Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Вычислительные системы и технологии»

ОПТИМИЗАЦИОННЫЕ АЛГОРИТМЫ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ

Методические материалы
к выполнению самостоятельных работ
по курсам «Дискретная математика», «Методы оптимизации»
и «Программирование на языке высокого уровня»
для студентов специальности 230101
«Вычислительные машины, комплексы, системы и сети»

Нижний Новгород
2007

Составители: Л.С.Ломакина, А.С.Суркова, П.И.Уваров

УДК 510.6

Оптимизационные алгоритмы теории графов в программировании: методические материалы к изучению курсов «Дискретная математика», «Методы оптимизации», «Программирование на языке высокого уровня» и выполнению самостоятельных работ для студентов специальности 230101 «Вычислительные машины, комплексы системы и сети» / НГТУ; Сост: Л.С. Ломакина, А.С. Суркова, П.И.Уваров, Н.Новгород, 2007. – 32 с.

Рассчитано на студентов специальности 230101 «Вычислительные машины, комплексы, системы и сети», может использоваться студентами других специальностей, а также магистрантами и аспирантами.

Научный редактор В.В. Кондратьев

Редактор Э.Б.Абросимова

Подп. к печ. 12.02.2007 . Формат 60x84 1/16. Бумага газетная.
Печать офсетная. Печ.л. 2. Уч.-изд.л. 1,8. Тираж 300 экз.
Заказ .

Нижегородский государственный технический университет
Типография НГТУ. 603950, Н.Новгород, ул. Минина, 24.

ВВЕДЕНИЕ

Среди дисциплин и методов дискретной математики теория графов и особенно алгоритмы на графах находят наиболее широкое применение в программировании. Теория графов предоставляет очень удобный язык для описания программных и многих других моделей. Стройная система специальных терминов и обозначений теории графов позволяют просто и доступно описывать сложные и тонкие вещи. Особенно важно наличие наглядной графической интерпретации понятия графа. Само название «граф» подразумевает наличие графической интерпретации. Картинки позволяют сразу «усмотреть» суть дела на интуитивном уровне, дополняя текстовые доказательства и сложные формулы.

Первой работой теории графов как математической дисциплины считают статью Эйлера (1736 г.), в которой рассматривалась задача о Кенигсбергских мостах. Эйлер показал, что нельзя обойти семь городских мостов и вернуться в исходную точку, пройдя по каждому мосту ровно один раз. Следующий импульс теория графов получила спустя почти 100 лет с развитием исследований по электрическим сетям, кристаллографии, органической химии и другим наукам, эти работы связаны с именами Г.Кирхгофа, А.Кэли, К.Жордана.

Графы служат удобным средством описания связей между объектами. Например, рассматривая граф, изображающий сеть дорог между населёнными пунктами, можно определить маршрут проезда от пункта А до пункта Б. Если таких маршрутов окажется несколько, хотелось бы выбрать в определённом смысле оптимальный, например, самый короткий или самый безопасный. Для решения задачи выбора требуется проводить определённые вычисления над графами. При решении подобных задач необходимо формализовать понятие графа.

Методы теории графов широко применяются в программировании. Без них невозможно обойтись при анализе и синтезе различных дискретных преобразователей: функциональных блоков компьютеров, комплексов программ и т. д.

В настоящее время теория графов охватывает большой материал и активно развивается.

1. НАХОЖДЕНИЕ КРАТЧАЙШИХ ПУТЕЙ

Существует задача, весьма характерная для планирования в крупных масштабах сетей связи транспорта и распределения. Аналогичная задача возникает также при использовании существующей тарифной системы на телефонных линиях. Она состоит в следующем.

Рассмотрим теперь задачу поиска кратчайшего пути в ориентированном графе. Пусть дан взвешенный орграф $G=(X,U,L)$ без параллельных дуг (то есть для любых двух его вершин из одной исходит, а в другую заходит не более одной дуги), L – множество весов дуг графа. В графе выделены две вершины – начальная, обозначаемая x_s и конечная,

обозначаемая x_t, необходимо найти в графе кратчайший путь между этими двумя вершинами. Общий подход к решению задачи о кратчайшем пути был развит американским математиком Р. Беллманом, эти же принципы были положены в основу методов решения задач динамического программирования.

1.1. АЛГОРИТМ ДЕЙКСТРЫ ПОИСКА КРАТЧАЙШЕГО ПУТИ

Этот алгоритм был предложен в 1959 году Е. Дейкстрой и считается одним из наиболее эффективных алгоритмов решения задачи о кратчайшем пути, однако в данном алгоритме существует одно ограничение – веса дуг должны быть положительны. Алгоритм Дейкстры называют также алгоритмом расстановки меток, в процессе его работы узлам (вершинам) сети $x_i \in X$ приписываются числа (метки) $d(x_i)$, которые служат оценкой длины (веса) кратчайшего пути от вершины x_s к вершине x_t. Если вершина x_i получила на некотором шаге метку $d(x_i)$, это означает, что в графе G существует путь из x_s в x_i, имеющий вес $d(x_i)$. Метки бывают двух типов – временные или постоянные. Превращение временной метки в постоянную означает, что кратчайшее расстояние от начальной вершины до соответствующей вершины найдено. Учитывая указанные замечания, можно формально описать алгоритм Дейкстры, который состоит из двух этапов.

Этап 1. Нахождение длины кратчайшего пути.

Шаг 1. Присвоение вершинам начальных меток. Полагаем $d(x_s)=0^*$ и считаем эту метку постоянной (постоянные метки помечаем звездочкой). Для остальных вершин $x_i \in X$, $x_i \neq x_s$ метки являются временными и считаются равными бесконечности $d(x_i)=\infty$. Обозначим y – текущая вершина, тогда на первом шаге $y=x_s$.

Шаг 2. Перерасчет меток.

Для текущей вершины y записываем множество S – множество вершин, непосредственно следующих за вершиной y, и для всех элементов из этого множества пересчитываем метки по формуле:

$$d(x_i)=\min\{d_{cmap}(x_i),\ d(y)+l(y,x)\}. \qquad (1)$$

Шаг 3. Превращение метки из временной в постоянную.

Из всех вершин графа, имеющих временные метки, выбираем вершину x_j^* с наименьшим значением метки:

$$d(x_j^*)=\min\{d(x_j) |\ x_j \in X,\ d(x_j) \text{ – временные метки}\}. \qquad (2)$$

Эта метка становится постоянной, а вершина x_j^* – текущей.

Шаг 4. Проверка на завершение первого этапа.

Если $y=x_t$, то $d(y)$ – длина кратчайшего пути из x_s в x_t; в противном случае происходит возвращение ко второму шагу.

Этап 2. Построение кратчайшего пути.

Шаг 5. Последовательный поиск дуг кратчайшего пути.

Среди вершин с постоянными метками, непосредственно предшествующих y, находим вершину x_i, удовлетворяющую соотношению:

$$d(y)=d(x_i)+l(x_i,y). \qquad (3)$$

Включаем дугу (x_i, y) в искомый путь и полагаем $y=x_i$.

Шаг 6. Проверка на завершение второго этапа.

Если $y=x_s$, то кратчайший найден путь – его образует последовательность дуг, полученных на пятом шаге алгоритма и выстроенных в обратном порядке (начиная с x_s). Если $y\neq x_s$, то возвращаемся к пятому шагу.

Отметим, что каждый раз, когда помечается некоторая вершина (не считая вершины x_s), помечается и некоторая дуга, заходящая в данную вершину. Таким образом, на любом этапе алгоритма в каждую вершину заходит не более чем одна помеченная дуга. Кроме того, дуги с метками не могут образовывать в исходном графе цикл, так как в алгоритме не может помечаться дуга, концевые вершины которой уже помечены. Следовательно, окрашенные дуги образуют в исходном графе ориентированное дерево с корнем в вершине x_s. Это дерево называется ориентированным деревом кратчайших путей. Единственный путь от вершины x_s до любой вершины x_i, принадлежащей дереву кратчайших путей, то есть является кратчайшим путем между указанными вершинами.

Поскольку на всех этапах алгоритма Дейкстры помеченные дуги образуют в исходном графе ориентированное дерево, алгоритм можно рассматривать как процедуру наращивания ориентированного дерева с корнем в вершине x_s. Когда в этой процедуре наращивания достигается вершина x_t, процедура может быть остановлена.

Если бы мы хотели определить кратчайшие пути из вершины x_s во все вершины исходного графа, то процедуру наращивания дерева следовало бы продолжить до тех пор, пока все вершины графа не были бы включены в ориентированное дерево кратчайших путей. При этом для исходного графа было бы получено покрывающее ориентированное дерево (при условии, что в этом графе содержится хотя бы одно такое дерево).

Пример 1. Нахождение кратчайшего пути в графе с использованием алгоритма Дейкстры.

Задана весовая матрица L для сети G. Найти кратчайший путь из вершины x_1 в вершину x_6.

$L=$

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	∞	9	∞	6	11	∞
x_2	∞	∞	8	∞	∞	∞
x_3	∞	∞	∞	∞	6	9
x_4	∞	5	7	∞	6	∞
x_5	∞	6	∞	∞	∞	4
x_6	∞	∞	∞	∞	∞	∞

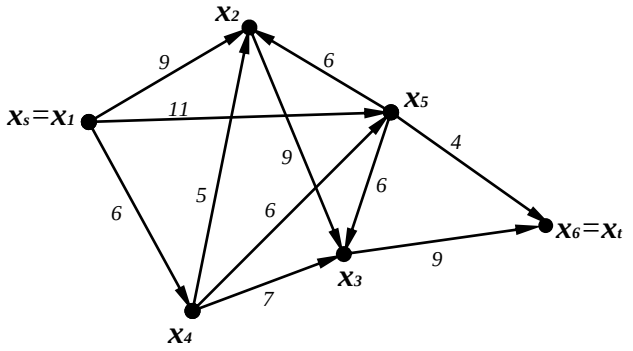


рис. 1

Граф, задаваемый матрицей весов, изображен на рис. 1. Распишем работу алгоритма Дейкстры по шагам для данного примера.

Этап 1.

Шаг 1. Полагаем $d(x_1)=0^*$, $y=x_1$,

$d(x_2)= d(x_3)= d(x_4)= d(x_5)= d(x_6)= \infty$.
1-я итерация.
Шаг 2. Множество $S=\{x_2, x_4, x_5\}$. Пересчитываем временные метки для этих вершин: $d(x_2)=\min\{\infty, 0^*+9\}=9$
$d(x_4)=\min\{\infty, 0^*+6\}=6$
$d(x_5)=\min\{\infty, 0^*+11\}=11$
Шаг 3. Находим вершину x^*, для которой временная метка превращается в постоянную: $d(x^*)=\min\{d(x_2), d(x_3), d(x_4), d(x_5), d(x_6)\}= \min\{9, \infty, 6, 11, \infty\}= 6^*= d(x_4)$. Текущая вершина $y= x_4$.
Шаг 4. $y= x_4 \neq x_t=x_6$, возвращение на шаг 2.
2-я итерация.
Шаг 2. $S=\{x_2, x_3, x_5\}$
$d(x_2)=\min\{9, 6^*+5\}=9$
$d(x_3)=\min\{\infty, 6^*+7\}=13$
$d(x_5)=\min\{11, 6^*+6\}=11$
Шаг 3. $d(x^*)= \min\{9, 13, 11, \infty\}=9^*= d(x_2)$; $y= x_2$
Шаг 4. $y= x_2 \neq x_6$, возвращение на шаг 2.
3-я итерация.
Шаг 2. $S=\{ x_3 \}$
$d(x_3)=\min\{13, 9^*+8\}=13$
Шаг 3. $d(x^*)= \min\{13, 11, \infty\}=11^*= d(x_5)$; $y= x_5$
Шаг 4. $y= x_5 \neq x_6$, возвращение на шаг 2.
4-я итерация.
Шаг 2. $S=\{ x_6 \}$
$d(x_6)=\min\{\infty, 11^*+4\}=15$
Шаг 3. $d(x^*)= \min\{13, 15\}=13^*= d(x_3)$; $y= x_3$
Шаг 4. $y= x_3 \neq x_6$, возвращение на шаг 2.
5-я итерация.
Шаг 2. $S=\{ x_6 \}$
$d(x_6)=\min\{15, 13^*+9\}=15$
Шаг 3. $d(x^*)= \min\{15\}=15^*= d(x_6)$; $y= x_6$
Шаг 4. $y= x_6= x_t$. Конец первого этапа; длина кратчайшего пути $d(x_6)=15$.
Этап 2.
1-я итерация.
Шаг 5. Множество вершин, непосредственно предшествующих $y= x_6$ с постоянными метками $S'=\{ x_3, x_5\}$. Проверяем выполнение условия (3):
$d(x_3)+l(x_3,x_6)=13^*+9=21 \neq d(y)$
$d(x_5)+l(x_5,x_6)=11^*+4=15=d(y)$
Дугу (x_5,x_6) включаем в кратчайший путь; $y= x_5$.
Шаг 6. $y= x_5 \neq x_1$, возвращение на пятый шаг.
2-я итерация.
Шаг 5. $S'=\{ x_1, x_4\}$
$d(x_1)+l(x_1,x_5)=0^*+11=11= d(y)$
$d(x_4)+l(x_4,x_5)=6^*+6=12 \neq d(y)$
Дугу (x_1,x_5) включаем в кратчайший путь; $y= x_1$.

Шаг 6. $y = x_1 = x_s$. Конец второго этапа.
Кратчайший путь от вершины x_1 к вершине x_6 построен, его образует последовательность дуг: $(x_1,x_5) – (x_5,x_6)$, и его вес равен 15.

1.2. АЛГОРИТМ БЕЛЛМАНА-МУРА

Алгоритм Беллмана-Мура похож на алгоритм Дейкстры, однако может быть применен для случая, когда некоторые дуги имеют отрицательный вес. Алгоритм Беллмана-Мура называют также алгоритмом корректировки меток. Как и в алгоритме Дейкстры, вершинам графа приписываются метки, однако нет деления меток на временные и постоянные. Корректировка меток осуществляется по правилу (1), но выполнение условия (2) не гарантирует, что найдено кратчайшее расстояние от x_s до x_j, так как наличие в графе дуг с отрицательными весами может уменьшить эту метку на последующих шагах. Поэтому в алгоритме Беллмана-Мура формируется очередь вершин, в которых по правилу (1) анализируются возможности уменьшения меток вершин, не находящихся в данный момент в очереди, но достижимых из нее за один шаг. В процессе работы алгоритма одна и та же вершина может несколько раз вставать в очередь, а затем покидать ее.
Этап 1. Нахождение длин кратчайших путей от вершины x_s до всех остальных вершин графа.
Шаг 1. Присвоение начальных значений.
$d(x_s)=0$; $d(x_i)=\infty$, для всех $x_i \in X$, $x_i \neq x_s$.
Обозначим Q – множество вершин в очереди.
Текущая вершина $y=x_s$.
Шаг 2. Корректировка меток в очереди.
Удаляем из очереди Q вершину, находящуюся в самом начале очереди, она считается текущей y. Для каждой вершины x_i, непосредственно достижимой из y, корректируем ее метку по формуле (1):

$$d(x_i)=\min\{d_{стар}(x_i),\ d(y)+l(y,x)\} \qquad (1)$$

Если получившаяся метка удовлетворяет условию

$$d(x_i)<d_{стар}(x_i), \qquad (4)$$

то корректируем очередь вершин, если условие (4) не выполняется, продолжаем перебор вершин и изменение меток.
Корректировка очереди выполняется по следующему правилу. Если вершина x_i не была ранее в очереди и не находится в ней в данный момент, то ставим ее в конец очереди; если же x_i уже была когда-нибудь ранее в очереди или находится там в настоящий момент, то переставляем ее в начало очереди.
Шаг 3. Проверка на завершение первого этапа.
Если множество вершин в очереди не пусто ($Q \neq \varnothing$), происходит возвращение к началу второго шага; если $Q=\varnothing$, значит найдены минимальные расстояния от x_s до всех вершин графа, конец первого этапа.
Этап 2. Построение кратчайшего пути.
Второй этап алгоритма Беллмана-Мура совпадает с соответствующим этапом в алгоритме Дейкстры, состоит из двух шагов и основан на применении формулы (3).

Пример 2. Использование алгоритма Беллмана-Мура для нахождения кратчайшего пути в графе с произвольными значениями весов.

Граф, изображенный на рис. 2, задан весовой матрицей. Найти кратчайший путь из вершины x_1 до x_6.

$L=$

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	∞	4	∞	6		∞
x_2	∞	∞	7	-8	6	∞
x_3	∞	∞	∞	∞	-7	5
x_4	∞	∞	8	∞	9	∞
x_5	∞	∞	∞	∞	∞	3
x_6	∞	∞	∞	∞	∞	∞

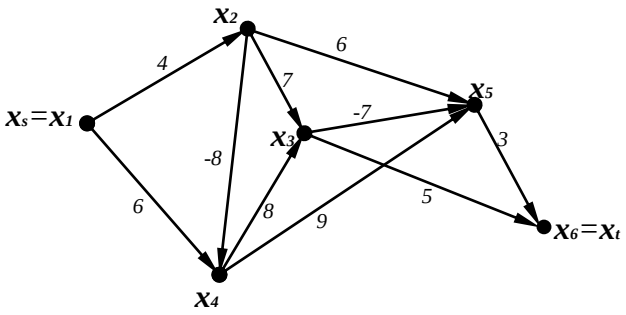


рис. 2

Этап 1.

Шаг 1. Полагаем $d(x_1)=0$, $y=x_1$,

$d(x_2)= d(x_3)= d(x_4)= d(x_5)= d(x_6)= \infty$, $Q=\{ x_1\}$.

1-я итерация.

Шаг 2. $y= x_1$, $Q=Q\setminus\{ x_1\}=\varnothing$.

Множество S – множество вершин, непосредственно достижимых из y. $S=\{x_2, x_4\}$. Пересчитываем метки последовательно для этих вершин: $d(x_2)=\min\{\infty, 0+4\}=4$, и проверяем условие *(4):* $4 < \infty$? Условие выполняется, корректируем очередь вершин: x_2 не была ранее в очереди, то есть ставим ее в конец очереди, но поскольку Q пусто, конец очереди совпадает с началом. $Q=\{ x_2\}$.

$d(x_4)=\min\{\infty, 0+6\}=6$. $6 < \infty$? Да. $Q=\{ x_2, x_4\}$.

Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

2-я итерация.

Шаг 2. $y= x_2$, $Q=Q\setminus\{ x_2\}=\{x_4\}$. $S=\{x_3, x_4, x_5\}$.

$d(x_3)=\min\{\infty, 4+7\}=11$, $11 < \infty$? Да. $Q=\{x_4, x_3\}$.

$d(x_4)=\min\{6, 4 – 8\}= –4$. $–4 < \infty$? Да. Вершину x_4 надо поставить в начало очереди, но она уже стоит там. $Q=\{x_4, x_3\}$.

$d(x_5)=\min\{\infty, 4+6\}=10$, $10 < \infty$? Да. $Q=\{x_4, x_3, x_5\}$.

Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

3-я итерация.

Шаг 2. $y= x_4$, $Q= Q\setminus\{y\}=Q\setminus\{x_4\}=\{x_3, x_5\}$. $S=\{x_3, x_5\}$.

$d(x_3)=\min\{11, –4+8\}=4$, $4 < 11$? Да. $Q=\{x_3, x_5\}$. Вершину x_3, надо поставить в начало очереди, но она уже там.

$d(x_5)=\min\{10, –4 + 9\}= 5$. $5 < 10$? Да. $Q=\{x_3, x_5\}$. Вершину x_5 передвигаем из конца очереди в начало. $Q=\{x_5, x_3\}$.

Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

4-я итерация.

Шаг 2. $y= x_5$, $Q= Q\setminus\{y\}=Q\setminus\{x_5\}=\{x_3\}$. $S=\{x_6\}$.

$d(x_6)=\min\{\infty, 5+3\}=8$, $8 < \infty$? Да. $Q=\{x_3, x_6\}$.

Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

5-я итерация.

Шаг 2. $y= x_3$, $Q= Q\backslash\{y\}=Q\backslash\{x_3\}=\{x_6\}$. $S=\{x_5, x_6\}$.
$d(x_5)=\min\{5, 4 – 7\}=–3$, $–3 < 5$? Да. $Q=\{x_5, x_6\}$.
$d(x_6)=\min\{8, 4 + 5\}= 8$. $9 < 8$? Нет. $Q=\{x_5, x_6\}$.
Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

6-я итерация.

Шаг 2. $y= x_5$, $Q= Q\backslash\{y\}=Q\backslash\{x_5\}=\{x_6\}$. $S=\{x_6\}$.
$d(x_6)=\min\{8, –3 + 3\}= 0$. $0 < 8$? Да. $Q=\{x_6\}$. Q содержит только вершину x_6, и она встала в начало очереди.
Шаг 3. $Q= \varnothing$? Нет. Переход на начало второго шага.

7-я итерация.

Шаг 2. $y= x_6$, $Q= Q\backslash\{y\}=Q\backslash\{x_6\}=\varnothing$. $S=\varnothing$.
Шаг 3. $Q= \varnothing$? Да. Конец первого этапа. Найдены минимальные расстояния до всех вершин от первой: $d(x_2)=4$, $d(x_3)=4$, $d(x_4)=–4$, $d(x_5)=–3$, $d(x_6)=0$.

Этап 2.

$y= x_6$.

1-я итерация.

Шаг 4. Множество вершин, непосредственно предшествующих $y= x_6$: $S'=\{x_3, x_5\}$.
$d(x_3)+l(x_3,x_6)=4+5=9 \neq 0 = d(y)$.
$d(x_5)+l(x_5,x_6)= –3+3 = 0 =d(y)$.
Дугу (x_5,x_6) включаем в кратчайший путь; $y= x_5$.
Шаг 5. $y= x_5 \neq x_1$, возвращение на четвертый шаг.

2-я итерация.

Шаг 4. Множество вершин, непосредственно предшествующих $y= x_5$: $S'=\{x_2, x_3, x_4\}$.
$d(x_2)+l(x_2,x_5)=4+6=10 \neq –3 = d(y)$.
$d(x_3)+l(x_3,x_5)=4 – 7= –3 = d(y)$.
$d(x_4)+l(x_4,x_5)= – 4+9 =5 \neq –3 =d(y)$.
Дугу (x_3,x_5) включаем в кратчайший путь; $y= x_3$.
Шаг 5. $y= x_3 \neq x_1$, возвращение на четвертый шаг.

3-я итерация.

Шаг 4. Множество вершин, непосредственно предшествующих $y= x_3$: $S'=\{x_2, x_4\}$.
$d(x_2)+l(x_2,x_3)=4+7=11 \neq 4 = d(y)$.
$d(x_4)+l(x_4,x_3)= – 4+8 = 4 =d(y)$.
Дугу (x_4,x_3) включаем в кратчайший путь; $y= x_4$.
Шаг 5. $y= x_4 \neq x_1$, возвращение на четвертый шаг.

4-я итерация.

Шаг 4. Множество вершин, непосредственно предшествующих $y= x_4$: $S'=\{x_1, x_2\}$.
$d(x_1)+l(x_1,x_4)=0+6=6 \neq –4 = d(y)$.
$d(x_2)+l(x_2,x_4)= 4 – 8 = –4 =d(y)$.
Дугу (x_2,x_4) включаем в кратчайший путь; $y= x_2$.

Шаг 5. $y = x_2 \neq x_1$, возвращение на четвертый шаг.

5-я итерация.

Шаг 4. Множество вершин, непосредственно предшествующих $y = x_2$: $S' = \{x_1\}$.

$d(x_1)+l(x_1,x_2)=0 + 4 = 4 = d(y)$.

Дугу (x_1,x_2) включаем в кратчайший путь; $y = x_1$.

Шаг 5. $y = x_1$, Конец второго этапа.

Искомый кратчайший путь от вершины x_1 к вершине x_6 имеет нулевой вес и состоит из следующей последовательности дуг: $(x_1,x_2) – (x_2,x_4) – (x_4,x_3) – (x_3,x_5) – (x_5,x_6)$.

2. ЗАДАЧА КОММИВОЯЖЕРА

Рассмотрим задачу коммивояжера, которая заключается в определении последовательности объезда n пунктов, при которой требуется минимизировать некоторый критерий эффективности – стоимости проезда, время в пути, суммарное расстояние и т.п. Требуется выбрать один или несколько оптимальных маршрутов из $(n-1)!$ возможных вариантов. В качестве вариантов решений выбирается последовательность объезда городов, причем допустимыми являются такие варианты, при которых в каждом городе коммивояжер должен побывать ровно один раз и вернуться в исходный пункт.

Пусть длина c_{ij} - стоимость проезда из i-го города в j-й. Если прямого пути между городами k и l не существует, то $a_{kl}=\infty$. Расстояния между городами удобно записывать в виде матрицы $A=(a_{ij})_{n\times n}$, где $a_{ii}=\infty$ (выезжая из любого города, в него нельзя вернуться непосредственно, не объехав другие города). При построении математической модели задачи коммивояжера определим переменные x_{ij} следующим образом.

Тогда математическая модель задачи коммивояжера будет следующая:

$$V=\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min$$

при ограничениях

$$\sum_{i=1}^{n} x_{ij}=1, j=1,\dots n$$

$$\sum_{j=1}^{n} x_{ij}=1, i=1,\dots n$$

$$x_{ij}\geq 0, i=1,\dots n, j=1,\dots n$$

Если городам поставить в соответствие вершины графа, а соединяющим их дорогам – дуги, то в терминах теории графов задача заключается в определении Гамильтонова контура минимальной длины. **Гамильтоновым контуром** называется путь, проходящий через все вершины графа, у

которого начальная вершина совпадает с конечной. Здесь под длиной контура понимается не количество дуг, входящих в контур, а сумму их весов (длин). Опишем алгоритм ветвей и границ для решения задачи коммивояжера.

1. Находим в каждой строке матрицы стоимостей проезда минимальный элемент и вычитаем его из каждого элемента этой строки. Если в полученной матрице окажутся столбцы, не содержащие нулевых элементов, то в каждом из них находим минимальный элемент и вычитаем его из всех элементов этого столбца. Таким образом, приходим к матрице, каждая строка и каждый столбец которой содержат, по меньшей мере, один нулевой элемент. Такое преобразование называется **приведением** матрицы по строкам и столбцам, а полученная матрица – **приведенной**.

1. Суммируем все минимальные элементы строк и столбцов, которые вычитали для приведения матрицы. Эта сумма является нижней границей H множества решений, которая берется в качестве корня дерева решений.

1. Если в каждой строке и в каждом столбце было ровно по одному нулевому элементу, то эти элементы образовали бы оптимальный маршрут, и оптимальная стоимость проезда равнялась бы H.

1. Если это не так, то по матрице стоимости строят одно звено оптимального маршрута. Если выбирают звено (i, j) (где i - строки, а j - номер столбца матрицы стоимостей), то решение не должно содержать других звеньев соответствующих элементам i-ой строки и j-го столбца. Для выбора звена оптимального, маршрута, выбираем вершины (i, j), для которых $c_{ij}=0$. Рассмотрим маршруты, не содержащие звено (i, j). Пункт i должен быть связан с некоторым другим пунктом и поэтому каждый маршрут, не содержащий узел (i, j) должен содержать звено, у которого стоимость не меньше минимального значения элемента i-й строки, не считая c_{ij}, равного нулю. Аналогично, для того чтобы в пункт j можно было бы попасть из некоторого другого пункта, маршрут, не содержащий узел (i, j), должен содержать звено, у которого стоимость не меньше, чем стоимость минимального значения элемента j-го столбца, не считая c_{ij}. Сумму минимальной стоимости в i-й строке (за исключением c_{ij}) и минимальной стоимости в j-ом столбце (за исключением c_{ij}) называют штрафом F_{ij}. Значение F_{ij} равно минимальному штрафу, которому мы подвергаемся, если не включаем звено (i, j) в оптимальный маршрут. Если штраф за неиспользование звена вычислить для всех звеньев, у которых $c_{ij}=0$, то можно сравнить соответствующие значения F_{ij} и включить в текущий маршрут такое звено (i, j), для которого штраф за не использование звена максимальный. То есть, включая звено (i, j), мы получаем выигрыш в стоимости, равный максимальному значению F_{ij}. Таким образом, нижняя граница ветви дерева маршрутов, по ветке, не включающей узел (i, j), должна быть равна сумме текущей нижней границы и максимального штрафа за не использование звена (i, j).

2. Чтобы определить новую нижнюю границу для маршрутов,

включающих звено (i, j), необходимо преобразовать матрицу стоимости. Если в маршрут включено некоторое звено (i, j), то в дальнейшем мы не рассматриваем i-ю строку и j-й столбец. Кроме того, звено (i, j) является звеном некоторого ориентированного цикла, и оно не может принадлежать этому же маршруту. Последнее условие можно выполнить, положив $c_{ij}=\infty$. Из рассмотрения следует исключить и так называемые запрещенные звенья, с помощью которых в дальнейшем могут быть образованы замкнутые циклы, включающие в себя неполное множество пунктов (могут быть образованы под маршруты) элементы матрицы стоимости, соответствующие этим звеньям, берут равными ∞. Нижняя граница для маршрута, содержащего звено (i, j), вычисляется после приведения матрицы, полученной вычеркиванием i-й строки и j-го столбца, как H с предыдущего уровня и констант приведения (то есть минимумов строк и столбцов, которые вычитались из новой матрицы для получения нулевых элементов в каждой строке и в каждом столбце). Далее процесс выбора звеньев оптимального решения повторяется, пока не построится замкнутый передвижения коммивояжера.

3. Построенный таким образом полный маршрут будет оптимальным, если его стоимость не превосходит стоимости любого маршрута, соответствующего другим звеньям дерева решений. Если есть ветви дерева стоимость движения, по которым меньше или равна, чем стоимость движения по оптимальному маршруту, следует возвратиться на эти ветки и просчитать маршруты по ним. Процедуру анализа предыдущих промежуточных точек ветвления, которые могли бы определить более дешевый маршрут или другой маршрут с минимальной стоимостью, называют возвратом. Матрицу стоимости в данном случае называют матрицей стоимости возврата (в ней в соответствии с веткой отвержения звена (m, k), ставится запрещение движения по этому звену $c_{mk}=\infty$). С новой матрицей стоимости возврата выполняют описанные процедуры ветвления и построения границ. В результате выбирают маршрут с наименьшим значением нижней границы.

Пример. Задана матрица стоимости переездов между пятью городами. Найти путь коммивояжера с наименьшей стоимостью.

	1	**2**	**3**	**4**	**5**	
1	∞	5	7	9	2	2
2	4	∞	5	7	6	4
3	7	4	∞	5	8	4
4	6	6	7	∞	4	4
5	3	7	8	5	∞	3

	1	**2**	**3**	**4**	**5**
1	∞	3	5	7	0
2	0	∞	1	3	2
3	3	0	∞	1	4
4	2	2	3	∞	0
5	0	4	5	2	∞
	0	0	1	1	0

	1	**2**	**3**	**4**	**5**
1	∞	3	4	6	0/3
2	0/0	∞	0/2	2	2
3	3	0/2	∞	0/1	4
4	2	2	2	∞	0/2
5	0/1	4	4	1	∞

Приводим матрицу и вычисляем нижнюю границу H=2+4+4+4+3+1+1=19. Для каждого нуля в приведенной матрице находим штраф: F_{15}=3, F_{21} =0,

F_{23}=2, F_{32}=2, F_{34}=1, F_{45}=2, F_{51}=1. Максимальный штраф, равный 3, соответствует звену цепи (*1, 5*), включаем это звено в маршрут и не рассматриваем далее первую строку и последний столбец. Чтобы исключить возврат из города 5 в город *1*, полагаем $c_{51}=\infty$. Новая матрица – неприведенная, приводим ее, при этом нижняя граница увеличивается на 3: H=21. Далее по алгоритму находим штрафы для всех «нулей», включаем в маршрут звено с максимальным штрафом, преобразовываем матрицу.

	1	2	3	4	5	
1						
2	0	∞	0	2		
3	3	0	∞	0		
4	2	2	2	∞		2
5	∞	4	4	1		1

	1	2	3	4	5
1					
2	0/0	∞	0/0	2	
3	3	0/0	∞	0/0	
4	0/0	0/0	0/0	∞	
5	∞	3	3	0/3	

	1	2	3	4	5
1					
2	0/3	∞	0/0		
3	3	0/3	∞		
4	∞	0/0	0/0		
5					

	1	2	3	4	5
1					
2					
3		0	∞		
4		∞	0		
5					

В результате получили путь, состоящий из звеньев (*1, 5*), (*5, 4*), (*4, 3*), (*3, 2*), (*2, 1*). Стоимость этого пути равна 22. Проверяем маршруты, в которые не включены звенья из найденного пути: путь, в который не включено звено (*1, 5*), имеет первоначальную оценку стоимости 22. Проверяем данный маршрут, исключая из рассмотрения звено (*1, 5*): $c_{15}=\infty$, находим новый маршрут:

	1	2	3	4	5	
1	∞	5	7	9	∞	*5*
2	4	∞	5	7	6	*4*
3	7	4	∞	5	8	*4*
4	6	6	7	∞	4	*4*
5	3	7	8	5	∞	*3*

	1	2	3	4	5
1	∞	0	2	4	∞
2	0	∞	1	3	2
3	3	0	∞	1	4
4	2	2	3	∞	0
5	0	4	5	2	∞
	0	0	*1*	*1*	0

	1	2	3	4	5
1	∞	0/1	1	3	∞
2	0/0	∞	0/1	2	2
3	3	0/2	∞	0/1	4
4	2	2	2	∞	0/4
5	0/1	4	4	1	∞

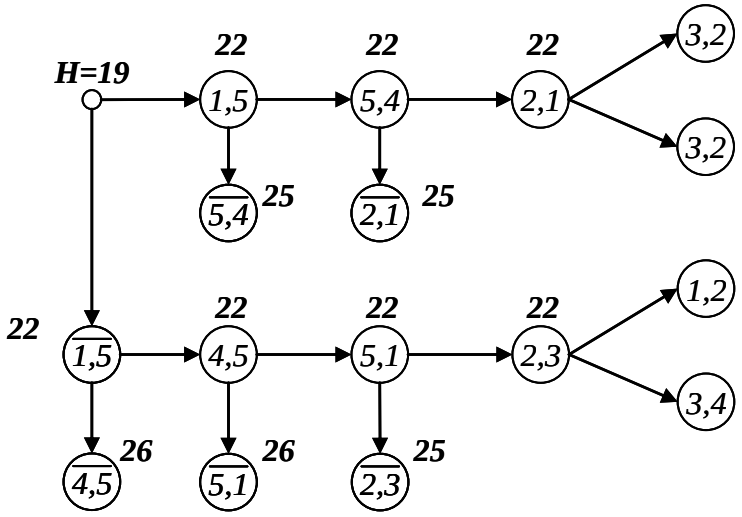
	1	2	3	4	5
1	∞	0/1	1	3	
2	0/0	∞	0/1	2	
3	3	0/0	∞	0/2	
4					
5	0/4	4	4	∞	

	1	2	3	4	5
1		0/1	1	∞	
2		∞	0/3	2	
3		0/0	∞	0/2	
4					
5					

	1	2	3	4	5
1		0		∞	
2					
3		∞		0	
4					
5					

Найденный маршрут состоит из звеньев (*1, 2*), (*2, 3*), (*3, 4*), (*4, 5*), (*5, 1*). Стоимость его равна 22. Таким образом нашли два оптимальных пути обхода городов по заданной матрице стоимостей. Других оптимальных маршрутов

нет. Последовательность поиска звеньев цепи можно представить схемой на рис. 3, где $\overline{(i,j)}$ – включение звена (i, j) в маршрут, а $\overline{\overline{(i,j)}}$ – исключение его из маршрута, рядом указана нижняя оценка стоимости маршрута.



Маршрут:

1–5–4–3–2–1

Маршрут:

1–2–3–4–5–1

Рис. 3

3. РАСКРАСКА ГРАФОВ

Пусть G – конечный связный граф; обозначим: m – число ребер графа, n – число вершин. Тогда $\gamma(G)=m-n+1$ есть **цикломатическое число** графа.

Если граф не является связным и состоит из q связных компонент, то

Свойства цикломатического числа.

1. Пусть G' – граф, полученный из графа G добавлением ребра между вершинами x_i и x_j. Если x_i и x_j связаны в G, то $\gamma(G')=\gamma(G)+1$, в противном случае $\gamma(G')=\gamma(G)$.

Доказательство. Если вершины x_i и x_j связаны, то добавление нового ребра изменяет величину m и не изменяет n и q, то есть $\gamma(G')=\gamma(G)+1$. Если x_i и x_j не связаны, то добавление ребра увеличивает на *1* величину m и уменьшает на *1* величину q, те есть $\gamma(G')=\gamma(G)$.

2. Для любого графа $\gamma(G)\geq 0$.

Доказательство. Цикломатическое число графа, состоящего из изолированных вершин равно 0, $\gamma(G)=m-n+q=0-n+n=0$. Добавление ребра может только увеличивать $\gamma(G)$, то есть $\gamma(G)\geq 0$.

3. Любой связный ациклический граф имеет $\gamma(G)=0$. Если граф имеет только один цикл, то $\gamma(G)=1$.

Пусть G – неориентированный граф с однократными ребрами и без петель. Граф называется ***k*-раскрашенным**, если множество его вершин X

можно разбить на k непересекающихся классов $X_1, ..., X_k$ так, чтобы ребра графа соединяли вершины только из разных классов.

$$X = \bigcup_{i=1}^{k} X_i, \quad X_i \cap X_j = \varnothing$$

Такое разложение множества вершин называется **хроматическим**, а классы $X_1, ..., X_k$ – **хроматическими классами.**

Будем считать, что каждый класс имеет свой цвет. При этом вершины раскрашиваются так, чтобы концы каждого ребра имели различный цвет. Перенумеруем все цвета целыми числами *1, 2, ... k.* Наименьшее число $k=k(G)$ классов в возможной раскраске графа называется **хроматическим числом** графа. Таким образом, граф называется r-раскрашиваемым, если $k(G) \le r$ и r-хроматическим, если $k(G)=r$.

Пример. Граф на рис. 4 является двухроматическим $k(G)=2$, для его раскраски необходимо 2 краски (указаны цифрами 1 и 2 около каждой вершины).

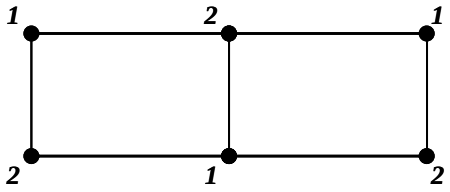


Рис. 4

Граф является однохроматическим тогда и только тогда, когда он состоит только из изолированных вершин. Полный граф с n вершинами имеет хроматическое число n.

Теорема о пяти красках. Любой плоский граф 5-раскрашиваем.

В соответствии с этой теоремой любой граф, обладающий плоской реализацией, может быть раскрашен в 5 цветов. Эта теорема дает ответ на следующую задачу. Дана географическая карта страны, в которой имеется несколько областей. Требуется окрасить каждую область так, чтобы любые 2 области, граничащие между собой, были раскрашены в разные цвета, при этом необходимо использовать минимальное число красок. В терминах теории графов эта задача может быть сформулирована так. Поставим в соответствие каждой области вершину графа и соединим ребрами те из них, которые соответствуют граничащим областям. Возникает вопрос о раскраске графа. По теореме о 5 красках для раскраски карты 5 красок достаточно. Вопрос о том, достаточно ли для любой карты 4-х красок, не решен до сих пор. Это проблема получила название задачи о четырех красках.

Примеры графов, не раскрашиваемых в 3 цвета, строятся очень легко – полный граф на 4х вершинах.

Примеры. В рассмотренных ниже примерах на рис. 5 хроматические числа графов равны соответственно: $k(G_1)=3$, $k(G_2)=2$, $k(G_3)=3$, $k(G_4)=4$, $k(G_5)=4$, $k(G_6)=2$, $k(G_7)=4$.

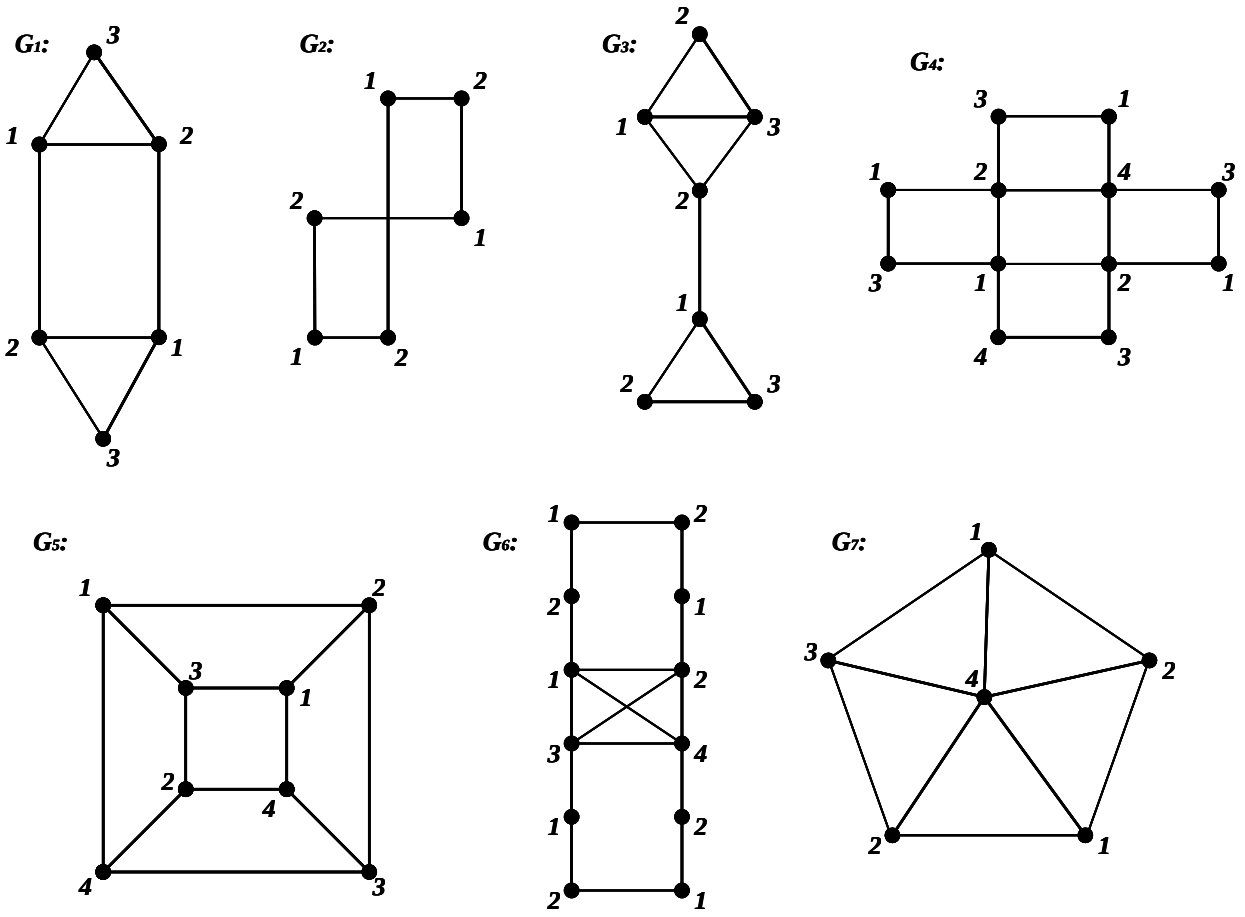


Рис.5

4. ПОТОКИ В СЕТЯХ

Задача о поиске максимального потока

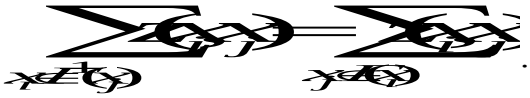
Функциональное назначение большинства физически реализованных сетей состоит в том, что они служат носителями систем потоков, т.е. систем, в которых некоторые объекты текут, движутся или транспортируются по системе каналов (дуг сети) ограниченной пропускной способности. Примерами могут служить потоки автомобильного транспорта по сети автодорог, грузов по участку железнодорожной сети, воды в городской сети водоснабжения, электрического тока в электросети, телефонных или телеграфных сообщений по каналам связи, программ в вычислительной сети. Ограниченная пропускная способность означает, что интенсивность перемещения соответствующих объектов по каналу ограничена сверху определенной величиной.

Пусть некоторая сеть представлена графом $G(X, U)$, при этом граф удовлетворяет следующим условиям.

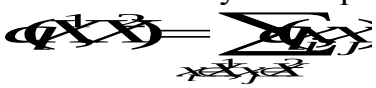
1. G – связный граф без петель.
2. Существует ровно одна вершина, не имеющая предшествующих; эта вершина называется **источником** и обозначается s.

3. Существует ровно одна вершина, не имеющая последующих; эта вершина называется **стоком** и обозначается *t*.
4. Каждой дуге $(x_i,x_j)\in U$ поставлено в соответствие неотрицательное число $q_{ij}=q(x_i\,x_j)$, называемое **пропускной способностью** дуги.

Обозначим величину потока через дугу $z_{ij} = z(x_i,x_j)$. Естественно, что поток по дуге не может превышать ее пропускной способности, то есть $z_{ij} \le q_{ij}$, $\forall(x_i,x_j)\in U$.

Обозначим через $\Gamma(x)$ – множество преемников вершины x (множество вершин, в которые заходят исходящие из вершины x дуги), а через $\Gamma^{-1}(x)$ – множество предшественников вершины x (множество вершин, из которых исходят дуги, заходящие в x). Тогда для источника $\Gamma^{-1}(s)=0$, а для стока – $\Gamma(t)=0$. Для нормального функционирования сети необходимо выполнение условия сохранения потока:  . Это условие обозначает, что сумма всех входящих в вершину потоков должна быть равна сумме всех исходящих из нее потоков, если эта вершина не является стоком или источником, то есть в промежуточных вершинах потоки не создаются и не исчезают.

Величина $\Delta_{ij}=q_{ij}-z_{ij}$ называется **остаточной пропускной способностью** дуги. Если $q_{ij}=z_{ij}$, то дуга называется **насыщенной.**

Пусть вершины графа разбиты на два непересекающихся множества X^1 и X^2. Тогда множество дуг, таких, что каждая из них берет свое начало в X^1, а оканчивается в X^2, называется **разрезом** графа G.  **Пропускной способностью** разреза или **величиной разреза** называется сумма пропускных способностей дуг, образующих этот разрез.  **Минимальным** называется разрез с минимальной пропускной способностью. Путь от источника к стоку, допускающий приращение потока называется **аугментальным.**

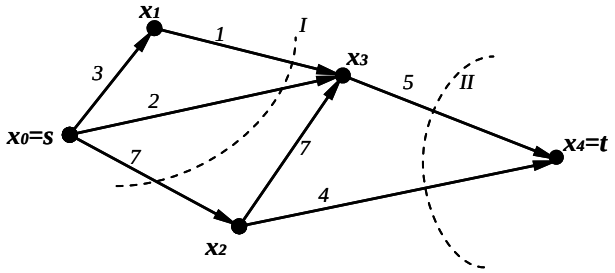


Рис. 6

На рис. 6 изображена сеть, на которой около каждого ребра указана его пропускная способность. Пунктиром показаны два разреза . При разрезе I оказались разбиты на подмножества $X^1=\{x_0, x_1\}$ и $X^2=\{x_2, x_3, x_4\}$; ребра, образующие разрез I: (x_0, x_2), (x_0, x_3), (x_1, x_3); пропускная способность этого разреза равна $1+2+7=10$. Разрез II образуют ребра (x_2, x_4), (x_3, x_4), $X^1=\{x_0, x_1, x_2, x_3\}$ и $X^2=\{x_4\}$, пропускная способность разреза равна 9.

Алгоритм поиска максимального потока следует из теоремы о максимальном потоке и минимальном разрезе или теоремы Форда–Фалкерсона.

Теорема Форда–Фалкерсона: для любой сети с одним источником и одним стоком величина максимального потока в сети от источника к стоку равна пропускной способности минимального разреза (минимальному разрезу).

Алгоритм поиска максимального потока состоит в поиске аугментальных путей, пока это возможно. В процессе работы вершины помечаются парами символов вида $[\pm x_j,\delta]$. Знак «+» в метке означает, вдоль дуги (x_i,x_j) допускается увеличение потока на величину δ, а знак «–» означает, что вдоль дуги (x_j,x_i) допускается уменьшение потока на ту же величину. Рекурсивный алгоритм поиска максимального потока записывается следующим образом:

1. Помечаем вершину x_0 (т.е. источник) меткой $[x_0,\infty]$.
2. Просматриваем вершину x_0. Для некоторой вершины x_i процесс просмотра заключается в следующем:

   a. Все непомеченные вершины $x_j \in \Gamma(x_i)$ помечаем метками $[+x_i, \min(\delta,q_{ij} - z_{ij})]$, где δ – значение метки при вершине x_i. Вершины помечаем только в том случае, если $q_{ij} > z_{ij}$.

   b. Все непомеченные вершины множества предшественников x_i: $x_j \in \Gamma^{-1}(x_i)$ помечаем метками $[-x_i, \min(\delta, z_{ij})]$, если $z_{ij} > 0$.

   c. Просматриваем аналогичным образом все вершины $x_j \in \Gamma(x_i)$ и $x_j \in \Gamma^{-1}(x_i)$ (рекурсивно).
3. После просмотра вершины x_0 проверяем, помечен ли сток.
4. Если сток не помечен, то найденный поток является максимальным. В этом случае алгоритм завершает работу, запоминая распределение потоков по дугам. Можно также найти величину самого потока путем сложения потоков всех входящих в сток дуг.
5. Если сток помечен, значит найден аугментальная цепь. Увеличиваем вдоль нее значение потока согласно установленным меткам.
6. Стираем все метки и возвращаемся к шагу 2.

Пример.

Приведем пример работы алгоритма на графе с десятью вершинами, изображенном на рис. 7:

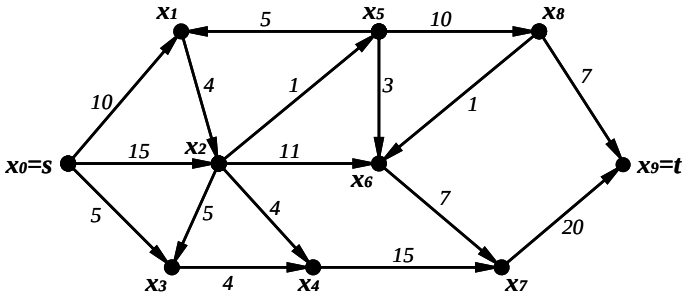


Рис. 7.

Пропускные способности заданы таблицей:

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_0		10	15	5						
x_1			4							
x_2				5	4	1	11			
x_3					4					
x_4								15		
x_5		5					3		10	
x_6								7		
x_7										20
x_8							1			7
x_9										

Первая итерация.

1. Помечаем вершину x_0 меткой $[x_0, \infty]$.
2. Просматриваем вершину x_0:
   a) $\Gamma(x_0) = \{x_1, x_2, x_3\}$:
   Вершину x_1 помечаем меткой $[+x_0, \min(\infty, 10 - 0) = 10] = [+x_0, 10]$.
   Вершину x_2 помечаем меткой $[+x_0, \min(\infty, 15 - 0) = 15] = [+x_0, 15]$.
   Вершину x_3 помечаем меткой $[+x_0, \min(\infty, 5 - 0) = 5] = [+x_0, 5]$.
   b) $\Gamma^{-1}(x_0) = \varnothing$.
   2.1. Просматриваем вершину x_1:
   a) $\Gamma(x_1) = \{x_2\}$, но x_2 уже помечена $[+x_0, 15]$.
   b) $\Gamma^{-1}(x_1) = \{x_0, x_5\}$. Из них x_0 уже помечена, а $z_{5,0} = 0$, так что не допускает уменьшение потока.
   *2.2.*Просматриваем вершину x_2:
   a) $\Gamma(x_2) = \{x_3, x_4, x_5, x_6\}$.
   Вершина x_3 уже помечена.
   Помечаем вершину x_4 меткой $[+x_2, \min(15, 4 - 0) = 4] = [+x_2, 4]$.
   Вершину x_5 – меткой $[+x_2, 1]$.
   Вершину x_6 – меткой $[+x_2, 11]$.
   b) $\Gamma^{-1}(x_2) = \{x_0, x_1\}$ – обе эти вершины помечены.
   2.2.1. Просматриваем вершину x_4:
   a) $\Gamma(x_4) = \{x_7\}$:
   Вершину x_7 пометим меткой $[+x_4, 4]$.
   b) $\Gamma^{-1}(x_4) = \{x_2, x_3\}$ – обе вершины уже помечены
   2.2.1.1. Просмотрев вершину x_7, можно пометить вершину x_9 (то есть сток) меткой $[+x_7, 4]$.
3. Нашли аугументальный путь, который допускает приращение потока на 4. Вершина x_9 помечена меткой $[+x_7, 4]$, поэтому увеличиваем поток на 4 для следующих дуг: $z_{7,9} = 4$; $z_{4,7} = 4$; $z_{2,4} = 4$; $z_{0,2} = 4$.
4. Стираем все метки и заново помечаем вершины. На рис. 8 представлено новое распределение (пунктирными линиями помечены дуги, на которых поток равен нулю; жирными – насыщенные дуги).

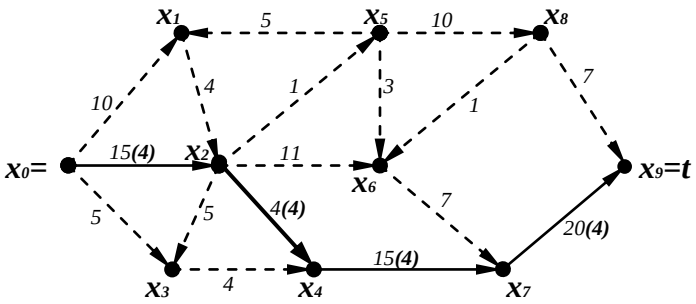


Рис. 8.

Вторая итерация.

1. Помечаем вершину x_0 меткой $[x_0, \infty]$.
2. Просмотр вершины x_0:
   a) $\Gamma(x_0) = \{x_1, x_2, x_3\}$:
   Вершина x_1 помечаем меткой $[+x_0,\ 10]$.
   x_2 помечаем $[+x_0, \min(\infty,\ 15 - 4) = 11] = [+x_0,\ 11]$.
   x_3 помечаем $[+x_0,\ 5]$.
   b) $\Gamma^{-1}(x_0) = \varnothing$.
   2.1. Просмотр вершины x_1:
   a) $\Gamma(x_1) = \{x_2\}$, но x_2 уже помечена
   b) $\Gamma^{-1}(x_1) = \{x_0, x_5\}$: x_0 уже помечена, а $z_{5,0} = 0$, так что поток по этой дуге уменьшить нельзя.
   2.2. Просмотр вершины x_2:
   a) $\Gamma(x_2) = \{x_3, x_4, x_5, x_6\}$:
   x_3 уже помечена.
   Дуга (x_2, x_4) уже насыщена – поток по ней нельзя увеличить.
   Вершине x_5 приписываем метку $[+x_2, \min(11, 1 - 0) = 1] = [+x_2,\ 1]$.
   Вершине x_6 – метку $[+x_2, \min(11, 11 - 0) = 11] = [+x_2,\ 11]$.
   b) $\Gamma^{-1}(x_2) = \{x_0, x_1\}$ – обе вершины помечены.
   2.2.1. Просмотр вершины x_5:
   a) $\Gamma(x_5) = \{x_6, x_8\}$:
   x_6 уже помечена.
   Вершину x_8 пометим как $[+x_5,\ 1]$.
   b) $\Gamma^{-1}(x_5) = \{x_2\}$ – уже помечена.
   2.2.1.1. При просмотре вершины x_8 вершина x_9 (сток) помечается меткой $[+x_8,\ 1]$.
   Дальнейшая пометка вершин не нужна. Выходим из рекурсии.
3. Сток помечен как $[+x_8, 1]$. Увеличиваем поток вдоль найденного аугментального пути: $z_{8,9} := 1$; $z_{5,8} := 1$; $z_{2,5} := 1$; $z_{0,2} := 5$.
4. Стираем все метки и расставляем заново с новым распределением потоков, которое выглядит следующим образом, представленном на рис. 9.

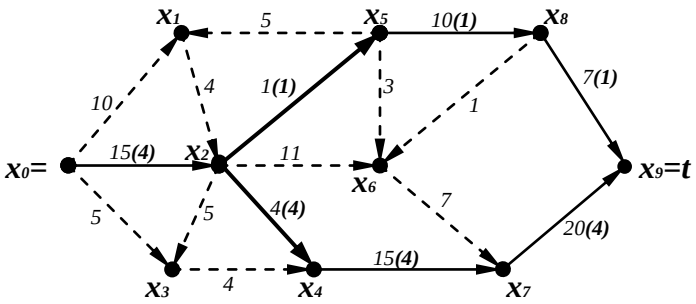


Рис. 9.

Третья итерация.

1. Действуя так же, как ранее, получаем следующие метки для вершин:
   x_1 : [+x_0, 10]. x_2 : [+x_0, 10]. x_3 : [+x_0, 5]. x_6 : [+x_2, 10]. x_7 : [+x_6, 7]. x_9 : [+x_7, 7].
2. В результате опять помечен сток. Так как метка при нем имеет вид [+x_7, 7], то можно увеличить поток вдоль найденного аугментального пути на 7: $z_{7,9}$: =11; $z_{6,7}$: = 7; $z_{2,6}$: = 7; $z_{0,2}$: = 12.
3. Новое распределение показано на рис. 10:

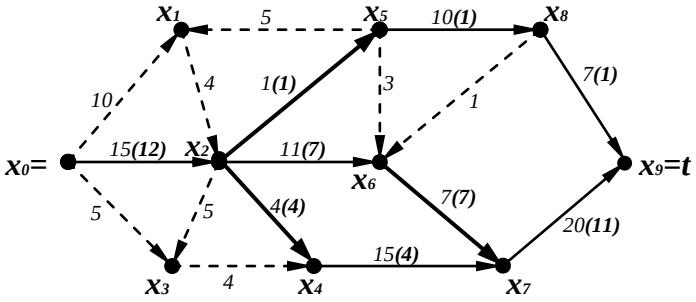


Рис. 10.

Четвертая итерация.

1. Помечая вершины с учетом нового распределения, получаем следующий набор меток для вершин:
   x_1 : [+x_0, 10]. x_2 : [+x_0, 3]. x_3 : [+x_0, 5]. x_6 : [+x_2, 3]. x_4 : [+x_3, 4]. x_7 : [+x_4, 4]. x_9 : [+x_7, 4].
2. Сток помечен. Вдоль найденного аугментального пути увеличим поток на 4: $z_{7,9}$: = 15; $z_{4,7}$: = 8; $z_{3,4}$: = 4; $z_{0,3}$: = 4.
3. Новое распределение на рис. 11:

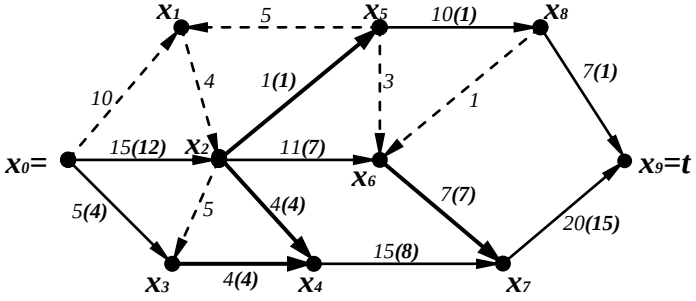


Рис. 11.

Пятая итерация.

1. По этому распределению видно, что дальнейшее увеличение потока невозможно. Действительно, если стереть метки и попытаться их расставить заново, то окажется, что сток пометить нельзя:
   x_1 : [+x_0, 10]. x_2 : [+x_0, 3]. x_3 : [+x_0, 1]. x_6 : [+x_2, 3].
   Однако остальные вершины пометить не получится: либо исходящие дуги насыщены, либо поток по входящим равен нулю.

Вывод: найденный поток является максимальным, его величина равна сумме потоков, входящих в сток, т.е. 15+1 = 16.

5. ПРИМЕНЕНИЕ ТЕОРИИ ГРАФОВ В ПРОГРАММИРОВАНИИ

При конструировании и отладке программ часто возникают задачи, либо сводящиеся к задачам теории графов, либо использующие таковые в качестве основы для решения. К ним в первую очередь относятся задачи анализа потока управления в программе, задачи тестирования и проверки правильности программы, оценки сложности и времени исполнения.

5.1. Размещение блоков программы в оперативной памяти ЭВМ (сведение к задаче коммивояжера)

Пусть $X_1, ..., X_n$ – отдельные блоки программы, p_{ij} – вероятность исполнения блока X_j после X_i. Если X_1 – первый блок, а X_n– последний, то $p_{i1}=1$ и $p_{nj}=0$ для любых i и j. Если для марковского процесса, описываемого матрицей переходов $P=||p_{ij}||$, существует предельное распределение, и оно не зависит от начального состояния процесса, то частоты исполнения отдельных блоков $X_1, ..., X_n$ определяются решением $N=(N_1, ..., N_n)$ матричного уравнения $NP=N$, нормализованного так, что сумма частот исполнения равна 1. Пусть величины N_i найдены. Тогда, если τ_i– время исполнения блока X_i, то ожидаемое время исполнения всей программы равно $\sum_{i=1}^{n} \tau_i N_i$. Если p_{ij} и N_i известны, то можно рассмотреть задачу оптимизации: минимизировать ожидаемое число передач управления, производимых при выполнении программы. Если блок X_j следует в памяти за блоком X_i, то передача управления нужна всякий раз, когда после блока X_i будет выполняться блок X_k, отличный от X_j. Ожидаемое число таких передач равно $c_{ij}=N_i(1 – p_{ij})$.

Пусть $x_{ij}=1$, если блок X_j непосредственно следует за блоком X_i в памяти, и $x_{ij}=0$ в противном случае; а блок X_{n+1}– блок, состоящий из кода начальных данных и кодов, которые не выполняются в виде команд. Тогда для всех i имеем $c_{i,n+1}=c_{n+1,i}=0$. Полагаем $x_{i,n+1}=1$, если X_i – последний блок в памяти, и $x_{n+1,i}=1$, если X_i – первый блок в памяти, в остальных случаях – $x_{i,n+1}=x_{n+1,i}=0$. Тогда рассматриваемая задача сводится к минимизации ожидаемого числа выполнения передач управления $\sum_{i=1}^{n+1}\sum_{j=1}^{n+1} c_{ij}x_{ij}$ при условии, что $||x_{ij}||$ есть матрица циклической перестановки, а это и есть задача коммивояжера. Задача коммивояжера является эталонной при исследовании

трудоемкости в том смысле, что многие задачи сводятся к ней за полиномиальное время.

5.2. Минимизация памяти при вычислении арифметических выражений (сведение к задаче построения укладки ордерева)

Арифметические выражения могут быть изображены корневыми ордеревьями, в каждую вершину которых заходит не более двух дуг. Начальные данные соответствуют висячим вершинам дерева, а промежуточные результаты – внутренним вершинам. Каждая внутренняя вершина изображает бинарную операцию над аргументами, соответствующими ее предшественникам. Порядок, в котором нужно брать аргументы, в простейших моделях не учитывается, в более сложных моделях он может играть важную роль.

Пример. Процесс вычисления арифметического выражения *((a+b)–c*d)/(e*(f–g))* может быть изображен деревом, представленным на рис. 12.

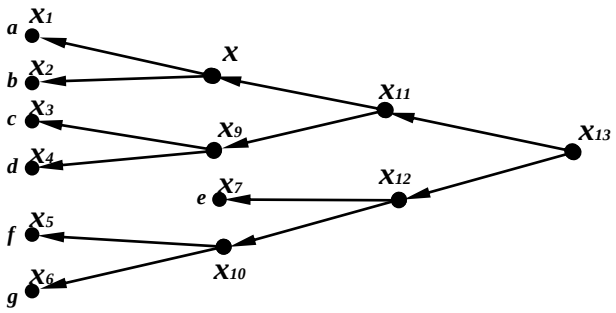


Рис.12

Выбрать возможный порядок операций – топологически упорядочить дерево, то есть расположить его вершины в целочисленных точках прямой в такой последовательности, что вершина *x* предшествует вершине *x'*, если существует дуга из *x'* в *x*. Это условие эквивалентно требованию, что каждый промежуточный результат должен быть вычислен раньше, чем использован. Топологически упорядоченное дерево выражения *((a+b)–c*d)/(e*(f–g))* показано на рис. 13.

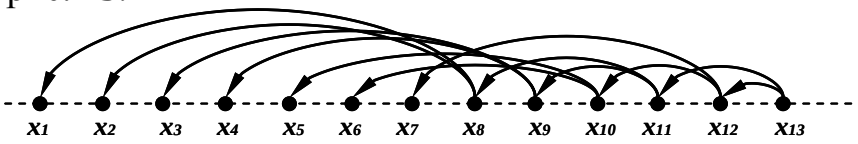


Рис.13

Пусть в некоторый момент времени выполняется операция, изображаемая вершиной x_i. Величины, не использующиеся в качестве аргументов в операции x_i, должны храниться в памяти; на рис. 13 этим величинам соответствуют дуги, проходящие над вершиной x_i. Если число этих дуг равно $w(x_i)$, то мы должны иметь не менее $w+1$ ячеек памяти в момент выполнения операции x_i (w ячеек для хранения промежуточных результатов и одну для результата операции x_i). Таким образом, возникает оптимизационная задача: минимизировать количество ячеек памяти при организации вычисления арифметического выражения. Эта задача сводится к построению укладки ордерева с минимальной шириной.

5.3. Задача экономии памяти (сведение к задаче раскраски графа)

Пусть необходимо написать программу для вычисления некоторой функции $f(x_1, x_2, \ldots x_n)$. Вычисление этой функции разбито на ряд блоков, у каждого из блоков имеются входные и выходные переменные, взаимосвязь блоков представлена на рис. 14. Так у блока 2 входные переменные *a* и *b*, выходная – *c*, а у блока 3 входная переменная *a*, а выходная – *d*. Схему представленную на рис. 14 можно назвать «блок-схемой по информации».

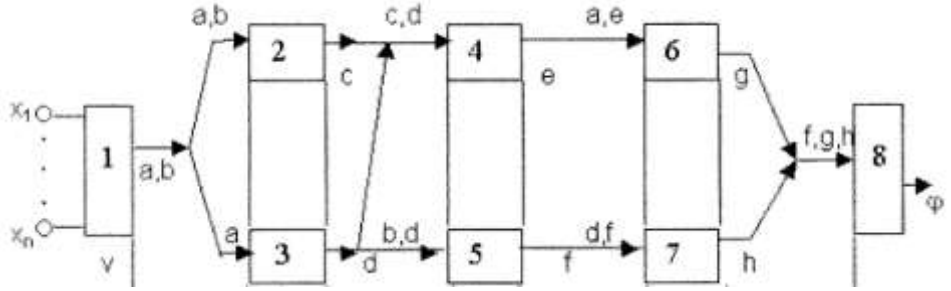


Рис. 14

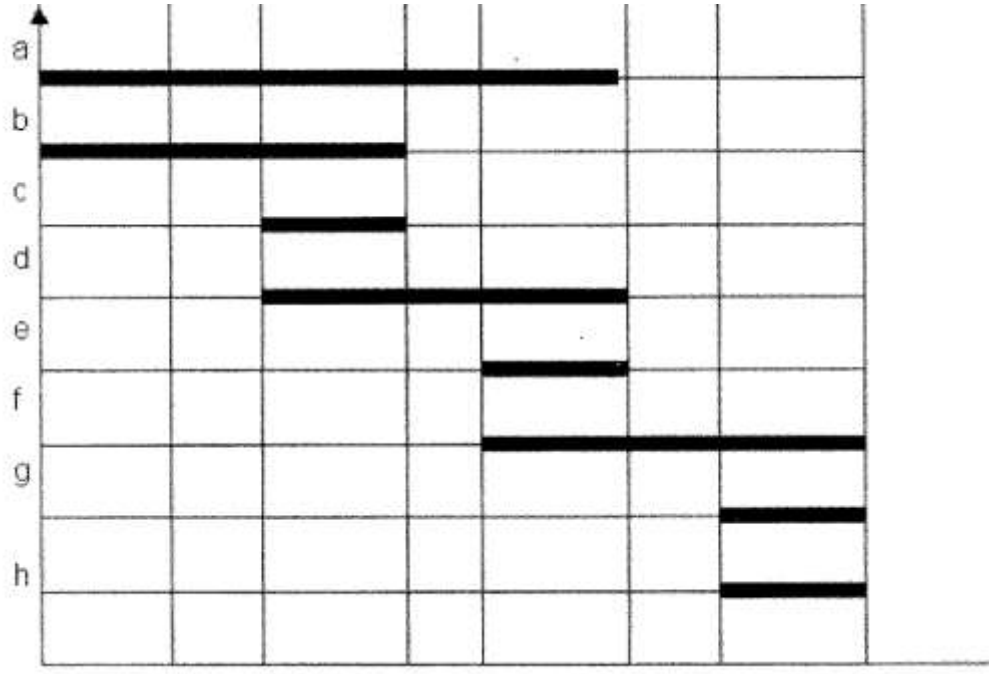


Рис. 15

На рис. 15 представлен график в системе координат, в которой по вертикальной оси отложены введенные в информационной блок-схеме переменные, а по горизонтальной оси – «время их жизни» при вычислении значения функции *f*.

Предположим, что значения переменной занимают одну ячейку памяти. Задача состоит в том, чтобы определить наименьшее число ячеек памяти, необходимое для хранения введенных переменных. На множестве переменных $V=\{a,b,\ldots g,h\}$ введем структуру графа: две переменные соединим ребром, если времена их жизни пересекаются. Полученный граф будем называть графом несовместимости переменных. Значения переменных не могут занимать одну ячейку памяти тогда и только тогда, когда переменные соединены ребром в графе несовместимости. Задача экономии

памяти сводится к нахождению оптимальной раскраски графа несовместимости. В рассмотренном примере достаточно четырех ячеек памяти (граф, представленный на рис. 16 является 4-раскрашиваемым).

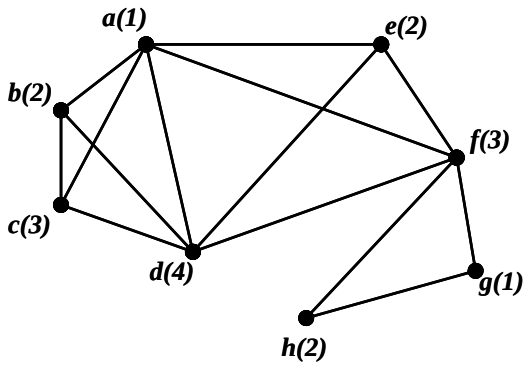


Рис. 16

5.4. Задача о распределении регистров (сведение к задаче о раскраске графов)

При системном программировании удобно считать, что в распоряжении программиста находится неограниченное число регистров, но, прежде чем осуществлять генерацию окончательного кода, необходимо отобразить это неограниченное множество виртуальных регистров на конечное множество регистров реальной машины. Первым шагом в распределении регистров является подготовка графа несовместимости. Каждый виртуальный регистр, так же как и реальный, представляется вершиной графа. Два виртуальных регистра, которые не могут сосуществовать в одном реальном, называются **несовместимыми;** соответствующие вершины в графе несовместимости соединяются ребром. Затем отыскивается минимальная раскраска графа несовместимости. Если потребовавшееся число цветов не превышает числа реальных регистров, то распределение регистров осуществляется закреплением за каждым виртуальным регистром реального регистра того же цвета.

Построение графа несовместимости в данном случае отличается от задачи, к которой сводится задача распределения памяти при составлении программ, тем, что в данном случае отсутствует требование минимального числа цветов. Будем говорить, что величина *x* **живет** в точке программы, если в управляющем графе существует путь из входа в вершину, соответствующую оператору, вырабатывающую величину *x*, далее в некоторую другую вершину и, наконец, в вершину *V*, где величина *x* используется, причем нигде на этом пути величина *x* не перевычисляется. Очевидно, что две величины, живущие одновременно в одной точке программы, несовместимы. Если в некоторой точку программы живут одновременно *k* величин, то эта точка программы изображается полным *k*-вершинником, что определяет хроматическое число графа несовместимости не меньше *k*. Если хроматическое число графа несовместимости больше, чем число реальных регистров, требуется перестройка программы для получения нужного хроматического числа.

5.5. Анализ и тестирование программ

Управляющие графы

Управляющий граф – основная модель программы, представляющая в виде орграфа систему управляющих связей в программе; в которой сохраняется членение программы на операторы, а также информация о тождественности операторов и возможных передачах управления между операторами.

Управляющий граф программы сохраняет в себе информацию о внутренней структуре программы и дает пользователю наглядное представление о своей программе. Эта информация полезна в процессе выделения путей по определенному критерию структурного тестирования. Применительно к объектно-ориентированным языкам управляющий граф представляет пользователю очень полезную информацию о взаимоотношениях между объектами на уровне классов и на уровне методов. При этом граф управления программы представляет внутриклассовые отношения и связи между методами различных классов. Такие отношения синтезируются из графов методов и моделей состояния. Это означает, что пути потока данных между методами всех классов могут быть идентифицированы и инструментированы для охвата.

Тестирование

Тестирование – проверка правильности программы с помощью множества контрольных примеров – **тестов.**

Определение. **Тестом**, проверяющим программу, называется тройка $\tau = \langle \hat{x}, \mu(G), \hat{y} \rangle$, где $\hat{x} \subset \hat{X}$ – подмножество значений входных данных, $\mu(G)$ – (s–t)-путь в управляющем графе G, соответствующий входным данным $\hat{x}$, $\hat{y} \subset \hat{Y}$ – подмножество значений выходных данных при прохождении программой пути $\mu(G)$.

Из определения вытекает, что для полного тестирования программы желательно проверить все (s–t)-пути, но это невозможно даже для небольших программ, поэтому существует несколько различных стратегий тестирования.

В первом случае задача тестирования сводится к проверке предписанного числа операторов в программе. Это требует построения кратчайшего пути в управляющем графе, который проходит через предписанное множество вершин (если такой путь существует). В общем случае предполагается, что каждой вершине сопоставлен вес $w>0$, который можно интерпретировать как сложность оператора (или отдельного блока) программы, изображаемого данной вершиной. Отыскание тестирующих данных для тестирующего пути тем проще, чем меньше вершин будет в нем содержаться из множества K вершин, не подлежащих проверке. Это соответствует задаче отыскания (s–t)-пути с наибольшим весом при условии, что вершинам из множества K приписан небольшой отрицательный вес.

В общем случае тестирования программ требуется построить тестирующие пути, или все требуемые пути (заданные отрезки путей в

управляющем графе), или все дуги, или все вершины управляющего графа. Эти задачи сводятся к построению различного вида покрытий графа путями.

Задача о контрольных точках

Задача возникает при отладке больших программ и состоит в нахождении наименьшего числа точек программы, в которые нужно поместить операторы выдачи контрольных распечаток для обнаружения ошибок в функционировании циклических частей программы. В теории графов эта задача соответствует задаче отыскания наименьшего по мощности множества дуг, удаление которых разрывает все контуры и тем самым превращает орграф в бесконтурный. Нахождение точного решения представляет собой *NP*-полную проблему и для ее решения неизвестен эффективный алгоритм. Однако существует приближенное решение, при котором отыскивается множество дуг, принадлежащих как можно большему числу контуров, алгоритм его можно записать следующим образом.

Пусть K – множество дуг, разрывающих все контуры. На вход алгоритма поступает множество простых контуров, заданных перечислением дуг, входящих в контуры.

Шаг 1. Построить матрицу $H=||h_{ij}||$, контуры-дуги которой определяются:

Шаг 2. Включить в множество K те дуги, которым в матрице H соответствуют строки, содержащие только одну единицу (тем самым в множество K включаются все петли графа).

Шаг 3. Удалить из H строки и столбцы, соответствующие этим петлям.

Шаг 4. Исключить из H все мажорируемые столбцы, то есть те, для которых найдется хотя бы один столбец, имеющий единицу в тех же строках, что и рассматриваемые.

Шаг 5. Включить в множество K дугу u_r, соответствующую столбцу с наименьшим числом единиц.

Шаг 6. Вычеркнуть из H строки, соответствующие единицам в r-м столбце, и сам r-й столбец.

Шаг 7. Проверка условия $H=\emptyset$, если оно не выполняется, необходимо вернуться к шагу 4. Если условие выполнено, конец алгоритма, K – искомое множество дуг, принадлежащих как можно большему числу контуров.

6. ОСОБЕННОСТИ АЛГОРИТМОВ В ТЕОРИИ ГРАФОВ. СЛОЖНОСТЬ ЗАДАЧ ТЕОРИИ ГРАФОВ

Рассмотренные задачи позволяют сформулировать следующие свойства алгоритмов теории графов.

1. Каждый алгоритм состоит из совокупности конечного числа правил и предписаний. Действия над графом (матрицей описывающей его), производимые в соответствии с правилами, должны быть достаточно просты.

2. Алгоритм применяется в дискретном времени, правила алгоритма – по шагам, число шагов конечно.

3. Какое правило будет применено на данном шаге или какое действие будет совершено в соответствии с некоторым правилом, зависит только от результатов предыдущих шагов.

4. Алгоритмы обладают свойством локальности: действие в соответствии с правилом или установление непротиворечивости некоторого действия правилам алгоритма происходит на основе анализ дуг, инцидентных данной вершине, или вершин смежных с данной.

5. Алгоритмы обладают свойствами массовости: применяются либо для всех, либо для некоторого бесконечного множества графов.

Для задач дискретного характера необходимо задаваться следующими вопросами.

1. Существует ли решение?
2. Как найти решение, наилучшее в том или ином смысле?

Для конечных графов почти всегда принципиальной проблемы решения задач такого рода не существует. Достаточно перебрать все возможные варианты (что в принципе возможно из-за конечности графа) и выбрать наилучшее в том или ином смысле решение. Однако в большинстве случаев «переборный» способ решения реальных задач не осуществим, даже если использовать самые современные вычислительные машины. Поэтому необходимо отходить от переборного способа решения задач. При этом большое значение имеет этап анализа того или иного способа решения задачи и выяснения ее трудности. Часто под трудностью понимают число необходимых операций или время работы алгоритма. Если требуется построить схему, которая реализовывала бы некоторую функцию, то под сложностью схемы можно понимать число элементов, из которых она собрана.

Предлагается объединять в один класс задачи, которые «достаточно быстро» сводятся друг к другу. Тогда, если хотя бы для одной задачи этого класса удастся получить простое решение, то такое же простое решение найдется и для любой другой задачи. Для более точного определения сложности задачи необходимо ввести основные понятия теории NP-полных задач.

В практике определения сложности той или иной задачи приходится почти всегда иметь дело с двумя классами функций – не более чем полиномы и не менее чем экспонента. При этом такая функция, как, например, $n\sqrt{n}\ln n$, считается полиномиальной, а $n^{\log n}$ – экспоненциальной. Известно, что при стремлении аргумента к ∞ рано или поздно любая экспоненциальная функция начнет принимать значения большие, чем любой из полиномов. Поэтому мы условно будем считать, что полином меньше, чем экспонента, а следовательно полиномиальная сложность всегда лучше, чем экспоненциальная. Будем говорить, что две модели **полиномиально эквивалентны**, если время вычисления некоторой массовой задачи одной из

них можно представить как полином от времени решения с помощью другой. Можно показать, что любые модели вычислений полиномиально эквивалентны и при кодировании входных последовательностей всегда одну из них можно перевести в другую не более чем за полиномиальное время. Переход от операций высокого уровня к двоичным тоже всегда возможен за полиномиальную сложность.

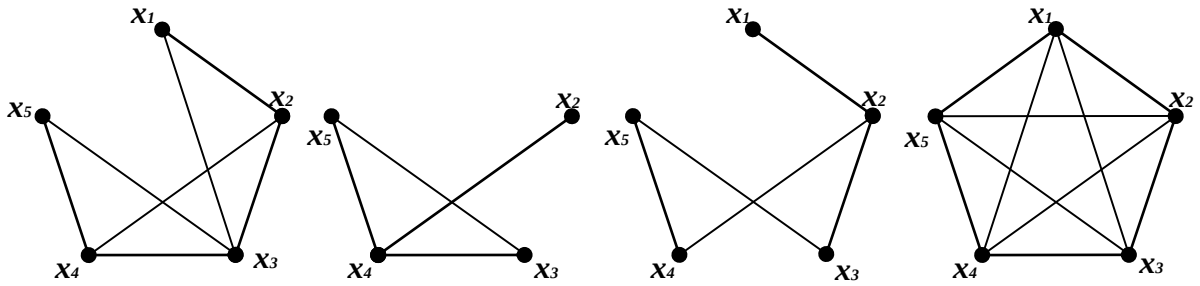
Пусть имеем некоторое устройство (алгоритм), перерабатывающее двоичные последовательности. Будем считать, что память алгоритма не ограничена. Состояние алгоритма определим как адрес выполняемой в данный момент команды и значения всех переменных. Одно из самых существенных свойств алгоритма – его детерминированность. Это означает, что для каждого имеющегося в данный момент состояния либо не существует никакого следующего (если состояние заключительное), либо существует в точности одно следующее состояние. Введем определение недетерминированного алгоритма. Предположим, что алгоритм для каждого текущего состояния имеет больше одного следующего, это означает, что если в процессе вычисления алгоритм доходит до того места, где требуется выбрать одну из нескольких альтернатив, он не исследует их как детерминированный алгоритм последовательно друг за другом, а просматривает их все одновременно, копируя самого себя. В свою очередь эти копии могут порождать новые, и так далее. Полученные копии продолжают работать все вместе до тех пор, пока одна (или несколько) из них не найдет решения. После этого дается команда на остановку работы всех копий. Конечно, никакое реально существующее устройство не может вести себя как недетерминированный алгоритм, но такая абстракция позволяет избежать трудностей, возникающих при анализе алгоритмов.

Алгоритм будем называть полиномиальным, если время его работы с некоторой массовой задачей ограничено сверху полиномом. Все задачи, которые могут быть решены с помощью полиномиальных алгоритмов, объединим в один класс и обозначим буквой P. Класс задач, которые могут быть решены с помощью недетерминированного алгоритма за полиномиальное время, обозначим через NP. Если любая задача из класса NP сводится к некоторой задаче из класса P за полиномиальное время, то она называется NP-трудной. Если же она принадлежит классу NP, то она называется NP-полной. Вопрос о том, существует ли для какой-либо NP-трудной или NP-полной задачи детерминированный полиномиальный алгоритм её решения, является чрезвычайно важным. Имеется большое число очень важных задач теории графов, которые являются NP-полными, и если хотя бы для одной из них будет найден детерминированный полиномиальный алгоритм её решения, то из этого автоматически будет следовать, что и для многих других задач будет существовать полиномиальный алгоритм решения.

7. ГЛОССАРИЙ. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Пусть задано некоторое конечное множество $X=\{x_1, x_2, ..., x_n\}$, элементы которого будем называть **вершинами**. Образуем из него новое множество $U=\{u_1, u_2, ..., u_m\}$, состоящее из пар элементов (x_i, x_j) множества X. Тогда пара $G=(X,U)$ называется **неориентированным конечным графом**. Элементы множества U называются **ребрами**.

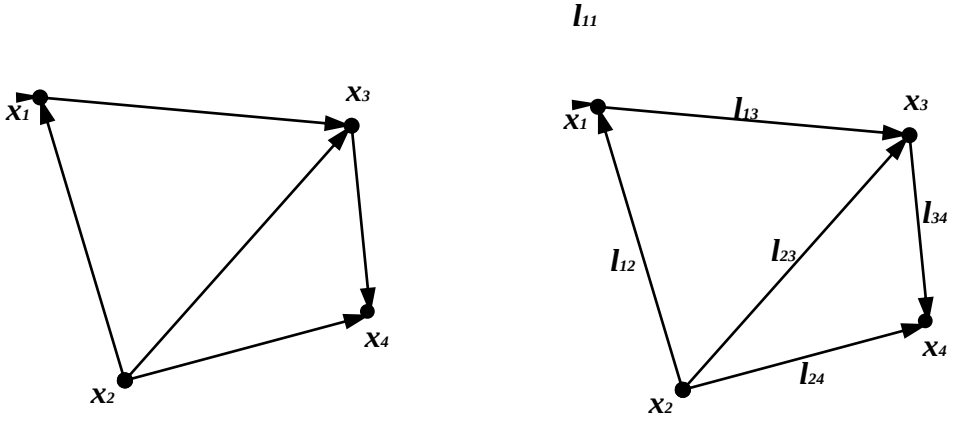
Подграфом графа $G=(X,U)$ называется граф $G'=(X',U')$ такой, что X' является подмножеством множества X, а U' – подмножеством множества U. Если при этом X' совпадает с X, то граф G' называется **остовным подграфом** графа G. Граф называется **полным**, если для любых двух его вершин существует соединяющее их ребро. На рис. 17,*a* изображен неориентированный граф, на рис. 17,*b* – его подграф, а на рис. 17,*c* – остовной подграф этого графа. На рис. 17,*d* приведен пример полного графа с пятью вершинами.



a) *b*) *c*) *d*)

рис. 17

Если в паре вершин x_i и x_j указано направление связи, то есть какая из вершин является первой, то соединяющий их отрезок u_k называется **дугой**, а вершины, определяющие дугу u_k, называют **концевыми вершинами**. Если концевые вершины совпадают, то дугу называют **петлей**. Если в графе $G=(X,U)$ все элементы множества U изображаются дугами, то граф называется **ориентированным** или **орграфом**. Пример орграфа представлен на рис 18,*a*.



a) *b*)

рис. 18

Если каждой дуге (x_i, x_j) орграфа $G=(X,U)$ поставлено в соответствие некоторое число $l(x_i, x_j)$, то граф G называется **графом со взвешенными дугами** или **сетью**, а число $l(x_i, x_j)$ – **вес** дуги (x_i, x_j). В зависимости от контекста решаемой задачи вес дуги характеризует конкретный параметр (пропускная способность, длина, стоимость и т.д.). Если в графе не существует некоторая дуга, ее вес считается равным ∞ или 0 в зависимости от приложений. На рис. 18,*b* изображен взвешенный ориентированный граф.

Задание графов с помощью матрицы смежности.

Любой граф G может быть определен с использованием так называемой матрицы смежности. Матрица смежности – квадратная матрица, ее строки и столбцы соответствуют вершинам графа. Если в графе существует ребро, соединяющее вершины x_i и x_j, то в матрице смежности на пересечении i-й строки и j-го столбца ставится единица, в противном случае – ноль. Если G – неориентированный граф без параллельных ребер, то его матрица смежности будет симметрична относительно главной диагонали. Аналогично определяется матрица смежности орграфа.

Граф со взвешенными дугами может быть представлен матрицей весов $L=|l_{ij}|$, где l_{ij} – вес ребра, соединяющего вершины x_i и x_j. Ниже приведены матрицы смежности для графов, изображенных на рис. 17,*c* и 18,*a*, и матрица весов для графа с рис. 18,*b*:

	x_1	x_2	x_3	x_4	x_5
x_1	0	1	0	0	0
x_2	1	0	1	1	0
x_3	0	1	0	0	1
x_4	0	1	0	0	1
x_5	0	0	1	1	0

рис. 17,*c*

	x_1	x_2	x_3	x_4
x_1	1	0	1	0
x_2	1	0	1	1
x_3	0	0	0	1
x_4	0	0	0	0

рис. 18,*a*

	x_1	x_2	x_3	x_4
x_1	l_{11}	∞	l_{13}	∞
x_2	l_{21}	∞	l_{23}	l_{24}
x_3	∞	∞	∞	l_{34}
x_4	∞	∞	∞	∞

рис. 18,*b*

Многие прикладные задачи представляются в виде отдельного класса теории графов – деревьев. Связанный неориентированный граф называется **деревом**, если он не имеет циклов. В дереве любые две вершины связаны единственной цепью. Если у дерева выделена одна вершина, то она называется **корнем дерева**, а сам граф – **корневым деревом**. Пример корневого дерева представлен на рис. 19.

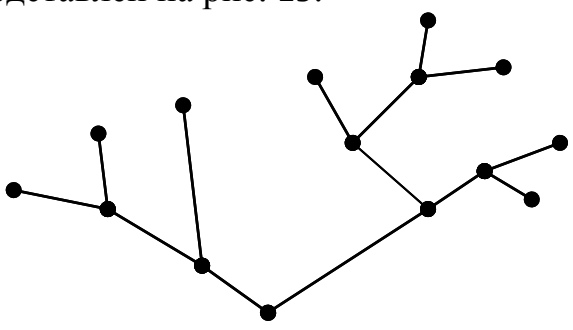


рис. 19

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Касьянов, В.Н. Графы в программировании: обработка, визуализация и применение / В.Н.Касьянов, В.А.Евстигнеев. – СПб.: БХВ-Петербург, 2003.

2. Касьянов, В.Н. Применение графов в программировании //Программирование, 2001. №3. С.51-76.

3. Ершов, Н.М. Построение графов вычислительных алгоритмов методом автотрассировки //Программирование, 2001, №6. С.8-16.

4. Буч, Г. Объектно-ориентированное проектирование с примерами применения /Г.Буч – М.: Конкорд, 1992.

5. Липаев, В.В. Надежность программных средств / В.В.Липаев. – Синтег, 1998.

6. Оре, О. Теория графов / О.Оре. – М.: Наука, 1980.

СОДЕРЖАНИЕ