Г Л А В А IV
ПРОБЛЕМЫ ПОСТРОЕНИЯ ЭФФЕКТИВНЫХ АЛГОРИТМОВ НА ЧИСЛОВЫХ ГРАФАХ

В этой главе исследуется важная проблема об оценке сложности алгоритмов на числовых графах. Получены результаты, которые позволяют в дальнейшем на этих графах для некоторых сложных задач строить эффективные алгоритмы.

4.1. Об общем представлении числовых графов

Из всех числовых графов натуральные арифметические (*NA*-графы) и натуральные модульные (*NM*-графы) графы по способу задания являются самыми простыми. Уже обычные арифметические (*A*-графы) и модульные (*M*-графы) требуют задания способа (функции), по которому из натурального ряда чисел изымаются те, которые соответствуют вершинам, не принадлежащим множеству X. В общем случае, в зависимости от вида числового графа, существуют два принципиально различных их задания.

Первый, наиболее общий способ, задается следующим образом.

Числовым графом $G=(X,U,F,g)$ называется $n-$вершинный граф, представленный двумя множествами $X=\{1,2,\ldots,n,\ldots,N\}=N_n$ – множеством вершин и $U\in N$ – множеством образующих, функцией смежности $F(x_i,x_j)$ и функцией исключения $g(x)$. Вершина $x_k\notin X$ если $g(x_k)=0$, а вершины $x_i,x_j\in X$ смежны, если $F(x_i,x_j)\in U$.

Если $F(x_i,x_j)=x_i+x_j$, то такой числовой граф называется арифметическим, если же $F(x_i,x_j)=|x_i-x_j|$, то соответственно – модульным. Относительно функции $g(x)$ никаких определенных свойств не предполагается, в конце концов, она может просто перечислять множество вершин, не принадлежащих X.

В большинстве случаев, когда заданный граф имеет определенную струк-

туру, имеющую несколько осей симметрии, или периодически повторяющиеся части, задание функций $F(x_i, x_j)$ и $g(x)$ не представляет труда.

Применение теории графов в различных областях практической деятельности человека показало, что подавляющее большинство графов, которые при этом использовалось, носили именно такую специфическую структуру. Можно перечислить множество солидных монографий из области распознавания образов, математического моделирования параллельных процессов, теоретических основ создания информационных технологий и многих других, где рассматриваются графы, которые можно представить в виде числовых графов указанным выше способом.

Рассмотрим пример из книги [16, рис.2.4, стр. 42]. В ней уравнение распознаваемой реализации сигнала x_i с эталоном слова производилось с помощью графа, представленного (в уменьшенном виде) на рис. 4.1.

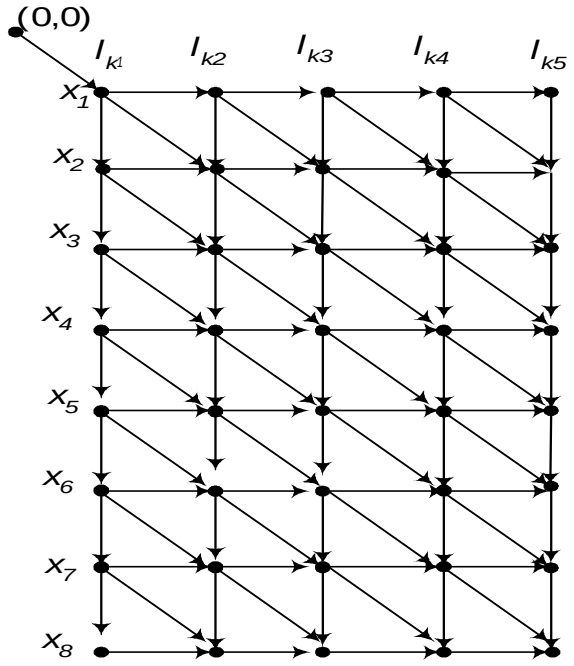

Рис. 4.1. Часто используемый развернутый граф слова

По своей структуре это довольно простой граф, который можно представить в виде числового графа следующим образом.

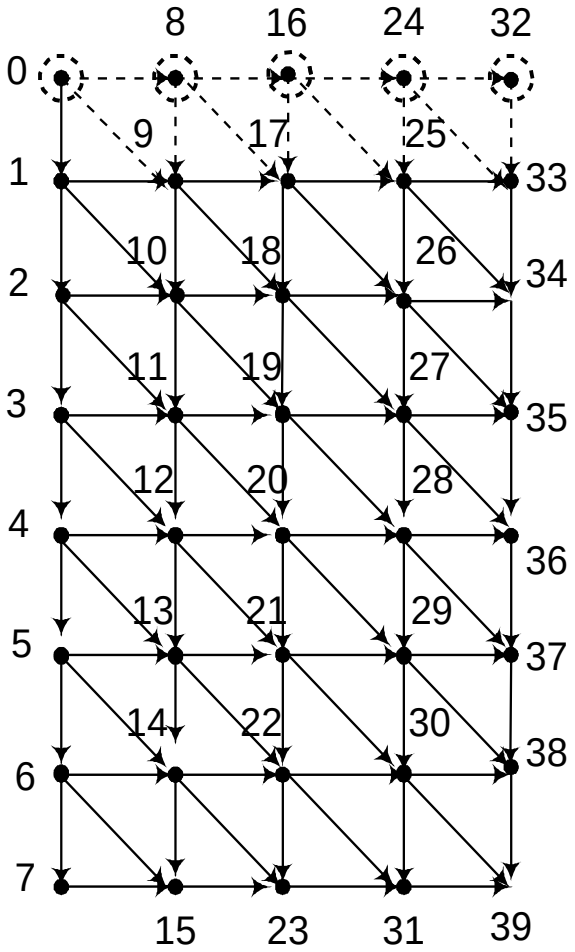

Рис. 4.2. Реализация предыдущего графа в виде числового

В данном случае, получаем граф с числом вершин $n = 39$, то есть $X = \{1,2,\ldots,39\}$, $F(x_i, x_j) = x_j - x_i$, $g(x) \equiv 0 (\mathrm{mod}\ 8)$, а $U = \{1,8,9\}$. Таким образом, информация о графе укладывается в 6 чисел. В цитируемой книге граф по размерам больше, он по вертикали имеет 21 вершину, и по горизонтали – 14 вершин, то есть всего – 294. Числовой граф, (с учетом добавленных фиктивных вершин) будет иметь $n = 308$ вершин и будет задаваться таким образом:

$$X = \{1,2,\ldots,308\},\ F(x_i, x_j) = x_j - x_i,\quad g(x) \equiv 0(\mathrm{mod}\ 22),\quad U = \{1,22,23\}.$$

Аналогичное представление допускает и граф, представленный на рис.4.3, взятый из той же книги [16 , рис. 9.6, стр.185].

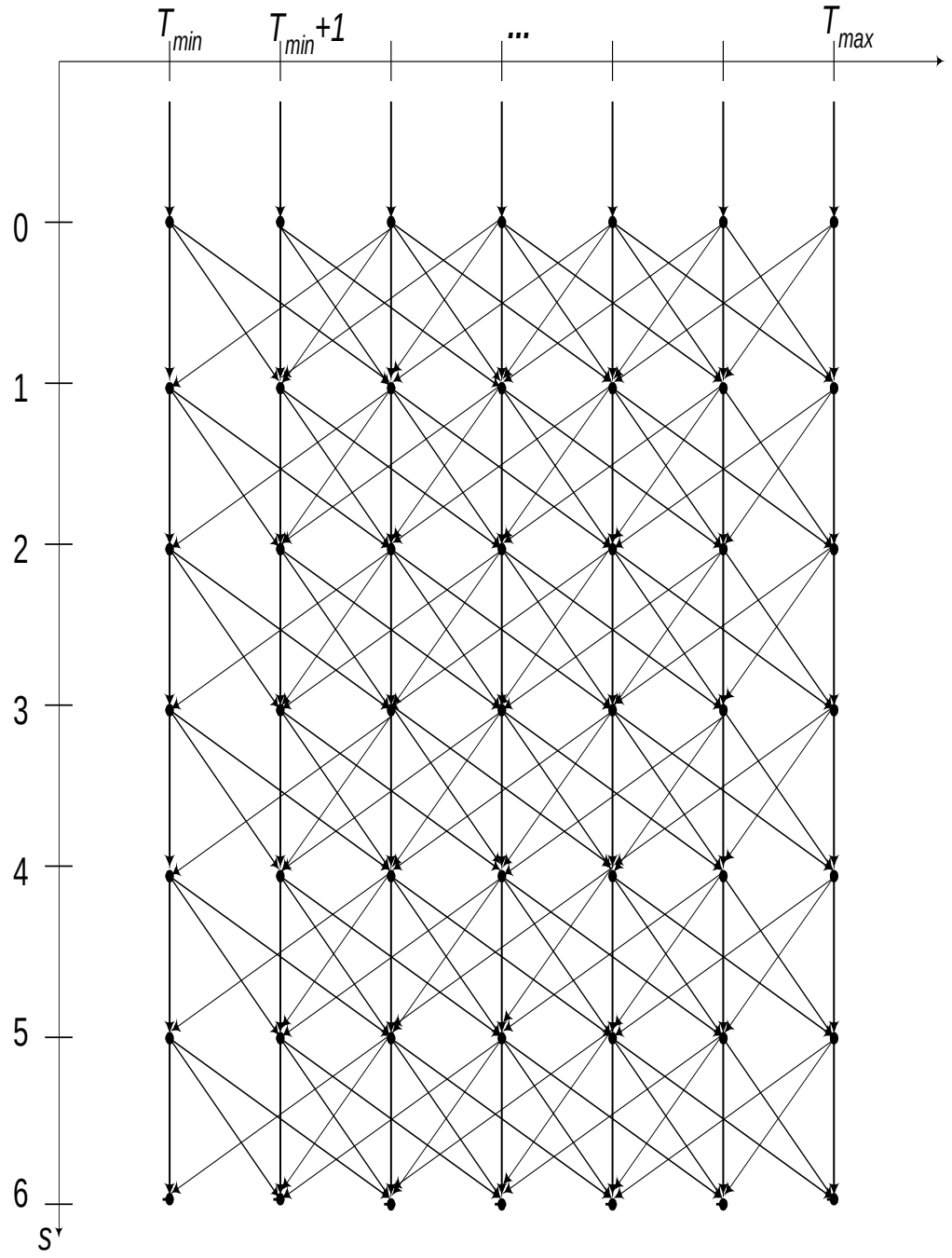

Рис.4.3. Граф решения задачи о мгновенном периоде основного тона

Этот граф можно представить в виде числового графа с той же самой порождающей функцией, что и в предыдущем примере, то есть $F(x_i, x_j) = x_j - x_i$, но с другим множеством образующих. Этот граф указан на рис. 4.4.

Здесь на рисунке сохраняются все стрелки, что и на основном рисунке, но они для удобства лишь обозначены. Граф представляется в виде $n = 62$, то есть $X = \{1,2,\ldots,62\}$, $F(x_i, x_j) = x_j - x_i$, $g(x) \equiv 0 (\mathrm{mod}\ 9)$, $U = \{-17,-8,1,10,19\}$, пос-

кольку $F(x_i, x_j)$ – функция несимметричная, то граф ориентирован.

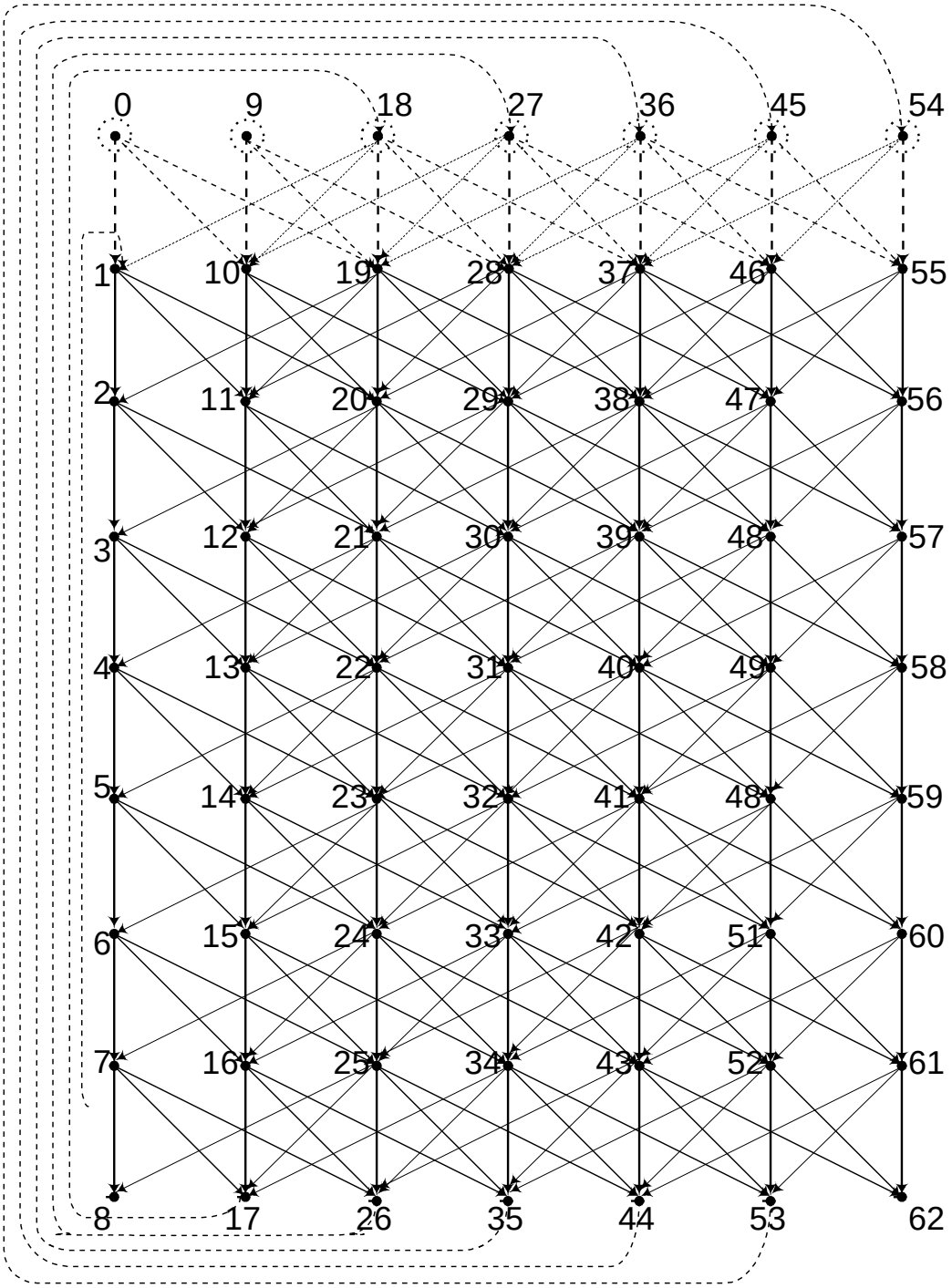

Рис.4.4. Реализация предыдущего графа в виде числового

Если бы он не был ориентированным, то его можно было бы представить и другой функцией – $F_1(x_i, x_j) = | x_j - x_i - 1 |$, а $U = \{0,9,18\}$.

Рассмотрим еще один пример из этой же книги [16 рис. 8.1, стр. 159], где функция смежности не является элементарной. На нем нанесены пунктиром стрелки, которые соединяют вершины цело численной решетки по правилам:

а) вершину $(0,0)$ с вершинами $(1,i)$, $1 \le i \le 8$;

б) вершину (i, j) с вершинами $(i+1,k)$, $8 \ge k \ge j \ge 1$;

в) вершины $(8, j)$, с вершиной $(9,9)$.

Чтобы реализовать этот граф в виде числового графа, рассмотрим следующий граф из 80 вершин (рис.4.5), который представляет собой 10 колонок по 8 вершин в каждой. Нумерация вершин начинается с нуля до 79, идет снизу вверх.

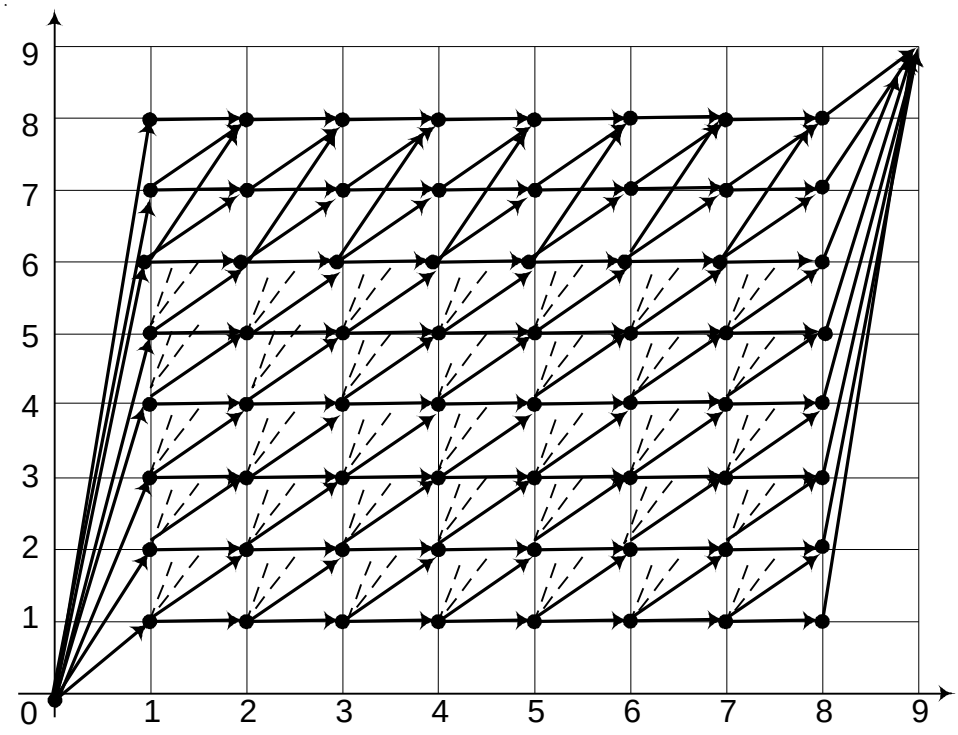

Рис. 4.5. Граф для оценивания оператора настройки S

Вершины {1,2,…,7} и {72,73,…,78} будут фиктивными. Вершина 0 соответствует вершине $(0,0)$ на рис. 4.5, а вершина 79 – вершине $(9,9)$. Остальные связи легко переносятся на новый граф. В результате получим граф на рис. 4.6.

Граф представляется следующим образом:

$n = 79$, , $g(x) = \{1,2,\ldots,7; 72,73,\ldots,78\}$, $U = \{8,9,10,11,12,13,14,15\}$.

Покажем, что любой подобный граф, у которого $n = st$, можно реализовать в виде числового графа со следующими параметрами:

$n = st - 1,\ X = \{0,1,\ldots, st-1\},\quad ,\ g(x) = \{1,2,\ldots, s-1; (s-1)t, (s-1)t+1,\ldots, st-1\},$

$U = \{s, s+1,\ldots, 2s-1\}.$

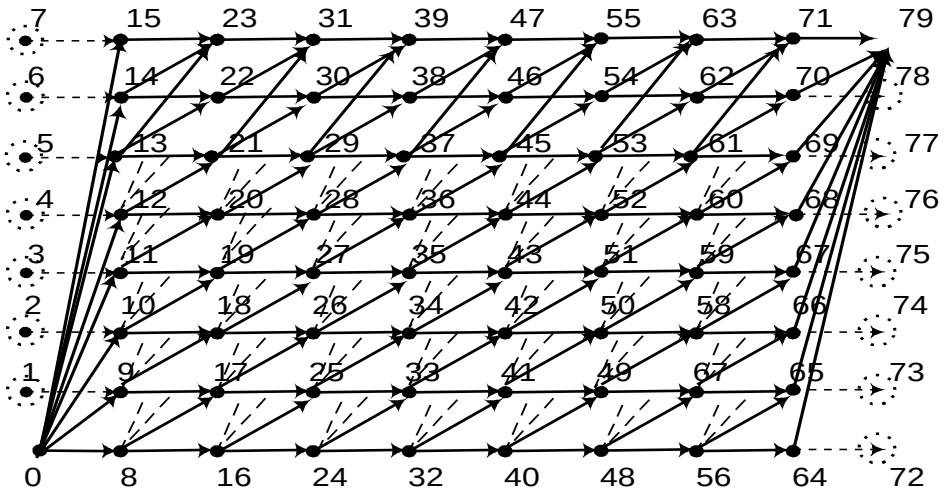

Рис. 4.6. Числовой граф, эквивалентный графу на рис.4.5

Действительно, U представляет собой образующие, которые связывают вершины двух соседних столбцов, минимальная разность кодов которых равна s. Номер каждого столбца, в котором находится вершина x_i, равен ks (номера начинаются с нуля). Поэтому, если найдутся две вершины x_i и x_j, разность которых принадлежит U, но они находятся не в соседних столбцах, то множитель k будет больше 1, и тогда $\left|F(x_j, x_i)\right| \geq 2s$, то есть вершины графа x_i и x_j будут несмежными.

Во многих монографиях, посвященных построению и исследованию математических моделей параллельных вычислительных систем, особенно успешным оказалось использование графов, в частности при описании графов систем, реализующих заданный алгоритм. В той или иной мере графы для этих целей применяются давно, однако их использование носило фрагментарный характер и не получило должного развития. Это происходило из-за того, что сравнение систем и алгоритмов на уровне графов неизбежно приводит к

необходимости исследовать графы на изоморфизм, гомоморфизм. Кроме того, многие задачи на графах являются *NP*-полными. Если алгоритм записывается произвольным графом, то отсюда возникают пессимистические выводы о невозможности решить задачу эффективными методами и необходимости привлекать для этого полный перебор вариантов.

В основе этих выводов лежит методологическая ошибка, связанная с предположением о произвольности графа алгоритма. На самом деле графы алгоритмов таковыми не являются, многие из них, как правило, обладают такими свойствами как симметричность, периодичность или подобие отдельных частей. В этом отношении числовые графы и способы их представления могут оказать пользу, идет ли речь об объеме компьютерной памяти, или быстродействии алгоритмов.

Рассмотрим в качестве примера элементарный решетчатый граф, который повсеместно используется в книге [17 рис.15.1,стр.153] (рис.4.7)

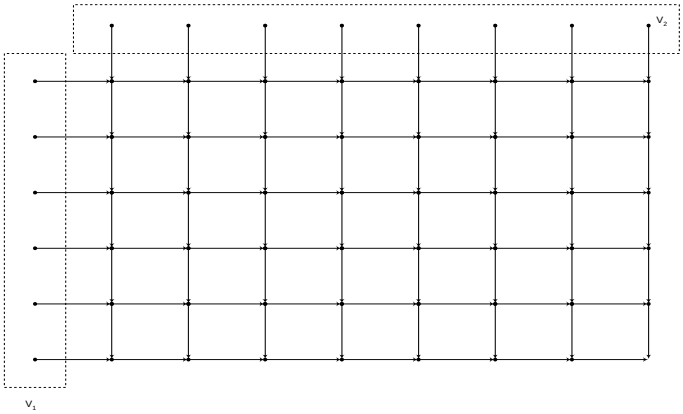

Рис. 4.7. Элементарный двумерный решетчатый граф

Очевидно, что такой граф легко реализовать с помощью *M*-графа, используя только две образующие. В частности можно использовать представление, показанные на рис.4.2, где подобный граф отличается от решетчатого наличием диагональных дуг. Существуют алгоритмы, графы которых можно задавать решетчатыми графами размерностью 3. К такому алгоритму может привести задача об умножении двух прямоугольных матриц. Такой алгоритм приведен в книге [17 на рис 6.2, стр.70].

На рис.4.8 приведена реализация такого графа в виде М-графа. Здесь $n = 71,\ \ X = \{0,1,2,...,71\},\quad g(x) \equiv 0(\mathrm{mod}\,4),\quad F(x_i, x_j) = x_i - x_j,\quad U = \{1,4,12\}$. Фиктивные вершины, которые составляют верхний слой, легко определяются свойством делимости на 4.

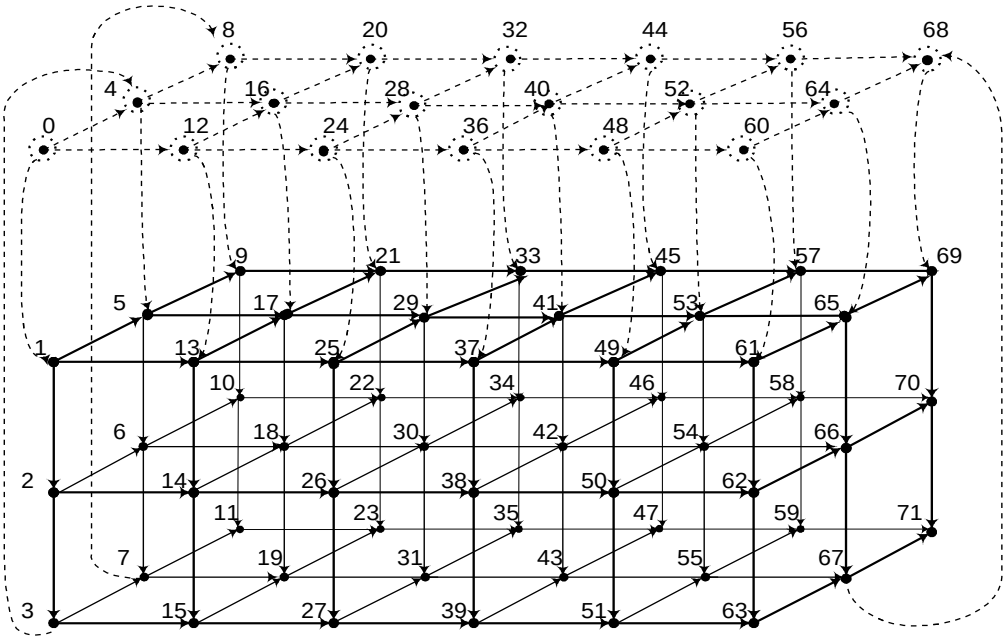

Рис. 4.8. Реализация трехмерного решетчатого графа

Одним из перспективных направлений повышения производительности и эффективности параллельных вычислительных систем являются их специализация на классе решаемых задач. Возрастающие требования к скорости решения современных задач и связанных с этим переход к новейшим технологиям сверхбольших интегральных схем (СБИС) привели к разработке специализированных локально связанных систем. В частности, это относится к систолическим массивам [73], обладающих высоким уровнем быстродействия, который достигается благодаря тому, что структура систолического массива адекватна структуре реализуемого им параллельного алгоритма и максимально учитывает особенности технологии СБИС [74].

Как пример реализации быстрого алгоритма для умножения матриц и синтеза новых архитектур систолических массивов, можно привести из [52,53], где используются трехмерные решетчатые графы.

Можно еще привести многочисленные примеры из книги [16], где используемые графы можно легко представить в виде числовых графов. Однако по мере усложнения структуры графов, это, как указывалось раньше, не всегда возможно. Существует более сложный способ представления числовых графов отличный от описанного раньше. В этом случае функция исключения имеет в качестве аргумента не вершины, а образующие. Обозначим ее $h(u)$. При этом $h(u_i)$ указывает те вершины, к которым нельзя применять данную образующую. В качестве иллюстрации рассмотрим рис.4.9, взятый из книги [17, рис.23.7, стр.236]. На этом рисунке дуги, идущие слева на право, вниз не всюду имеют место.

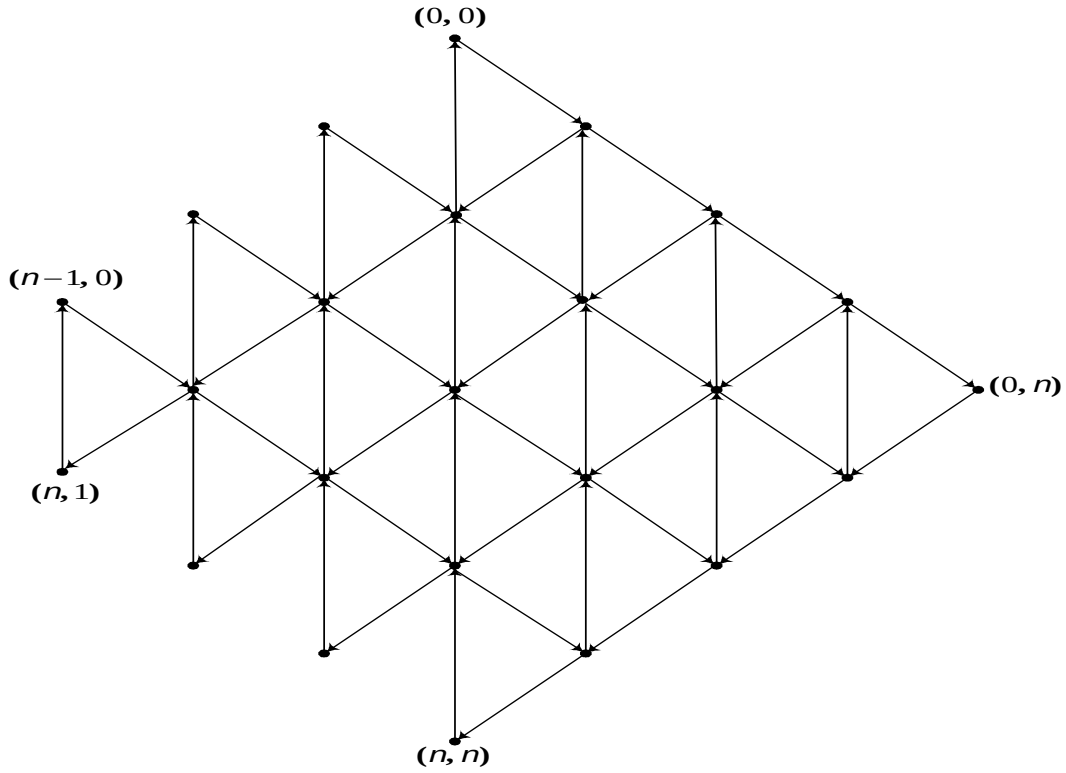

Рис. 4.9. Пример систолического массива

То же самое можно сказать и про дугу, идущую справа налево вниз. Если этим дугам поставить в соответствие образующие, то для некоторых вершин эти образующие не могут быть применены, хотя вершины и не фиктивные. В этом случае используем функцию g для указанных образующих. В результате этого получим числовой граф (рис.4.10) в виде некоторого решетчатого графа.

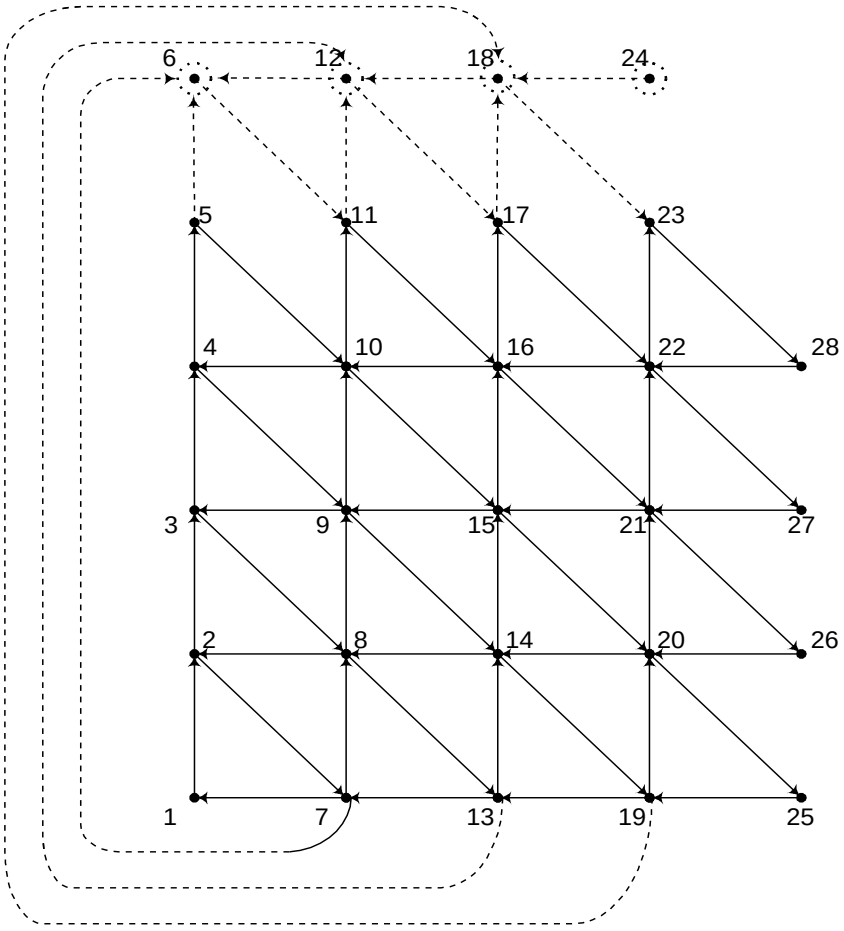

Рис. 4.10. Реализация графа на рис.4.9

Этот граф можно реализовать в виде М-графа с параметрами: $n = 28$, $X = \{1,2,...28\}$, $g(x) \equiv 0(\mathrm{mod}\,6)$, $F(x_i, x_j) = |x_i - x_j|$, $U = \{-6,1,5\}$, $h(1) = \{x > 25\}$, $h(-6) = \{x \equiv 5(\mathrm{mod}\,6)\}$. Здесь введена функция $h(u)$, которая применяется к двум образующим. Она запрещает проводить горизонтальные дуги на пятой горизонтали. Аналогично, запрещаются вертикальные дуги на последней вертикали.

Приведенные примеры показывают, что числовые графы можно успешно использовать в различных областях научных исследований, а также и в различных прикладных вопросах.

4.2. Оценка сложности базового алгоритма поиска в глубину на *NM*-графах

Понятие числового графа, введенное в [23], получило широкое применение. Все известные алгоритмы на графах построены с учетом

традиционных представлений графов. То же относится и к оценке трудоемкости этих алгоритмов. Поскольку представление графов в виде числовых графов отличается от известных представлений, возникает необходимость построить для них наиболее употребляемые алгоритмы и оценить их относительную сложность.

Опишем вкратце один из таких типичных алгоритмов, каким является поиск в глубину. Целью этого алгоритма является построение остовного леса для данного графа. Для этого рассмотрим следующее прохождение вершин неориентированного графа. Выбираем и посещаем начальную (произвольную) вершину графа x_0. Затем выбираем произвольное ребро (x_0, x_i) инцидентное x_0, и посещаем вершину x_i. Вообще, пусть x - последняя посещенная вершина. Выбираем какое-нибудь не рассмотренное ребро (x, y), инцидентное x. Если вершина y уже посещалась, ищем новое ребро, инцидентное x. Если y не посещалось, идем в вершину y и начинаем поиск с нее. В случае невозможности продвижения вперед возвращаемся к вершине x (к вершине, от которой мы пришли к y) и начинаем поиск с нее. В итоге вернемся к вершине x_0 и обнаружим невозможность продолжать от нее поиск. Ищем теперь новую, ранее не посещаемую вершину x_i и вновь ведем от нее поиск. Поиск закончится, если посетим таким образом все вершины графа.

Этот метод обхода называется поиском в глубину, поскольку процесс идет в направлении вперед (вглубь) до тех пор, пока это возможно.

Поиск в глубину в неориентированном графе $G = (X, U)$ разбивает ребра U на два множества T и B. Ребро $u_i = (x, y)$ помещается в T только в том случае, если в процессе поиска мы прошли по этому ребру. Подграф $G' = (X, T)$ представляет собой лес, который называется глубинным остовным лесом. Если лес состоит из единственного дерева, G' будем называть глубинным остовным деревом. В этом случае вершина x_0, с которой начинается обход, является корнем этого дерева.

Оценка трудоемкости этого алгоритма для натуральных арифметических графов (*NA*-графов) дана в [42]. В данном разделе приводится такая оценка для

натуральных модульных графов (*NM*-графов). Наиболее часто употребляемое представление графов осуществляется в виде списков смежностей.

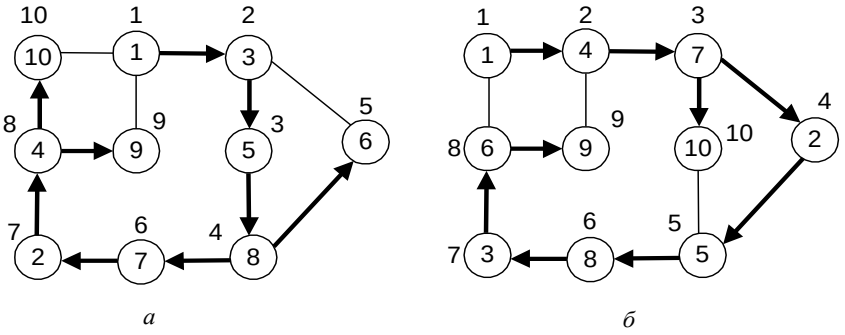

Рис. 4.11. Построение остовного дерева

В качестве примера рассмотрим граф на рис. 4.11, а, где номера вершин записаны внутри кружков. Заданный граф будет представлен массивом НАЧАЛО размерностью n и двумя массивами КОНЕЦ и СЛЕДУЮЩИЙ размерностью $2m$ (рис. 4.12.).

Вершины

	НАЧАЛО
1	1
2	4
3	6
4	9
5	12
6	14
7	16
8	18
9	21
10	23

Ребра

	КОНЕЦ	СЛЕДУЮЩИЙ
1	3	2
2	9	3
3	10	0
4	4	5
5	7	0
6	1	7
7	5	8
8	6	0
9	2	10
10	9	11
11	10	0
12	3	13
13	8	0
14	3	15
15	8	0
16	2	17
17	8	0
18	5	19
19	6	20
20	7	0
21	1	22
22	4	0
23	1	24
24	4	0

Рис. 4.12. Массивы ребер и вершин

В $i-$ячейке массива НАЧАЛО указан номер ячейки массива КОНЕЦ, с

которой начинается запись конечных вершин j для всех ребер (i, j). Конец записи отмечается нулем в ячейке массива СЛЕДУЮЩИЙ.

Теперь можно сформулировать задачу.

ПОИСК В ГЛУБИНУ на неориентированном графе.

Вход: граф $G = (X, U)$, представленный таблицами списка смежностей с помощью массивов НАЧАЛО, КОНЕЦ и СЛЕДУЮЩИЙ.

Выход: разбиение множества U на два множества T и B.

При работе алгоритма, реализующего решение этой задачи, будет использована рекурсивная процедура ПОИСК(x). Процедура ПОИСК(x) добавляет ребро (x, y) в T, если вершина y впервые достигнута во время прохождения по ребру из вершины x. Все вершины графа первоначально помечены как «новые».

ПРОЦЕДУРА ПОИСК(x)

1) пометить вершину *x* как "старую";

2) найти очередную вершину *y*, смежную с x;

3) если вершина y помечена как «старая», перейти к 2;

4) добавить ребро (x, y) к T;

5) перейти к ПОИСК(y);

6) если не все вершины, смежные с x, рассмотрены, перейти к 2;

7) конец.

Общий алгоритм, решающий задачу, выглядит так:

1) $T \Leftarrow \varnothing$;

2) пометить все вершины графа как «новые»;

3) найти вершину x, отмеченную как «новая»;

4) ПОИСК(x);

5) если не все вершины графа «старые», перейти к 3;

6) конец.

Это описание алгоритма пригодно для всех структур исходных данных. Конкретный алгоритм (алгоритм Р) расписан детальнее в [47] и состоит

из 24 шагов. Этот алгоритм по смысловой нагрузке можно разбить на три части. К первой части можно отнести шаги 1 – 7, где выполняется предварительная обработка. Вторая, основная, часть охватывает шаги 8 – 19, где осуществляется обход по вершинам графа. Третья часть алгоритма шаги 20 – 24 работает в том случае, если граф не связен. Тогда для каждой компоненты графа поиск (обход вершины графа) начинается заново.

После работы алгоритма массив T содержит ребра остовного дерева исходного графа, отмеченные на рис.4.11,а жирными линиями. Эти ребра имеют вид $w_i = (T[i], i)$, где $2 \le i \le n$, т. е.

$w_2 = (6,2)$, $w_3 = (1,3)$, $w_4 = (2,4)$,
$w_5 = (3,5)$, $w_6 = (8,6)$, $w_7 = (8,7)$, $w_8 = (5,8)$, $w_9 = (4,9)$, $w_{10} = (4,10)$. Стрелки на этих ребрах и порядковые номера вне кружков указывают порядок обхода вершин графа. К множеству B относятся оставшиеся ребра, а именно: $B = [(9,1), (10), (3,6)]$.

Рассмотрим теперь тот же алгоритм, но записанный для другой структуры исходных данных. Пусть задан тот же граф, но нумерация вершин будет такой как на рис. 4.11,б (номер вершины – в кружке). Его можно представить в виде *NM*-графа с множеством вершин $X = \{1,2,\ldots,9,10\}$ и множеством образующих $U = \{3,5\}$.

Поиск в глубину можно организовать различным способом, все зависит от порядка выбора смежных вершин. Будем полагать, что в данном алгоритме продвижение в глубину идет как можно дальше вдоль ребер, соответствующих первой образующей u_1.

Алгоритм Q. (Поиск в глубину). Натуральный модульный граф $G = (X, U)$ с n вершинами и p образующими задан массивом $U(2p)$, где $U = \{-5,-3,3,5\}$. Используются два рабочих массива $S(n)$ и $T(n)$; первый – для отметок вершин графа, второй – для получения ребер множества T. Используются четыре переменных r, r_1, r_2 и r_3. Как и в алгоритме P, предполагается, что $n \ge 1, \quad p \ge 1$. Алгоритм предназначен для поиска T - остовного леса заданного

графа. Первые части алгоритмов P в [47] и Q совпадают буквально, третья часть алгоритма Q (шаги 23-25) соответствует шагам 20-24 алгоритма P, поэтому приводится только основная часть Q. В r_1 первоначально находится номер первой вершины, $r_1 = 1, r_2 = p$.

8. Установить $r_3 = n$.

9. [Отмечается вершина как «старая». В качестве отметки будем пользоваться числом 1] $S[r_1] = 1$.

10. [Начальная установка] $i = 1$.

11. [Пропуск образующей, использованной на предыдущем шаге]. Если $i = T[i]$, то перейти к шагу 14.

12. [Определение смежной вершины] $r = u[i] + r_1$.

13. [Отрицательная вершина?] Если $r > 0$, то перейти к шагу 19.

14. [Продвижение] $i = i + 1$.

15. [Использованы все образующие?] Если $i \le r_2$, то вернуться к шагу 11.

В противном случае возвращаемся по ребру множества T в предыдущую вершину.

16. [Вернулись в начальную вершину?] Если $T[r_1] = 0$, что является признаком того, что вернулись в начальную вершину, от которой никуда нельзя продвинуться, то необходимо поискать «новую» начальную вершину. Для этого надо перейти к 23.

17. [Продолжение поиска предыдущей вершины] $i = T[r_1]$.

18. [Восстановление номера предыдущей вершины] Если $T[r_1] = 0$, то перейти к 25. Установить $r_1 = r_1 - u[i]$ и вернуться к шагу 14.

19. [Превышает ли номер вершины максимальный номер?] Если $r > r_3$, вернуться к шагу 18.

20. [Пропуск «старой» вершины]. Если $S[r] = 1$, то вернуться к шагу 14.

21. [Добавление ребра в T] $T[r] = i$.

22.[Начало поиска с «новой» вершины] Установить $r_1 = r$ и вернуться к 9.

23. [Продвижение] $i = i + 1$.

24. [Пропуск «старой» вершины]. Если $S[r] = 1$, то вернуться к шагу 14.

25. Конец.

После работы алгоритма Q массив T содержит информацию о ребрах остовного дерева исходного графа, отмеченных на рисунке жирными линиями. Эти ребра имеют вид $w_i = (i - u[T[i]], i)$ где $2 \le i \le n$. Для придания им вида, соответствующего виду массива T в алгоритме P, надо выполнить следующие операции.

01. [Начальная установка] Установить $i = 2$.

02. [Расшифровка первой вершины ребра] Установить $T[i] = i - u[T[i]]$.

03. [Продвижение] Увеличить i на 1.

04. [Конец массива T?] Если $i \le r_3$, вернуться к шагу 02.

05. Конец.

В результате получаем ребра вида $w_i = (T[i], i)$, т.е. $w_2 = (7,2)$, $w_3 = (8,3)$, $w_4 = (1,4)$, $w_5 = (2,5)$, $w_6 = (3,6)$, $w_7 = (4,7)$, $w_8 = (5,8)$, $w_9 = (6,9)$, $w_{10} = (7,10)$. Оставшиеся ребра принадлежат множеству $B = [(1,6), (4,9), (5,10)]$. Порядковые номера возле кружков и направление стрелок указывают порядок обхода вершин графа.

Попытаемся сделать сравнительную оценку двух описанных алгоритмов. Двумя важными мерами сложности алгоритмов являются временная и емкостная сложности, рассматриваемые как функции от размера исходных данных. Чтобы точно определить временную и емкостную сложности, надо указать время, необходимое для выполнения каждого шага алгоритма, и объем памяти, используемый для исходных данных и рабочих массивов. Существует несколько весовых критериев для оценки алгоритмов. Воспользуемся логарифмическим весовым критерием, описанным в [6], суть которого заключается в следующем. Пусть $e(i)$ – логарифмическая функция на целых числах, заданная равенствами для всех остальных i.

Логарифмический весовой критерий основан на допущении, что цена

выполнения операции (ее вес) пропорциональна длине ее операндов. В работе [47] подсчитано время поиска начального адреса массивов НАЧАЛО, КОНЕЦ, СЛЕДУЮЩИЙ, S, T, SS, U соответственно $e(M_1)$, $e(M_2)$, $e(M_3)$, $e(M_4)$, $e(M_5)$, $e(M_6)$ и $e(M_7)$, которые являются постоянными величинами.

Чтобы подсчитать суммарное время, необходимо оценить значения, которые принимают параметр i, и переменные r, r_1 и r_2. Оказывается, что их среднее значение равно $\frac{n+1}{2}$. Тогда можно с очень малой погрешностью принять $e(i)=e(r)=e(r_1)=e(r_2)=e(r_3)=e(n)-1$. Подставляя эти значения в таблицу 4.1 и суммируя, получим оценки для алгоритмов P и Q.

Таблица 4.1

Номер шага	Алгоритм P	Число выполнений	Алгоритм Q	Число выполнений
08	$e(M_3)+e(r_1)+1$	n	$e(p)$	1
09	$e(M_4)+e(r_1)$	n	$e(M_3)+e(r_1)+1$	n
10	$e(M_1)+e(r)$	n	1	$n-1$
11	$e(M_2)+e(r)$	m	$e(M_4)+e(i)+e(i)$	m
12	$e(M_4)+e(r_1)+1$	m	$e(M_7)+e(r_1)+e(i)$	m
13	$e(M_6)+e(r_1)+e(r)$	$n-1$	$e(r)+1$	m
14	$e(M_3)+e(r_2)+e(r_1)$	$n-1$	$e(i)+1$	$n-1$
15	$e(r_2)$	$n-1$	$e(i)+e(p)$	m
16	$e(r)+1$	m	$e(M_4)+e(r_1)+1$	$n-1$
17	$e(M_3)+e(r_1)+1$	$n-1$	$e(M_4)+e(r_1)+e(i)$	$n-1$
18	$e(M_3)+e(r_1)$	$n-1$	$e(M_7)+e(r_1)$	m
19	$e(M_6)+e(r_1)$	$n-1$	$e(n)+e(r)$	m
20	-	-	$e(M_3)+e(r)+1$	$n-1$
21	-	-	$e(M_4)+e(r)+e(i)$	$n-1$
22	-	-	$e(r)$	$n-1$

$$t_1=e(n)(11n+3m-8)+n\left[4e(M_3)+2e(M_4)+e(M_1)+e(M_2)+2e(M_6)-9\right]+$$
$$+m\left[e(M_4)+e(M_2\_-1\right]+7,$$
$$t_2=e(n)(9n+9m-8)+n\left[2e(M_3)+3e(M_4)-4\right]+m\left[e(M_4)+e(M_7)e(p)-7\right]+$$
$$+e(p)+4.$$

Теперь нетрудно подсчитать временную сложность для общих частей алгоритмов: $e(n)(9n-1)+n\left[3e(M_3)-1\right]+3$.

Окончательно получаем оценки временной сложности алгоритмов:

$$t(P)=e(n)(20n+3m-9)+n\left[7e(M_3)+2e(M_4)+e(M_2)+2e(M_6)-10\right]+$$
$$+m\left[e(M_4)+2e(M_2)-1\right]+10.$$

$$t(Q)=e(n)(18n+9m-9)+n\left[5e(M_3)+3e(M_4)-5\right]+m\left[e(M_4)+2e(M_7)e(p)-7\right]+$$
$$+e(p)+7.$$

Для подсчета емкостной сложности алгоритмов будем считать, что она равна сумме величин $e(x_i)$ по всем используемым элементам памяти, где x_i – наибольшее число, хранящееся в этой памяти. Очевидно, что это число равно n. Таким образом, для алгоритма P требуется память объемом $n+4m+n+n+n=4(n+m)$ ячеек. Это дает оценку $V(P)=4(n+m)e(n)$. Для алгоритма Q требуется память объемом $2n+p$, то есть $V(Q)=e(n)(2n+p)$.

Можно сделать вывод, что в вычислительном смысле алгоритм Q немного лучше алгоритма P, но по емкостной оценке он намного превышает алгоритм P, потому что $p\approx\frac{4m}{n}$.

Мы сделали оценку алгоритма Q для поиска в глубину на *NM*-графах, однако эта оценка справедлива и для *M*-графов. Это видно из того, что алгоритм Q работает и для *M*-графов, если в массиве S вначале пометить номера несуществующих вершин как «старые».

Различие структур данных приводит к различным оценкам трудоемкости и других алгоритмов. В алгоритмах на графах наиболее часто встречаются следующие операции, которые приводят к различным оценкам.

P_1 – для вершины x найти очередную смежную с ней вершину.

Q_1 – проверить, являются ли вершины x и y смежными.

Оценим эти операции для первой структуры данных.

Поиск начала массивов для алгоритма P_1 имеет вес

$$\left[e(M_4)+e(x)\right]+\left[e(M_5)+e(r_1)\right].$$

Если необходимо определить очередную вершину, смежную с x, то вес этих операций равен $\left[e(M_5)+e(r)\right]+\left[e(r_2)+e(r)\right]$.

Для второй структуры поиск первой смежной вершины достигается путем поиска такого числа u_i, что $u_i + x > 0$. Отсюда вес алгоритма P равен

$$1+\frac{p+1}{2}\left[e(i)+e(U[i])+e(r)+e(x)+e(M_1)+2\right].$$

Оценим теперь операцию Q_1 для двух случаев. Для первой структуры данных каждый шаг в среднем повторяется $\frac{n+1}{2}$ раз, поэтому общий вес алгоритма Q_1 равен

$$1+\frac{n+1}{2}\left[e(M_3)+e(M_4)+e(M_5)+2e(r)+2e(r_1)+e(x)+e(y)+1\right].$$

Для структуры М-графа вес алгоритма Q_1 с учетом того, что шаги повторяются в среднем $\frac{p+1}{2}$ раз, будет равен

$$1+e(x+y)+\frac{p+1}{2}\left[e(M_1)+e(U[i])+2e(i)+e(i+1)+e(p)\right].$$

Для окончательной оценки веса алгоритмов учтем, что средние значения величин x, y, r, r_1 и $U[i]$ равны $\frac{n+1}{2}$, а переменной i – $\frac{p+1}{2}$. Подставляя эти числа, получим $e(x)=e(y)=e(r_1)=e(r)=e(U[i])\approx \log_2 n$.

В формуле определения весов алгоритмов есть множитель $\frac{p+1}{2}$, который получается как среднее число шагов при последовательном поиске. Однако, ввиду упорядоченности множества U, можно организовать поиск со средним числом шагов, равным $\log_2 p$.

При сравнении двух структур данных можно заметить, что веса $t(P_1)$ для двух структур данных одного порядка. Если же p зависит от n, то P_1 несколько лучше для традиционных структур данных. Однако Q_1 значительно лучше для *NM*-графов.. Даже полагая, что $p\approx O(\sqrt{n})$, получим порядки трудоемкости

алгоритма Q_1 соответственно $O(n\log_2 n)$ и $O(\log_2^2 n)$), т.е. для *NM*-графов выигрыш значительный. Для подсчета емкостной сложности алгоритмов будем считать, что она равна сумме величин $e(x_i)$ по всем используемым элементам памяти, где x_i – наибольшее число, хранимое в этой памяти. Очевидно, что это число равно n. Таким образом, для алгоритма поиска в глубину для первой структуры данных требуется память объемом $n+4m+3n=4(n+m)$ ячеек. Это дает оценку $V_1 = 4(n+m)\log_2 n$.

Для второй структуры данных требуется память объемом $2n+p$, то есть

$$V_2 = (2n+p)\log_2 n.$$

Можно сделать вывод о том, что в вычислительном смысле представление графа в виде *NM*-графов по всем параметрам дает несомненное преимущество по сравнению с традиционными способами представления. Этот вывод справедлив и для *M*-графов.

Здесь показано преимущество представления графов в виде *M*-графов для алгоритмов, разработанных и применяемых на графах с традиционным представлением в виде списков смежностей. Однако можно построить такие же алгоритмы специально для *M*-графов, которые имеют еще больше преимуществ по сравнению с рассмотренными.

4.3. Оптимальный алгоритм поиска в глубину на *NM*-графах с двумя образующими

Основное преимущество числовых графов по сравнению с обычными состоит в том, что для них проблема размещения данных решается значительно проще, особенно это относится к натуральным графам, множество вершин которых можно задавать одним числом. Как было доказано раньше [85], в силу различных законов изоморфизма для произвольных числовых графов можно так модифицировать множество их вершин, что оно будет представлено как натуральный ряд за исключением некоторых чисел. Однако, множество образующих в общем случае может иметь произвольный объем. На практике

оно значительно меньше множества вершин, и это дает большое преимущество по сравнению с обычным представлением графов в виде списков смежностей, где объем памяти пропорционален сумме чисел вершин и ребер. Преимущество это проявляется при разработке на графах различных алгоритмов. Можно привести сравнение типичных алгоритмов, записанных для графов, представленных в виде списков смежностей и в виде числовых графов. Такими типичными алгоритмами являются алгоритмы, в основе которых лежит систематический просмотр всех вершин графа, при этом каждая вершина просматривается ровно один раз. Такими алгоритмами, которые являются базовыми для конструкции более сложных алгоритмов, являются алгоритмы поиска в глубину и поиска в ширину. Первое такое сравнение для арифметических графов алгоритма поиска в глубину было проведено в [42], где и подтвердилось преимущество числовых графов. Затем [47] была проделана аналогичная работа для модульных графов и с тем же результатом. В обеих работах алгоритм был записан в том виде, в каком он был раньше разработан для обычных графов, то есть преимущество числовых графов было чисто техническим за счет удобной записи исходных данных и меньшего времени работы алгоритма. В последней работе была высказана гипотеза о том, что многие алгоритмы для числовых графов не нужно строить в привычном понимании, а можно благодаря специфической структуре данных выдать готовое решение поставленной задачи. Тогда сложность алгоритма будет оцениваться только временем вычисления некоторых простых параметров и записи ответа. Здесь этот результат излагается для алгоритма поиска в глубину на натуральных модульных графах.

Общая идея алгоритма для связного графа состоит в том, что, начиная с какой-либо фиксированной вершины, по ребру переходим на новую вершину и от нее продолжаем поиск новых вершин. Если это невозможно, то возвращаемся пройденным путем до тех пор, пока появится возможность перейти на новую вершину. Процесс заканчивается, когда окажемся в исходной вершине и нет возможности продолжить поиск. В результате работы алгоритма выдается

список пар вершин исходного графа, которые в совокупности представляют ориентированное остовное дерево, корнем которого является начальная вершина поиска.

Таким образом, поиск в глубину в неориентированном графе $G=(X,U)$ разбивает ребра на два множества T и B. Ребро r помещается в T в том и только в том случае, если в процессе поиска мы прошли по этому ребру. Подграф $H=(X,T)$ называется глубинным остовным деревом, поэтому алгоритм заканчивается построением множества T, состоящего из $n-1$ ориентированных ребер вида $r[i]=(x_i, y_i)$.

Рассмотрим произвольный связный NM-граф. Известно [87], что связность его обеспечивает наименьшее количество образующих $(u_{i_1}, u_{i_2}, \ldots, u_{i_k}), (k \le m)$, наибольший общий делитель которых есть 1.

Рассмотрим сначала случай $k=2$. Пусть это образующие u_1 и u_2, где $u_1 \le u_2$, и они взаимно просты. Подграф $G'=(X,Y')$, где $U=\{u_1,u_2\}$, является связным графом, степени вершин которого не больше 4. Поэтому в алгоритме поиск новых вершин не всегда имеет альтернативу, что упрощает решение задачи.

В процессе поиска ребра остовного дерева будем помещать вершины в двумерный массив T. Номер первого элемента этого массива будем называть текущим номером, а значение второго элемента – текущим кодом вершины. Кроме того, используем массив S объемом u_1.

Вычислим параметры $s=u_1+u_2$, $p=\left\lfloor \frac{n}{s} \right\rfloor$, $k=\left\lfloor \frac{u_2}{u_1} \right\rfloor$ и положительный вычет $\Delta=-u_2 (\mathrm{mod}\, u_1)$. Назовем шагом вперед сложение текущего кода вершины с образующей u_1 и шагом назад вычитание из текущего кода вершины образующей u_2. Рассмотрим алгоритм Q, который состоит из такой последовательности операций.

1. [Начальная установка.] В качестве текущего кода установим $i=1$, текущий код вершины установим $l=1,\ j=1,\ v(1)=1.$

2. [Прямой шаг.] Вычислить $r=v(i)+u_1.$

3. [Заполнение массива T .] Установить $T[i]=\big(v(i),r\big),\ i=i+1,\quad v(i)=r$.

4. [Сравнить.] Если $r\le u_2$, то перейти к шагу 2.

5. [Запомнить.] Установить $S\big[j\big]=r,\ j=j+1.$

6. [Обратный шаг.] Вычислить $r=v(i)-u_2$. Если $r>1,$ перейти к шагу 3.

7. Конец.

Лемма 4.1. В результате работы алгоритма Q получится цикл Z, в котором перечисляются все вершины интервала $[1,2,...,u_1+u_2]$.

Доказательство. Цикл получится ввиду того, что код текущей вершины в конце алгоритма становится равным 1. Так как u_1 и u_2 взаимно просты, то это относится и к Δ с u_1. По условию шага 2 код вершины не должен быть больше u_1+u_2. В начале алгоритма идет перечисление всех кодов равных $1(\mathrm{mod}\,u)$. Затем идет переход (после шага назад) к множеству кодов $(1+\Delta)(\mathrm{mod}\,u_1)$ и так далее. Так как Δ и u_1 взаимно просты, то линейная форма $(1+c\Delta)(\mathrm{mod}\,u_1)$ при подстановке в нее вместо c положительных вычетов $0,1,...,\ u_1-1$ также пробегает все эти вычеты. В сумме это и дает весь интервал $[1,2,...,u_1+u_2]$.

Лемма 4.2. Последовательность $[1,2,...,u_1+u_2]$ в цикле Z занимает места, номера которых составляют арифметическую прогрессию по $\mathrm{mod}(u_1+u_2)$ с началом $a=1$ и разницей $d>k$.

Доказательство. Поскольку обход графа начинается с вершины 1, то к вершине 2 мы придем через a прямых и b обратных шагов, тогда a и b являются решением диофантова уравнения

$$au_1-bu_2=1 \qquad (4.1)$$

Так как u_1 и u_2 взаимно просты, то решение этого уравнения существует.

Чтобы найти его, можно воспользоваться представлением дроби $\frac{u_2}{u_1}$ в виде конечной непрерывной дроби [78]. В результате решения получим $d = a + b$. Очевидно, $d > k$, что определяется самим алгоритмом Q, где сначала делается не меньше k прямых шагов и один обратный. Сделав d шагов, придем в вершину 2. Ясно, что после $\alpha d\,(\alpha > 1)$ шагов придем в вершину $\alpha + 1$, если продвижение будет осуществляться по циклу Z. Если сделать $1 + u_1$ шагов, то окажемся во второй точке цикла. Это равносильно соотношению

$$du_1 = (u_1 + u_2)x + 1 \qquad (4.2)$$

Если подставить $d = a + b$ и $x = b$, то получим уравнение (4.1), что доказывает справедливость леммы 4.2.

В результате работы алгоритма Q мы придем в вершину $1 + u_2$. Если теперь к ее коду прибавить образующую u_1, то получим вершину $1 + u_1 + u_2$ которая является следующей после наибольшей вершины в цикле Z. Согласно лемме 4.1, применяя алгоритм Q, получим обход всех вершин в интервале $[1 + u_1 + u_2, \ldots, 2u_1 + 2u_2]$. Этот алгоритм можно применять p раз и получить обход всех вершин графа от 1 до $p(u_1 + u_2)$. Определить p можно из условия $p(u_1 + u_2) < n$, откуда $p = \left\lfloor \frac{n}{u_1 + u_2} \right\rfloor$. Для того, чтобы использовать это свойство, заменим шаги 3 и 7 алгоритма Q на последовательность операций, которая организует соответствующий цикл.

Покажем на примере (см. рис. 4.3) как работает алгоритм. Здесь $n = 51, U = \{4, 17\}$. Вычислим $s = u_1 + u_2 = 21$. Отсюда $p = \left\lfloor \frac{51}{21} \right\rfloor = 2$. За первый цикл алгорим проходит цепь, в которой перечисляются вершины от 1 до 21. Во втором цикле перечисляются вершины от 22 до 42. На рис. 4.13 эти два множества вершин определяют разрезы (а) и (б).

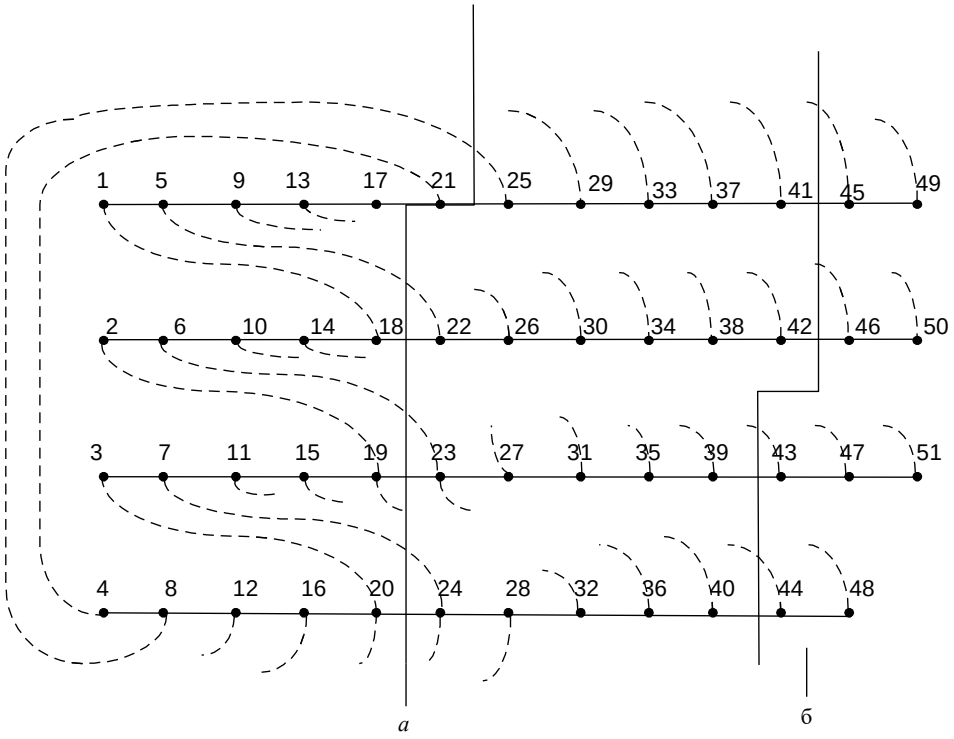

Рис. 4.13. *NM*-граф с $n = 51, \quad U = \{4, 17\}$

В результате работы алгоритма Q и двух указанных замен получим в массиве T цепь из $p(u_1 + u_2)$ ориентированных ребер с начальной вершиной 1 и конечной $p(u_1 + u_2) - u_1 + 1$. Если номер последней вершины меньше n, то остаются еще непосещенные вершины.

Чтобы их посетить, применим новый алгоритм R, описанный ниже и результат работы которого отражен на рис.4.14.

1. [Начальная установка.] Установить $i = 1, \; j = ps$.

2. [Найти начало ветви дерева.] Вычислить $v = S[i] + k$.

3. [Прямой шаг.] Установить $i = i + 1$, вычислить $r = v + u_1$.

4. [Сравнить.] Если $r < n$, то перейти к шагу 6.

5. [Текущие установки.] Положить $T[j] = (v, r), v = r, j = j + 1$ и перейти к шагу 3.

6. [Сравнить.] Если $i < u_1$, то перейти к шагу 2.

7. Конец.

О корректности алгоритма утверждает

Теорема 4.1. Модульный граф, у которого существуют две образующие, обеспечивающие его связность, алгоритм Q с расширениями и алгоритм R выдают решение задачи о поиске в глубину.

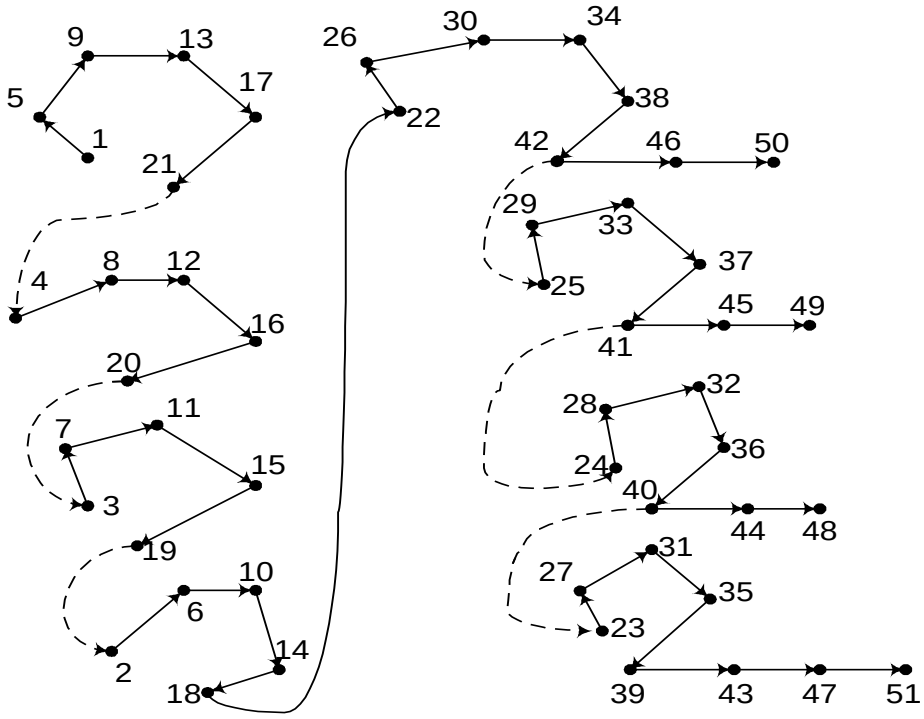

Рис.4.14. Порядок обхода вершин при поиске в глубину

Если модульный граф содержит l образующих $(l > 2)$, обеспечивающих его связность, то для него решение задачи поиска в глубину достигается немного сложнее, но почти с тем же объемом вычислительных операций.

4.4. Оптимальный алгоритм поиска в глубину для произвольных *NM*-графов

Обычно, при построении алгоритма поиска в глубину довольствуются предъявлением двух множеств ребер – множества T, представляющего остовное дерево, и множества B, представляющего остальные ребра графа. Сам алгоритм, в основном, зависит от способа выбора смежных вершин на каждом шаге.

Алгоритм, описанный в разделе 4.3, определяется тем, что доминирующим предпочтением при таком выборе являлась первая образующая *NM*-графа из двух возможных.

Чтобы воспользоваться полученными результатами, необходимо еще составить порядок обхода вершин *NM*-графа. Для этого изобразим дерево, соответствующее множеству T (рис. 4.15).

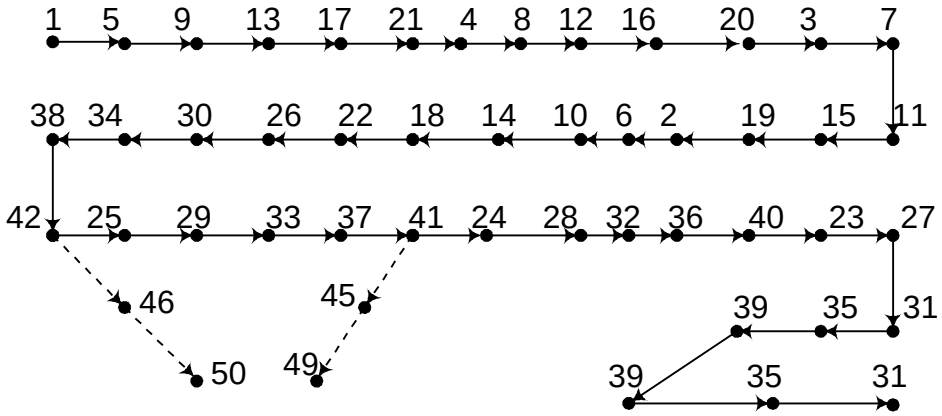

Рис. 4.15. Дерево обхода графа после поиска в глубину

В алгоритме Q из предыдущего раздела в массиве S сохраняются номера тех вершин, в которых дерево T разветвляется. В нашем примере это вершины (в порядке обхода) 42, 41, 40, 39. Обозначим их $S(i)$, $i = 1,2,3,\ldots,u_1$. Тогда от вершины $S(i)$ идет ветвь длиною $\lambda_i = \left\lfloor \dfrac{n - S(i)}{u_1} \right\rfloor$, заканчивающаяся висячей вершиной $S(i) + \lambda u_1$. Чтобы продолжить обход, необходимо вернуться по этой ветви в вершину $S(i)$. Все вершины цепи обозначены первыми в массиве ребер T. Таким образом весь обход представляется последовательностью соответствующих вершин, если после вершины $S(i)$ вставить вершины ветви в прямом и обратном порядке, за исключением последней ветви, где заканчивается обход.

В нашем примере она выглядит следующим образом:

$L = [1,5,9,13,17,21,4,8,12,16,20,3,7,11,15,19,2,6,10,14,18,22,26,30,34,38,42,$
$(46,50,46,42),25,29,33,37,41,(45,49,45,41),24,28,32,36,40,(44,48,44,40),$
$23,27,31,35,39,43,47,51]$

Каждый раз при построении алгоритма поиска в глубину на любом графе возникает дилемма: либо легко представить остовое дерево T, которое, может быть, неудобно обходить, либо очень сложно представить такое дерево, но

которое очень легко обходить. В последнем случае самым удобным деревом является гамильтонова цепь. Но вопрос о том, существует ли в *NA*-графе гамильтонова цепь, пока остается открытым. Вопрос об уменьшении числа вершин, принадлежащих ветвям в дереве T, полученным с помощью алгоритма Q в предыдущем пункте, можно частично решить с помощью такого приема. Увеличим на единицу длину каждой цепи, полученной благодаря применению только образующей u_1. На рисунке 4.14 первая цепь заканчивается в 21. Продлим ее до 25. Следующая цепь в том же столбце будет начинаться не с вершины 4, а с вершины 8 $(25-17=8)$, а заканчиваться вершиной 24. Точно также будут пропущены вершины 3 и 2. Они будут висячими по отношению к основной цепи. На рисунке 4.13 это будет отмечено переносом разреза *а* на один интервал правее. Если проделать ту же операцию с аналогичными цепями во втором столбце рисунка 4.14, то получим висячие вершины 26, 27 и 28, а разрез *б* передвинется на два интервала правее. И останется еще одна висячая вершина - 51. Всего получим 7 вершин, не принадлежащих основной цепи, что на две меньше, чем на рис. 4.15.

Таким образом, если алгоритм Q дает решение в виде основной цепи $p(u_1+u_2)$ и $n-p(u_1+u_2)$ вершин на ответвлениях, то с помощью описанного приема можно получить основную цепь с $p(u_1+u_2+1)$ вершинами и на p вершин меньше на ответвлениях. Последовательность L преобразуется в $L_1=[1, 5, 9, 13, 17, 21, 25, 8, (4,8), 12, 16, 20, 24, 7, (3,7), 11, 15, 19, 23, 6, (2,6), 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 33, (29,33), 37, 41, 45, 49, 32, (28, 32), 36, 40, 44, 48, 31, (27, 31), 35, 39, 43, 47, 51]$.

Самый простой способ построения остовного дерева T определяет следующая

Теорема 4.2. Следующий набор ребер *NM*-графа с двумя образующими определяет остовное дерево: $[i+(\lambda-1)u_1, i+\lambda u_1]$, где , а также ребра $(i, i+u_2)$ для $i=1,2,\ldots,u_1-1$.

Действительно, для каждого фиксированного i эта последовательность

определяет цепь с началом в вершине i и которая заканчивается максимально большим номером. Последние $u_1 - 1$ ребер связывают эти цепи в общее остовное дерево. Очевидно, что образующие u_2 не в этих ребрах не образуют цикл, иначе существовало бы ненулевое решение уравнения

$$au_2 \equiv 0(\mathrm{mod}\, u_1),\ 0 < a \le u_1 - 1. \qquad (4.3)$$

Но это возможно тогда и только тогда, когда u_1 и u_2 имеют общий множитель, что противоречит начальным условиям.

Для этого дерева легко организовать обход по всем вершинам, хотя в нем проходы вперед и возвраты занимают больше вершин, чем в предыдущих решениях. Для этого обозначим $\alpha_i = (\alpha_{i-1} + u_2)(\mathrm{mod}\, u_1) > 0$, $\alpha_0 = 1$, $(i = 1,2,\ldots,u_1 - 1)$, а $\beta_i = \max\limits_k \{\alpha_i + ku_1\} \le n$. Тогда последовательность обхода вершин *NA*-графа будет схематически выглядеть следующей:

$$L_3 = \beta_0 \to 1 \to \bigcup_{i=1}^{u_1-1} [\alpha_{i-1} + u_2 \to \beta_i \to \alpha_i]. \qquad (4.4)$$

Здесь стрелками обозначен путь вдоль цепи с помощью образующей u_1. Для примера (рис. 4.13) вычислим параметры

$\beta_0 = 49, \alpha_1 \equiv (1+17)(\mathrm{mod}\, 4) = 2,$
$\beta_1 = 50, \alpha_2 = 2 + 17 \equiv 3(\mathrm{mod}\, 4), \beta_2 = 51, \alpha_3 = 3 + 17 \equiv 4(\mathrm{mod}\, 4)$. В результате обход всех вершин будет осуществляться по маршруту

$L_4 = [$ 49, 45, 41, 37, 33, 29, 25,21, 17, 13, 9, 5, 1, 18, (22, 26, 30, 34, 38, 42, 46, 50, 42, 38, 34, 30, 26, 22,18), 14, 10, 6, 2, 19, (23, 27, 31, 35, 39, 43, 47, 51, 47,43, 35, 31, 27, 23, 19), 15, 11, 7, 3, 20, (24, 28, 32, 36, 40, 44, 48, 44, 36, 28, 24, 20), 16, 12, 8, 4].

Теперь, когда получены два способа построения остовных деревьев и, соответственно, два способа обхода всех вершин *NM*-графа, можно перейти к построению таких деревьев и их обходов для *NM*-графов с произвольным числом образующих, большим двух.

Пусть задан связный натуральный модульный граф, у которого $U = \{u_1, u_2, \ldots, u_m\}$, из которых l – максимальное число образующих,

обеспечивающих связность. Рассмотрим подграф G' на подмножестве этих образующих, которые перенумеруем $U' = \{u'_1, u'_2, \ldots u'_l\}$. По теореме 3.3 граф G' также будет связен и для него справедливо

$$\begin{aligned} &\text{а) НОД}(u_1, u_2, \ldots, u_l) = 1; \\ &\text{б) НОД}(u_{i_1}, u_{i_2}, \ldots, u_{i_k}) = d > 1;\ k < l. \end{aligned} \qquad (4.5)$$

Возьмем наименьшие образующие u_1 и u_2. Обозначим $\text{НОД}(u_1, u_2) = d(1,2) > 1$. Это означает, что подграф на множестве образующих $\{u_1, u_2\}$ будет состоять из $d(1,2)$ компонент. Построим на каждой из этих компонент остовное дерево по алгоритму Q, то есть по типу как на рис. 4.15. Каждое $i-$е дерево $(i = 1,2,\ldots,d(1,2))$ содержит все вершины $i[\mathrm{mod}\, d(1,2)]$, и каждое такое дерево содержит $\dfrac{u_1}{d(1,2)}$ поддеревьев, образованных одной образующей u_1. Добавление третьей образующей u_3 приведет к уменьшению компонент связности до числа, равного наибольшему общему делителю $\text{НОД}(u_1, u_2, u_3) = d(1,2,3)$. И в каждой такой компоненте будет $\dfrac{d(1,2)}{d(1,2,3)}$ поддеревьев, образованных с помощью образующих u_1 и u_2. Продолжая ввод новых образующих, будем все время уменьшать число компонент связности получаемого графа. На $j-$м шаге при вводе образующей u_j получим $d(1,2,\ldots,j)$ компонент связности, а в каждой из них по $\dfrac{d(1,2,\ldots,j-1)}{d(1,2,\ldots,j)}$ поддеревьев, образованных с помощью образующих $u_1, u_2, \ldots, u_{j-1}$. И наконец, при вводе образующей u_l получим связный граф.

Эти рассуждения помогут нам последовательно построить остовное дерево. Для этого докажем следующую теорему.

Теорема 4.3. Следующий набор из l типов ребер *NM*-графа, у которого l образующих, обеспечивающих связность, определяет остовное дерево:

1) $[i+(\lambda-1)u_1, i+\lambda u_1]$, $i=1,2,\ldots,u_1$; $\lambda=1,2,\ldots,\left\lfloor \frac{n-i}{u_1} \right\rfloor$;

2) $(i, i+u_2)$, $i=1,2,\ldots,u_1 - d(1,2)$;

3) $(i, i+u_3)$ $i=1,2,\ldots,d(1,2)-d(1,2,3)$;

……

l) $(i, i+u_l)$, $i=1,2,\ldots,d(1,2,\ldots,l)-1$.

Доказательство вытекает из способа построения остовного дерева, описанного выше.

4.5. Алгоритм раскраски натуральных модульных графов

Задача поиска хроматического числа графа для обычных графов является *NP*-полной [29]. Однако метод разностей, примененный для натурального модульного графа, позволяет решить эту задачу эффективно.

Как и в предыдущем разделе, разобьем весь алгоритм нахождения хроматического числа на процедуры и, представив алгоритмы процедур, определим вычислительную емкость всего алгоритма. Как видно из главы 2, известные сейчас методы раскраски графов либо обладают *NP*-полнотой, либо имеют другие сложности, что приводит к ограничению в их использовании. В предыдущих главах данной работы выполнены теоретические обоснования, позволяющие для натуральных модульных графов построить алгоритмы, имеющие полиномиальную сложность:

для алгоритма проверки изоморфизма графов $n^2 \log n$ при линейных затратах на память;

для алгоритма нахождения хроматического числа $n^2 m$ при затратах на память n^2.

Построим процедуру раскраски *NM*-графа методом разностей.

Пусть $G_n(U)$ – заданный *NM*-граф, который предстоит раскрасить набором из k красок. Процедура раскраски графа методом разностей содержит несколько этапов. Вначале необходимо выбрать код раскраски. Затем,

проверить совместимость кодов для каждого цвета. И наконец, выбрав код для каждого цвета, построить наборы одноцветных вершин, получив, таким образом, раскраску графа.

Код раскраски должен соответствовать свойству 3.1 и 3.3. Это значит, что раскраска кодами одинаковой длины позволяет определить число, разбиение которого может быть принято в качестве подходящего кода раскраски. Так, при попытке раскрасить граф k цветами, нам следует проверить коды, суммы элементов которых имеют вид $k, 2k, 3k, \ldots, rk$, где r – максимальная допустимая длина кода. В общем случае, величина r определяется как $\left\lfloor \frac{n-1}{k} \right\rfloor$, поскольку построить разбиение числа, превышающего n для кода раскраски не представляется возможным.

Определим алгоритм нахождения кода раскраски. Для этого, определим код раскраски длины 1. Проверим его на соответствие свойству 3.1. Если этот код подходит – переходим к следующему этапу. Иначе – строим следующие код длины 2 и повторяем все снова. Формально, все можно это можно записать так.

Алгоритм 4.1. Нахождение подходящего кода раскраски.

Шаг 0. На входе – количество цветов для раскраски k, количество вершин n, множество образующих U. Полагаем длину кода раскраски l равным 1.

Шаг 1. Определяем число – сумму кода раскраски как lk. Строим код раскраски Δ как разбиение числа lk на кортеж из l элементов.

Шаг 2. Если код соответствует свойству 3.1, выходим и возвращаем код Δ в качестве результата.

Шаг 3. Если есть еще возможность взять код длины l - переходим на шаг 1. Иначе – увеличиваем l на 1.

Шаг 4. Если $l > r$ выходим с пустым кодом – граф не раскрашиваем k цветами, иначе переходим на шаг 1.

Шаг 1 алгоритма 4.3 предполагает алгоритм разбиения числа на сумму длины l. Этот алгоритм нетривиален и требует в общем случае экспоненциального времени для получения всех возможных разбиений. Однако чтобы определить подходящее разбиение нет необходимости рассматривать все возможные коды раскрасок. Используя рекурсивную процедуру пошагового разбиения с проверкой на каждом шаге можно получить алгоритм со сложностью $O(l\log m)$ для нахождения разбиения длины l. Учитывая, что таких поисков в худшем случае может быть r, определим верхнюю оценку времени вычисления разбиения числа lk. Имеем $O(lk\log(lk)\log m)$. Благодаря полученной оценке можно определить вычислительную емкость раскраски графа $G_n(U)$ k цветами.

Алгоритм 4.1 имеет вычислительную емкость $O(nk^2u_m^3\log m)$

NM-графы могут быть связными и несвязными. Для связных графов и таких, которые распадаются на изоморфные связные компоненты можно применять метод разностей, поскольку в этом случае имеется граф, множество вершин которого необходимо разбить на подмножества несмежных вершин. Если граф распадается на изоморфные связные компоненты, то в таком случае можно применить метод разностей к графу-компоненте, а полученный код раскраски адаптировать к исходному графу.

Например, пусть исходный граф $G_{25}(5,10,20)$ требуется раскрасить четырьмя красками. В этом случае граф $G_5(1,2,4)$ представляет собой граф-компоненту исходного графа, в котором пять таких компонент. Этот граф легко окрашивается кодом $(3,5)$ и его циклическим сдвигом. Отметим, что для исходного графа данный код раскраски неприемлем. В то же время код $(15,25)$ соответствует требованиям свойств 3.1 и 3.3. Итак, для раскраски графа, состоящего из l изоморфных компонент, необходимо применить метод разностей для компоненты графа, а затем адаптировать код раскраски, умножив все элементы полученного кода раскраски на количество компонент такого графа.

Рассмотрим несвязный *NM*-граф другого вида, состоящего из нескольких неизоморфных связных компонент. В этом графе не хватает вершин для связности. Определив количество вершин такого графа таким, чтобы достичь связности, к этому графу можно применить метод разностей в стандартном виде. Полученная раскраска приемлема и для исходного графа, в котором вершины без ребер красятся цветом 1.

Для определения хроматического числа при помощи итеративного алгоритма очень важны нижние и верхние оценки для заданного натурального модульного графа. Определение нижней оценки является сравнительно простой задачей. Поскольку для *NM*-графа с нечетными образующими хроматическое число равно 2, то это число и является нижней оценкой для хроматического числа. Таким образом, возникает вопрос о верхней оценке хроматического числа для заданного *NM*-графа.

Самый простой способ, позволяющий определить наибольшее хроматическое число связано с наибольшей степенью графа. Для *NM*-графа, наибольшая степень в графе составляет $2m$, где $m = |U|$. Следовательно, $\lambda(G) = 2m+1$, если количество вершин в графе соответствует условию $2u^{\max} \le n$. То есть, мы получили правдоподобную оценку, не зависящую от количества вершин графа.

Вне всяких сомнений, полученная оценка является очень грубой, так как нет прямой зависимости между степенью вершин графа и его хроматическим числом. Так, несложно привести раскраску графа гораздо меньшим количеством цветов, чем максимальная оценка. На рисунке 4.16 приведен такой граф. Максимальная степень вершин этого графа составляет 6, а следовательно, оценка хроматического числа для него составляет 7. Однако, очевидно, что граф $G_{21}(3, 6, 8)$ имеет хроматическое число не превышающее 3. Значит, оценка $2m+1$ весьма далека от настоящего хроматического числа.

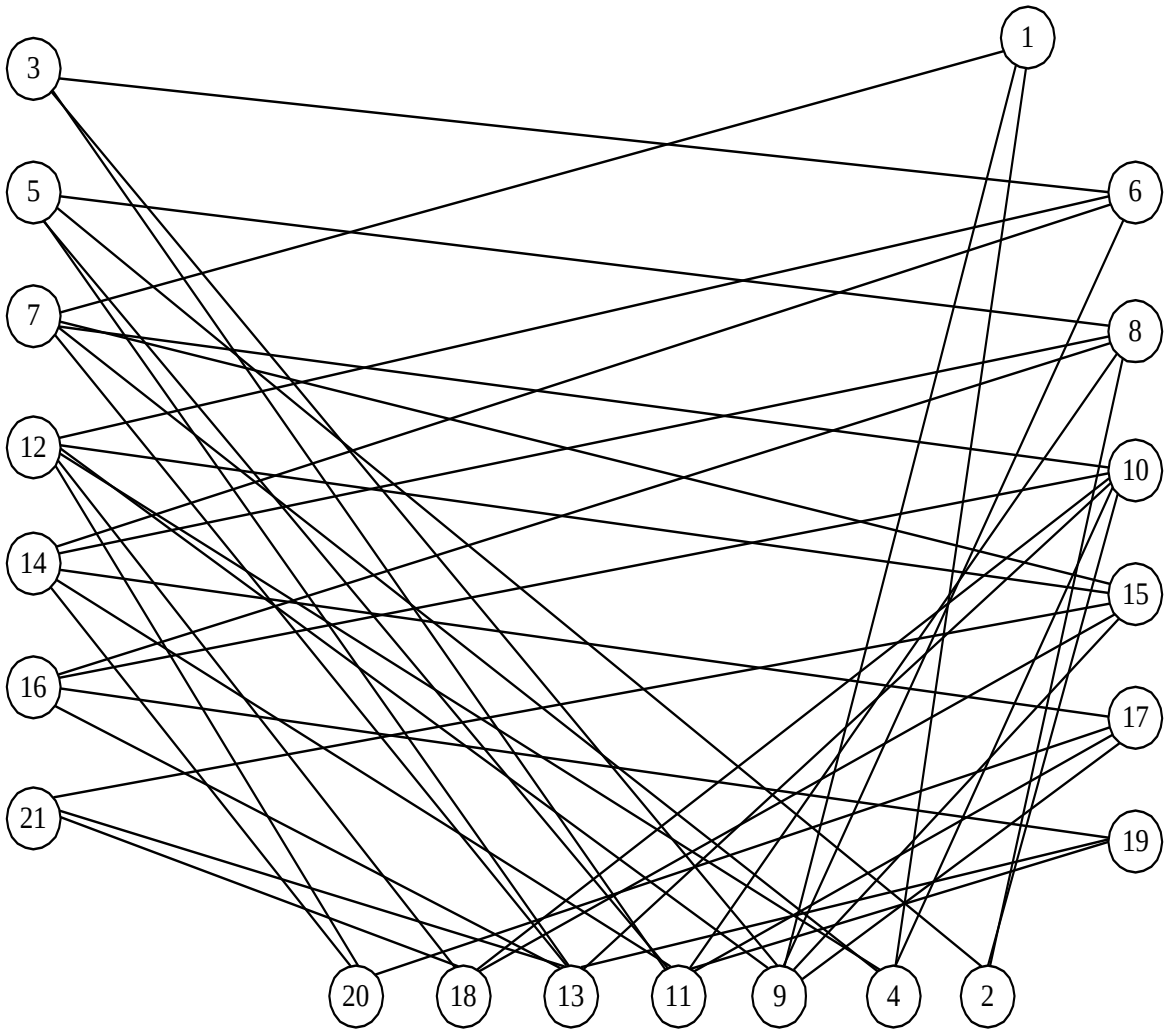

Рис. 4.16. Граф с $n = 21$ и $U = \{3, 6, 8\}$

Еще одна оценка хроматического числа связанная с образующими, основана на свойстве смежности вершин натурального модульного графа. Связность вершин исчерпывается множеством образующих, следовательно можно разбить множество вершин на некоторое число подмножеств, не являющееся делителем ни одной из образующих. Например, для графа, приведенного на предыдущем рисунке, это 4. Разбив множество вершин на четыре подмножества $(1,5,9,13,17,21)$, $(2,6,10,14,18)$, $(3,7,11,15,19)$ и $(4,8,12,16,20)$, мы получим раскраску, то есть вершины из подмножеств не имеют общих ребер. Такая оценка в ряде случаев бывает ближе к хроматическому числу, нежели $2m+1$. Выразим утверждение об оценке хроматического числа более формально.

Лемма 4.3. Хроматическое число *NM*-графа $\lambda(G)$ не превышает наименьшего натурального числа, не являющегося делителем ни одной образующей.

Доказательство. В самом деле, пусть ψ – число, не являющееся делителем ни одной образующей. В таком случае, мы можем раскрасить граф ψ цветами, окрашивая каждую вершину x цветом $x(\mathrm{mod}\,\psi)$. Очевидно, что одинаково раскрашенные вершины не будут смежными и такая раскраска состоятельна. Наименьшее число ψ и будет обеспечивать искомую оценку.

В некоторых случаях данная оценка будет весьма далека от хроматического числа, особенно для графов с небольшим количеством вершин и большими образующими. Так, например, для графа $G_{21}(5,6,8)$ оценка по $2m+1$ и оценка по лемме 4.3 совпадают и равны 7, а хроматическое число данного графа равно 3. В таких случаях можно воспользоваться методом определения оценки по наименьшей образующей. Раскраска в этом случае производится так. Первые u_1 вершин, а также последние $n-u_1$ вершин красятся в один цвет, исключая, естественно, вершины связанные наибольшей образующей $x > u_m + 1; x \le u_1$. Остальные вершины можно окрасить $\left\lfloor \frac{n}{u_1} \right\rfloor$ цветами. Таким образом, имеем оценку $\left\lfloor \frac{n}{u_1} \right\rfloor + 1$. Эта оценка бывает лучше в случаях, когда величина наименьшей образующей близка к половине n.

На рис. 4.17 приведен такой граф. Очевидно, что его хроматическое число равно 3, что соответствует оценке. К сожалению, из-за зависимости оценки от n, в общем случае эта оценка далека от истинного значения, особенно при больших n и сравнительно меньшей первой образующей. С другой стороны, данная оценка не зависит от количества образующих в NM-графе.

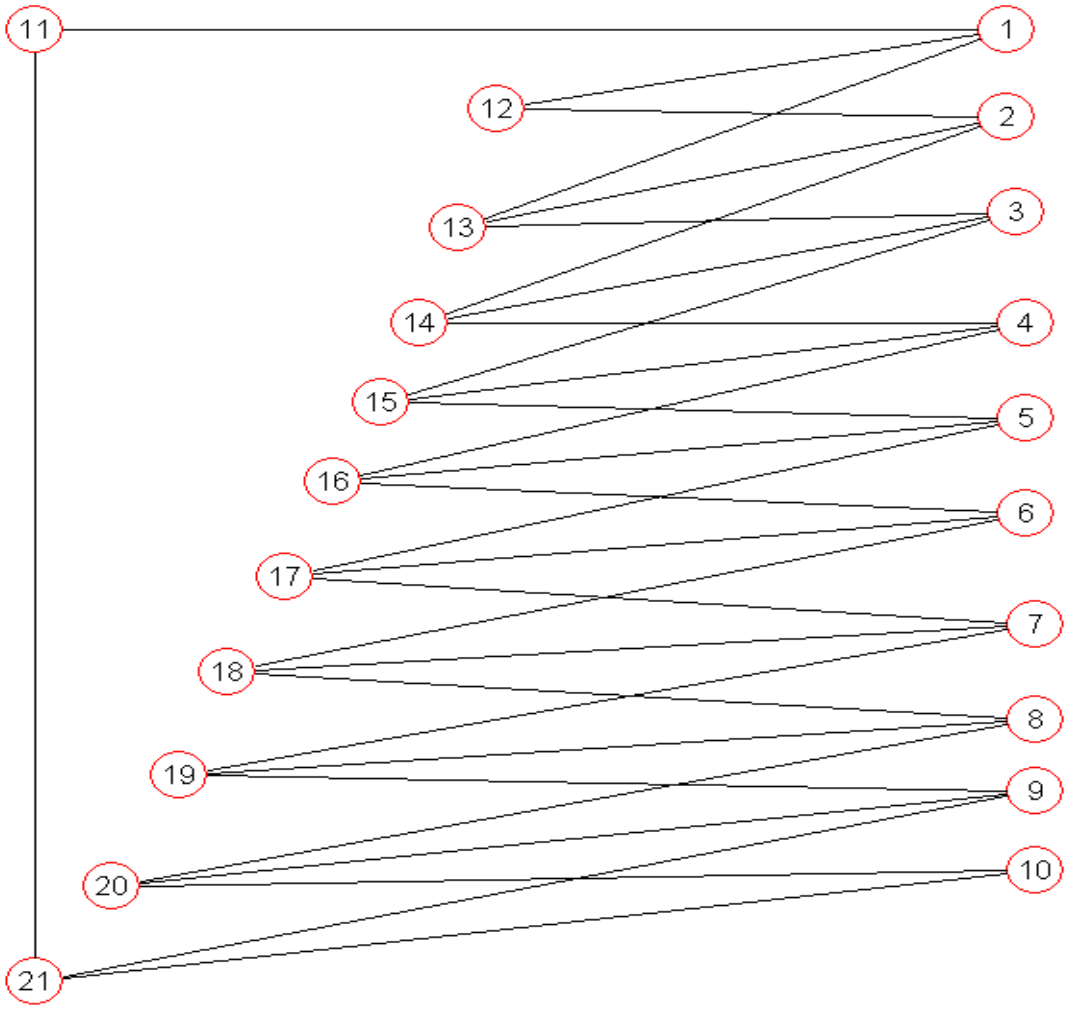

Рис. 4.17. Пример графа $G_{21}(10,11,12)$

Верхняя оценка хроматического числа графа весьма важна для ограничения перебора вариантов в обобщенном алгоритме определения минимального набора красок раскраски. Применив алгоритм поиска сначала для 3 красок и вплоть до оценочного значения, мы сможем точно определить хроматическое число произвольного *NM*-графа.

Чтобы определить хроматическое число графа $G_n(U)$ воспользуемся методом направленного перебора в пределах определенных верхней и нижней оценок хроматического числа. Для поиска используем метод дихотомии.

Пусть $P(n,U,k)$ – процедура раскраски графа $G_n(U)$ набором из k красок. В этом случае эта процедура возвращает одно из значений:

1, если $G_n(U)$ раскрашиваем k красками;

0, в противном случае.

Обозначим нижнюю оценку хроматического числа $\underline{\lambda}$, а верхнюю – $\overline{\lambda}$.

Применим процедуру $P(n,U,k)$ для $k \in [\underline{\lambda}, \overline{\lambda}]$ и в качестве хроматического числа λ^* выберем наименьшее из k, таких, что процедура возвращает 1.

Обозначим нижнюю оценку хроматического числа $\underline{\lambda}$, а верхнюю – $\overline{\lambda}$. Применим процедуру $P(n,U,k)$ для $k \in [\underline{\lambda}, \overline{\lambda}]$ и в качестве хроматического числа λ^* выберем наименьшее из k, таких, что процедура возвращает 1.

Процедура поиска хроматического числа осуществляет уточнение верхней оценки хроматического числа $\overline{\lambda}$ и нижней оценки $\underline{\lambda}$. Рекурсивная процедура поиска может быть задана следующим образом:

Алгоритм 4.2. Поиск хроматического числа *NM*-графа.

Шаг 0. На входе – отрезок $[\underline{\lambda}, \overline{\lambda}]$ – искомых хроматических чисел.

Шаг 1. Если $\underline{\lambda}$ равно $\overline{\lambda}-1$, то в качестве λ^* вернуть $\overline{\lambda}$ и остановиться.

*Шаг 2***.** Выбрать θ – среднее значение отрезка $[\underline{\lambda}, \overline{\lambda}]$.

Шаг 3. Если $P(n,U,\theta)$ возвращает 1 – то $\overline{\lambda}$ присвоить θ, а в противном случае – $\underline{\lambda}$ присвоить θ.

Шаг 4. Перейти на шаг 1.

Если длина отрезка оценок составляет Λ, то требуется $O(\log \Lambda)$ операций для выполнения алгоритма 4.1. Совокупная вычислительная емкость нахождения хроматического числа *NM*-графа составляет $O(n\Lambda k^2 u_m^3 m \log m \log \Lambda)$, что подтверждает полиномиальную оценку вычислительной емкости.

Список литературы

1. Алгебраические исследования в комбинаторике //Под редакцией И.А.Фараджева, – М.; Наука, 1978, – 187 с.
2. Асельдерова И.М. Об оптимальном кодировании некоторых арифметических графов //Теория оптимальных решений, Киев; Ин-т кибернетики им. В.М.Глушкова АН УССР, 1987, – С.47-54.
3. Асельдеров З.М., Донец Г.А. Представление и восстановление графов. – Киев.: Наук. думка, 1991. – 178 с.
4. Асельдерова И.М., Донец Г.А. Об оптимальном кодировании циклов и однородных деревьев // Теория и практика разработки и внедрения интегрированных АСУ, Киев. – 1988. – С. 56–62.
5. Асельдерова И.М., Донец Г.А. Об оптимальном кодировании однородных деревьев // Математические методы дискретной оптимизации. – Киев: Киевский госуниверситет им. Т.Г.Шевченка, 1986. – С. 33–37.
6. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 270 с.
7. Баннаи Э, Ито Т. Алгебраическая комбинаторика. Схемы отношений. – М.: Мир, 1987, – 373 с.
8. Белага Э.Г. К раскраске плоских графов // УМЖ, – 1972, –№3, – . 125- 128.
9. Белага Э.Г. Мини-геометрии (четыре фрагмента математики XX века), –М.: Знание, – 1977, – С. 22.
10. Белага Э.Г. Об одной интерпретации 4-раскраски плоского графа. – Успехи мат. наук, 1972, – 27, вып. 3, С. 191.
11. Берж К. Теория графов и ее применение. – М.: ИЛ, 1962. – 319 с.
12. Ван дер Варден Б.Л. Алгебра, М.: Наука, 1979. – 624 с.
13. Бончев Д.Г. Характеризация химических структур с помощью теории информации и теории графов: Автореф. дис. д.ф.-м. наук: 01.01.09. – М.: МГУ, 1984 – 28 с.
14. Боревич З.И., Шафаревич И.Р. Теория чисел. – М.: Наука, 1985.–503с.

15. Бусленко Н.П. Автоматизация имитационного моделирования сложных систем. – М.: Наука, 1977. – 288 с.
16. Винцюк Т.К. Анализ, распознавание и интерпретация речевых сигналов. – Киев: Наук. думка, 1987. – 264 с.
17. Воеводин В.В. Математические модели и методы в параллельных процессах . – М.: Наука, 1986. – 295 с.
18. Глушков В.М. О кибернетике как науке. – Вып. Кибернетика, мышление, жизнь. – М.: Мысль, 1964, С. 53–61.
19. Глушков В.М., Капитонова Ю.В., Летичевский Л.А. Теоретические основы проектирования дискретных систем // Кибернетика. – 1977. – №6. – С. 5–20.
20. Глушков В.М., Стогний А.А. и др. Системы автоматизации творческих процессов в научных исследованиях, проектировании и задачах управления роботами // Кибернетика. – 1981. – №6. – С. 110–115.
21. Горбатов В.А. Основы дискретной математики. – М.: Высш. школа, 1986. – 311 с.
22. Григорьян Ю.Г. Использование вычислительных машин для синтеза цифровых автоматов. – Известия АН АрмССР (серия тех.науки) XVI, 1963. – №6. – С. 41–47.
23. Григорьян Ю.Г. Арифметический метод минимизации булевских функций. – Материалы научных семинаров по теоретическим и прикладным вопросам кибернетики // Теория автоматов. – Киев: Ин–т кибернетики, 1964. – С. 3–24.
24. Григорьян Ю.Г. Вариационная задача функций алгебры логики и метод ее реализации на ЭВМ // Кибернетика. – 1967. – №1. – С. 26–30.
25. Григорьян Ю.Г., Маноян Г.К. Некоторые вопросы арифметической интерпретации неориентированных графов // Кибернетика. – 1977. – № 3. – С. 129–131.
26. Григорьян Ю.Г. Отображение сокращенных дизъюнктивных нормальных форм на графы // Кибернетика. – 1979. – № 3. – С. 105–107.
27. Григорьян Ю.Г. Классификация и статистические свойства арифметических графов // Кибернетика. – 1979. – №6. – С. 9–12.

28. Григорьян Ю.Г. Геометрия арифметических графов // Кибернетика. – 1982. – №4. – С. 1–4.
29. Григорьян Ю.Г. Задача существования и вопросы представления натуральных арифметических графов // Журнал выч. матем. и матем. физики АН СССР. – 1984. – №11. – С. 1751–1756.
30. Григорьян Ю.Г. Группы автоморфизмов булевских графов.Вторая Всесоюзная конференция "Математические методы распознавания образов" (тезисы докладов). Дилижан (АрмССР), 1985, С. 47–48.
31. Григорьян Ю.Г., Адонц А.М. Группы автоморфизмов арифметических графов. – Труды ВЦ АН АрмССР и ЕГУ. Математические вопро-ы кибернетики и вычислительной техники, т.15, 1988, С. 174–179.
32. Григорьян Ю.Г., Адонц А.М. Свойства регулярних натуральних арифметических графов // Кибернетика. – 1990. – №5. – С. 112–113.
33. Григорьян Ю.Г. Группы арифметических автоморфизмов простых циклов. // Кибернетика. – 1990. – № 4. – С. 9–16.
34. Григорьян Ю.Г. Группы, индуцируемые булевскими графами. // Кибернетика. – 1990. – № 6 – С. 111–113.
35. Григорьян Ю.Г. Информационные методы оценки сложности дискретных объектов в стандартных представлениях. – II Всесоюзная конференция по актуальным проблемам информатики и вычислительной техники "Информатика-87" (тезисы докладов). – Ереван, 1987. – С. 69–70.
36. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 416 с.
37. Дмитриев А.Н., Журавлев Ю.И., Кренделев Ф.П. О математических принципах классификации предметов и явлений // Дискретный анализ, вып.7. – Новосибирск: ИМ СО АН СССР, 1966. – С. 3–11.
38. Донец Г.А. О графах, задаваемых аналитическим способом // Теория оптимальных решений. – Киев: Ин-т кибернетики им.В.М.Глушкова АН УССР, 1987. – С. 20–27.
39. Донец Г.А. Об оптимальном кодировании однородных деревьев в

арифметических графах // Методы решения экстремальных задач и смежные вопросы. – Киев: Ин-т кибернетики им. В.М.Глушкова АН УССР, 1987. – С. 72–77.

40. Донець Г.О., Неженцев Ю.І. Арифметичні графи та їх представлення // Доп. АН УРСР.Сер.А. – 1990. – №11. – С. 5–8.
41. Донець Г.А., Неженцев Ю.Г.Об оптимальном кодировании циклов и наборов цепей в арифметических графах // Теория оптимальных решений. – Киев: Ин-т кибернетики им.В.М.Глушкова АН УССР, 1990. – С. 73–80.
42. Донец Г.А.,Неженцев Ю.И. Об оценке сложности алгоритмов в арифметических графах // Методы решения задач нелинейного и дискретного программирования. – Киев: Ин-т кибернетики им.В.М.Глушкова АН УССР, 1991. – С. 79–88.
43. Донец Г.А. Необходимые и достаточные условия связности арифметических графов // Теория оптимальных решений.-Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 1992. – С. 69–74.
44. Донец Г.А. Асельдерова И.М. Условия однородности арифметических графов // Оптимизация и ее приложения. – Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 1993. – С. 23–30.
45. Донец Г.А. О зависимости связности арифметических графов от числа образующих // Методы исследования экстремальных задач.-Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 1994. – С. 47–52.
46. Донец Г.А., Шулинок И.Э.. Однородные натуральные арифметические графы // Препринт 98-7.-Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 1998. – С. 18.
47. Донец Г.А., Шулинок И.Э.. Об оценке сложности алгоритмов для натуральных модульных графов // Теория оптимальных решений.-Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2001. – С. 61–68.
48. Донец Г.А., Шулинок И.Э.. Оптимальное представление однородных деревьев первого ранга в классе А-графов // Комп’ютерна математика. Оптимізація обчислень. – Київ: Ін-т кібернетики ім. В.М. Глушкова НАН

України, 2001. – С. 127–134.

49. Донец Г.А., Шулинок Г.А. Об изоморфизме натуральных арифметических графов // Теория оптимальных решений. – Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2003. – С. 3–10.
50. Донец Г.А., Шулинок И.Э. Об оценке сложности алгоритмов для натуральных модульных графов // Теория оптимальных решений.-Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2001. – С. 61–68.
51. Донец Г.А., Шулинок И.Э. Об общем представлении числовых графов // Теория оптимальных решений. – Киев: Ин-т кибенетики им. В.М.Глушкова НАН Украины, 2004. – С.11 - 18.
52. Елфимова Л.Д., Капитонова Ю.В. Быстрый алгоритм для умножения матриц и его эффективная реализация на систолических массивах // Кибернетика и системный анализ. – 2001. – № 1. – С. 135–150.
53. Елфимова Л.Д., Капитонова Ю.В. Интегрированный подход к проектированию процессорных массивов с систолической организацией вычислений // Кибернетика и системный анализ. – 2002. – № 6. – С. 3–15.
54. Емеличев В.А, Мельников О.И., Сарванов В.И., Тышкевич Р.И. Лекции по теории графов. – М.: Наука, 1990. – 384 с.
55. Журавлев Ю.И. Теоретико-множественные методы в алгебре логики. – В кн.: Проблемы кибернетики, вып.8. – М.: Наука, 1962. – С. 5–44.
56. Журавлев Ю.И. Об алгебраическом подходе к решению задач распознавания и классификации. – В кн.: Проблемы кибернетики, выл. 33. -М.: Наука, 1978. – С. 5–68.
57. Зыков А.А. Реберно-вершинные функции и распределительные свойства графов // ДАН СССР, т.139. – 1961. – №4. – С. 787–791.
58. Зыков А.А. Теория конечных графов.–Новосибирск: Наука, 1969.–543с.
59. Зыков А.А. Основы теории графов. – М.: Наука, 1987. – 384 с.
60. Каюров В.Ю, Шулинок Г.А. Полиномиальный алгоритм проверки изоморфизма деревьев // Теория оптимальных решений. – Киев: Ин-т кибенетики им. В.М.Глушкова НАН Украины, 2002. – С.92 - 97.

61. Кострикин А.И. Введение в алгебру. – М.: Наука, 1977. – 495 с.
62. Кэртис Ч.. Райнер И. Теория представлений конечных групп и ассоциативных алгебр. – М.: Наука, 1969. – 668 с.
63. Курош А.Г. Теория групп. – М.: Наука, 1967. – 648 с.
64. Михелович Ш.Х. Теория чисел. – М.:Высш. школа, 1962. – 260 с.
65. Неженцев Ю.И. Об оценке сложности алгоритмов поиска в арифметических графах // Математические методы и программное обеспечение в системах принятия решения и проектирования.-Киев: Ин-т кибернетики им. В.М. Глушкова АН УССР, 1990. – С. 14–24.
66. Пападимитриу Х. Стайглиц К. Комбинаторная оптимизация Алгоритмы и сложность. – М.: Мир, 1985. – 512 с.
67. Оре.О. Теория графов. – М: Наука, 1980.– 336 с.
68. Поваров Г.И. Математическая теория синтеза контактных (1,К) – полюсников // ДАН СССР. – 1955. – т.100. – №5. – С. 909–912.
69. Розенфельд Б.А. История неевклидовой геометрии. – М.: Наука, 1976. – 413 с.
70. Сапоженко А.А., Асатрян А.С., Кузюкин Н.Н. Обзор некоторых результатов по задачам о покрытии // Дискретный анализ, вып.30. – Новосибирск: ИМ СО АН СССР, 1977. – С. 46–75.
71. Свами М., Тхуласираман К. Графы, сети и алгоритмы. – М.: Мир, 1984. – 455 с.
72. Серр Ж.-П. Линейные представления конечных групп. – М.: Мир, 1970. – 131 с.
73. Теория и методы автоматизации проектирования вычислительных систем /Под ред. М.Брейера. – М.: Мир, 1977. – 283 с.
74. Ульман Д. Вычислительные аспекты СБИС. – М.: Радио и связь, 1990. – 480 с.
75. Хамермеш М. Теория групп и ее применение к физическим проблемам. – М.: Мир, 1969. – 587 с.
76. Хан Г., Шапиро С. Статистические модели в инженерных задачах. – М.: Мир, 1969. – 281 с.

77. Харари Ф. Теория графов. – М.: Мир, 1973. – 301 с.
78. Хассе Г. Лекции по теории чисел. – М.: ИЛ, 1953. – 527 с.
79. Чегис И.А., Яблонский С.В. Логические способы контроля электрических схем. – Труды матем. ин-та им.В.А.Стеклова АН СССР, 51, 1958. – С.270–360.
80. Шулінок Г.О. Про ізоморфізм натуральних модульних графів. // Теория оптимальных решений. – Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2004. – С.69 – 73.
81. Шулінок Г.О. Про ізоморфізм одного підкласу числових графів. // Праці міжн. конф. “Питання оптимізації обчислень(ПОО-XXXIII)”, присвяченої пам’яті академіка В.С.Михалевича (Крим, Велика Ялта, смт. Кацивелі, 19-23 вересня 2005 р.) .В.М.Глушкова НАН України, 2005. – С.216.
82. Шулинок Г.А. Об изоморфизме регулярных NM-графов. // Теория оптимальных решений. – Киев: Ин-т кибернетики НАН Украины, 2005. – С.100 – 106.
83. Шулинок Г.А. Об одном методе раскраски натуральных модульных графов // Компьютерная математика. – 2006. – №1. – С.78 – 89.
84. Шулинок И. Э. Структура натуральных арифметических графов с нечетным числом вершин // Оптимизация и ее приложения. – Киев: Ин-т кибернетики им.В.М. Глушкова НАН Украины, – 1997. – С. 54–60.
85. Шулинок И.Э. Об одном классе числовых графов // Теория и приложения методов оптимизации.- Киев: Ин-т кибернетики им.В.М.Глушкова НАН Украины, 1998. – С. 24–29.
86. Шулинок И.Э. О связности натуральных модульных графов // Кибернетика и системный анализ. – 1998 – №5. – С. 50–53.
87. Шулинок И.Э.О связности и цикломатическом числе натуральных модульных графов // Теория оптимальных решений. – Киев: Ин-т кибернетики им.В.М. Глушкова НАН Украины, – 1999. – С. 51–57.
88. Шулинок И.Э. Полное описание структуры одного подкласса *NM*-графов // Теория оптимальных решений. – Киев: Ин-т кибернетики

им.В.М. Глушкова НАН Украины, – 1999. – С. 75–81.

89. Шулинок И.Э. О сложности алгоритмов на числовых графах // Прикладная математика: Тез, докл. Междунар. конференции, посв. 65-летию со дня рождения Б.Н.Пшеничного. 25 – 28 июня, 2002. – Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, НТУ Украины «КПИ», ИПСА, 2002. – С. 110–111.

90. Яблонский С.В. Функциональные построения в k-значной логике. – Труда матем.ин-та им.В.А.Стеклова АН СССР, 51,1958. – С. 5–143.

91. Kantz W.H. Optimized data ancoding for digital computers // IRE Conv. Rec. – 1954. – Vol. 4. – P. 47–56.

92. Kung H.T., Leiserson C.E. Systolic arrays for VLSI // Sparse Matrix Symp., 1978. Philadelphia: SIAM, 1979. – P. 252–282.

93. Kung H.T. Why systolic architectures? // Computer. – 1982. – **15**, №1. – P. 37–46.

94. Monien B. The banwidth minimization problems for caterpillars with hair length 3 is NP- complete // SIAM J. Alg. Disc. Meth. – 1986. – Vol. 7, № 4. – P. 505–512.

95. Papadimitrion C.H. The NP–completeness of the bandwidth minimization problem // Computing. – 1976. – Vol.16. – P. 263–270.