предшествующую вершине x_i в текущем кратчайшем пути от начальной вершины x_s к вершине x_i.

Пусть $L(x_i)$ – текущая длина пути от начальной вершины x_s к вершине x_i.

Алгоритм 1

Шаг 1. Инициализация. Положим длину пути к вершине x_s $L(x_s) = 0$, а для всех остальных вершин x_i – $L(x_i) = \infty$, активность вершины x_s

становится равной $a_s = 1$ (вершина x_s становится первым компонентом вектора A^t). Положим $A^0 = [x_s]$, $k = 1$, $a_i = 0$ для всех $x_i \neq x_s$.

Шаг 2. Выбираем активную вершину x_i из вектора A^t. В случае отсутствия активных вершин конец работы алгоритма.

Шаг 3. Находим вершину $x_j \in \Gamma(x_i)$, $x_i \neq x_j$. В случае, если просмотрены все вершины x_j, то положить $a_i = 2$ и перейти на шаг 2.

Комментарий. После того как рассмотрены все смежные с x_i вершины, активность вершины x_i устанавливается в $a_i = 2$. Это означает, что вершина x_i была рассмотрена в векторе A^t. При этом вершина x_i остается в векторе A^t, но на шаге 2 производится переход к следующей активной вершине.

Шаг 4. Изменяем длину пути к вершине x_j исходя из следующего правила:

1. Если $a_j = 1$ и $L(x_j) > L(x_i) + m(x_i, x_j)$, то положить $L(x_j) = L(x_i) + m(x_i, x_j)$, $d_{x_j} = x_i$;

2. Если $a_j = 0$, то положить $L(x_j) = L(x_i) + m(x_i, x_j)$, $a_j = 1$, $A^{t+1} = [A^t : x_j]^T$, $k = k + 1$, $d_{x_j} = x_i$;

3. Если $a_j = 2$ и $L(x_j) > L(x_i) + m(x_i, x_j)$, то положить $L(x_j) = L(x_i) + m(x_i, x_j)$, $a_j = 1$, $A^{t+1} = [A^t : x_j]^T$, $k = k + 1$, $d_{x_j} = x_i$.

Если длина пути к вершине x_j не изменилась, то перейти на шаг 3.

Комментарий. Для второго и третьего случаев помещаем вершину x_j в вектор A^{t+1} (в первом случае x_j уже находится в векторе A^t и все вершины, которые могут изменить длину пути к x_j будут рассмотрены в векторе A^t раньше нее).

При поиске очередной активной вершины достаточно перейти к следующему компоненту вектора A^t. Данный подход позволяет на шаге 2 отказаться от перебора всех вершин при поиске очередной активной вершины. Следует отметить, что размерность вектора A^t увеличивается каждый раз на 1 при поступлении новой активной вершины.

Шаг 5. Поиск цикла отрицательного суммарного веса.

5.1. Положить $r = x_i$.

5.2. Если $r = x_s$ перейти на шаг 3.

5.3. Если $r = x_j$, конец работы алгоритма. Обнаружен цикл отрицательного веса.

5.4. Положить $r = d_r$. Перейти на 5.2.

Введение понятия активности позволяет на шаге 3 исключить возможность нахождения в векторе A^t одновременно нескольких активных копий вершины, что заметно снижает вычислительные затраты при работе алгоритма и делает эффективной организацию очереди активных вершин.

После окончания работы алгоритма множество $L = \{L(x_1),\ldots,L(x_n)\}$ содержит длины кратчайших путей от начальной вершины x_s к каждой вершине. Из вектора D можно получить маршруты от вершины x_s ко всем остальным вершинам графа.

Отличие алгоритма 2 от алгоритма 1 заключается в том, что в первом алгоритме при поиске смежных вершин на шаге 3 приходится просматривать все вершины графа. Данный подход оправдан в тех случаях, когда выполняется поиск кратчайших путей один раз.

В случаях, если изменения характеристик графа во времени не носят глобальный характер, например, появились или исчезли только несколько дуг, и поиск кратчайших путей на графе производится многократно, для каждой вершины предварительно формируется $(z \times 1)$-вектор смежных вершин $Y_i = [y_1,\ldots,y_z]^T$, где z число смежных вершин для вершины x_i, а компоненты $y_1,\ldots,y_z$ вектора Y_i непосредственно сами смежные вершины:

Шаг 1. Положить $z = 0$.

Шаг 2. Выбрать вершину x_i. Если просмотрены все вершины – останов.

Шаг 3. Найти вершину $x_j \in \Gamma(x_i)$, $x_i \neq x_j$. Если просмотрены все вершины, перейти на шаг 1.

Шаг 4. Положить $z = z+1$, $Y_i = [Y_i : x_j]^T$. Перейти на шаг 3.

Данный подход позволяет отказаться от полного перебора всех вершин на шаге 3 алгоритма 1. Если характеристики графа изменились, то достаточно скорректировать соответствующие элементы матрицы M и компоненты соответствующих векторов Y_i, после чего снова повторить работу алгоритма с шага 1. Данный подход позволяет резко снизить затраты времени на работу алгоритма (например, при решении задач маршрутизации).

Алгоритм 2

Шаг 1 Алгоритма 1.

Шаг 2 Алгоритма 1.

Шаг 3. Выбираем вершину x_j из вектора $Y_i = [y_1,\ldots,y_z]^T$.

3.1. Если просмотрены все компоненты вектора Y_i, то положить $a_i = 2$, $h = 0$ и перейти на шаг 2.

3.2. Положить $h = h + 1$, $x_j = y_h$.

Шаг 4 Алгоритма 1.

Шаг 5 Алгоритма 1.

Методика моделирования и программная реализация. Экспериментальное моделирование было проведено с целью подтверждения теоретических выкладок, изложенных выше, проверки эффективности и сравнительного анализа работы предложенных алгоритмов и алгоритма Дейкстры.

При проведении эксперимента были проанализированы два параметра работы алгоритмов: затраты машинного времени $t(N)$ и число операций сравнения и сложения $H(N)$ в зависимости от числа вершин произвольного графа с неотрицательной матрицей связей.

Данный эксперимент проводился на ПЭВМ Pentium II-800 под управлением ОС Windows 2000, ОЗУ 128 Мбайт. Алгоритмы были реализованы на языке Object Pascal с применением визуальной среды программирования Delphi 5.

При программной реализации алгоритмов предусматривалось ограничение на число вершин графа до 2500 вершин, что связанно с применяемой техникой программирования и использованными типами переменных.

Для формирования положительной матрицы весов графа применялся метод “случайного места” [6], суть которого состоит в том, что предварительно вершины распределялись случайным образом внутри заданного прямоугольника, а затем определялись расстояния между ними. При этом для каждой вершины было введено ограничение в 5 смежных вершин. Количество вершин в графах варьировало от 100 до 1600 с шагом в 100 вершин. При проведении эксперимента случайным образом генерировались 20 графов для каждого значения числа вершин. При этом для каждого сгенерированного графа выборка значений параметров работы алгоритмов осуществлялась по 50 различным начальным вершинам.

Анализ полученных результатов. В ходе проведенного эксперимента были получены следующие результаты:

1. Алгоритм Дейкстры показал меньшие затраты машинного времени, чем алгоритм 1, но следует отметить, что алгоритм 1 в отличие от алгоритма Дейкстры применим для общей матрицы весов.

2. Алгоритм 2 показал намного меньшие затраты машинного времени, чем алгоритм Дейкстры. При этом относительный выигрыш по времени быстро увеличивается с ростом числа вершин графа.

Полученные зависимости затрат машинного времени от числа вершин графа приведены в табл. 1, а также на рис. 1 (приведены усредненные данные по 1000 выборкам для каждого числа вершин графа).

Таблица 1

Число вершин	Время выполнения, ms.		
	Дейкстра	Алгоритм 1	Алгоритм 2
100	0	0	0
200	1	1	0
300	3	3	0
400	6	6	0
500	9	11	0
600	14	22	1
700	18	32	1
800	24	43	2
900	31	61	3
1000	38	85	3
1100	46	131	4
1200	54	184	6
1300	64	237	6
1400	75	275	7
1500	85	300	9
1600	97	338	9

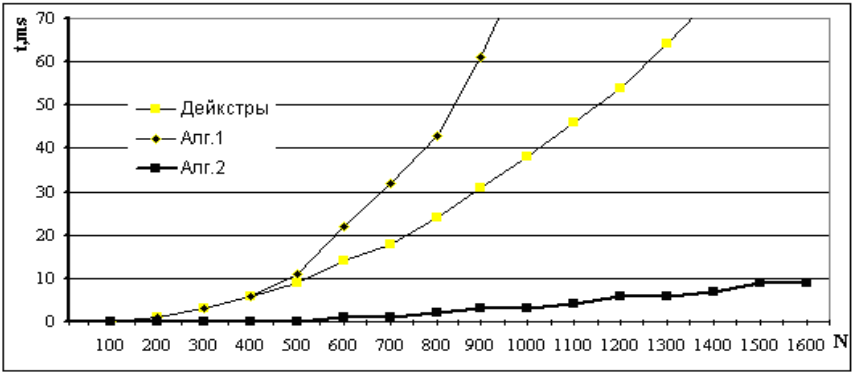

Рис. 1

3. Число операций сравнения и сложения, необходимых для нахождения всех кратчайших длин путей на графе от исходной вершины, для алгоритма 1

больше, чем у алгоритма Дейкстры, что связанно с поиском циклов с отрицательным суммарным весом.

4. Число операций сравнения и сложения для алгоритма 2 значительно меньше, чем для алгоритма Дейкстры.

Полученные данные по чилу операций сложения и сравнения представлены на рис. 2.

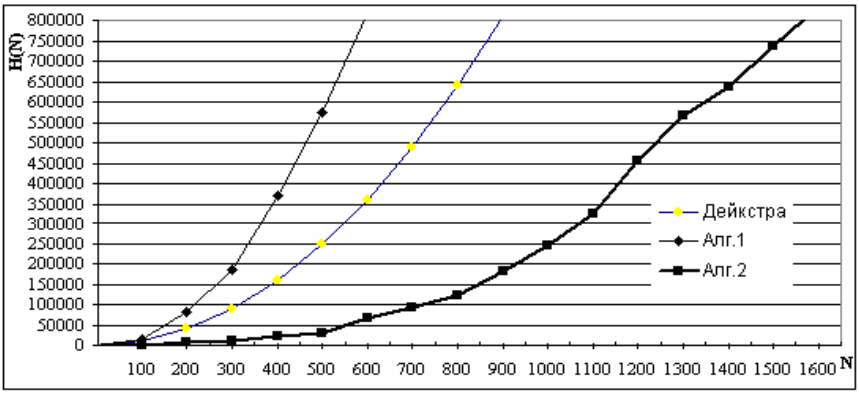

Рис. 2

Таблица 2

Число вершин	Время выполнения, ms.		
	Дейкстра	Алгоритм 1	Алгоритм 2
100	0	0	0
200	1	1	0
300	3	3	0
400	6	6	0
500	9	11	0
600	14	19	1
700	18	29	1
800	24	40	1
900	31	55	1
1000	38	75	2
1100	46	124	2
1200	54	175	2
1300	64	227	3
1400	75	261	3
1500	85	289	4
1600	97	325	4

В случае, если матрица весов имеет только положительные элементы, то целесообразно отказаться от поиска циклов с отрицательным суммарным весом. При этом для алгоритма 2 произойдет дополнительное снижение необходимого числа операций, и как следствие, снижение затрат машинного времени.

Результаты работы алгоритма 1 и алгоритма 2 без поиска циклов с отрицательным суммарным весом приведены в табл. 2, а также на рис. 3 и 4.

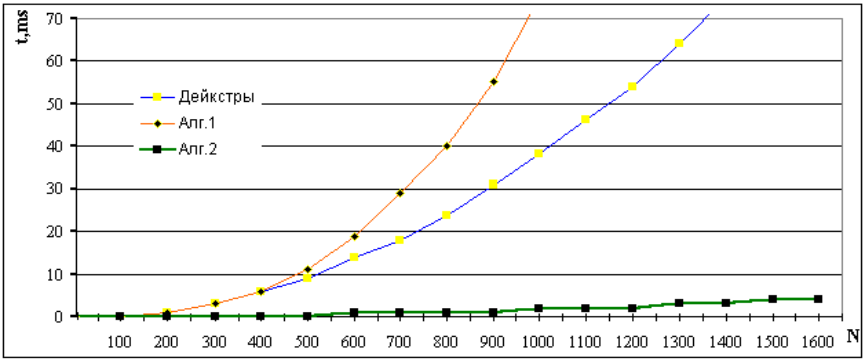

Рис. 3

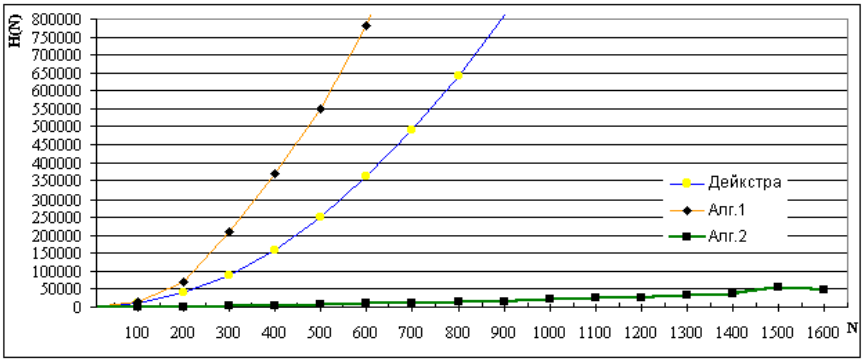

Рис. 4

Анализ полученных результатов работы алгоритмов показал, что при однократных расчетах кратчайших путей на графе для случаев с положительной матрицей весов более целесообразно использовать алгоритм Дейкстры, т.к. для алгоритма 2 необходимо предварительно построить вектора смежных вершин, на что требуется дополнительно выполнить N^2 операций сравнения и сложения.

В случае, если производятся многократные поиски и характеристики графа изменяются во времени не резко, например, появились или исчезли несколько дуг графа, более предпочтительным по отношению к алгоритму Дейкстры оказывается применение алгоритма 2. Если характеристики графа изменились, то достаточно скорректировать соответствующие элементы матрицы весов графа, а также необходимые компоненты векторов смежных вершин, после чего снова повторить работу алгоритма. Следует также отметить, что предложенные алгоритмы применимы для графов с общей матрицей весов.

Особо следует отметить ситуацию, когда требуется найти кратчайшие пути на графе для всех пар вершин. В этом случае при использовании алгоритма 2 (n-кратное выполнение алгоритма) затраты необходимого машинного времени оказываются очень малыми, за счет отказа от полного перебора при поиске смежных вершин.

Следует отметить возможность усовершенствования предложенных алгоритмов, если при их работе производить учет накопленного опыта, т.е. за счет использования информации полученной на предыдущих итерациях.

Несмотря на то, что в данную статью не вошли результаты, полученные для случая полных связных графов, следует отметить, что затраты машинного времени в этом случае для алгоритма 1, алгоритма 2 и алгоритма Дейкстры отличаются незначительно независимо от числа вершин графа. Данный эффект объясняется тем, что число смежных вершин для любой вершины составляет $n-1$ для графа имеющего n вершин.

Заключение. Программная реализация предложенных алгоритмов показала их высокую эффективность. Применение алгоритма 2 позволяет существенно сократить затраты времени при решении задач, связанных с расчетами кратчайших путей применительно к задачам маршрутизации в сетях передачи данных.

Однако, проведенные вычисления носят частный характер и не претендуют на глобальность исследований, поскольку представляют собой лишь экспериментальную проверку работоспособности предложенных алгоритмов и могут служить основой для решения реальной задачи.

Список литературы: 1. *Таненбаум Э.* Компьютерные сети. – С.-Пб.: Питер, 2002. – 848 с. **2.** *Кульгин М.* Практика построения компьютерных сетей. Для профессионалов. – С.-Пб.: Питер, 2001. – 304 с. **3.** *Олифер В.Г., Олифер Н.А.* Компьютерные сети. Принципы, технологии, протоколы. – С.-Пб.: Питер, 2002. – 672 с. **4.** *Кристофидес Н.* Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с. **5.** *Бурков В.Н., Новиков Д.А.* Теория графов в управлении организационными системами. – М.: Синтег, 2001. **6.** *Агеев Д. В.* Модернизированная методика синтеза начальной структуры транспортной сети передачи данных // Труды УНДИРТ. – Одесса, 2001. – №2 (26). – С. 42 – 47.

Поступила в редакцию 28.09.2004