ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

На правах рукописи

УДК 004

БЕКЧАНОВА ИРОДА АБДУЛЛАЕВНА

Исследование и оптимизация маршрутизации в глобальной сети Интернет

Специальность 5А330202 – "Информационные и мультимедийные технологии"

Диссертация
на соискание академической степени магистра

Научный руководитель

к.т.н. Кабильджанов А.С.

Ташкент 2013

СОДЕРЖАНИЕ

Введение

Развитие информационно-коммуникационных технологий (ИКТ) является одним из основных факторов благосостояния и экономического роста страны. Сегодня ИКТ становится одним из основных приоритетов государственной политики Узбекистана [1].

В Постановлении Президента Республики Узбекистан "О мерах по дальнейшему внедрению и развитию современных информационно-коммуникационных технологий. (Собрание законодательства Республики Узбекистан, 2012 г., № 13, ст. 139), одной из основных задачи дальнейшего внедрения и развития информационно-коммуникационных технологий, в частности, является программа мер по коренному и качественному улучшению функционирования национальной информационно-поисковой системы, увеличению количества ее пользователей [2].

В настоящее время происходит бурный рост вычислительных сетей. Необходимость обеспечения качественного обслуживания современного трафика, передаваемого через IP-сети, предъявляет высокие требования к эффективности передачи пакетов данных от отправителя к получателю. Предлагаются различные методы ускорения маршрутизации, поддержки требуемого качества обслуживания, передачи голосового и видео трафика, повышения уровня безопасности сети и т.д.

Загрузка и пропускная способность линий связи вычислительной сети динамически меняется. Традиционно применяемый в вычислительных сетях метод статической маршрутизации оказывается в этих условиях неэффективным. Применение современных адаптивных протоколов маршрутизации в ряде случаев наталкивается на проблему быстродействия при решении задачи поиска кратчайших путей. Изменение характеристик линий связи приводит к полному перерасчету оптимальных маршрутов. Как показали исследования, в ряде случаев удается уменьшить трудоемкость расчета оптимальных маршрутов путем использования

дополнительной информации и изменения алгоритма поиска кратчайших путей в графе. Маршрутизация является неотъемлемой частью составных вычислительных сетей. Осуществление процессов маршрутизации является одной из наиболее важных проблем, возникающих при определении направлений и непосредственной передаче информационных потоков. Особую важность имеет эффективная маршрутизация сообщений в условиях отказов отдельных элементов сети, всплесков трафика и локальных перегрузок.

Имеет место прямая зависимость производительности сети от производительности элементов, обрабатывающих межсетевой трафик - маршрутизаторов. Многие как отечественные, так и зарубежные сети не используют алгоритмов маршрутизации, адаптивных к перегрузкам каналов. В значительной мере это объясняется тем, что большинство вычислительных сетей использует при выборе маршрута передачи *критерий минимума стоимости передачи*, а стоимость каналов связи в распределенных системах не зависит от нагрузки. В то же время, выбор маршрута, учитывающий перегрузки в сети, улучшает характеристики сети.

Основные вопросы в области маршрутизации и обеспечения функционирования сетей освещаются в работах отечественных и зарубежных ученых В.М.Вишневского, А.И.Ляхова, Р.В.Павлова, А.В.Умнова, А.В.Гуреева и др., а также зарубежных M.Gerla, C.Perkins, D.Johnson, C.-K.Toh, V.D.Park, N.H.Vaidya. В результате предлагается множество способов маршрутизации, но нахождение маршрута в них сводится лишь к отысканию кратчайшего пути с обеспечением доставки данных только "по возможности" т.е. передачи пакетов без соблюдения требований качества обслуживания. При этом передача данных приложений, чувствительных к величинам пропускной способности, задержки и потерь пакетов, при нагрузках становится практически невозможной.

Исходя из всего выше сказанного, можно сделать вывод об ***актуальности выбранной темы диссертационной работы.***

Объектами исследований являются глобальная компьютерная сеть Интернет, алгоритмы поиска кратчайших путей.

Методы исследований. В работе использованы методы теории алгоритмов, теории графов и комбинаторики и теории моделирования.

Цель диссертационной работы состоит в построении моделей сети и создание алгоритмов маршрутизации с поддержкой качества обслуживания данных в сетях, увеличения производительности маршрутизаторов путем разработки новых методов и алгоритмов поиска оптимальных маршрутов в вычислительных сетях в условиях изменяющейся загрузки и реальной пропускной способности линий связи.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Анализ целей и задач маршрутизации в вычислительных сетях. Провести исследование существующих адаптивных алгоритмов, протоколов маршрутизации, применяемых в составных вычислительных сетях для того, чтобы выявить достоинства, недостатки и область их применения.

2. Создание и формализованное описание модели сети с обеспечением качества обслуживания и модели сети с учетом изменяющихся характеристик и топологии.

3. Решение задач маршрутизации с множественными ограничениями на искомый маршрут и разработка алгоритмического обеспечения для маршрутизации с поддержкой качества обслуживания данных в сетях.

4. Разработка алгоритма поиска оптимальных маршрутов при изменении значения метрики линии связи, учитывающий факт ее использования в дереве кратчайших путей.

5. Модернизация алгоритма Дейкстры с целью увеличения его производительности.

Научная новизна. В рамках выполнения диссертационной работы были получены следующие результаты:

1. Предложена сетевая модель, позволяющая учитывать особенности динамически изменяющейся топологии и характеристик сети.

2. Разработано алгоритмическое обеспечение для маршрутизации с поддержкой качества обслуживания данных в сетях.

3. Введено понятие "локально кратчайших путей", что позволило улучшить производительность алгоритма Дейкстры для нахождения кратчайшего пути в графе.

4. Разработан алгоритм уменьшения размерности задачи поиска кратчайших путей в графе.

Практическая ценность работы. Разработанная в диссертационной работе модель и алгоритмы позволяют адекватно описывать процессы изменения топологии, происходящие в сетях. Алгоритм уменьшения размерности задачи нахождения оптимальных маршрутов позволяет сократить трудоемкость решения задачи поиска.

Структура диссертационной работы

Диссертационная работа состоит из введения, трех глав, заключения, списка использованной литературы и приложений.

В первой главе рассмотрены основные протоколы маршрутизации, параметры оптимизации маршрута, показан основной принцип оптимальности маршрута, дано понятие метрики маршрута, рассмотрен широковещательный алгоритм и показана его низкая эффективность. Рассмотрены основные алгоритмы поиска Беллмана-Форда, Дейкстры и др. Сделан вывод, что в больших сетях лучше себя зарекомендовал алгоритм Дейкстры (OSPF), основанный на использовании в каждом маршрутизаторе информации о состоянии всей сети. Здесь же рассмотрено использование протокола RIP, как эвристического алгоритма динамического программирования Беллмана-Форда. Показаны его преимущества и недостатки.

В Главе 2 рассматриваются задачи маршрутизации с множественными ограничениями. Предлагается графовая модель сети, допускающая использование схем обеспечения качества обслуживания. Описываются следующие правила для используемых при маршрутизации весовых функций. Во-первых, для любой выбранной функции, должно предполагаться существование эффективных и масштабируемых алгоритмов маршрутизации и сложность используемого алгоритма маршрутизации не должна расти пропорционально размерам сети. Во-вторых, функция должна отражать основные характеристики сети. Любые требования качества обслуживания, будут отображаться как выраженные в значениях весовых функций ограничения на маршрут. Следовательно, используемые при маршрутизации весовые функции позволяют определить те типы качества обслуживания, которые сеть может поддерживать.

В третьей главе рассмотрены вопросы анализа и модернизации алгоритма Дейкстры. Введено понятие локально кратчайших путей в графе. Рассмотрены две леммы и доказаны теоремы о локально кратчайших путях в графе. Описывается разработанный алгоритм поиска с использованием локально кратчайших путей.

В приложениях приведен материал, дополняющий и поясняющий основной текст диссертационной работы.

Глава 1. Анализ протоколов доступа и классификация алгоритмов маршрутизации в глобальной сети Интернет

1. Алгоритмы маршрутизации. Метрики маршрутов.

Цель маршрутизации - доставка пакетов по назначению с максимизацией эффективности. Чаще всего эффективность выражена взвешенной суммой времен доставки сообщений при ограничении снизу на вероятность доставки. Маршрутизация сводится к определению направлений движения пакетов в маршрутизаторах. Выбор одного из возможных в маршрутизаторе направлений зависит от текущей топологии сети (она может меняться хотя бы из-за временного выхода некоторых узлов из строя), длин очередей в узлах коммутации, интенсивности входных потоков и т.п.

Алгоритмы маршрутизации можно дифференцировать, основываясь на нескольких ключевых характеристиках. Во-первых, на работу результирующего протокола маршрутизации влияют конкретные задачи, которые решает разработчик алгоритма. Во-вторых, существуют различные типы алгоритмов маршрутизации, и каждый из них по-разному влияет на сеть и ресурсы маршрутизации. И наконец, алгоритмы маршрутизации используют разнообразные показатели, которые влияют на расчет оптимальных маршрутов.

Алгоритмы маршрутизации включают процедуры:

- измерение и оценивание параметров сети;
- принятие решения о рассылке служебной информации;
- расчет таблиц маршрутизации (ТМ);
- реализация принятых маршрутных решений.

В зависимости от того, используется ли при выборе направления информация о состоянии только данного узла или всей сети, различают алгоритмы ***изолированные*** и ***глобальные***. Если ТМ реагирует на

изменения состояния сети, то алгоритм ***адаптивный (динамический),*** иначе ***фиксированный (статический***), а при редких корректировках - ***квазистатический***. В статических маршрутизаторах изменения в ТМ вносит администратор сети [3-7].

Простейший алгоритм - изолированный, статический. ***Алгоритм кратчайшей очереди*** в отличие от простейшего является адаптивным, пакет посылается по направлению, в котором наименьшая очередь в данном узле. ***Лавинный алгоритм*** - многопутевой, основан на рассылке копий пакета по всем направлениям, пакеты сбрасываются, если в данном узле другая копия уже проходила. Очевидно, что лавинный алгоритм обеспечивает надежную доставку, но порождает значительный трафик и потому используется только для отдельных пакетов большой ценности.

Наиболее широко используемые адаптивные протоколы (методы) маршрутизации - RIP (Routing Information Protocol) и OSPF (Open Shortest Path First). Метод RIP иначе называется методом рельефов. Он основан на ***алгоритме Беллмана-Форда*** и используется преимущественно на нижних уровнях иерархии сети. В сетях, работающих в соответствии с методом OSPF, информация о любом изменении в сети рассылается лавинообразно.

Алгоритм Беллмана-Форда относится к алгоритмам DVA (Distance Vector Algorithms). В DVA *рельеф Ra(d)* - это оценка кратчайшего пути от узла a к узлу d. Оценка (условно назовем ее расстоянием) может выражаться временем доставки, надежностью доставки или числом узлов коммутации (измерение в хопах) на данном маршруте. В ТМ узла a каждому из остальных узлов отводится одна строка со следующей информацией:

- узел назначения;
- длина кратчайшего пути;
- номер N ближайшего узла, соответствующего кратчайшему пути;
- список рельефов от a к d через каждый из смежных узлов.

Хотя алгоритм Беллмана-Форда сходится медленно, для сетей сравнительно небольших масштабов он вполне приемлем. В больших сетях лучше себя зарекомендовал алгоритм OSPF [7]. Он основан на использовании в каждом маршрутизаторе информации о состоянии всей сети. В основе OSPF ***лежит алгоритм Дейкстры поиска кратчайшего пути в графах***. При этом сеть моделируется графом, в котором узлы соответствуют маршрутизаторам, а ребра - каналам связи. Веса ребер - оценки (расстояния) между инцидентными узлами.

Находит применение еще один алгоритм маршрутизации - IGRP (Interior Gateway Routine Protocol), разработанный фирмой Cisco [9]. Он аналогичен алгоритму RIP, но развивает его в направлениях: а) возможны различные метрики (целевые функции); б) трафик может распределяться по нескольким каналам с близкими значениями метрики.

В начале работы сети и в дальнейшем с определенной периодичностью маршрутизаторы обмениваются маршрутной информацией, на основе которой формируются таблицы маршрутизации. Информация передается волнообразно, и в больших сетях обновление таблиц может происходить медленно. Для устранения этого недостатка сеть разбивают на части (области OSPF) и обмен информацией происходит только внутри частей. При этом уменьшаются также размеры таблиц маршрутизации. Между собой части они связаны через пограничные маршрутизаторы, работающие по типу мостов.

Метрики маршрутов

Если адресат достижим более чем одним путем, маршрутизатор должен сделать выбор, этот выбор осуществляется на основании оценки маршрутов-кандидатов. Обычно каждому сегменту, составляющему маршрут, присваивается некоторая величина - оценка этого сегмента. Каждый протокол маршрутизации использует свою систему оценки маршрутов. Оценка сегмента маршрута называется метрикой. Здесь следует обратить внимание на то, что при выборе маршрута всем

сегментам пути должны быть даны сопоставимые значения метрики. Недопустимо, чтобы одни сегменты оценивались числом шагов, а другие - по величине задержки в миллисекундах. В пределах автономной системы это обычно не создает проблем, ведь это зона ответственности одного администратора. Но в региональных сетях, где работает много администраторов, проблема выбора метрики может стать реальной трудностью. Именно по этой причине в таких сетях часто используется вектор расстояния, исключающий субъективность оценок метрики.

Помимо классической схемы маршрутизации по адресу места назначения, часто используется вариант выбора маршрута отправителем (данный вариант получил дальнейшее развитие при введении стандарта IPv6). В этом случае IP-пакет содержит соответствующий код опции и список промежуточных адресов узлов, которые он должен посетить по пути к месту назначения [7-11].

2. Широковещательный алгоритм оптимизации маршрута. Таблицы маршрутизации. Маршруты по умолчанию

Существуют и другие схемы, например, использующие широковещательные методы адресации (flooding), где каждый приходящий пакет посылается по всем имеющимся исходящим каналам, за исключением того, по которому он получен. С тем чтобы исключить беспредельное размножение пакетов в заголовок вводится поле-счетчик числа шагов. В каждом узле содержимое поля уменьшается на единицу. Когда значение поля становится равным нулю, пакет ликвидируется. Исходное значение счетчика определяется размером субсети. Предпринимаются специальные меры против возможного зацикливания пакетов. Существует усовершенствованная версия широковещательной маршрутизации, называемая селективной широковещательной рассылкой. В этом алгоритме рассылка производится не по всем возможным

направлениям, а только по тем, которые предположительно ведут в правильную сторону. Широковещательные методы не относятся к широко применимым. Но они используются там, где нужна предельно возможная надежность, например в военных приложениях, когда весьма вероятно повреждение тех или иных каналов. Данные методы могут использоваться лишь при формировании виртуального канала, ведь они всегда обеспечивают наикратчайший путь, так как перебираются все возможности. Если путь записывается в пакете, получатель может выбрать оптимальный проход и уведомить об этом отправителя.

Большинство алгоритмов учитывают топологию связей, а не их качество (пропускную способность, загрузку и пр.). Но существуют подходы к решению проблемы статической маршрутизации, учитывающие как топологию, так и загрузку (flow-based routing). В некоторых сетях потоки между узлами относительно стабильны и предсказуемы. В этом случае появляется возможность вычислить оптимальную схему маршрутов заранее. Здесь на основе теории массового обслуживания производится оценка средней задержки доставки для каждой связи. Топология маршрутов оптимизируется по значению задержки доставки пакета. Исходными данными при расчете считается описание топологии связей, матрица трафика для всех узлов T_{ij} (в пакетах в секунду) и матрица пропускных способностей каналов $B_{i,j}$ в битах в секунду. Задержка t для каждой из связей оценивается по формуле

$$t_{ij} = 1/(p \cdot B_{ij} - T_{ij}),$$

где i и j - номера узлов;

$1/P$ - среднее значение ширины пакета в битах, произведение $p \cdot B_{ij}$ выражается в пакетах в секунду, а t измеряется в миллисекундах.

Сформировав матрицу t_{ij}, можно получить граф кратчайших связей. Так как вычисления производятся не в реальном масштабе времени, особых трудностей здесь не возникает.

Статические протоколы (примером реализации статических протоколов может служить первая версия маршрутизатора Netblazer) предполагают, что любые изменения в маршрутные таблицы вносит администратор сети. Динамическая маршрутизация, обладая очевидными преимуществами, к сожалению, облегчает задачу хакеру, пытающемуся проникнуть в сеть [10].

Рассмотрим для примера сеть, изображенную на рис. 1.1.

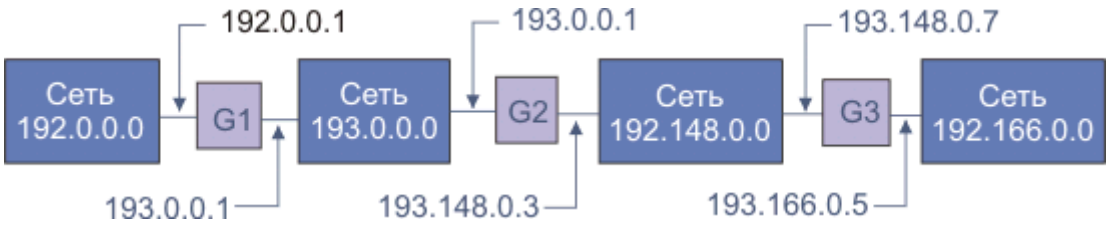


Рис. 1.1. Схема для иллюстрации методики составления маршрутных таблиц

G1, G2, G3 – маршрутизаторы.

Примитивная таблица маршрутизации для приведенного примера может иметь вид (для маршрутизатора G2): Сеть - адресат - Маршрут к этой сети

193.0.0.0	прямая доставка
193.148.0.0	прямая доставка
192.0.0.0	через адрес 193.0.0.1
192.166.0.0	через адрес 193.148.0.7

Маршруты по умолчанию

Заметно сокращают размер маршрутной таблицы маршруты по умолчанию. В этой схеме сначала ищется маршрут в таблицах, а если он не найден, пакет посылается в узел, специально выбранный для данного случая. Так, когда имеется только один канал за рубеж, неудачный поиск в таблице маршрутов в конкретной стране означает, что пакет следует послать по этому каналу и пусть там с ним разбираются. Маршруты по

умолчанию используются обычно тогда, когда маршрутизатор имеет ограниченный объем памяти или по какой-то иной причине не имеет полной таблицы маршрутизации. Маршрут по умолчанию может помочь реализовать связь даже при ошибках в маршрутной таблице. Это может не иметь никаких последствий для малых сетей, но для региональных сетей с ограниченной пропускной способностью такое решение может повлечь серьезные последствия.

Алгоритм выбора маршрута универсален и не зависит от протокола маршрутизации, который используется лишь для формирования маршрутной таблицы. Общий алгоритм выбора оптимального маршрута на основе метрик сегментов пути сформулировал американский математик Дейкстра в 1959 году [12]. Описание алгоритма выбора маршрута представлено ниже:

Извлечь IP-адрес (ID) места назначения из дейтограммы

Вычислить IP-адрес сети назначения (IN)

IF IN соответствует какому-либо адресу локальной сети, послать дейтограмму по этому адресу;

else if IN присутствует в маршрутной таблице, то послать дейтограмму к серверу, указанному в таблице;

else if описан маршрут по умолчанию, то послать дейтограмму к этому серверу;

else выдать сообщение об ошибке маршрутизации.

Если сеть включает в себя субсети, то для каждой записи в маршрутной таблице производится побитная операция <И> для ID и маски субсети. Если результат этой операции совпадет с содержимым адресного поля сети, дейтограмма посылается серверу субсети. На практике при наличии субсетей в маршрутную таблицу добавляются соответствующие записи с масками и адресами сетей.

3. Алгоритмы обработки сетей и графов как топологических моделей

Рассматриваются алгоритмы нахождения кратчайших путей, используемых для обработки графов и сетей, которые можно обозначить как модели сложных систем.

Существует большое количество практических задач, сводящихся к поиску кратчайших путей в сети (графе). К их числу можно отнести: поиск кратчайшего расстояния между городами; поиск пути передачи информации, обеспечивающего минимальную стоимость или минимальное время передачи, или максимальную надежность при распространении информации в разветвленной сети.

Алгоритм Форда-Беллмана

Исходными данными для поиска кратчайшего пути в сети (графе) является матрица весов дуг заданного ориентированного графа. Это означает, что каждой дуге (u,v)OE поставлено в соответствие некоторое вещественное число A(u,v), называемое весом данной дуги. Длину кратчайшего пути d(s,t) между вершинами s и t называют расстоянием от s до t (расстояние, определенное таким образом, может быть и отрицательным). Если не существует ни одного пути из s в t, то полагают d(s,t)=Ґ, где Ґ- некоторый символ.

Большинство алгоритмов поиска расстояний между двумя фиксированными вершинами s и t включают в себя следующие действия: по данной матрице весов дуг A[u,v] (u,vOV) вычисляют некоторые верхние ограничения D[v] на расстояние от s до всех вершин v. На каждом шаге, если D[v]+A[u,v]<D[v] оценку D[v] улучшают: D[v]=D[u]+A[u,v]. Процесс прекращается, когда дальнейшее улучшение ни одного из ограничений невозможно.

Алгоритм Форда-Беллмана позволяет найти расстояние от источника до всех вершин D[v]=d(s,v), vOV ориентированного графа при

условии, что граф не содержит контуров отрицательной длины (n - количество вершин в графе):

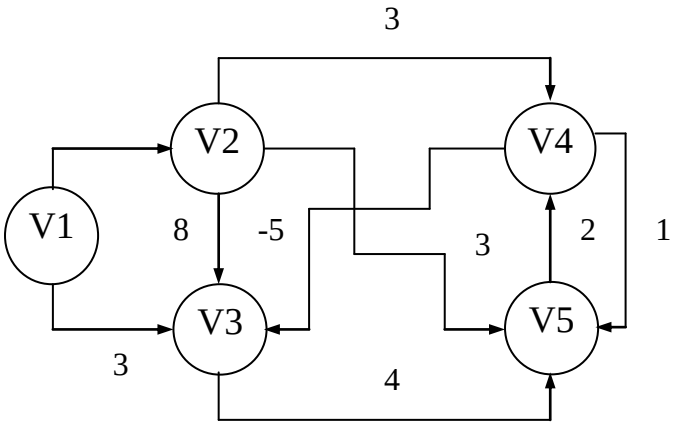
For vOV do D[v]:=A[s,v]; D[s]:=0;

For k:=1 to n-2 do

For veV\{s} do

For uOV do D[v]:=min(D[v],D[u]+A[u,v])

На рисунке 1.2. приведен: (а) граф; (б) соответствующая ему матрица весов дуг; (в) результаты работы алгоритма Форда-Беллмана.



а)

1	2	3	4	35
∞	1	∞	∞	8
∞	∞	3	3	6
∞	∞	∞	1	-5
∞	∞	2	∞	∞
∞	∞	∞	4	∞

б)

K	D[1]	D[2]	D[3]	D[4]	D[5]
	0	1	∞	∞	3
1	0	1	4	4	-1
2	0	1	4	3	-1
3	0	1	4	3	-1

б)

Рис. 1.2. Алгоритм Форда-Беллмана

Приведенный алгоритм отыскания кратчайших путей в графах с отрицательными длинами дуг, принадлежащий Форду, Муру и Беллману, может служить одним из возможных способов обнаружения контуров отрицательной длины (или циклов в неориентированном графе) [10,11].

Алгоритм Дейкстры

Более эффективным алгоритмом для решения данной задачи является алгоритм Дейкстры, применяемый в том случае, если веса всех дуг неотрицательны. Исходная информация и результаты работы аналогичны предыдущему алгоритму.

For vOV do D[v]:=A[s,v];

D[s]:=0; T:=V\{s};
While

T=0 do

begin

u:=Произвольная вершина rOT, такая, что D[r]=min

{D[p]:pOT}; T:=T\{u};

For vOT do D[v]:=min(D[v],D[v]+A[u,v])

end

Механизм работы алгоритма Дейкстры представлен на рисунке 1.3:

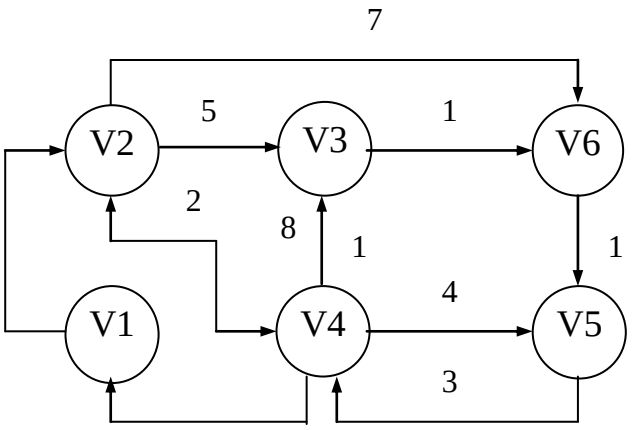


Рис. 1.3а. Граф алгоритма Дейкстры

D[1]	D[2]	D[3]	D[4]	D[5]	D[6]
0	(1)	(∞)	(∞)	(∞)	(∞)
0	1	(6)	(3)	(∞)	(∞)
0	1	(4)	3	(7)	(∞)
0	1	4	3	(7)	(5)
0	1	4	3	(6)	5

Рис. 1.3б. Матрица весов дуг Алгоритм Дейкстры

Пути в бесконтурной сети

Вторым случаем, для которого известен алгоритм нахождения расстояний от фиксированной вершины за время O(n2), является случай, когда граф является бесконтурным (теперь веса дуг могут быть произвольными). При описании алгоритма нахождения путей в бесконтурном графе мы можем предположить, что каждая дуга идет из вершины с меньшим номером в вершину с большим номером.

Исходной информацией для алгоритма поиска расстояний D[v1]=d(v1,vi),i=1,...,n от источника v1 до всех заданных вершин в бесконтурном графе является ориентированный граф (V,E), где V={v1,...,vn}, и для произвольной дуги <vi,vj>OE имеем i<j. Граф определяется линейными списками Предш[v], vOV.

D[v1]:=0;

For j:=2 to n do D[vj]:=Г;

For j:=2 to n do

For viOПредш[vj] do D[vj]:=min(D[vj],D[vi]+A[vi,vj])

Нахождение k кратчайших путей в графе

Есть неориентированный граф. Состоит из вершин и рёбер, рёбрам приписаны длины. Вершин несколько тысяч (N штук), количество рёбер известно (M штук). Дополнительных сведений о соотношении числа рёбер и числа вершин нет.

Надо найти несколько кратчайших путей без циклов между двумя заданными точками (K штук путей, K задается, K порядка нескольких десятков) структур фундаментальной физики.

Этот алгоритм предполагает, что мы умеем находить один кратчайший путь в графе.

Делается это так: Будем вести список кандидатов в кратчайшие пути. Находится первый кратчайший путь. Так как все другие пути не должны совпадать с первым путем, то эти остальные пути не содержат как минимум одно из ребер первого пути. Поэтому, выкидываем по одному ребру из первого пути и находим кратчайшие пути в получаемых графах. Найденные пути (с пометкой о том, какое ребро было выкинуто) добавляем в список кандидатов. Из списка кандидатов выбираем самый короткий путь - это второй самый короткий путь.

Далее находим следующий самый короткий путь аналогично. При нахождении каждого самого короткого пути в список кандидатов добавляется не более N новых путей (на самом деле конечно меньше).

При удалении ребра нахождение кратчайшего пути в полученном графе производится вроде за линейное время. Для этого в исходном графе алгоритм Дейкстры запускается как от начальной, так и от конечной вершины. При удалении одного ребра кратчайший путь в новом графе ищется с использованием деревьев, полученных для исходного графа.

Алгоритм Флойда-Уоршелла

Находит расстояние от каждой вершины до каждой за количество операций порядка n^3. Веса могут быть отрицательными, но у нас не может быть циклов с отрицательной суммой весов рёбер (иначе мы можем ходить по нему сколько душе угодно и каждый раз уменьшать сумму, так не интересно).

В массиве d[0… n - 1][0… n - 1] на i-ой итерации будем хранить ответ на исходную задачу с ограничением на то, что в качестве

«пересадочных» в пути мы будем использовать вершины с номером строго меньше i - 1 (вершины нумеруем с нуля). Пусть идёт i-ая итерация, и мы хотим обновить массив до I + 1-ой. Для этого для каждой пары вершин просто попытаемся взять в качестве пересадочной i - 1-ую вершину, и если это улучшает ответ, то так и оставим. Всего сделаем n + 1 итерацию, после её завершения в качестве "пересадочных" мы сможем использовать любую, и массив d будет являться ответом.

Псевдокод:

прочитать g // g[0 ... n - 1][0 ... n - 1] - массив, в котором хранятся веса рёбер, g[i][j] = 2000000000, если ребра между i и j нет

d = g

for i = 1 ... n + 1

for j = 0 ... n - 1

for k = 0 ... n - 1

if d[j][k] > d[j][i - 1] + d[i - 1][k]

d[j][k] = d[j][i - 1] + d[i - 1][k]

вывести d

4. Протоколы динамической маршрутизации

Общие сведения

Прежде чем вникать в подробности и особенности динамической маршрутизации обратим внимание на двухуровневую модель, в рамках которой рассматривается все множество машин Internet. В рамках этой модели весь Internet рассматривают как множество автономных систем (autonomous system - AS). Автономная система - это множество компьютеров, которые образуют довольно плотное сообщество, где существует множество маршрутов между двумя компьютерами, принадлежащими этому сообществу. В рамках этого сообщества можно говорить об оптимизации маршрутов с целью достижения максимальной скорости передачи информации. В противоположность этому плотному

конгломерату, автономные системы связаны между собой не так тесно как компьютеры внутри автономной системы. При этом и выбор маршрута из одной автономной системы может основываться не на скорости обмена информацией, а надежности, безотказности и т.п.

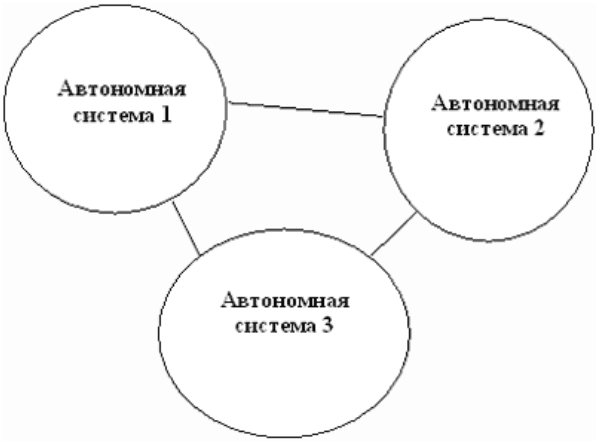


Рис.1.4. Схема взаимодействия автономных систем

Сама идеология автономных систем возникла в тот период, когда ARPANET представляла иерархическую систему. В то время было ядро системы, к которому подключались внешние автономные системы. Информация из одной автономной системы в другую могла попасть только через маршрутизаторы ядра. Такая структура до сих пор сохраняется в MILNET.

На рисунке 1.4. автономные системы связаны только одной линией связи, что больше соответствует тому, как национальный сектор подключен к Internet. В классических публикациях по Internet взаимодействие автономных частей чаще обозначают пересекающимися кругами, подчеркивая тот факт, что маршрутов из одной автономной системы в другую может быть несколько.

Обсуждение этой модели Internet необходимо только для того, чтобы объяснить наличие двух типов протоколов динамической маршрутизации:

внешних и внутренних.

Внешние протоколы служат для обмена информацией о маршрутах между автономными системами.

Внутренние протоколы служат для обмена информацией о маршрутах внутри автономной системы.

В реальной практике построения локальных сетей, корпоративных сетей и их подключения к провайдерам нужно знать, главным образом, только внутренние протоколы динамической маршрутизации. Внешние протоколы динамической маршрутизации необходимы только тогда, когда следует построить закрытую большую систему, которая с внешним миром будет соединена только небольшим числом защищенных каналов данных.

К внешним протоколам относятся Exterior Gateway Protocol (EGP) и <Protocol Gateway> .

EGP предназначен для анонсирования сетей, которые доступны для автономных систем за пределами данной автономной системы. По данному протоколу шлюз одной AS передает шлюзу другой AS информацию о сетях, из которых состоит его AS. EGP не используется для оптимизации маршрутов. Считается, что этим должны заниматься протоколы внутренней маршрутизации.

BGP - это другой протокол внешней маршрутизации, который появился позже EGP. В своих сообщениях он уже позволяет указать различные веса для маршрутов, и, таким образом, способствовать выбору наилучшего маршрута. Однако, назначение этих весов не определяется какими-то независимыми факторами типа времени доступа к ресурсу или числом шлюзов на пути к ресурсу. Предпочтения устанавливаются администратором и потому иногда такую маршрутизацию называют политической маршрутизацией, подразумевая, что она отражает техническую политику администрации данной автономной системы при доступе из других автономных систем к ее информационным ресурсам.

Протокол BGP используют практически все российские крупные IP-провайдеры, например крупные узлы сети Relcom.

К внутренним протоколам относятся протоколы Routing Information Protocol (RIP), HELLO, Intermediate System to Intermediate System (ISIS), Shortest Path First (SPF) и Open Shortest Path First (OSPF).

Протокол RIP (Routing Information Protocol) предназначен для автоматического обновления таблицы маршрутов. При этом используется информация о состоянии сети, которая рассылается маршрутизаторами (routers). В соответствии с протоколом RIP любая машина может быть маршрутизатором. При этом, все маршрутизаторы делятся на активные и пассивные. Активные маршрутизаторы сообщают о маршрутах, которые они поддерживают в сети. Пассивные маршрутизаторы читают эти широковещательные сообщения и исправляют свои таблицы маршрутов, но сами при этом информации в сеть не предоставляют. Обычно в качестве активных маршрутизаторов выступают шлюзы, а в качестве пассивных - обычные машины (hosts).

В основу алгоритма маршрутизации по протоколу RIP положена простая идея: чем больше шлюзов надо пройти пакету, тем больше времени требуется для прохождения маршрута. При обмене сообщениями маршрутизаторы сообщают в сеть IP-номер сети и число "прыжков" (hops), которое надо совершить, пользуясь данным маршрутом. Надо сразу заметить, что такой алгоритм справедлив только для сетей, которые имеют одинаковую скорость передачи по любому сегменту сети. Часто в реальной жизни оказывается, что гораздо выгоднее воспользоваться оптоволокном с 3-мя шлюзами, чем одним медленным коммутируемым телефонным каналом.

Другая идея, которая призвана решить проблемы RIP, - это учет не числа hop'ов, а учет времени отклика. На этом принципе построен, например, протокол OSPF. Кроме этого OSPF реализует еще и идею

лавинной маршрутизации. В RIP каждый маршрутизатор обменивается информацией только с соседями. В результате, информации о потере маршрута в сети, отстоящей на несколько hop'ов от локальной сети, будет получена с опозданием. Лавинная маршрутизация позволяет решить эту проблему за счет оповещения всех известных шлюзов об изменениях локального участка сети.

К сожалению, многовариантную маршрутизацию поддерживает не очень много систем. Различные клоны Unix и NT, главным образом ориентированы на протокол RIP. Достаточно посмотреть на программное обеспечение динамической маршрутизации, чтобы убедится в этом. Программа routed поддерживает только RIP, программа gated поддерживает RIP, HELLO, OSPF, EGP и BGP, в Windows NT поддерживается только RIP.

Внутренний протокол маршрутизации RIP

Этот протокол маршрутизации предназначен для сравнительно небольших и относительно однородных сетей (алгоритм Белмана-Форда). Протокол разработан в университете Калифорнии (Беркли), базируется на разработках фирмы Ксерокс и реализует те же принципы, что и программа маршрутизации routed, используемая в ОС UNIX (4BSD). Маршрут здесь характеризуется вектором расстояния до места назначения. Предполагается, что каждый маршрутизатор является отправной точкой нескольких маршрутов до сетей, с которыми он связан. Описания этих маршрутов хранится в специальной таблице, называемой маршрутной. Таблица маршрутизации RIP содержит по записи на каждую обслуживаемую машину (на каждый маршрут). Запись должна включать в себя:

IP-адрес места назначения. Метрика маршрута (от 1 до 15; число шагов до места назначения). IP-адрес ближайшего маршрутизатора (Gateway) по пути к месту назначения. Таймеры маршрута.

Первым двум полям записи мы обязаны появлению термина вектор

расстояния (место назначение – направление; метрика – модуль вектора). Периодически (раз в 30 сек) каждый маршрутизатор посылает широковещательно копию своей маршрутной таблицы всем соседям-маршрутизаторам, с которыми связан непосредственно. Маршрутизатор-получатель просматривает таблицу. Если в таблице присутствует новый путь или сообщение о более коротком маршруте, или произошли изменения длин пути, эти изменения фиксируются получателем в своей маршрутной таблице. Протокол RIP должен быть способен обрабатывать три типа ошибок:

Циклические маршруты. Так как в протоколе нет механизмов выявления замкнутых маршрутов, необходимо либо слепо верить партнерам, либо принимать меры для блокировки такой возможности.

Для подавления нестабильностей RIP должен использовать малое значение максимально возможного числа шагов (<16).

Медленное распространение маршрутной информации по сети создает проблемы при динамичном изменении маршрутной ситуации (система не поспевает за изменениями). Малое предельное значение метрики улучшает сходимость, но не устраняет проблему.

Несоответствие маршрутной таблицы реальной ситуации типично не только для RIP, но характерно для всех протоколов, базирующихся на векторе расстояния, где информационные сообщения актуализации несут в себе только пары кодов: адрес места назначение и расстояние до него.

Основное преимущество алгоритма вектора расстояний - его простота. Действительно, в процессе работы маршрутизатор общается только с соседями, периодически обмениваясь с ними копиями своих таблиц маршрутизации.

Получив информацию о возможных маршрутах от всех соседних узлов, маршрутизатор выбирает путь с наименьшей стоимостью и вносит его в свою таблицу.

Достоинство этого элегантного алгоритма - быстрая реакция на хорошие новости (появление в сети нового маршрутизатора), а недостаток – очень медленная реакция на плохие известия (исчезновение одного из соседей).

Выводы по Главе 1:

В ходе работы по Главе 1 были сделаны следующие выводы:

1. При решении задачи оптимизации маршрутизации можно получить результаты, не соответствующие желаемым, так как не всегда прямой путь от компьютера-отправителя к компьютеру-получателю является наилучшим. Он может в частности иметь малую пропускную способность или быть сильно перегружен.

2. Протоколы маршрутизации отличаются друг от друга тем, где хранится и как формируется маршрутная информация. Оптимальность маршрута достижима лишь при полной информации обо всех возможных маршрутах, но такие данные потребуют слишком большого объема памяти.

3. В итоге исследования алгоритмов Дейкстры и Беллмана-Форда можно сделать заключение, что оба алгоритма поиска кратчайшего пути приводят к одинаковому результату. По алгоритму Беллмана-Форда результат достигается за меньшее количество шагов. Однако, достоинством алгоритма Дейсктры является то, что на каждом шаге поиска находится кратчайшее расстояние еще до одной вершины, а по алгоритму Беллмана-Форда кратчайшее расстояние до любой вершины определяется только после прохождения всего алгоритма.

4. В целом, алгоритм Дейкстры, по сравнению с алгоритмом Беллмана-Форда, обеспечивает более реальную оценку ситуации в сети, более быструю реакцию на важные изменения в сети (такие, как включение новой линии связи) и уменьшает зацикливание пакетов; однако он сложнее в реализации и требует в несколько раз больше памяти.

Глава 2. Разработка графовой модели сети, использующая схемы обеспечения качества обслуживания.

1. Графовая модель обеспечения условий качества обслуживания. Состояние канала связи, описываемого тройкой переменных

Под термином "состояние канала связи" понимается совокупность используемых при маршрутизации характеристик данного канала. На рис. 2.1 состояние канала связи описывается тройкой переменных, состоящей из пропускной способности, задержки и вероятности потерь пакетов [13,14]. Предположим, что информация о состоянии каналов связи хранится в каждом отдельном узле, и описывается относительно данного узла.

Под допустимым маршрутом понимается такой маршрут, который удовлетворяет накладываемым на него требованиям качества обслуживания. Основной функцией маршрутизации с поддержкой качества обслуживания является нахождение именно такого маршрута.

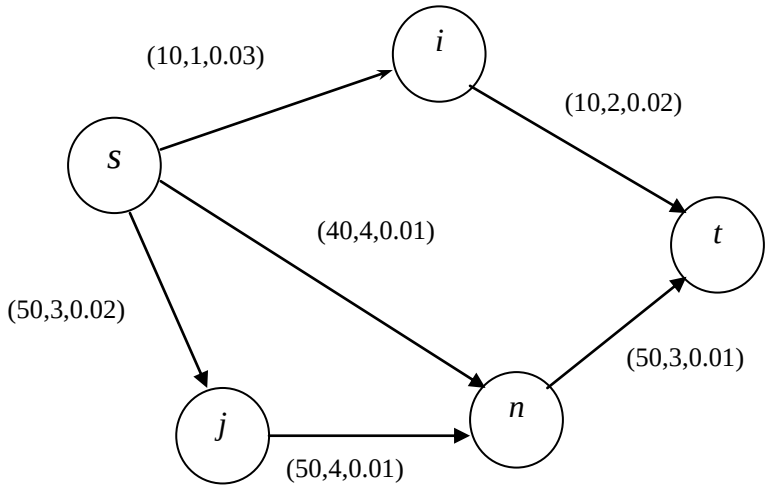


Рис. 2.1. Графовая модель сети

Для построения графовой модели (см. рис. 2.1) представим сеть передачи данных как взвешенный граф *G(V,E)*. Вершины графа *(V)*

представляют собой узлы сети. Ребра графа *(E)* представляют собой каналы связи, соединяющие между собой узлы *(V)*. Ребра графа ненаправленные только в том случае, если каналы связи всегда симметричны и имеют одинаковые характеристики (пропускную способность, задержку и т.д.) в оба направления.

Схема обеспечения условий качества обслуживания описывается следующим образом. Когда вершина *s* хочет отправить данные с некоторыми требованиями качества обслуживания для маршрута к вершине *t*, генерируется запрос на соответствующее соединение. Сначала система идентифицирует путь от *s* до *t*, который может удовлетворить требованиям качества обслуживания, и затем резервирует ресурсы по маршруту для установления соединения. Процесс идентификации маршрута относится к обязанностям маршрутизации с поддержкой качества обслуживания данных, а процесс резервирования - к механизмам сигнализации и резервирования ресурсов. После того, как подключение будет установлено, узел *s* посылает данные по новому маршруту к узлу *t*. Качество обслуживания данных гарантируется найденными и зарезервированными ресурсами по всей длине маршрута.

Используемые при маршрутизации весовые функции, согласно применяемым к ним правилам композиции классифицируются следующим образом. Пусть $w(i, j)$, будет весовой функцией для канала связи *(i, j)*. Тогда для любого пути $P = i \to j \to k \to ... \to 1 \to t$ эта функция будет:

аддитивной, если для маршрута она рассчитывается по формуле:

$$w(P) = w(i, j) + w(j, k) + ... + w(1, t) \quad (2.1)$$

мультипликативной, если

$$w(P) = w(i, j)^{*} w(j, k)^{*} ...^{*} w(1, t) \quad (2.2)$$

вогнутой, если

$$w(P) = Min[w(i, j), w(j, k), ..., w(1, t)] \quad (2.3)$$

Если приложение критично более чем к одной весовой функции, возникает проблема поиска маршрута с множественными ограничениями,

которая становится тем сложнее, чем больше число требований к искомому маршруту.

Формализовано задача маршрутизации с множественными ограничениями или многокритериальной маршрутизации (МКМ), выглядит следующим образом. Пусть *R* будет набором положительных вещественных чисел и *I* набором положительных целых чисел. Рассмотрим ориентированный граф $G(V \cdot E)$, узел-источник *s*, узел-адресат *t*, весовые функции: $w_1 : E \to R$ и $w_2 = E \to R$, две константы $B \in R\ D \in R$; задача $MKM = (G, s, t, w_1, w_2, B, D)$ заключается в нахождении маршрута *P* от *s* к *t*, удовлетворяющего условиям $w_1(P) \geq B$ и $w_2 \leq D$, при условии существования такого маршрута.

Весовой функцией $w_1(P)$ для маршрута $P = v_0 \to v_1 \to \cdots, \to v_k$, считается функция, полученная в результате следующего соотношения:

$$w_1(P) = Min[w_1(v_{i-1}, v_i) \tag{2.4}$$

а весовой функцией $w_2(P)$ для маршрута $P = v_0 \to v_1 \to \cdots, \to v_k$,

$$w_2(P) = \sum_{i=1}^{k} w_2(v_{i-1}, v_i) \tag{2.5}$$

$w_1(P)$ называется весом w_1 и $w_2(P)$ - весом w_2 маршрута P.

В качестве весовых функций, используются пропускная способность, ограниченная пределом *B* и задержка, ограниченная пределом *D*. Данные весовые функции являются наиболее важными для большинства приложений, что доказывает актуальность их использования.

Маршрут P, удовлетворяющий условиям

$w_1(P) \geq B$ и $w_2(P) \leq D$

называется *решением* (допустимым маршрутом) для

$MKM = (G, s, t, w_1, w_2, B, D)$.

2. Разработка схемы алгоритма для нахождения пути для заданных двух ограничений

На рис. 2.2 приведена схема разработанного алгоритма нахождения пути для заданных двух ограничений.

Входными данными для алгоритма являются:

- множество вершин V графа G(V,E);
- множество ребер E графа G(V,E);
- узел-источник *s*;
- узел-адресат *t*;
- две константы ограничения *B* и *D*.

Сначала алгоритм формирует массив пропускных способностей *B[k]*. Такой шаг нужен для увеличения вероятности нахождения допустимого маршрута *P*. В существующих алгоритмах, при несоблюдении условия $w_1(P) \geq B$, алгоритм прекращают свою работу, и выдает как результат отсутствие допустимого маршрута. Однако использование адаптивных схем сжатия видео- и аудиоданных позволяет подстраиваться современным приложениям под существующие ресурсы, которые могут быть немного меньше требуемых, а не полностью отказываться от работы при отсутствии требуемого значения.

Для поиска кратчайшего пути на созданном алгоритмом подграфе используется модифицированный алгоритм Дейкстры, отличающийся простотой реализации и квадратичной вычислительной сложностью. В видоизмененном алгоритме снято обязательное ограничение на связность графа - несколько переборов по небольшим массивам, общая вычислительная сложность алгоритма может быть оценена как квадратичная.

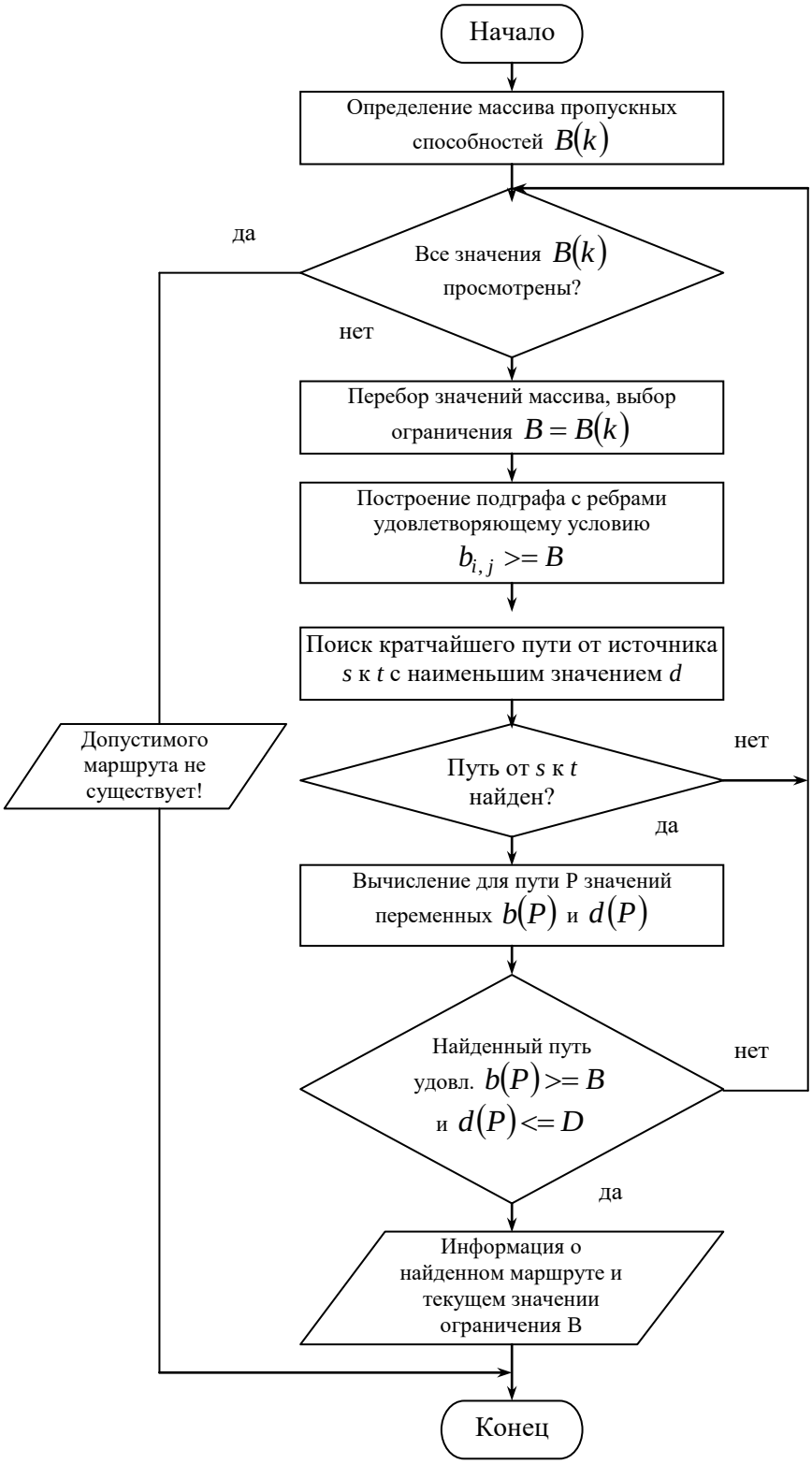


Рис 2.2. Схема алгоритма для заданных двух ограничений

Существуют приложения, для нормального функционирования которых недостаточно соблюдения ограничений на маршрут только по двум весовым функциям. Возникает проблема нахождения маршрута,

удовлетворяющего трем ограничениям. Подобная задача гораздо сложнее задачи поиска маршрута с двумя ограничениями, поскольку все маршруты, найденные алгоритмом для поиска маршрута с двумя ограничениями, могут не соответствовать третьему ограничению.

Для того чтобы найти маршрут с тремя ограничениями, используется прием, суть которого заключается в следующем: - чтобы обеспечить поиск маршрута одним из простых алгоритмов, нужно обеспечить сведение сложной задачи к более простой, которую подобный алгоритм может решить. Задача поиска по трем ограничениям сводится к задаче поиска по двум, обеспечив тем самым нахождение маршрута алгоритмом без его дополнительного усложнения. Для этого предлагается использовать новую, смешанную весовую функцию $u(i, j)$.

Для вычисления смешанной функции предлагается используемую в качестве весовой функции вероятность потерь пакетов, трактовать как некоторую задержку доставки пакета исходя из следующего предположения. При существующем транспортном протоколе, гарантирующем повторную доставку пакета в случае его потери, можно утверждать, что потерянный пакет все равно будет доставлен, но уже с некоторой задержкой $m_{i,j}$ равной,

$$m_{i,j} = g \cdot d_{i,j}$$

где $g \in R$ коэффициент, подбираемый алгоритмом, а $d_{i,j}$ - средняя задержка для ребра (i, j).

Таким образом, смешанную весовую функцию можно представить следующим образом,

$$u_{i,j} = d_{i,j} + s_{i,j} \cdot m_{i,j} = d_{i,j} + s_{i,j} + g \cdot d_{i,j} = d_{i,j}\left(1 + s_{i,j} \cdot g\right) \quad (2.6)$$

где $s_{i,j}$ величина потерь пакетов. Весовая функция *u* для маршрута *P*, вычисляется по аддитивному правилу, как и при расчете аналогичной функции для задержки,

$$U(P) = \sum_{i=1}^{k} u(v_{i-1}, v_i) \tag{2.7}$$

В результате, задача нахождения маршрута с тремя ограничениями, сводится к задаче нахождения маршрута с двумя ограничениями и решается алгоритмом, представленным на рис. 2.2. В качестве второй весовой функции, алгоритм использует смешанную функцию. В предлагаемом решении, результаты работы алгоритма не зависят от точной информации о топологии сети, что особенно актуально в мобильных сетях и решается проблема композиции весовых функций, одна из которых является аддитивной, а другая мультипликативной.

Сети также характеризуются часто изменяющимися характеристиками каналов связи. Для обеспечения работы разработанных алгоритмов с учетом изменяющихся характеристик сети, предлагается использовать переменные $\Delta b_{i,j}$ и $\Delta d_{i,j}$. Данные переменные поддерживаются в каждом узле вместе с $b_{i,j}$ и $d_{i,j}$ характеризующими, соответственно, пропускную способность и задержку. Все переменные обновляются одновременно и отражают действительное состояние канала связи. Переменные $\Delta b_{i,j}^{старое}$ и $\Delta b_{i,j}^{новое}$ используются для хранения значений $\Delta b_{i,j}$ до, и после обновления. Аналогично исполняются переменные $\Delta d_{i,j}^{старое}$ и $\Delta d_{i,j}^{новое}$. Идентичным вышеописанному способом, добавлены переменные для обозначения предыдущего и текущего состояний и $b_{i,j}$ - это $b_{i,j}^{старое}$ и $b_{i,j}^{новое}$, и $d_{i,j}$ - это $d_{i,j}^{старое}$ и $d_{i,j}^{новое}$.

Значения $b_{i,j}^{новое}$ и $d_{i,j}^{новое}$ вычисляются при использовании протокола, учитывающего состояние канала связи. Значения $b_{i,j}^{старое}$ и $d_{i,j}^{старое}$ также известны, поскольку легко вычислить разницу между предыдущим и текущим состояниями переменных $b_{i,j}$ и $d_{i,j}$. Для расчета значений переменных $\Delta b_{i,j}$ и $\Delta d_{i,j}$ используется формула, схожая с формулой Якобсона для протокола TCP

$$\Delta b_{i,j}^{новое} = k \cdot \Delta b_{i,j}^{старое} + (1-k)\left| b_{i,j}^{новое} - b_{i,j}^{старое} \right| \quad (2.8)$$

где коэффициент $k < 1$ показывает, насколько быстро уничтожается информация об истории изменения переменной $\Delta b_{i,j}^{старое}$, а значение $(1-k)$ - насколько быстро $\Delta b_{i,j}^{новое}$ стремится к значению $\left| b_{i,j}^{новое} - b_{i,j}^{старое} \right|$. Формула для вычисления $\Delta d_{i,j}$ аналогична формуле 2.7,

$$\Delta d_{i,j}^{новое} = k \cdot \Delta d_{i,j}^{старое} + (1-k)\left| d_{i,j}^{новое} - d_{i,j}^{старое} \right| \quad (2.9)$$

Предлагаемый алгоритм гораздо более прост, чем существующие решения и предполагает поддержание в каждом узле только переменных b_{ij}, d_{ij}, и $\Delta b_{i,j}$, $\Delta d_{i,j}$, без каких-либо функций распределения вероятности.

Проводится сравнение разработанного алгоритма (МКМ) с алгоритмом, ищущим маршрут последовательно по каждому ограничению. Он известен как алгоритм 3М. Например, в случае трех ограничений, алгоритм, с учетом определенного класса обслуживания находит маршрут между двумя узлами с минимальной задержкой; если найдено больше одного маршрута, алгоритм выбирает маршрут с максимальной пропускной способностью; если и в этом случае найдено более одного маршрута, алгоритм выбирает маршрут с минимальными потерями. В настоящее время, алгоритм 3М наиболее популярен и позволяет осуществлять поиск маршрута с тремя ограничениями.

Как критерий для сравнения, используется **коэффициент успешности алгоритма**, равный количеству доставленных пакетов, разделенному на общее количество отправленных пакетов. Данный критерий позволяет оценить, насколько быстро алгоритм справляется с нахождением маршрута, поскольку время поиска маршрута является одним из слабых мест реактивной маршрутизации.

При моделировании сети из 50 узлов (рис. 2.3) среднее суммарное значение коэффициента успешности алгоритма МКМ превосходит аналогичный показатель алгоритма 3М примерно на 16 процентов.

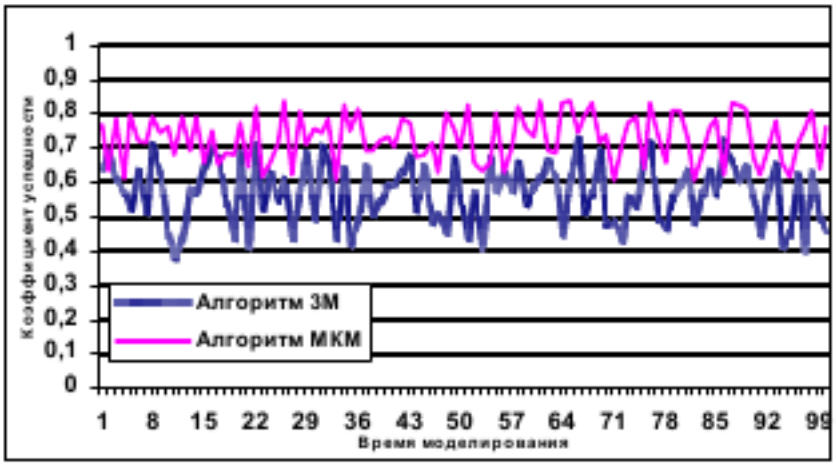


Рис.2.3. Сравнение алгоритмов на модели сети из 50 узлов

Для выяснения зависимости коэффициента успешности от количества узлов в сети был поставлен дополнительный эксперимент. В данном эксперименте сравнивались те же алгоритмы, что и в предыдущем эксперименте, но значение коэффициента успешности усреднялось после окончания всего процесса моделирования. Тестировались сети с количеством узлов от 5 до 100 с нарастающим шагом в 5 узлов. Результат эксперимента представлен на рис. 2.4.

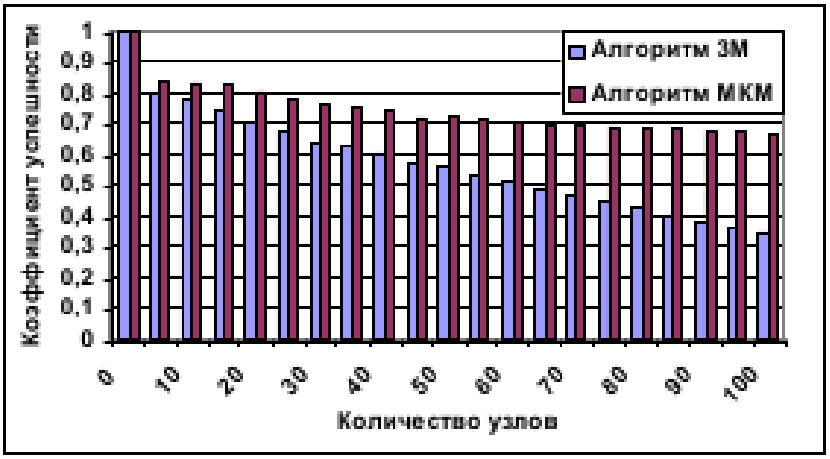


Рис.2.4. Зависимость коэффициента успешности от размеров сети

Во всех предложенных алгоритмах при отсутствии в сети маршрута, жестко удовлетворяющего ограничению, заданному на пропускную способность, используется механизм нахождения альтернативного маршрута. Для исследования эффективности работы данного решения,

сгенерированные пакеты, относящиеся к классу критичных к пропускной способности, помимо ограничения на пропускную способность содержали некоторую дельту. Значение дельты позволяет приложению определить, возможна ли работа на предложенных алгоритмом условиях или нет.

Для реализации данного эксперимента, приложению разрешалось принимать как допустимые, маршруты с пропускной способностью немного ниже (на одну единицу), чем значение, затребованное первоначально.

На рис. 2.5. показано, что использование МКМ с возможностью поиска альтернативного маршрута дает дополнительный выигрыш, приблизительно в 12 процентов по отношению к МКМ без использования такой возможности. Суммарно получилось, что при заданных условиях моделирования, алгоритм МКМ с использованием возможности альтернативного маршрута превосходит алгоритм ЗМ примерно на 30 процентов.

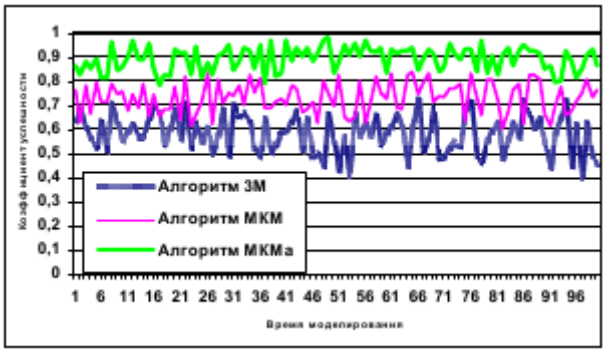


Рис.2.5. Сравнение алгоритмов ЗМ, МКМ без поиска альтернативного маршрута и МКМ с поиском альтернативного маршрута (МКМа) для модели сети из 50 узлов

Выводы по Главе 2:

В ходе работы по Главе 2 были сделаны следующие выводы:

1. Предложена графовая модель сети, позволяющая применять схемы качества обслуживания данных, включая типы ограничений и требования к ним.

2. Сформулированы задачи маршрутизации с множественными ограничениями и описаны правила композиции различных видов ограничений на маршрут.

3. Разработаны алгоритмы для решения задач многокритериальной маршрутизации с двумя и тремя заданными на маршрут ограничениями. Рассмотрена ситуация с изменяющимся состоянием сети и разработан алгоритм маршрутизации с множественными ограничениями при изменяющемся состоянии сети.

Глава 3. Модернизация алгоритма Дейкстры поиска кратчайшего пути

1. Анализ и модернизация алгоритма Дейкстры

Несмотря на последние достижения в области структур данных, которые привели к более быстрым реализациям (см., например, кучи Фибоначчи Фридмана [15]), алгоритмическое изобретение Дейкстры все еще актуально спустя 45 лет [16], обеспечивая повсеместно стандартный алгоритм эффективного поиска кратчайшего пути. Алгоритм Дейкстры вычисляет кратчайшие пути из данной исходной вершины за время $O(m + n \cdot \log n)$, время худшего случая в графе с *n* вершинами и *m* ребрами. Чтобы вычислить все пары кратчайшего пути, мы получаем $O(mn + n^2 \cdot \log n)$, в худшем случае просто повторяется алгоритм одного источника от каждой вершины.

На рисунке 3.1. и 3.2 показан простой вариант алгоритма Дейкстры, где все пары кратчайшего пути вычислены поочередно. Алгоритм использует матрицу *d*, которая соответствует верхней границе расстояния в графе. Верхняя граница *d [x, y]* для любой пары из вершины *(x, y)* первоначально равно граничному весу *w (x, y)*, если есть ребро между *x* и *y*, и +∞ в противном случае. Алгоритм также поддерживает приоритетную очередь H каждой пары *(x, y)* с приоритетом *d [x, y]*.

Основной цикл алгоритма неоднократно извлекает из *H* пару *(x, y)* с минимальным приоритетом и совершает попытку к расширению при каждой итерации соответствующего кратчайшего пути ровно на одно ребро во все возможные направления. Это требует сканирования всех ребер, покидая y и используя x, выполняя классический релаксационный шаг (смена меток), чтобы уменьшить верхние границы расстояния *d*. Не трудно доказать, что если граничные веса неотрицательны, *d* содержит точные расстояния.

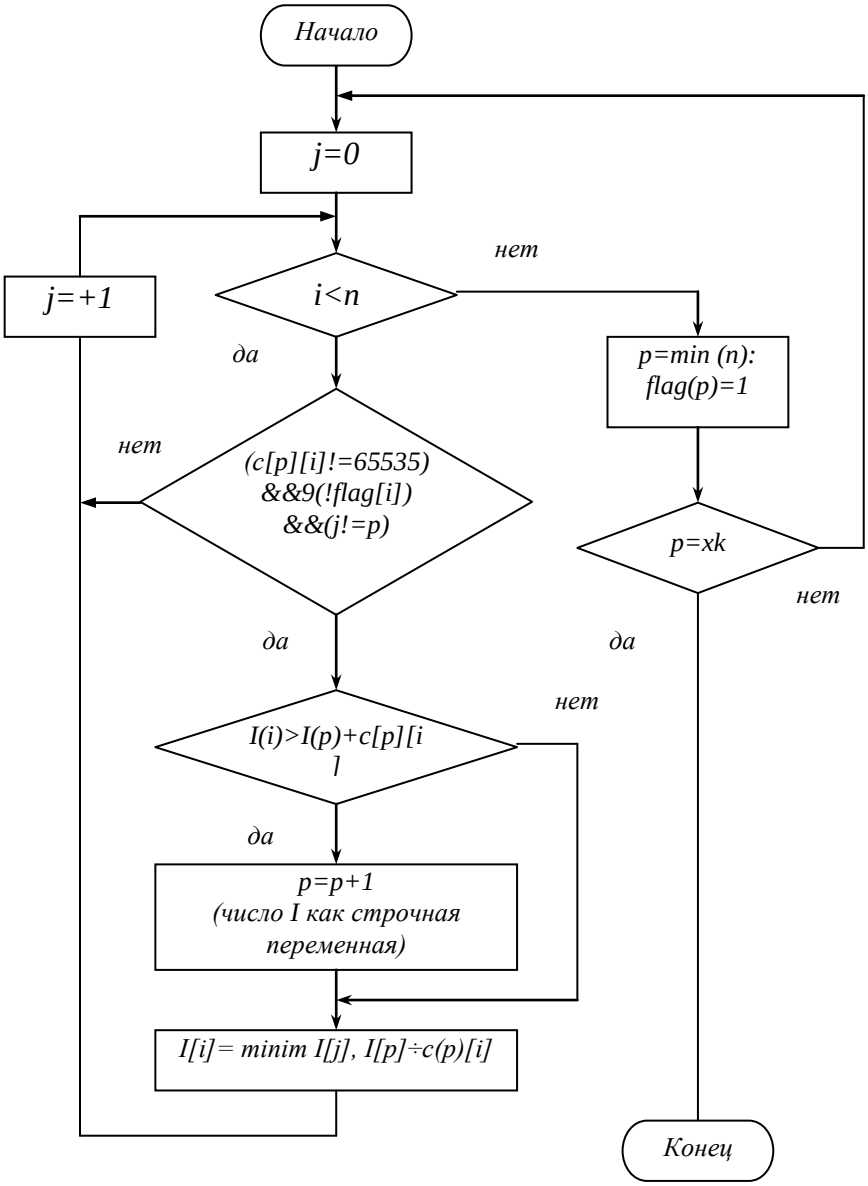


Рис. 3.1. Алгоритм Дейкстры

Время, необходимое для загрузки и разгрузки приоритетной очереди в худшем случае равно $O\left(n^2 \cdot \log n\right)$. Каждое ребро отсканировано за время O(n), и для каждой отсканированной вершины мы тратим постоянное время, если H – например, куча Фибоначчи, это равно $O\left(mn + n^2 \cdot \log n\right)$, время худшего случая.

Для того, чтобы определить постоянность или временность узлов введем понятие ранга. Каждому найденному кратчайшему пути в этом случае присваивается соответствующий ранг. В этом случае путь, имеющий самый низший ранг, и будет являться наикратчайшим, т.е.

$$d[x,y]+w[y,z](link(z_i,z_{i+1})(i=1,...,n\cdot 2\log_{n+2}))<d[x,z]$$

```
Алгоритм Дейкстры (граф  G = (V,E,w)) →матрица
1.      пусть d матрица размером  n × n и  H пустая  приоритетная очередь
2.      For each (x, y) € V × V do                          {инициализация}
3.              If (x, y) 2 E then d[x, y]   w(x, y) else d[x, y]   +1
4.              Add вершину (x, y) to H с приоритетом d[x, y]
5.      While H не пустое do                                {основной цикл}
6.              извлечь из H вершину (x, y) с min приоритетом
7.      for each  (y, z) €  E покинувших y do               {расширение}
8.              if d[x, y] + w(y, z) < d[x, z] then
9.              d[x, z]   d[x, y] + w(y, z)
10.             уменьшение приоритета of (x, z) in H to d[x, z]
11.     For each (z, x) € E ввод x do                       {обратное расширение}
12.             if w(z, x) + d[x, y] < d[z, y] then
13.             d[z, y]   w(z, x) + d[x, y]
14.             уменьшение приоритета (z, y) in H to d[z, y]
15.     return d
```

Рис. 3.2. Алгоритм Дейкстры.
w (x, y) обозначает вес вершины *(x, y)* в *G*.

Для того, чтобы определить постоянность или временность узлов введем понятие ранга. Каждому найденному кратчайшему пути в этом случае присваивается соответствующий ранг. В этом случае путь, имеющий самый низший ранг, и будет являться наикратчайшим, т.е.

$$d[x,y]+w[y,z](link(z_i,z_{i+1})(i=1,...,n\cdot 2\log_{n+2}))<d[x,z]$$

Эта запись размещается в 12 строку алгоритма Дейкстры.

Применим алгоритм Фиббоначи для определения ранга матрицы.

$F_1 = 1;$

$F_2 = 1;$

$F_3 = F_2 + F_1;$

$F_4 = F_3 + F_2 = 2 + 1;$

$\dots$

$F_n = F_{n-1} + F_{n-2}.$

и построим матрицы рангов:

$$A = \begin{Bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{Bmatrix}$$

1) $\begin{Bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{Bmatrix}$ 2) $\begin{Bmatrix} a_1 & a_3 \\ a_7 & a_9 \end{Bmatrix}$ 3) $\begin{Bmatrix} a_1 & a_3 \\ a_7 & a_9 \end{Bmatrix}$ 4) $\begin{Bmatrix} a_5 & a_6 \\ a_8 & a_9 \end{Bmatrix}$

В то время как более быстрые алгоритмы существуют для очень разбросанных графов [8 – 10], алгоритм Дейкстры все еще хороший практический выбор для решения многих задач. Поэтому поиски быстрых реализаций привели исследователей к изучению методов для ускорения алгоритма Дейкстры на основе приоритетных очередей. Например, теперь понятно, что эффективные структуры данных играют важную роль в случае разбросанных графов, в то время как граничное сканирование имеет типичные слабые места в густых графах [17].

Пранализируем алгоритм, изображенный на рисунке 3.2. Исследуем эвристические методы для того, чтобы сократить количество проверяемых вершин.

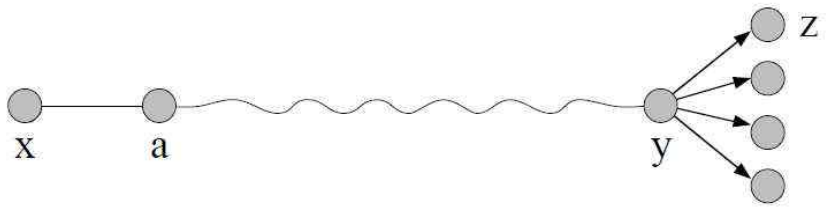


Рис. 3.3. Расширение кратчайшего пути от x до y одного ребра

Рассмотрим строку 7 из рисунка 3.2 (или, аналогично, строку 11): цикл сканирует все ребра (*y, z)*, ищет предварительные кратчайшие пути вида $x \rightarrow y \rightarrow z$ с весом $d[x, y] + w(y, z)$. Нужно ли нам сканировать все ребра *(y,z)*, оставляя *y* ? Пусть π_{xy} кратчайший путь из *x* в *y*, и пусть $\pi_{xy} = \pi_{xy} \cdot (y, z)$ получены путем перехода от *x* к *y* через *xy*, и от *y* к *z* через ребро *(y, z)* (см. рисунок 3.3).

Таким образом, ребро (*x,a*) считается первым узлом π_{xy}. Расширение кратчайшего пути проиллюстрируем на рисунках 3.4 и 3.5.

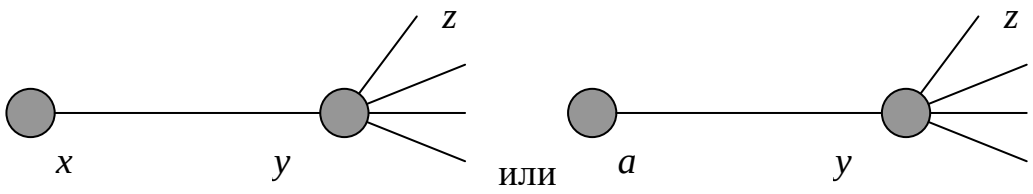


Рис. 3.4. Расширение кратчайшего пути

Так, как на основании формулы Фибоначчи $F_1 = 1$ а $F_2 = 1$, обозначим $F_1 = x$ а $F_2 = a$. Так как $F_1 = F_2$, то и $x = a$. Поясним это на рисунке:

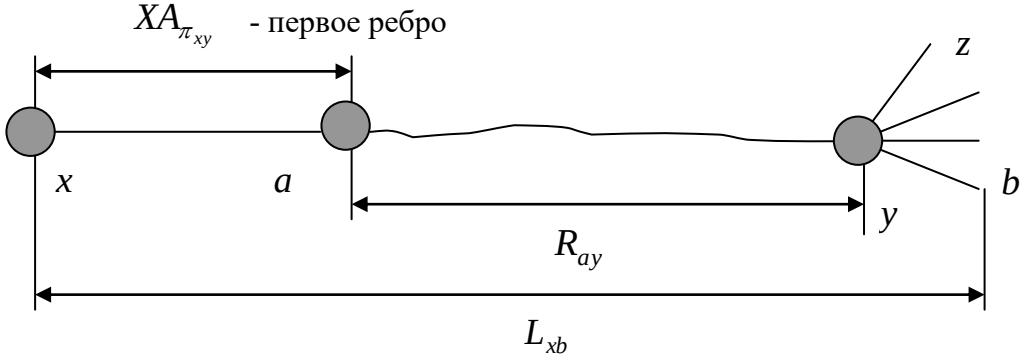


Рис. 3.5. Расширение кратчайшего пути

Таким образом $R_{ax} \neq L_{xb}$.

В данном алгоритме поиск расстояний между узлами осуществляется путем последовательного перебора всех узлов.

Рассмотрим теперь известное свойство оптимальной подструктуры кратчайших путей (см. например, [2]).

Лемма 1. Каждый подпуть кратчайшего пути – кратчайший путь.

Это свойство подразумевает, что, если π_{xy} – кратчайший путь, и мы удаляем любой из его конечных точек, мы все еще получаем кратчайший путь. Таким образом ребро *(y, z)* может быть последним ребром

кратчайшего пути π_{xy}, только если и $\pi_{xy} \subset \pi_{xy}$ и $\pi_{az} \subset \pi_{xy}$ являются самыми короткими, где (x, a) первое ребро в π_{xy}. Используем это свойство в нашем алгоритме поиска кратчайшего пути во избежание сканирования "ненужных" ребер. Предположим, что кратчайший путь в графе является уникальным. Мы можем просто поддержать для каждой пары вершин *(x, y)* список ребер

$$R_{xy} = \{(y,z) \cdot \pi_{xy} \cdot (y,z) \text{ кратчайший путь}\},$$

и список ребер

$$L_{xy} = \{(z,x)\pi_{zy} = (z,x) \cdot \pi_{xy} - \text{кратчайший путь}\}.$$

где π_{xy} кратчайший путь от *x* до *y*. Такие списки могут быть легко созданы, так алгоритм работает на каждой паре извлечения из приоритетной очереди H. Допустим, необходимо изменить строку 7 и строку 11 из рисунка 3.2, чтобы отсканировать просто ребра в R_{ay} и L_{xb}, соответственно, где *(x, a)* первая точка и *(b, y)* последняя точка в π_{xy}. Не трудно увидеть, что таким образом мы рассматриваем на релаксационном шаге только пути, которые являются подпутями кратчайших путей. Назовем такие пути **локально кратчайшими**.

Теорема 1. Путь *xy* является локально самым коротким в *G* если также:

1. *xy* состоит из единственной вершины, или

2. каждый надлежащий подпуть *xy* – кратчайший путь в G.

С модификациями выше, алгоритм требует, в худшем случае, $O(|LSP| + n^2 \cdot \log n)$, времени, где $|LSP|$ является числом локально кратчайших путей в графе. И естественным вопросом будет: сколько локально кратчайших путей может быть в графе?

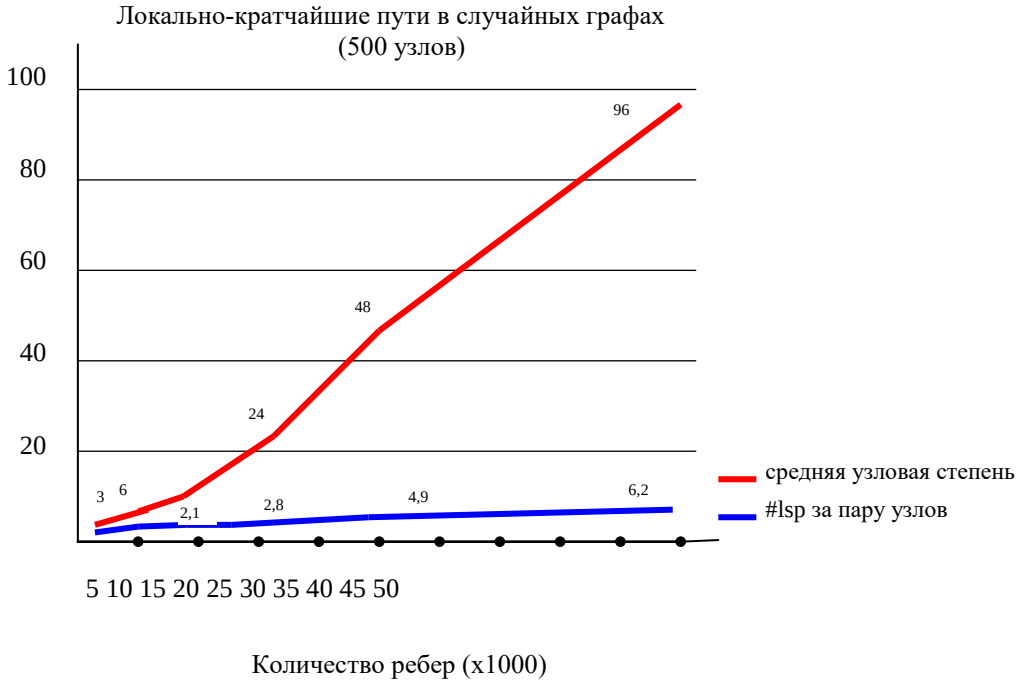


Рис. 3.6. Среднее число локально кратчайших путей, соединяющих пару вершин в сети случайных графов с увеличивающейся плотностью

Лемма 2. Если кратчайшие пути уникальны в G, то может быть не более mn локально кратчайших путей в G.

Это означает, что наша модификация алгоритма Дейкстры не является асимптотически более быстрым методом. В работе [18], произведен некоторый подсчет экспериментов и на случайных и на реальных графах (включая дорожные сети и интернет – подграфы автономных систем). В ходе изучения мы обнаружили, что в этих графах |LSP| на удивление очень близко к n^2. На рисунке 3.7 показано среднее количество локально кратчайших путей, соединяющих пары вершин в сети случайных графов с 500 вершинами и растущее число ребер. На этом рисунке мы сравниваем фактическое время работы алгоритма приведенного на рис 3.2 (S-DIJ), и тот же алгоритм, с локальной эвристической кратчайшего пути (S-LSP). Эти реализации описаны в [18]. Следует обратить внимание, что в случайном графе с плотностью 20% S-LSP может быть в 16 раз быстрее, чем S-DIJ. Это подтверждает наши ожидания, основанные на подсчете результатов, приведенных на рисунке

3.7. В очень разреженных графах, S-LSP кажется немного медленнее, чем S-DIJ из-за издержек структуры данных поддержания списков L_{xy} и R_{xy} для каждой пары вершин x и y.

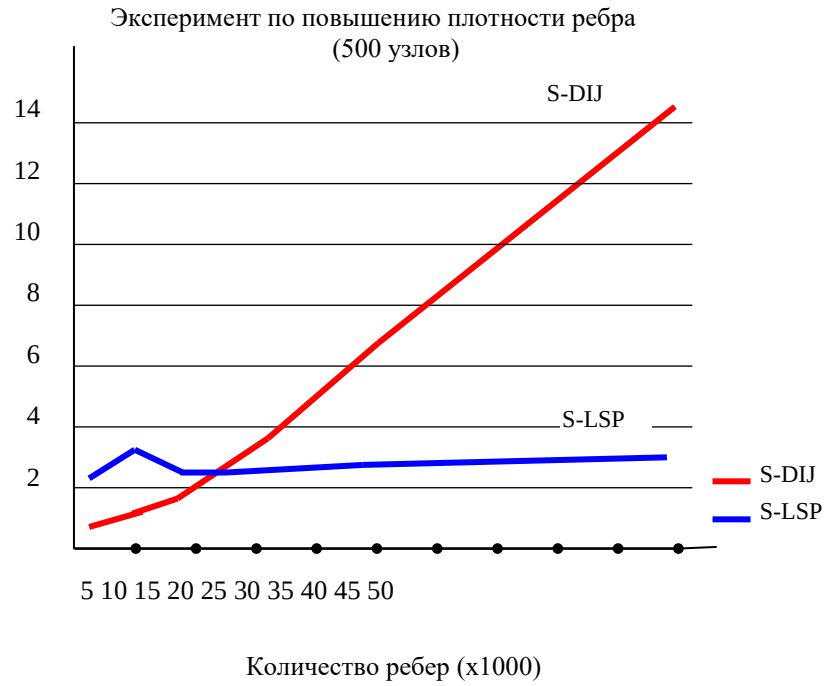


Рис. 3.7. Сравнение фактического времени выполнения реализации алгоритма Дейкстры с (S – LSP) и без (S – DIJ) локально кратчайшим эвристическим семейством случайных графов с 500 вершинами, при увеличении граничной плотности и веса ребра на [1, 1000].

2. Методика разработки нового динамического алгоритма с использованием локально кратчайших путей

Рассмотрим сценарий, в котором входной граф изменяется в течение долгого времени, подвергаясь последовательности граничного обновления веса. Цель динамического алгоритма поиска кратчайшего пути - это обновление расстояния в графе эффективнее, чем перерасчет всего решения с нуля после каждого изменения веса ребра.

Идея использования метода локально кратчайших путей, который оказывается полезным для вычисления всех пар кратчайших путей, может сыграть решающую роль в разработке асимптотически быстрых алгоритмов обновления для динамического решения этой проблемы.

Рассмотрим случай, когда вес ребер может быть увеличен (уменьшен). Обратим внимание, что после увеличения веса ребра, некоторые из кратчайших путей содержащих это ребро, могут перестать быть самыми короткими, в то время как другие пути могут стать самыми короткими, заменяя старые. Цель динамического обновления алгоритма – эффективно находить такие пути замены. Локально кратчайший путь является или самым коротким или нет. Поэтому локально кратчайшие пути являются наиболее подходящими для того, чтобы стать заменой после граничного увеличения веса. Один из возможных подходов имеет в структуре данных все локально кратчайшие пути, так, чтобы замена пути могла быть найдена максимально быстро после обновления ребра. С другой стороны, хранение таких структур данных не должно быть затруднительным. То есть, в первую очередь необходимо понять, сколько путей могут быть локально кратчайшими и сколько путей могут перестать быть локально кратчайшими после увеличения веса ребра?

Теорема 2. Пусть *G* граф с последовательным увеличением веса ребер. Если кратчайшие пути уникальны в *G*, то при каждом обновлении:

1. $O\left(n^2\right)$ пути могут перестать быть локально кратчайшими;

2. $O\left(n^2\right)$ пути могут быть локально кратчайшими, за Ω(n) операций [5]

В соответствии с теоремой 1, возможно сохранение явного множества локально кратчайших путей в графе за квадратичное амортизируемое время на операцию. Если кратчайший путь в графе локально кратчайший, то сохранение такого набора позволит нам хранить информацию и о кратчайших путях. Действительно, динамический вариант алгоритма, приведенный на рис 3.3, показывает возможность обновления локального кратчайшего пути (и, следовательно, кратчайшего пути) графа за время $O\left(n^2 \cdot \log n\right)$ с учетом увеличения веса [19]. Для графов с ребрами $m = \Omega\left(n \cdot \log_n\right)$, это асимптотически быстрее, чем

пересчет решения с нуля, используя алгоритм Дейкстры. Кроме того, если матрица расстояний должна сохраняться явно, это лишь полилогарифмический фактор далеко не наилучший.

Для нахождения всех пар кратчайших путей в динамике в общем случае, где последовательность обновления может содержать увеличение и уменьшение, локально кратчайший путь не может быть использован непосредственно. Действительно, если увеличение и уменьшение могут быть перемешаны, возможен наихудший случай с Ω(mn) изменений в наборе локально кратчайших путей во время каждого обновления. Однако, используя обобщение из локально кратчайших путей, которое содержит историю последовательности обновлений, чтобы справиться с патологическими экземплярами, разработан метод, для обновления кратчайшего пути за время обновления $O\left(n^2 \cdot \log 3 \cdot n\right)$ [5]. Эта оценка была улучшена на $O\left(n^2 \cdot \left(\log n + \log 2\left(m/n\right)\right)\right)$ по Thorup [20].

3. Сравнение времени работы алгоритма Дейкстры по "классическому" и модернизированному алгоритму

Сложность алгоритма Дейкстры зависит от способа нахождения вершины *v*, а также способа хранения множества непосещенных вершин и способа обновления меток. Обозначим через n количество вершин, а через *m* — количество ребер в графе *G*. В простейшем случае, когда для поиска вершины с минимальным *d[v]* просматривается все множество вершин, а для хранения величин *d* - массив, время работы алгоритма есть *O(n2+m)*. Основной цикл выполняется порядка *n* раз, в каждом из них на нахождение минимума тратится порядка *n* операций, плюс количество релаксаций (смен меток), которое не превосходит количества ребер в исходном графе.

Для разреженных графов (то есть таких, для которых *m* много меньше *n2*) непосещенные вершины можно хранить в двоичной куче, а в качестве ключа использовать значения *d[i]*, тогда время извлечения

вершины из *U* станет *log n*, при том, что время модификации *d[i]* возрастет до *log n*. Так как цикл выполняется порядка n раз, а количество релаксаций не больше m, скорость работы такой реализации $O\left(n^* \log n + m^* \log n\right)$.

Если для хранения непосещенных вершин использовать фибоначчиеву кучу, для которой удаление происходит в среднем за *O(log n)*, а уменьшение значения в среднем за *O(1)*, то время работы алгоритма составит $O\left(n^* \log n + m\right)$.

Граф-схема модернизированного алгоритма Дейкстры изображен на рисунке 3.8

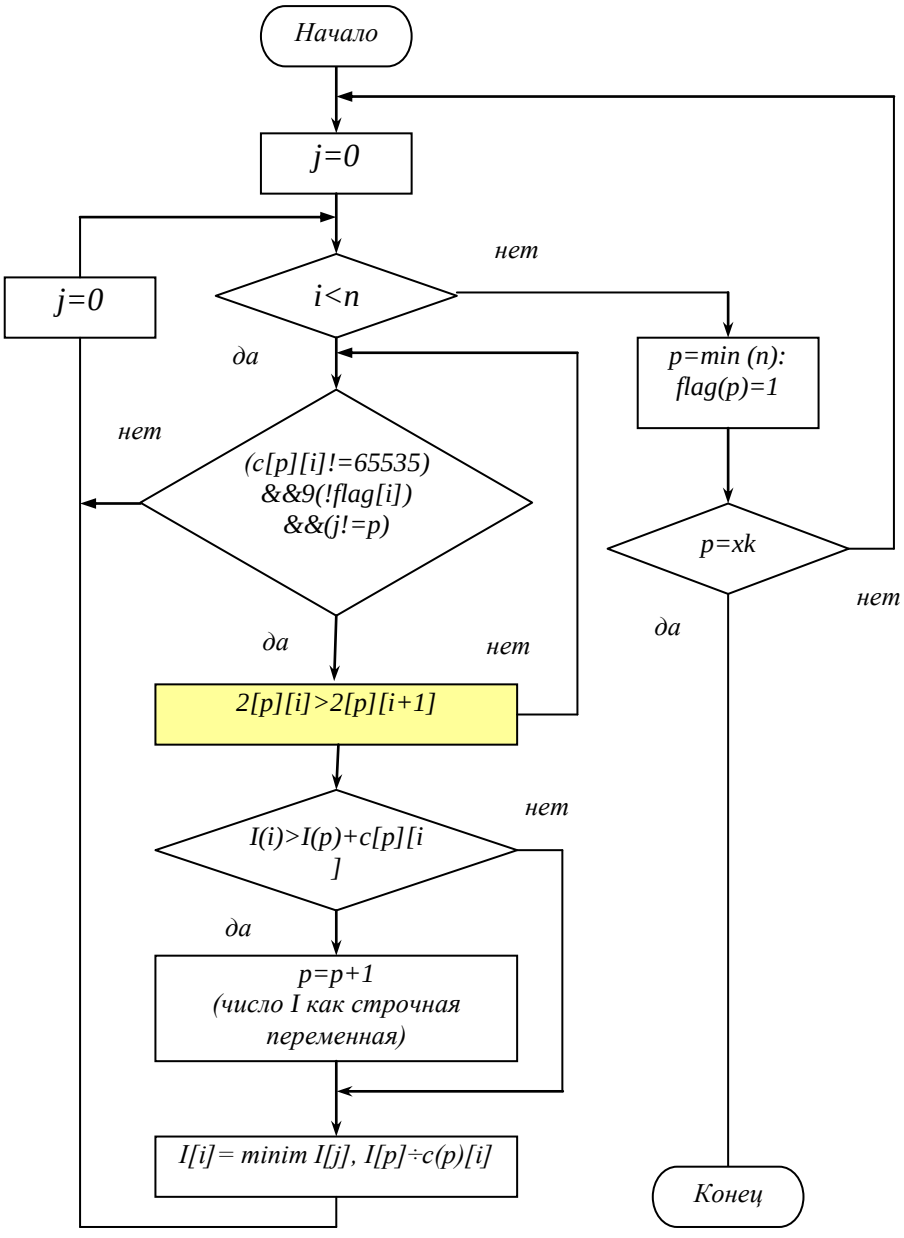


Рис. 3.8. Модернизированный алгоритм Дейкстры

Листинги программ реализаций алгоритма Дейкстры по "классическому" и модернизированному вариантам приведены в приложениях 3 и 4.

4. Результаты исследований свойств алгоритма маршрутизации Дейкстры сетевым методом

Для определения основных характеристик достаточно трудоемких алгоритмов маршрутизации сетевого трафика имеется возможность использования аналитических моделей вычислительного процесса.

В современных компьютерных сетях все большая доля сетевого трафика ложится на такие транзитные узлы, как маршрутизаторы и маршрутизирующие коммутаторы. Одним из путей повышения производительности этих устройств является оптимизация и модификация их вычислительной структуры. Для решения поставленной задачи необходимо исследовать поведение алгоритмов маршрутизации при изменении основных факторов, влияющих на трудоемкость и связность вычислительного процесса. Для определения свойств алгоритмов маршрутизации воспользуемся известными аналитическими моделями, а именно марковской и сетевой [22-25].

В качестве исследуемого алгоритма возьмем тот же алгоритм маршрутизации Дейкстры.

Идея сетевого подхода состоит в определении на графе алгоритма путей, соответствующих минимальной, средней и максимальной трудоемкости операторов. Для этого на графе выделяются циклы, которые затем заменяются операторами с эквивалентной трудоемкостью [26].

На укрупненной граф-схеме исследуемого алгоритма (ГСА) было выделено 4 цикла (рис. 3.9), причем два из них вложенные (C_2 *и* C_4).

Процесс анализа начинается с расчета данных циклов. Для примера рассматривается реализация алгоритма на сети, состоящей из 20 узлов. На

начальном этапе определяется вероятность перехода для каждой вершины ветвления алгоритма.

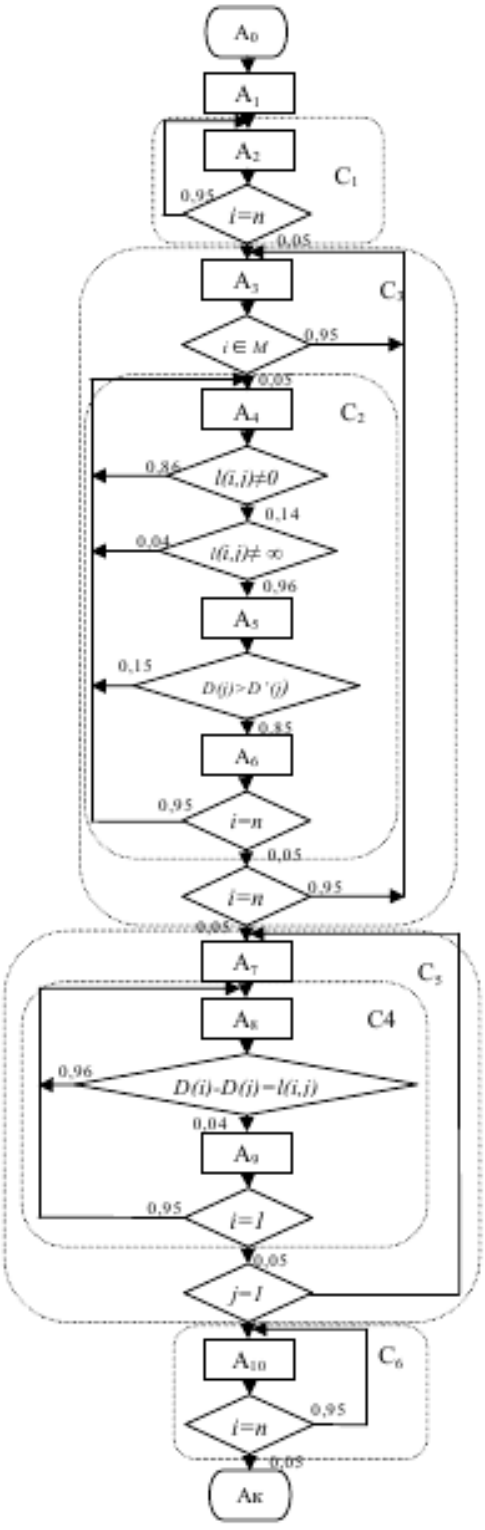


Рис. 3.9. Граф-схема (ГСА) алгоритма Дейкстры

Вероятность зависит от числа узлов сети и числа связей между

узлами.

Трудоемкость тела каждого цикла определяется по следующей формуле $Q_c^{'} = \sum n_i \cdot Q_i^V$, где Q_i^V - трудоемкость вершины, соответствующей оператору счета, а n_i - вероятность попадания в данную вершину.

Средняя трудоемкость всего цикла равна: $\Theta_{\tilde{N}} = n_C \cdot \Theta_C^{'}$ элементарные операции (элОп), где n_C - среднее число повторений цикла.

В качестве элементарной операции была взята операция арифметического сложения без переноса. Для оценки приняты следующие соотношения: одна операция регистровых пересылок (Рг→Рг) составляет одну эОп. Трудоемкость операций пересылки из регистра в ячейку памяти (Рг→ОЗУ) и из ячейки памяти в регистр (ОЗУ→Рг) составляет по три эОп, а трудоемкость операции сравнения («>», «<», «=», «+», «-») - 1 эОп.

Трудоемкость всего алгоритма определяется в соответствии с последовательностью выполнения выделенных циклов, исключая вложенные, т.к. их трудоемкость уже учтена при расчете C_2 и C_4. Таким образом, средняя трудоемкость алгоритма Дейкстры в соответствии с сетевым методом для рассмотренного примера равна:

$$\Theta_{ср} = \Theta_{A1} + \Theta_{C1} + \Theta_{C3} + \Theta_{C5} + \Theta_{C6} = 1856{,}9728(\text{элОп})$$

Аналогичным образом были определены трудоемкости алгоритма для сетей содержащих 50, 100 и 150 узлов. Они получились равными 10775,382 элОп, 50925,522 элОп и 92224,0866 элОп соответственно. На основе полученных результатов был построен график зависимости средней трудоемкости алгоритма Дейкстры от числа узлов в сети (рис. 3.10).

Полученные в ходе работы характеристики алгоритмов могут быть использованы для расчета необходимого быстродействия процессора вычислительной системы маршрутизатора.

Кроме того, результаты могут учитываться при синтезе структуры вычислительной системы с использованием методов оптимизации состава

систем обработки данных и параметров их устройств по заданным критериям. Так же данные расчеты будут использованы для сравнения известных и оптимизированных алгоритмов.

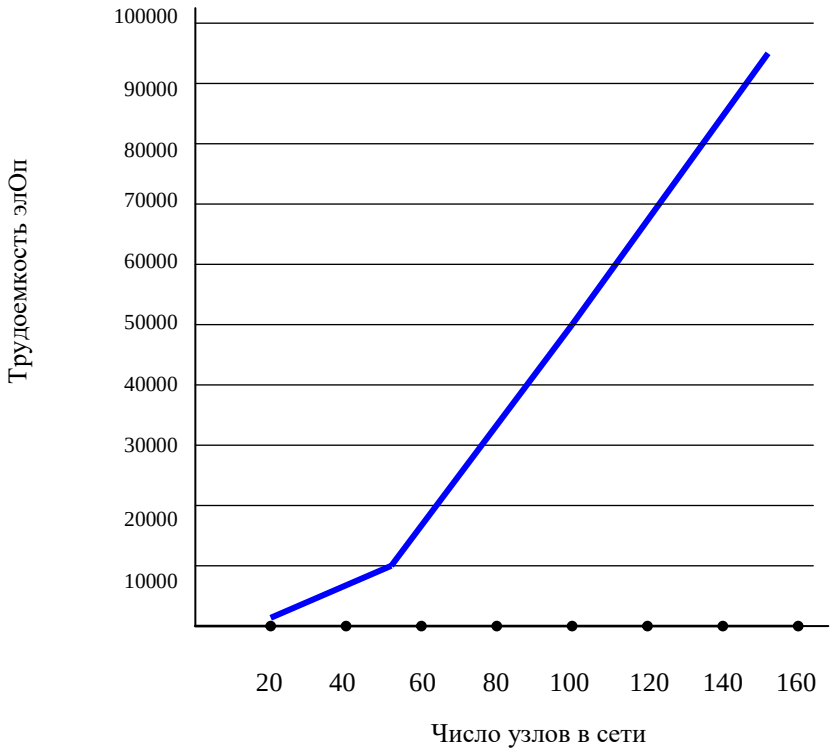


Рис. 3.10. Зависимость средней трудоемкости алгоритма Дейкстры от числа узлов в сети

Выводы по Главе 3:

В ходе работы по Главе 3 были сделаны следующие выводы:

1. На основе изучения взаимодействия теоретических и практических аспектов при использовании простого алгоритма Дейкстры для поиска кратчайшего пути, показано, что изученные эвристики, которые эффективны на практике, могут уступить результатам комбинаторных свойств проблемы, но, в конечном счете, могут привести к новым теоретически эффективным алгоритмам.

2. Описана предлагаемая методика резработки алгоритмов кратчайших путей на основе алгоритма Дейкстры. Введено понятие локально кратчайших путей, что позволило улучшить производительность алгоритма Дейкстры для нахождения кратчайшего пути в графе.

3. Сравнение фактического времени работы алгоритмов Дейкстры и того же алгоритма, с локальной эвристической кратчайшего пути показывает, что в случайном графе с плотностью 20% он может быть в 16 раз быстрее, чем классический вариант алгоритма.

4. При оптимизации и модификация вычислительной структуры алгоритма Дейкстры можно использовать сетевой подход, суть которого состоит в определении на графе алгоритма путей, соответствующих минимальной, средней и максимальной трудоемкости операторов. Вычисленные характеристики алгоритма могут быть использованы для расчета необходимого быстродействия процессора вычислительной системы маршрутизатора. Эти же результаты могут учитываться при синтезе структуры вычислительной системы с использованием методов оптимизации состава и параметров их устройств по заданным критериям. Так же полученные расчеты могут быть использованы при сравнения известных и оптимизированных алгоритмов.

Заключение:

В ходе выполнения диссертационной работы были получены следующие основные научные и практические результаты:

1. При решении задачи оптимизации маршрутизации можно получить результаты, не соответствующие желаемым - не всегда прямой путь от компьютера-отправителя к компьютеру-получателю является наилучшим. Он может иметь малую пропускную способность или быть сильно перегружен.

2. Протоколы маршрутизации отличаются друг от друга тем, как формируется и где хранится маршрутная информация. Оптимальность маршрута достижима лишь при полной информации обо всех возможных маршрутах, но такие данные потребуют слишком большого объема памяти.

3. В итоге исследования алгоритмов Дейкстры и Беллмана-Форда сделано заключение, что оба алгоритма поиска кратчайшего пути приводят к одинаковому результату. Достоинством алгоритма Дейсктры является то, что на каждом шаге поиска находится кратчайшее расстояние еще до одной вершины, а по алгоритму Беллмана-Форда кратчайшее расстояние до любой вершины определяется только после прохождения всего алгоритма. В целом, алгоритм Дейкстры, по сравнению с алгоритмом Беллмана-Форда, обеспечивает более реальную оценку ситуации в сети, более быструю реакцию на важные изменения в сети и уменьшает зацикливание пакетов; однако алгоритм Дейкстры сложнее в реализации и требует в несколько раз больше памяти.

4. В результате анализа проблем разработки алгоритмов поиска сделан вывод о необходимости разработки алгоритмического обеспечения маршрутизации с поддержкой качества обслуживания данных.

5. Предложена графовая модель сети, позволяющая применять схемы качества обслуживания данных, включая типы ограничений и требования к

ним.

6. Сформулированы задачи маршрутизации с множественными ограничениями и описаны правила композиции различных видов ограничений на маршрут.

7. Рассмотрена ситуация с изменяющимся состоянием сети и разработан алгоритм маршрутизации с множественными ограничениями.

8. Описана предлагаемая методика разработки алгоритмов кратчайших путей на основе алгоритма Дейкстры. Введено понятие локально кратчайших путей, что позволило улучшить производительность алгоритма Дейкстры для нахождения кратчайшего пути в графе.

9. Сравнение фактического времени работы алгоритмов Дейкстры и того же алгоритма, с локальной эвристической кратчайшего пути показывает, что в случайном графе с плотностью 20% он может быть в 16 раз быстрее, чем классический вариант алгоритма.

Список использованной литературы:

1. Закон Республики Узбекистан "О защите информации в автоматизированной банковской системе" // Собрание законодательства Республики Узбекистан.-Т., 2006, № 14.

2. Постановление Президента Республики Узбекистан "О мерах по дальнейшему внедрению и развитию современных информационно-коммуникационных технологий. (Собрание законодательства Республики Узбекистан, 2012 г., № 13, ст. 139).

3. А.П. Лунев «Телекоммуникационные технологии» Владивосток: ВГУЭиС, 1999, 407с.

4. Сети ЭВМ: протоколы стандарты, интерфейсы. Ю. Блэк; перев. с англ. - М.: Мир, 1990.

5. Коммутация и маршрутизация IP/IPX трафика. М. В. Кульгин, АйТи. - М.: Компьютер-пресс, 1998.

6. Протоколы Internet. С. Золотов. - СПб.: BHV - Санкт-Петербург, 1998.

7. Вычислительные системы, сети и телекоммуникации. Пятибратов и др. - ФИС, 2006.

8. Высокопроизводительные сети. Энциклопедия пользователя. Марк А. Спартак и др.; перев. с англ. - Киев, ДиаСофт, 1998.

9. Сэм Хелеби, Денни Мак-Ферсон. Принципы маршрутизации в Internet. Вильямс [Cisco Press], 2001, 404 с.

10. Джо Брокмайер. Маршрутизация в Linux- Вильямс, 2002, 240 с.

11. Джо Хабракен. Как работать с маршрутизаторами-ДМК: Пресс 2005, 317с.

12. E.W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

13. Поженко М.А. Разработка мобильных беспроводных сетей с использованием специфических алгоритмов маршрутизации. // Материалы Международной конференции «Проблемы человеко-компьютерного взаимодействия», Ульяновск— 2003 — С.99-101.

14. Диденко С.В и др. Моделирование эпизодических беспроводных сетей с динамической топологией. //Материалы международной конференции «Контроль и коммуникации (СИБКОН-2003)», Томск - 2003 - С.60-63.

15. Столлингс В. Современные компьютерные сети. – 2003. (Глава 14. Теория графов и поиск путей с минимальной стоимостью).

16. E.W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

17. B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. Mathematical Programming, 73:129–174, 1996.

18. C. Demetrescu, S. Emiliozzi, and G.F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. In Proceedings of the 15th Annual ACM – SIAM Symposium on Discrete Algorithms (SODA’04), 2004.

19. C. Demetrescu and G.F. Italiano. A new approach to dynamic all pairs shortest paths. In Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC’03), San Diego, CA, pages 159–166, 2003.

20. M. Thorup. Tighter fully – dynamic all pairs shortest paths, 2003. Unpublished manuscript.

21. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms. McGraw – Hill, 2001.

22. Компьютерные сети. Учебный курс, 2-е изд. (+CD-ROM). - MicrosoftPress, Русская редакция, 1998.

23. Сетевые средства Microsoft Windows NT Server 4.0; перев. с англ. СПб.: - BHV - Санкт-Петербург, 1997.

24. Ресурсы Microsoft Windows NT Server 4.0. Книга 1; перев. с англ.

СПб.: - BHV -Санкт-Петербург, 1997.

25. Основы теории вычислительных систем: Учебное пособие.- (Под ред. С.А. Майорова.) – М.: Высш. шк., 1978. – 408 с.

26. Власов А.А., Васяева Е.С., Васяева Н.С. Исследование моделей систем обработки данных. – Йошкар-Ола: МарГТУ, 2003.- 124 с.

27. Бекчанова И.А. Реализация маршрутизатора на основе протокола RIP/Молодой ученый. – 2013. - №6.

28. Бекчанова И.А. Создание объединенной сети с протоколом маршрутизации RIP для IP./Сборник докладов, часть 3. – 2013.

29. http://www.3com.ru/ (Архитектура сетей HP)

30. http://www.rednet.ru/ (Интернет провайдер, сети)

31. http://www.catv.org/ (Гид по Интернет сервисам)

32.http://www.citforum.ru/(Библиотека онлайн, аналитические данные)

Приложения

Приложение 1

Протоколы доступа

Опорные сети и автономные системы

Одна из базовых идей маршрутизации заключается в том, чтобы сконцентрировать маршрутную информацию в ограниченном числе (в идеале в одном) узловых маршрутизаторов-диспетчеров. Эта замечательная идея ведет к заметному увеличению числа шагов при пересылке пакетов. Оптимизировать решение позволяет backbone (опорная сеть), к которой подключаются узловые маршрутизаторы. Любая AS подключается к backbone через узловой маршрутизатор.

"Прозрачные" backbone не работают с адресами класса C (все объекты такой сети должны иметь один адрес, а для C-класса число объектов слишком ограничено). "Прозрачные" мосты трудно диагностировать, так как они не следуют протоколу ICMP (команда ping не работает, в последнее время такие объекты снабжаются snmp-поддержкой). За то они позволяют перераспределять нагрузку через несколько маршрутизаторов, что невозможно для большинства протоколов.

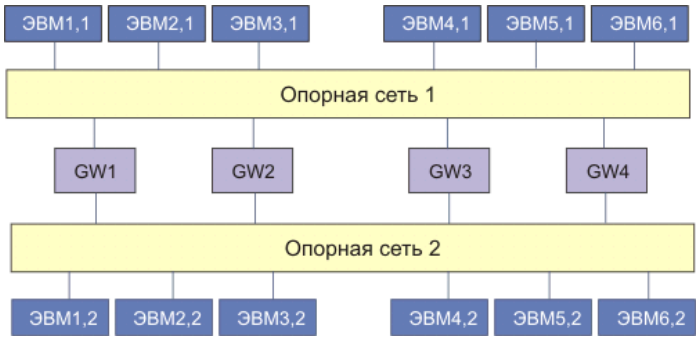


Рис. П1.1. Пример задачи маршрутизации

В примере, приведенном на рис. П1.1, задача маршрутизации достаточно сложна. ЭВМ 1,2 и ЭВМ 6,1 можно связать многими путями: ЭВМ 1,2 - GW1 - ЭВМ 6,1; ЭВМ 1,2 - GW2 - ЭВМ 6,1; ЭВМ 1,2 - GW3 - ЭВМ 6,1; ЭВМ 1,2 - GW4 - ЭВМ 6,1; ЭВМ 1,2 - GW1 - GW2 - GW3 - ЭВМ 6,1; и т.д. Трафик между двумя географически близкими узлами должен направляться кратчайшим путем, вне зависимости от направления глобальных потоков. Так ЭВМ 1,2 и ЭВМ 1,1 должны соединяться через GW1. Маршрутизация через опорные сети (backbone) требует индивидуального подхода для каждого узла. Администраторы опорных сетей должны согласовывать свои принципы маршрутизации. Ситуация, когда узел не владеет исчерпывающей маршрутной информацией, в сочетании с использованием маршрутов по умолчанию может привести к

зацикливанию пакетов. Например, если маршрут по умолчанию в GW1 указывает на GW2, а в GW2 на GW1, то пакет с несуществующим адресом будет циркулировать между GW1 и GW2 пока не истечет ttl (время жизни пакета), полностью блокируя канал. По этой причине желательно иметь полную таблицу маршрутизации, и, если не вынуждают обстоятельства, избегать использования маршрутов по умолчанию.

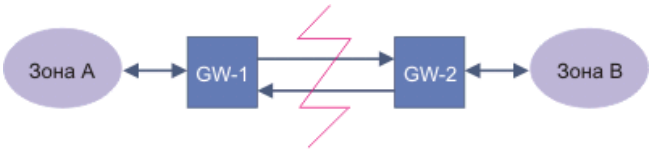


Рис. П1.2.

Протоколы маршрутизации отличаются друг от друга тем, где хранится и как формируется маршрутная информация. Оптимальность маршрута достижима лишь при полной информации обо всех возможных маршрутах, но такие данные потребуют слишком большого объема памяти. Полная маршрутная информация доступна для внутренних протоколов при ограниченном объеме сети. Чаще приходится иметь дело с распределенной схемой представления маршрутной информации. Маршрутизатор может быть информирован лишь о состоянии близлежащих каналов и маршрутизаторов.

Динамические протоколы (обычно используются именно они, наиболее известным разработчиком является компания CISCO):

В маршрутизаторе с динамическим протоколом (например, BGP-4) резидентно загруженная программа-драйвер изменяет таблицы маршрутизации на основе информации, полученной от соседних маршрутизаторов. В ЭВМ, работающей под UNIX и выполняющей функции маршрутизатора, эту задачу часто решает резидентная программа gated или routed (демон). Последняя - поддерживает только внутренние протоколы маршрутизации.

Применение динамической маршрутизации не изменяет алгоритм маршрутизации, осуществляемой на IP-уровне. Программа-драйвер при поиске маршрутизатора-адресата по-прежнему просматривает таблицы. Любой маршрутизатор может использовать два протокола маршрутизации одновременно, один для внешних связей, другой - для внутренних. Для стыковки внешнего маршрута с внутренним в большинстве протоколов предусматриваются специальные средства.

Внешние и внутренние протоколы маршрутизации

Может возникнуть вопрос, откуда возникло ограничение на число внешних и внутренних протоколов маршрутизации? Главная причина - согласование метрик сетевых каналов. С внешними протоколами все относительно просто. Во-первых, их мало, во-вторых, практически все они

используют для оценки каналов вектор расстояния, что потенциально может вообще снять рассматриваемое ограничение. А как быть с внутренней маршрутизацией? Ведь существуют протоколы, базирующиеся на векторе расстояния (RIP), и на состоянии канала (OSPF или IGRP). Предположим, что в одной зоне сети работает RIP, где канал оценивается числом шагов до цели, а в другой - OSPF с оценкой состояния канала, выполненной администратором сети. Если маршрут содержит фрагменты пути, пролегающие через обе указанные зоны, возникает проблема оценки такого пути. Как сложить метрики этих участков, ведь они несовместимы? В принципе задача имеет решение, для этого на границах зон с разными протоколами маршрутизации размещаются специальные маршрутизаторы, которые оптимизируют пути для каждого из протоколов (и зон) независимо. Но и в этом случае возможны трудно разрешимые ситуации. Один из таких вариантов показан на рис. П1.3.

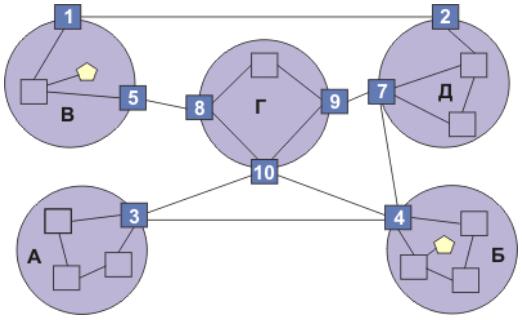


Рис. П1.3.

Пусть на рисунке П1.4 в сетевых зонах, обозначенных кругами используется протокол RIP, а в остальном пространстве - OSPF. ЭВМ обозначены пятиугольниками, внутренние маршрутизаторы прямоугольниками, а окрашенные прямоугольники отмечают пограничные маршрутизаторы зон. Любые маршруты из зоны А в зону Б проблем не вызовут, так как внутри зоны маршруты оптимизируются RIP, а между зонами - протоколом OSPF. Рассмотрим прокладку маршрута из зоны Б в зону В. Здесь следует рассмотреть варианты пути через зоны Г и Д. Важно то, что при выборе оптимального пути придется как-то учитывать метрики как OSPF-части пути, так и фрагменты транзитного пути внутри зон. При этом надо принять решение, с какими весами складывать метрики разных протоколов. Эта проблема снимается, если каждая зона имеет только один пограничный маршрутизатор. Маршрут прокладывается между пограничными маршрутизаторами, а различие маршрутов внутри зон игнорируется. Кстати именно эта логика лежит в основе рекомендации для автономной системы иметь только один пограничный маршрутизатор.

Это утверждение можно пояснить на примере, показанном на рис. П1.4.

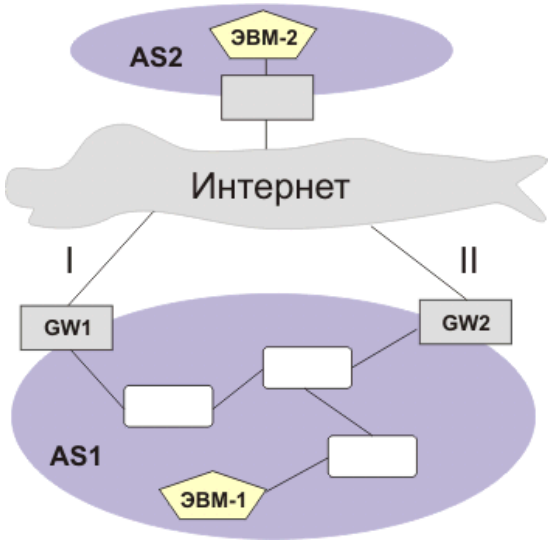


Рис. П1.4.

Рассмотрим процедуру выбора пути от ЭВМ-1 к ЭВМ-2 из автономной системы AS1 в автономную систему AS2. Имеются метрики, характеризующие путь до маршрутизаторов GW1 и GW2 (M1 и M2). Существует метрика пути от GW1 и GW2 до автономной системы AS2 M3 и M4 (они совсем необязательно равны между собой). В результате мы имеем характеристики двух путей между означенными машинами M1, M3 и M2, M4). Если внутренним протоколом маршрутизации является OSPF или IGMP, то складывать M1 и M4 (соответственно M2 и M4) нельзя, так как в одном случае (M1, M2) это характеристики состояния каналов (администратор назначил их, например, равными 35 и 55), а во втором (M3,M4) - это число шагов до автономной системы AS2 (пусть они равны, например, 6 и 4). Задача сопоставления метрик в этом простом случае может оказаться не по плечу даже суперЭВМ. Примером такой задачи может служить подключение к Интернет узлов ЮМОС (Южная Московская Опорная Сеть), имеющих выход и в АТМ-сеть ПРАН-МГУ.

Если бы у AS1 был только один пограничный шлюз (например, GW1), то задача решалась бы автоматически. Другим подходом может быть деление всего Интернет на две части, одна делается доступной только через GW1, а вторая - через GW2.

Любая автономная система (AS, система маршрутизаторов, ЭВМ или сетей, имеющая единую политику маршрутизации) может выбрать свой собственный протокол маршрутизации.

Деликатной процедурой алгоритмов маршрутизации является рассылка маршрутной информации. Если предположить, что один маршрутизатор получил пакет с новыми данными, а другой нет (потерялся или еще не дошел), то эти два прибора будут использовать разное представление о топологии сети, что может привести к циклам пакетов, осцилляции маршрутов и другим неприятностям. Первое, что приходит в голову для синхронизации маршрутной картины сети - это

широковещательная рассылка маршрутных данных. При этом каждому пакету можно присваивать порядковый номер. Анализ этого номера позволяет маршрутизатору избавляться от пакетов дубликатов и от кадров с устаревшей информацией. Получив новый пакет, маршрутизатор переадресует его на все свои интерфейсы кроме того, через который он пришел. Такой алгоритм может встать в тупик в случае переполнения номера пакета. Допустим, после номера N придет пакет с номером 1. В этом случае он будет отброшен как дубликат ранее пришедшего или как кадр, несущий устаревшие данные. Если использовать 32-разрядные номера пакетов, при частоте рассылки один пакет в секунду переполнение счетчика произойдет более чем через 136 лет. Такая угроза вряд ли кого-то напугает.

Другой проблемой является ситуация при которой маршрутизатор оказался временно отключен от сети переменного тока и был перезагружен. В этом случае он не будет знать, какой номер кадра следует ожидать. Кроме того, вполне реалистично предположить, что за 136 лет счетчик пакетов в маршрутизаторе может сбиться. Последствия могут оказаться плачевными.

Для решения всех этих проблем можно ввести помимо номера еще и возраст пакета и уменьшать его на единицу раз в секунду. Когда возраст станет равным нулю, маршрутная информация из маршрутизатора удаляется и процедура начинается с нуля.

Другим решением может быть схема, при которой входной пакет, предназначенный для рассылки, посылается соседям не сразу, а выдерживается некоторое время в буфере. Если за время выдержки придет очередной пакет, то их порядковые номера сравниваются. Если они равны, то дубликат отбрасывается. При разных номерах отбрасывается более старый. Для обеспечения надежности получение кадра с маршрутной информацией обязательно подтверждается.

Следует учитывать, что каждому сегменту пути может быть присвоено два значения метрики, по одному для каждого из направлений обмена.

Внутренний протокол маршрутизации IGP (Interior Gateway Protocol) определяет маршруты внутри автономной системы. Наиболее популярный IGP - RIP (Routing Information Protocol, RFC-1058), разработан Фордом, Фулкерсоном и Белманом (фирма XEROX) разработан в 1957-62 годах и использует в качестве метрики вектор расстояния. Протокол был базовым в рамках проекта ARPANET. В качестве метрики может выбираться время доступа, число пакетов в очереди, но обычно - это число шагов до места назначения. Если до цели имеется один промежуточный маршрутизатор, то число шагов считается равным 2. Расстояние до любого из соседей равно одному шагу. В этом протоколе всегда выбирается путь с наименьшим числом шагов до цели (наименьшее значение метрики). Маршрутизация на основе вектора расстояния в принципе позволяет получить положительный результат в любой ситуации, но она имеет одну

особенность - процесс сходится быстро при обнаружении нового более короткого пути и работает крайне медленно при исчезновении пути.

Алгоритм вектора расстояния неявно предполагает, что все каналы имеют равную пропускную способность.

Существует более новый протокол OSPF (Open Shortest Pass First, RFC-1131, -1245, -1247, -1253, -1584, -1850, -2328, -2740). базирующийся на оценках состояний каналов. Как во всех маршрутных протоколах, использующих состояние канала, многое зависит от того, как вычисляется метрика. Если определяющим фактором выбрать полосу пропускания канала, то при определенных обстоятельствах могут возникнуть трудно преодолимые проблемы. Рассмотрим топологию сети, показанную на рис. П1.5.

Рис. П1.5. Пример топологии сети, допускающей осцилляцию маршрутов

Здесь две субсети А и Б соединены двумя каналами 1 и 2. Кружочками обозначены маршрутизаторы. Если в исходный момент времени основной поток между сетями протекает по каналу 2, он может оказаться перегружен, в то время как канал 1 получит меньшее значение метрики из-за отсутствия загрузки. При очередной оценке каналов будет принято решение переключить поток между сетями на канал 1. После этого будет перегружен канал 1 и так может повторяться до бесконечности. Такая ситуация называется осцилляцией маршрутов и ее желательно избегать. Маршрутные таблицы в маршрутизаторах актуализуются вдоль пути с заметными задержками и отнюдь не синхронно. Такие осцилляции могут в разы понизить пропускную способность сети, что необходимо учитывать, выбирая параметры протоколов маршрутизации.

На практике этого не происходит, так как метрики сегментам пути в OSPF присваивает сетевой администратор. Но можно ожидать, что появятся как внутренние, так и внешние протоколы маршрутизации на основе алгоритма состояния канала, которые будут исследовать параметры канала в реальном масштабе времени.

Наиболее старые системы (IGP) используют протокол HELLO. Протокол HELLO поддерживался фирмой DEC, в качестве метрики он использует время, а не число шагов до цели.

Протокол IGRP (Interior Gateway Routing Protocol) разработан компанией CISCO для больших сетей со сложной топологией и

сегментами, которые обладают различной полосой пропускания и задержкой. Это внутренний протокол маршрутизации имеет некоторые черты сходства с OSPF.

IGRP использует несколько типов метрики, по одной на каждый вид QOS. Метрика характеризуется 32-разрядным числом. В однородных средах этот вид метрики вырождается в число шагов до цели. Маршрут с минимальным значением метрики является предпочтительным. Актуализация маршрутной информации для этого протокола производится каждые 90 секунд. Если какой-либо маршрут не подтверждает своей работоспособности в течение 270 сек, он считается недоступным. После семи циклов (630 сек) актуализации такой маршрут удаляется из маршрутных таблиц. IGRP аналогично OSPF производит расчет метрики для каждого вида сервиса (TOS) отдельно.

Для взаимодействия маршрутизаторов используются внешние протоколы (EGP - Exterior Gateway Protocols).

Одной из разновидностей EGP является протокол BGP (Border Gateway Protocol, RFC-1268 [BGP-3], RFC-1467 [BGP-4]).

Протокол IDPR (InterDomain Policy Routing Protocol, RFC-1477, -1479) представляет собой разновидность BGP-протокола. Протокол IS-IS (Intermediate System to Intermediate System Protocol, RFC-1195, -1142) является еще одним внутренним протоколом, который используется для маршрутизации CLNP (Connectionless Network Protocol, RFC-1575, -1561, -1526). IS-IS имеет много общего с OSPF. Смотри также бесклассовый протокол маршрутизации CIDR (RFC-1467, -1517-20).

Сразу после включения маршрутизатор не имеет информации о возможностях соседних маршрутизаторов. Статичесикие маршрутные таблицы могут храниться в постоянной памяти или загружаться из какого-то сетевого сервера. По этой причине первейшей задачей маршрутизатора является получение маршрутной информации от соседей, а для начала выявление наличия соседей и их адресов. Для этой цели посылается специальный пакет Hello через каждый из своих внешних интерфейсов. В ответ предполагается получить отклик, содержащий идентификационную информацию соответствующего маршрутизатора. Когда два или более маршрутизаторов объединены через локальную сеть, ситуация несколько усложняется (рис. П1.6). Маршрутизаторы E, F, G и H подключены непосредственно к локальной сети, некоторые из них имеют связи с другими маршрутизаторами (сети, которые они обслуживают, на рисунке не показаны).

Одним из способов смоделировать локальную сеть - рассматривать маршрутизаторы E, F, G и H, как соединеные непосредственно, введя виртуальный узел сети L (выделен зеленым цветом). На правой части рисунка показан граф такой сети. Возможность прохода из узла F к G обозначена путем FLG. Для протоколов, учитывающих состояние канала, желательно иметь исчерпывающую информацию о нем (загрузка, задержка, пропускная способность, надежность, стоимость и т.д.).

Некоторые из перечисленных параметров довольно легко измерить, например, задержку. Для этого вполне пригоден протокол ICMP. К сожалению многие из указанных параметров довольно сильно коррелированы и подвержены флуктуациям. В частности результаты измерения задержки зависят от загрузки канала (вариация времени ожидания в очереди).

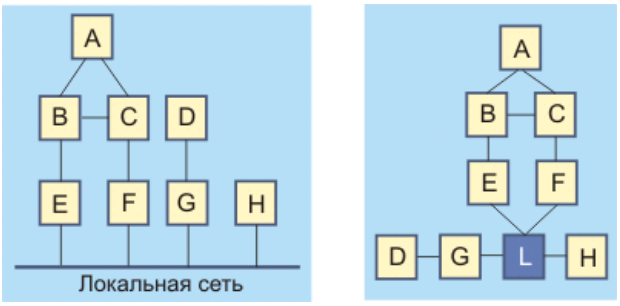


Рис. П1.6. Маршрутизаторы, подключенные к локальной сети

Рассмотрим трафик на пути A-H. Допустим на основании анализа состояния канал выбран путь через узел E. В этом случае он может оказаться перегружен, что приведет к большим задаржкам пакетов на пути A-H. Последующий анализ ситуации может привести к тому, что более оптимальным может оказаться маршрут через узел F. Если будет принято решение переключить трафик на маршрут ACFH, может перегрузиться участок ACF и история повторится. Данный сценарий описывает типичную ситуацию с осцилляциями маршрута. Осцилляции маршрутов не так безобидны, как это может показаться.

К сожалению, многие современные протоколы маршрутизации не имеют встроенных средств аутентификации (контроля доступа), что делает их уязвимыми для различных злоупотреблений.

В локальных или корпоративных сетях иной раз возникает необходимость разослать некоторую информацию всем остальным ЭВМ-пользователям сети (штормовое предупреждение, изменение курса акций, телеконференции с большим числом участников и т.д.). Отправителю достаточно знать адреса всех N заинтересованных пользователей и послать им соответствующее сообщение. Данная схема крайне не эффективна, ведь обычная широковещательная адресация предлагает решение в N раз лучше с точки зрения загрузки сети (посылается одно, а не N сообщений). Широковещательная адресация сработает, если в локальной сети нет маршрутизаторов, в противном случае широковещательные адреса MAC-типа заменяются на IP-адреса (что, впрочем, не слишком изящное решение) или применяется мультикастинг адресация. Для мультикастинг адресации в Интернет используются специальные адреса D-класса. Такие адреса позволяют организовать до 250 миллионов групп адресатов, функционирующих одновременно. При посылке пакета по такому адресу

доставка не гарантируется и некоторые члены группы могут не получить этот пакет. Маршрутизация для мультикастинга представляет собой отдельную задачу. Ведь здесь надо проложить маршрут от отправителя к большому числу получателей. Традиционные методы маршрутизации здесь применимы, но до крайности не эффективны. Для целей выбора маршрута можно с успехом применить алгоритм "дерево связей" (spanning tree; не имеет циклических структур). Когда на вход маршрутизатора приходит широковещательный пакет, он проверяет, является ли интерфейс, через который он пришел, оптимальным направлением к источнику пакета. Если это так, пакет направляется через все внешние интерфейсы кроме того, через который он пришел. В противном случае пакет игнорируется (так как, скорее всего это дубликат). Этот алгоритм называется Reverse Path Forwarding (переадресация в обратном направлении). Пояснение работы алгоритма представлено на рис. П1.7. (прямоугольниками на рисунке обозначены маршрутизаторы). Секция I характеризует топологию сети. Справа показано дерево маршрутов для маршрутизатора I (sink tree). Секция III демонстрирует то, как работает алгоритм Reverse Path Forwarding. Сначала I посылает пакеты маршрутизаторам B, F, H, J и L. Далее посылка пакетов определяется используемым алгоритмом.

При передаче мультимедиа информации используются принципиально другие протоколы маршрутизации. Здесь путь прокладывается от получателя к отправителю, а не наоборот. Это связано с тем, что там при доставке применяется мультикастинговый метод. Здесь, как правило, один отправитель посылает пакеты многим потребителям. При этом важно, чтобы размножение пакета происходило как можно ближе к кластеру адресатов. Такая стратегия иной раз удлиняет маршрут, но всегда снижает результирующую загрузку сети.

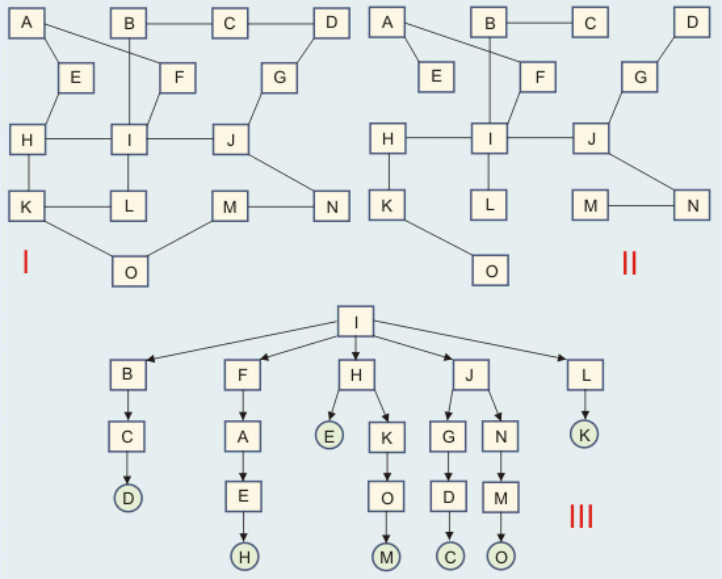


Рис. П1.7. Алгоритм Reverse Path Forwarding

Маршрутизация для мобильных объектов

В последнее время все больше людей обзаводятся компактными переносимыми ЭВМ, которые они берут с собой в деловые поездки, и хотели бы использовать в привычном режиме для работы в Интернет. Конечно, можно заставить модем дозвониться до вашего модемного пула в офисе, но это не всегда лучшее решение как по надежности так и по цене. Пользователи с точки зрения их подвижности могут быть разделены на три группы:

- стационарные, работающие всегда на своем постоянном месте в локальной сети;
- мигрирующие, меняющие время от времени свое рабочее место в рамках локальной сети или даже переходящие из одной LAN в другую (за время сессии перемещения машины не происходит);
- подвижные, перемещающиеся в пространстве и желающие работать в процессе перемещения.

Предполагается, что все эти пользователи имеют свою постоянную приписку к какой-то сети и соответствующий постоянный IP-адрес. (см. RFC-2794 Mobile IP Network Access Identifier Extension for IPv4. P. Calhoun, C. Perkins. March 2000). На рис. П1.8 показана схема подключения подвижных пользователей к Интернет. В этой схеме предполагается наличие в каждой области сети Интернет внешнего агента, обеспечивающего доступ к этой зоне подвижных ЭВМ (на рисунке такой агент помечен надписью "чужая LAN"). Доступ может осуществляться через мобильную телефонную сеть. Предполагается также наличие соответствующего агента в "домашней" LAN, куда стационарно приписана данная ЭВМ. Домашний агент отслеживает все перемещения своих пользователей, в том числе и тех, кто подключается к "чужим" LAN.

Рис. П1.8. Схема подключения к Интернет подвижных объектов

Когда к локальной сети подключается новый пользователь (непосредственно физически или через модем сотовой телефонной сети), он должен там зарегистрироваться. Процедура регистрации включает в себя следующие операции:

1. Каждый внешний агент периодически широковещательно рассылает пакет-сообщение, содержащее его IP-адрес. "Вновь прибывшая ЭВМ" может подождать такого сообщения или сама послать

широковещательный запрос наличия внешнего агента.

2. Мобильный пользователь регистрируется внешним агентом, сообщая ему свой IP- и MAC-адрес, а также некоторые параметры системы безопасности.

3. Внешний агент устанавливает связь с LAN постоянной приписки зарегистрированного мобильного пользователя, сообщая необходимую адресную информацию и некоторые параметры аутентификации.

4. Домашний агент анализирует параметры аутентификации и, если все в порядке, процедура установления связи будет продолжена.

5. Когда внешний агент получает положительный отклик от домашнего агента, он сообщает мобильной ЭВМ, что она зарегистрирована.

Когда пользователь покидает зону обслуживания данной LAN или MAN, регистрация должна быть анулирована, а ЭВМ должна быть автоматически зарегистрирована в новой зоне. Когда посылается пакет мобильному пользователю, "домашняя LAN", получив его, маршрутизирует пакет внешнему агенту, зарегистрировавшему данного пользователя. Этот агент переправит пакет адресату.

Процедуры переадресации выполняются с привлечением технологии IP-туннелей. Домашний агент предлагает отправителю посылать пакеты непосредственно внешнему агенту области, где зарегистрирована подвижная ЭВМ. Существует много вариантов реализации протокола с разным распределением функций между маршрутизаторами и ЭВМ. Существуют схемы и временным выделением резервного IP-адреса подвижному пользователю. Международный стандарт для решения проблемы работы с подвижными пользователями пока не разработан.

При широком внедрении IPv6 с практически неограниченным ресурсом адресов проблемы выделения IP-адреса вообще не будет.

В последнее время конфигурирование сетевого оборудования (маршрутизаторов, DNS и почтовых серверов усложнилось настолько, что это стало составлять заметную часть издержек при формировании коммуникационного узла. Заметного упрощения и удешевления маршрутизаторов можно ожидать при внедрении IPv6. Следующим шагом станет внедрение объектно-ориентированного языка описания маршрутной политики RPSL (Routing Policy Specification Language). Здесь конфигурирование маршрутизатора будет осуществляться на основе описанной маршрутной политики.

Приложение 2

Описание алгоритма Дейкстры

Алгоритм Дейкстры решает задачу о кратчайших путях из одной вершины для взвешенного ориентированного графа G = (V, E) с исходной вершиной s, в котором веса всех рёбер неотрицательны ((u, v) ≥ 0 для всех (u, v)E).

Формальное объяснение

В процессе работы алгоритма Дейкстры поддерживается множество S V, состоящее из вершин v, для которых δ(s, v) уже найдено. Алгоритм выбирает вершину u V\S с наименьшим d[u], добавляет u к множеству S и производит релаксацию всех рёбер, выходящих из u, после чего цикл повторяется. Вершины, не лежащие в S , хранятся в очереди Q с приоритетами, определяемыми значениями функции d. Предполагается, что граф задан с помощью списков смежных вершин.

Не формальное объяснение

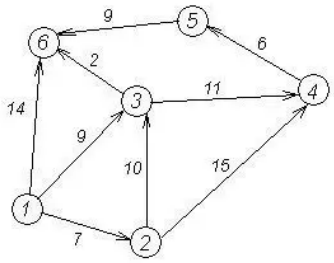
Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещенные.

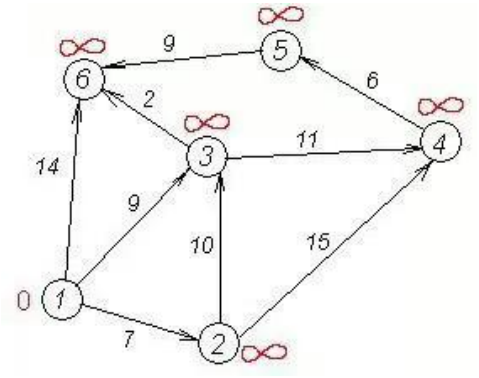
Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае из еще не посещенных вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, соединенные с вершиной u ребрами, назовем соседями этой вершины. Для каждого соседа рассмотрим новую длину пути, равную сумме текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученная длина меньше метки соседа, заменим метку этой длиной. Рассмотрев всех соседей, пометим вершину u как посещенную и повторим шаг.

ПРИМЕР

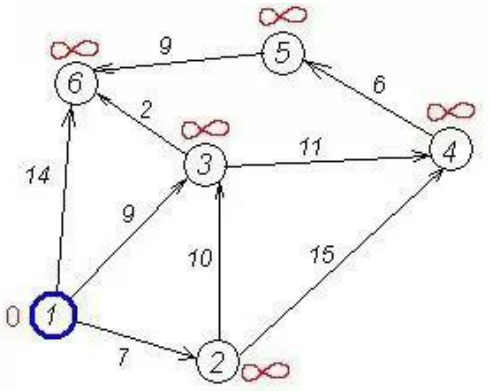
Рассмотрим работу алгоритма на примере графа, показанного на рисунке. Пусть требуется найти расстояния от 1-й вершины до всех остальных.



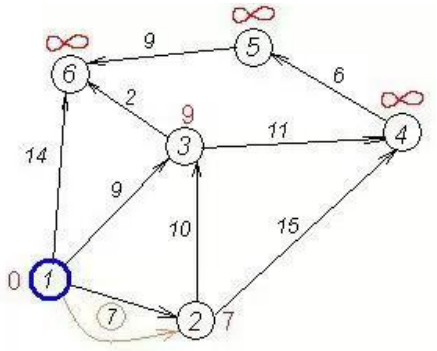
Кружками обозначены вершины, линиями — пути между ними (ребра графа). В кружках обозначены номера вершин, над ребрами обозначена их "цена" — длина пути. Рядом с каждой вершиной красным обозначена метка — длина кратчайшего пути в эту вершину из вершины 1.



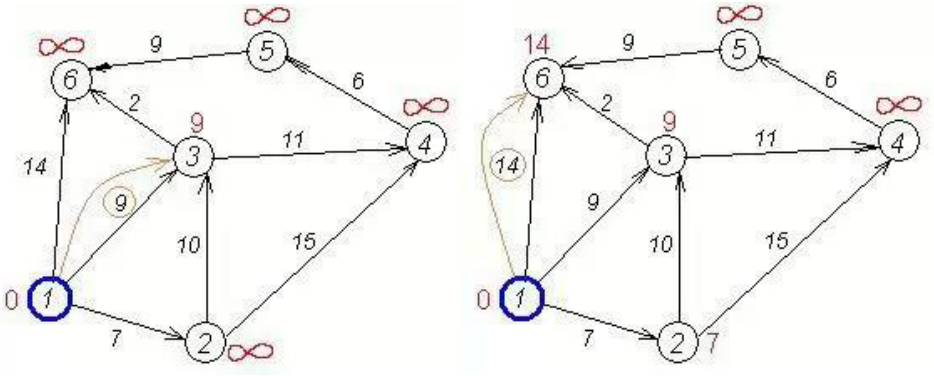
Первый шаг. Рассмотрим шаг алгоритма Дейкстры для нашего примера. Минимальную метку имеет вершина 1. Ее соседями являются вершины 2, 3 и 6.



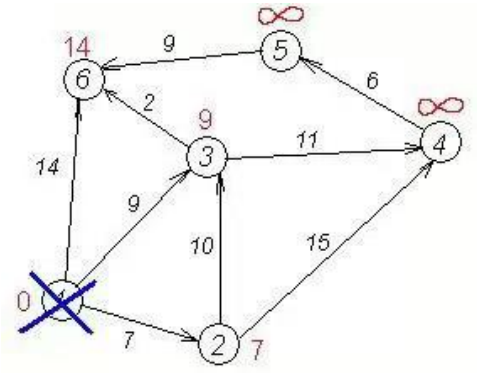
Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до нее минимальна. Длина пути в нее через вершину 1 равна кратчайшему расстоянию до вершины 1 + длина ребра, идущего из 1 в 2, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2, поэтому новая метка 2-й вершины равна 7.



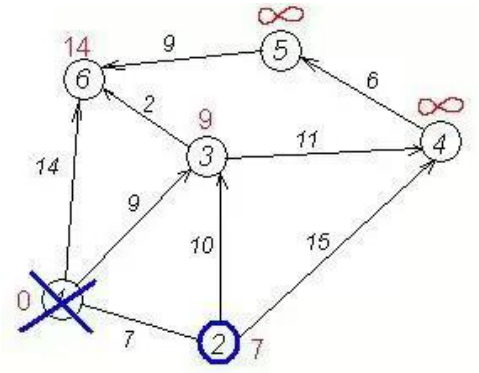
Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.



Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит (то, что это действительно так, впервые доказал Дейкстра). Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.



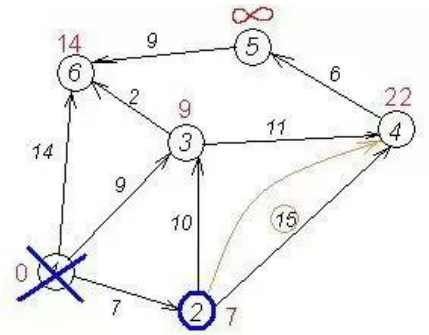
Второй шаг. Шаг алгоритма повторяется. Снова находим "ближайшую" из непосещенных вершин. Это вершина 2 с меткой 7.



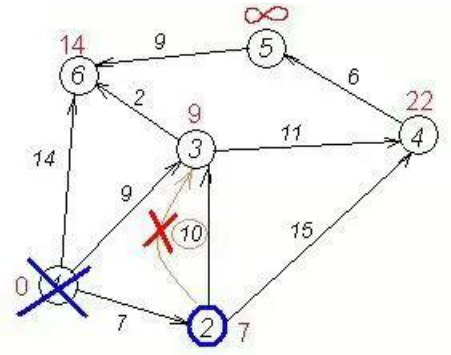
Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю. Соседями вершины 2 являются 1, 3, 4.

Первый (по порядку) сосед вершины 2 — вершина 1. Но она уже посещена, поэтому с 1-й вершиной ничего не делаем.

Следующий сосед вершины 2 — вершины 4 и 3. Если идти в неё через 2-ю, то длина такого пути будет = кратчайшее расстояние до 2 + расстояние между вершинами 2 и 4 = 7 + 15 = 22. Поскольку 22<∞, устанавливаем метку вершины 4 равной 22.



Ещё один сосед вершины 2 — вершина 3. Если идти в неё через 2, то длина такого пути будет = 7 + 10 = 17. Но текущая метка третьей вершины равна 9<17, поэтому метка не меняется.



Все соседи вершины 2 просмотрены, замораживаем расстояние до неё и помечаем ее как посещенную.

Третий шаг. Повторяем шаг алгоритма, выбрав вершину 3.

Дальнейшие шаги. Повторяем шаг алгоритма для оставшихся вершин (Это будут по порядку 6, 4 и 5).

Завершение выполнения алгоритма. Алгоритм заканчивает работу, когда вычеркнуты все вершины. Результат его работы: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Поскольку алгоритм Дейкстры всякий раз выбирает для обработки вершины с наименьшей оценкой кратчайшего пути, можно сказать, что он относится к жадным алгоритмам.

Листинг алгоритма Дейкстры на языке С++

```
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include<iomanip.h>

char NEWT[256];
char*RUS(char*TEXT) {
CharToOem(TEXT,NEWT);
return NEWT;}

int v;
int main()
{ int i,j;
 int infinity=1000; // Бесконечность
 // Количество вершин в графе
 int VES[100][100]; // Матрица весов графа

 int x[100]; //Массив, содержащий единицы и нули для каждой вершины,
 // x[i]=0 - еще не найден кратчайший путь в i-ю вершину,
 // x[i]=1 - кратчайший путь в i-ю вершину уже найден

 int DlinaPuti[100]; //t[i] - длина кратчайшего пути от вершины s в i

 int PredVertex[100]; //h[i] - вершина, предшествующая i-й вершине
 //на кратчайшем пути
 int VERTEX;
 int p;
cout<<RUS("Ввести количество вершин в графе ")<<endl;
cin>>VERTEX;p= VERTEX; //Число вершин в графе
cout<<RUS("Заполните матрицу весов графа ")<<endl; // Матрица весов графа
cout<<setw(4);
for (i=0;i<VERTEX;i++)
cout<<RUS("|x")<<i+1;
cout<<endl;

for(i=0;i<VERTEX;i++)
{cout<<RUS("X")<<i+1<<'|';
for(j=0;j<VERTEX;j++)
cin>>VES[i][j];}

 // Будем искать путь из вершины s в вершину g по циклу
 int start; // Номер исходной вершины
 int end; // Номер конечной вершины
N: cout<<RUS("Введите стартовую вершину: "); // Номер может изменяться от 0 до p-1
 cin>>start;
 if (start>(p-1) && start<0) {cout<<RUS("Нет такой вершины повторите ввод...")<<endl; goto N; } // на случай
неверных данных
 start=start-1; //так как массив начинается с 0 отнимаем от вводимой цифры 1
 for (int prosto=0;prosto<VERTEX;prosto++)
 {end=prosto; //цикл прогоняет алгоритм Флойда p-ое количество раз превращая его в алгоритм Дейкстры
 if (end==start) continue; //исключаем просчет растояния между одной и той же точкой
 else
 {

 // Инициализируем начальные значения массивов
 int u; // Счетчик вершин
 for (u=0;u<p;u++)
 {
 DlinaPuti[u]=infinity; //Сначала все кратчайшие пути из s в i
 //равны бесконечности
 x[u]=0; // и нет кратчайшего пути ни для одной вершины
 }
 PredVertex[start]=0; // s - начало пути, поэтому этой вершине ничего не предшествует
 DlinaPuti[start]=0; // Кратчайший путь из s в s равен 0
 x[start]=1; // Для вершины s найден кратчайший путь
 v=start; // Делаем s текущей вершиной
```

```
66
67  while(1)
68  {
69  // Перебираем все вершины, смежные v, и ищем для них кратчайший путь
70  for(u=0;u<p;u++)
71  {
72  if(VES[v][u]==0)continue; // Вершины u и v несмежные
73  if(x[u]==0 && DlinaPuti[u]>DlinaPuti[v]+VES[v][u]) //Если для вершины 'u' еще не
74  //найден кратчайший путь
75  // и новый путь в 'u' короче чем
76  //старый, то
77  {
78  DlinaPuti[u]=DlinaPuti[v]+VES[v][u]; //запоминаем более короткую длину пути в
79  //массив t[и]
80  PredVertex[u]=v; //запоминаем, что v->u часть кратчайшего
81  //пути из s->u
82  }
83  }
84
85  // Ищем из всех длин некратчайших путей самый короткий
86  int w=infinity; // Для поиска самого короткого пути
87  v=-1; // В конце поиска v - вершина, в которую будет
88  // найден новый кратчайший путь. Она станет
89  // текущей вершиной
90  for(u=0;u<p;u++) // Перебираем все вершины.
91  {
92  if(x[u]==0 && DlinaPuti[u]<w) // Если для вершины не найден кратчайший
93  // путь и если длина пути в вершину 'u' меньше
94  // уже найденной, то
95  {
96  v=u; // текущей вершиной становится 'u'-я вершина
97  w= DlinaPuti[u];
98  }
99  }
100 if(v==-1)
101 {
102 cout<<RUS("Нет пути из вершины ")<<start+1;cout<<RUS(" в вершину ")<<end+1<<"."<<endl;
    break;
103 }
104 if(v==end) // Найден кратчайший путь,
105 { // выводим его
106 cout<<RUS("Кратчайший путь из вершины ")<<start+1;cout<<RUS(" в вершину ")<<end+1<<":";
107 u=end;
108 while(u!=start)
109 {
110 cout<<" "<<u+1;
111 u=PredVertex[u];
112 }
113 cout<<" "<<start+1<<RUS(". Длина пути - ")<< DlinaPuti[end];cout<<endl;
114 break;
115 }
116 x[v]=1;
117 }}}
118
119 return 0;}
120
```

Приложение 4

Листинг модернизированного алгоритма Дейкстры на языке C++

```
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include<iomanip.h>

char NEWT[256];
char*RUS(char*TEXT) {
CharToOem(TEXT,NEWT);
return NEWT;}

int v;
int main()
{ int i,j;
 int infinity=1000; // Бесконечность
 // Количество вершин в графе
 int VES[100][100]; // Матрица весов графа

 int x[100]; //Массив, содержащий единицы и нули для каждой вершины,
 // x[i]=0 - еще не найден кратчайший путь в i-ю вершину,
 // x[i]=1 - кратчайший путь в i-ю вершину уже найден

 int DlinaPuti[100]; //t[i] - длина кратчайшего пути от вершины s в i

 int PredVertex[100]; //h[i] - вершина, предшествующая i-й вершине
 //на кратчайшем пути
 int VERTEX;
 int p;
cout<<RUS("Ввести количество вершин в графе ")<<endl;
cin>>VERTEX;p= VERTEX; //Число вершин в графе
cout<<RUS("Заполните матрицу весов графа ")<<endl; // Матрица весов графа
cout<<setw(4);
for (i=0;i<VERTEX;i++)
cout<<RUS("|x")<<i+1;
cout<<endl;

for(i=0;i<VERTEX;i++)
{cout<<RUS("X")<<i+1<<'|';
for(j=0;j<VERTEX;j++)
cin>>VES[i][j];}

 // Будем искать путь из вершины s в вершину g по циклу
 int start; // Номер исходной вершины
 int end; // Номер конечной вершины
N: cout<<RUS("Введите стартовую вершину: "); // Номер может изменяться от 0 до p-1
 cin>>start;
 if (start>(p-1) && start<0) {cout<<RUS("Нет такой вершины повторите ввод...")<<endl; goto N; } // на случай
неверных данных
 start=start-1; //так как массив начинается с 0 отнимаем от вводимой цифры 1
 for (int prosto=0;prosto<VERTEX;prosto++)
 {end=prosto; //цикл прогоняет алгоритм Флойда p-ое количество раз превращая его в алгоритм Дейкстры
 if (end==start) continue; //исключаем просчет растояния между одной и той же точкой
 else
 {

 // Инициализируем начальные значения массивов
 int u; // Счетчик вершин
 for (u=0;u<p;u++)
 {
 DlinaPuti[u]=infinity; //Сначала все кратчайшие пути из s в i
 //равны бесконечности
 x[u]=0; // и нет кратчайшего пути ни для одной вершины
 }
For (int:=0; i=0; i<100; i++
{
If (ves [P][i]>ves([P][i+1])
{
```

```
J=p;
While(u!=start)
Else
{
P=1
}
}
}

PredVertex[start]=0; // s - начало пути, поэтому этой вершине ничего не предшествует
DlinaPuti[start]=0; // Кратчайший путь из s в s равен 0
x[start]=1; // Для вершины s найден кратчайший путь
v=start; // Делаем s текущей вершиной

while(1)
{
// Перебираем все вершины, смежные v, и ищем для них кратчайший путь
for(u=0;u<p;u++)
{
if(VES[v][u]==0)continue; // Вершины u и v несмежные
if(x[u]==0 && DlinaPuti[u]>DlinaPuti[v]+VES[v][u]) //Если для вершины 'u' еще не
//найден кратчайший путь
// и новый путь в 'u' короче чем
//старый, то
{
DlinaPuti[u]=DlinaPuti[v]+VES[v][u]; //запоминаем более короткую длину пути в
//массив t[и]
PredVertex[u]=v; //запоминаем, что v->u часть кратчайшего
//пути из s->u
}
}

// Ищем из всех длин некратчайших путей самый короткий
int w=infinity; // Для поиска самого короткого пути
v=-1; // В конце поиска v - вершина, в которую будет
// найден новый кратчайший путь. Она станет
// текущей вершиной
for(u=0;u<p;u++) // Перебираем все вершины.
{
if(x[u]==0 && DlinaPuti[u]<w) // Если для вершины не найден кратчайший
// путь и если длина пути в вершину 'u' меньше
// уже найденной, то
{
v=u; // текущей вершиной становится 'u'-я вершина
w= DlinaPuti[u];
}
}
if(v==-1)
{
cout<<RUS("Нет пути из вершины ")<<start+1;cout<<RUS(" в вершину ")<<end+1<<"."<<endl;
break;
}
if(v==end) // Найден кратчайший путь,
{ // выводим его
cout<<RUS("Кратчайший путь из вершины ")<<start+1;cout<<RUS(" в вершину ")<<end+1<<":";
u=end;
while(u!=start)
{
cout<<" "<<u+1;
u=PredVertex[u];
}
cout<<" "<<start+1<<RUS(". Длина пути - ")<< DlinaPuti[end];cout<<endl;
break;
}
x[v]=1;
}}}

return 0;}
```