МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Дагестанский государственный университет»
Факультет математики и компьютерных наук
Кафедра дискретной математики и информатики

Магомедов А.М.

Алгоритмические аспекты решения головоломок

Махачкала − 2020 г.

УДК 519.68

Магомедов А. М.

Алгоритмические аспекты решения головоломок.

Пособие рассчитано на студентов начальных курсов бакалавриата по направлениям 010302 и 020302.

Рецензенты:

Рамазанов А.-Р.К. – проф., д-р физ.-мат. наук

Лугуев Т.С. – доцент, канд. физ.-мат. наук

Пособие одобрено учебно-методическим советом факультета математики и компьютерных наук Дагестанского государственного университета

Оглавление

Введение 4
Глава I. Базовые алгоритмы дискретной математики 5
§1.1. Алгоритмы теории графов 5
1.1.1. Эйлеров цикл 5
1.1.2. Максимальный поток 6
1.1.3. Кратчайшее расстояние 7
1.1.4. Лабиринт 7
1.1.5. Шахматный конь 8
1.1.6. Переправа 9
1.1.7. Допустимый поток 10
1.1.8. Связные компоненты 10
1.1.9. Кратчайшее остовное дерево 11
1.1.10. Размен купюры 12
1.1.11. Брачный маклер 12
§1.2. Алгоритмы комбинаторики 13
1.2.1. Множество пар 13
1.2.2. Единицы в пустых клетках 14
1.2.3. Разбиение на три подмножества 15
1.2.4. Минимальный набор линий с нулями 16
1.2.5. Рюкзак 16
§1.3. Алгоритмы для действий с числами 17
1.3.1. Извлечение квадратного корня 17
1.3.2. Возрастающая подпоследовательность 18
1.3.3. Местонахождение точки 18
1.3.4. Игра Ним 19
1.3.5. Знаки операций 20
1.3.6. k-й наименьший элемент 20
1.3.7. Сортировка 21
1.3.8. Алгоритм банкира 21
Глава II. Алгоритмы решения некоторых классических головоломок 23
§2.1. Построение графа для задачи о перевозке обезьян-каннибалов 23
§2.2. Алгоритм решения задачи "Ревнивые мужья" 27
§2.3. Построение графа для задачи о переливаниях 30
§2.4. «Компьютерное» доказательство существования терминального узла 33
Литература 36

Введение

Важной составляющей успеха на олимпиадах по программированию является владение базовыми алгоритмами дискретной математики. Не менее существенно знакомство с основными алгоритмами университетского курса дискретной математики и в работе профессионального программиста. Однако подготовка сборника алгоритмов с подробным их изложением следует признать трудновыполнимой задачей.

Поэтому при подготовке первой главы настоящей работы мы поставили целью отобрать алгоритмы, которые, на наш взгляд, сочетают доступность для начинающих программистов с определенной универсальностью применения к широкому ряду прикладных задач, и изложить их в максимально сжатой, конспективной форме. Несмотря на то, что идеи этих алгоритмов в значительной части заимствованы из литературы (в тексте приводятся подробные ссылки), не следует считать, что работа автора здесь сводится к редактированию известных текстов. Ограничимся одним примером: задача пункта 1.2.3 о проверке существования разбиения заданного множества натуральных чисел на три подмножества с равными суммами элементов нам в литературе не встречалась; в известной монографии [1] рассматривается задача разбиения на два подмножества (классическая NP-полная задача «Разбиение»), но и там изложение ограничивается лишь проверкой существования разбиения, мы же сформулировали и алгоритм генерации соответствующих подмножеств.

Предназначение второй главы принципиально иное: здесь мы стремились показать, что изложение трудных головоломок в терминах теории графов нередко позволяет свести их решение к поиску алгоритма построения пути из «начального» узла к некоторому «терминальному» узлу графа. При этом компьютерное сопровождение алгоритма (составление программы на одном из современных языков высокого уровня) служит ценной составляющей успеха.

Глава I. Базовые алгоритмы дискретной математики

Разбиение главы на параграфы носит условный характер: в первом сосредоточены не только задачи, непосредственно относящиеся к теории графов, но и задачи, решения которых могут быть получены с помощью алгоритмов теории графов; во втором параграфе рассмотрены комбинаторные задачи; объединение задач и алгоритмов третьего параграфа под названием «Алгоритмы для действий с числами» следует признать спорным (см., например, алгоритм пункта 1.3.8, применяемый в теории операционных систем для избежания тупиковых ситуаций).

§1.1. Алгоритмы теории графов

1.1.1. Эйлеров цикл

В связном неориентированном графе с четными степенями вершин указать цикл, который проходит через каждое ребро точно один раз [6], [10].

Алгоритм 1

Начиная с некоторой вершины v_0, будем обходить ребра графа, стирая пройденные ребра, пока не придем снова в вершину v_0. Это случится обязательно, т.к. заходя в вершину по непройденному ранее ребру, мы имеем возможность выйти из него в силу четности степеней вершин.

Если при возвращении в v_0 не существует вершина v_1, принадлежащая построенному циклу φ и являющаяся началом еще не пройденного ребра, то нужный цикл построен. В противном случае, начиная с v_1, будем проходить не принадлежащие φ ребра до тех пор, пока не придем снова в v_1 (это случится, т.к. в оставшейся после удаления ребер цикла φ части графа все степени вершин четные). Построим цикл φ', объединяя циклы φ и φ_1 (вновь построенный) следующим образом: берем часть цикла φ от v_0 до v_1, цикл φ_1 и затем оставшуюся часть цикла φ от v_1 до v_0. Если после этого в графе останутся не пройденные ребра, процесс продолжается, взяв в качестве φ цикл φ'. Через конечное число шагов процесс завершится построением эйлерова цикла.

Алгоритм 2. Начинать с некоторой вершины и каждый раз вычеркивать пройденное ребро. При этом а) не проходить по ребру, если удаление этого ребра

приводит к разбиению графа на две связные компонента (не считая изолированных вершин); б) не проходить по ребру, ведущему к вершине степени 1, пока еще остались ребра.

1.1.2. Максимальный поток

Вычислить максимальный поток f в заданной сети с целочисленными пропускными способностями c [7].

Сначала положим поток f равным нулю и приступим к приписыванию узлам сети пометок вида (x^+, ε) или (x^-, ε), где $x \in N$ (множество узлов), а ε – натуральное число или ∞, в соответствии с правилами операции A.

Во время операции A каждый узел находится в одном из следующих состояний: не помечен; помечен и просмотрен; помечен и не просмотрен.

Вначале все узлы не помечены.

Операция A. Источник S получает пометку $(0^-, \infty)$, после этого S помечен и не просмотрен.

Выберем произвольный помеченный и непросмотренный узел x. При выборе помеченного узла из двух узлов необходимо отдать предпочтение тому узлу, который был помечен ранее. Пусть он имеет пометку $(z^{\pm}, \varepsilon(x))$. Всем узлам y, которые не помечены и для которых $f(x,y) < c(x,y)$, припишем $(x^+, \varepsilon(y))$, где

$$\varepsilon(y) = \min\{\varepsilon(x), c(x,y) - f(x,y)\}.$$

Всем узлам y, которые после этого не помечены и для которых $f(y,x) > 0$, приписываем пометку $(x^+, \varepsilon(y))$, где

$$\varepsilon(y) = \min\{\varepsilon(x), f(y,x)\}$$

(такие y теперь помечены и не просмотрены, а узел x после этого помечен и просмотрен).

Повторяем этот шаг, пока не окажется помеченным и не просмотренным сток t, или же до тех пор, пока нельзя будет больше пометить ни один узел, а сток останется непомеченным. В первом случае переходим к операции B, а во втором алгоритм завершается.

Операция B. Сток t имеет пометку $(y^{\pm}, \varepsilon(t))$. Если $(y^+, \varepsilon(t))$, то $f(y,t)$ заменяем на $f(y,t) + \varepsilon(t)$; если $(y^-, \varepsilon(t))$, то $f(t,y)$ заменяем на $f(y,t) - \varepsilon(t)$. Затем в любом из этих случаев переходим к узлу y.

Вообще, если узел y имеет пометку $(x^+, \varepsilon(y))$, то $f(x,y)$ заменяем на $f(x,y) + \varepsilon(t)$, а если он имеет пометку $(x^-, \varepsilon(y))$, то $f(y,x)$ заменяем на $f(y,x) -$

$\varepsilon(t)$ и переходим к узлу x. Когда достигнем источника S, нужно стереть все старые пометки и вновь перейти к операции A.

1.1.3. Кратчайшее расстояние

Найти кратчайшее расстояние и кратчайший путь между двумя заданными вершинами v_0 и u_0 графа $G = (V, E)$, если задана длина $l(i,j)$ для каждой дуги $(i,j) \in E$ (если $(i,j) \notin E$, то $l(i,j) = \infty$) [3].

Выполним помечивание вершин; каждой вершине v сопоставляем две метки: $D(v)$ – кратчайшее расстояние от вершины v_0, $E(v)$ – вершина, непосредственно предшествующая вершине v в кратчайшем пути от v_0 до v; но таковыми эти две метки станут в конце помечивания, более точно – после того, как вершина v_0 будет включена в некоторое множество S, которое вначале включает только v_0, постепенно расширяется и в конце помечивания включает u_0.

$S \leftarrow \{v_0\};\ D(v_0) := 0;$

Для каждого $v \in V - v_0$ выполнить

$\{D(v) \coloneqq l(v_0, v)$; если $l(v_0, v) \neq \infty$, то $E(v) = v_0;\}$

пока $u_0 \notin S$ выполнить следующие три действия:

{

а) выбрать узел $w \in V - S$ с $\min D(w)$;

б) добавить w к S;

в) для $v \in V - S$

если $D(w) + l(w, v) < D(v)$, то $\{E(v) = w; D(v) = D(w) + l(w, v)\}$;

}

В результате $D(u_0)$ – расстояние от v_0 до u_0 по кратчайшему пути, а $v_0, \dots, E\big(E(u_0)\big), E(u_0), u_0$ – кратчайший путь от v_0 к u_0.

1.1.4. Лабиринт

Имеется лабиринт из n комнат, некоторые из которых соединены проходами. Даны «вход» и «выход»: S и T. Требуется найти путь из S в T. Лабиринт задан матрицей A $n * n$, где $a_{ij} = 1$, если i-я и j-я комнаты соединены проходом, в противном случае $a_{ij} = 0$ [4].

Понятно, что задача может быть решена с использованием предыдущего пункта. Предложим иное решение.

Пусть имеется массив номеров комнат $b_1, b_2, \dots, b_k$. Если это тупик, то либо b_k должно совпадать с некоторым из $b_1, \dots, b_{k-1}$, либо в матрице A все элементы

$a_{b_k,1}, \dots, a_{b_k,n}$, кроме $a_{b_k,b_{k-1}}$, равны нулю. Если тупик, то делается попытка найти среди элементов $a_{b_{k-1},b_{k+1}}, a_{b_{k-1},b_{k+2}}, \dots, a_{b_{k-1},n}$, равный единице.

Если единичные элементы имеются, то берется первый из них – пусть это будет $a_{b_{k-1},m}$. Тогда $b_k := m$.

Если единиц нет, то массив укорачивается на один элемент и процедура применяется уже к $b_1, \dots, b_{k-1}$ и т.д.

Если не тупик, то к массиву $b_1, \dots, b_k$ присоединяется еще один элемент b_{k+1} следующим образом: среди $a_{b_k,1}, \dots, a_{b_k,n}$ отыскивается первый, равный единице, пусть это $a_{b_k,m}$, тогда $b_{k+1} := m$.

1.1.5. Шахматный конь

Составить программу поиска обхода всех клеток шахматной доски конем так, чтобы очередной ход являлся продолжением предыдущего и на каждую клетку можно ступить ровно один раз [8].

Решение: каждый раз с текущей клетки сделать ход на ту из достижимых из нее непройденных клеток, откуда меньше всего ходов.

Указания:

а) Матрицу $D(1{:}8, 1{:}8)$ первоначально обнулить, если x-й ход сделан на поле $D(i,j)$, то $D(i,j)$ присвоить x. Предусмотреть два вектора A и B с начальными значениями $-1, -1, 1, 1, 2, 2, -2, -2$ и $2, -2, 2, -2, 1, -1, 1, -1$ соответственно. Тогда результат k-го из 8 ходов с клетки $D(i,j)$ вычисляется так: $k_i = i + A(k)$, и $k_j = j + B(k)$. Сделав ход на 64-ю по счету клетку, остановиться.

б) Нельзя проверять допустимость хода на клетку $D(k_i, k_j)$ следующим образом:

if $k_i > 0 \wedge k_i < 9 \wedge k_j > 0 \wedge k_j < 9 \wedge D(k_i, k_j) = 0$, поскольку и при нарушении, например, условия $k_i > 0$, будет проверено условие $D(k_i, k_j) = 0$, что приведет к прерыванию из-за выхода индекса за допустимые границы. Рекомендуется вместо операции конъюнкции использовать в этом условии операцию сокращенного логического умножения, которая имеется в некоторых языках программирования (например, операция && в языке C#).

в) Варьировать начало обхода. Предусмотреть переход к следующему варианту при возникновении тупиковой ситуации. Предварительно сделать попытку избежать тупиковые ситуации, начиная с той же клетки, но переставив элементы векторов A и B одновременно (тем самым мы влияем на выбор хода, когда клеток, откуда меньше всего ходов, оказывается несколько).

1.1.6. Переправа

Перевозчику нужно переправить на правый берег реки волка, козу и капусту. Лодка так мала, что кроме перевозчика может взять только один из этих объектов. Кроме того, нельзя оставлять капусту вместе с козой, а козу вместе с волком.

Различные позиции могут быть описаны объектами, находящимися на левом берегу. Каждую позицию будем представлять в виде двоичного кода длины 4, где наличие (отсутствие) единицы в первом, втором, третьем, четвертом бите означает наличие (отсутствие) на левом берегу соответственно волка, козы, капусты, перевозчика. Будем строить ориентированный граф, где каждая вершина обладает двоичным кодом длины 4, четностью r (0 или 1), знаком («+» или «-») и номером.

Алгоритм построения графа.

Ввести минусовую нечетную ($r = 1$) вершину v_1 с кодом 1111. Номер:$= 1$.

Пока не получена вершина с нулевым кодом, выполнять пункты $1 - 3$, где через (A) обозначена совокупность действий:

номер:=номер $+1$;

ввести минусовую вершину $v_{\text{номер}}$ с кодом * и четностью $1 - r$ и дугу $(v_i, v_{\text{номер}})$;

1. Отыскать минусовую вершину v_i с наименьшим номером;

$r:=$ четность v_i; *:=код v_i;

2. Если $r = 1$, то

а) обнулить 4-й бит кода *;

б) для $i = 1,2,3$ выполнить:

если в результате обнуления i-го бита в * не соседствуют единичные биты и полученный код не совпадает с кодом одной из вершин, то обнулить i-й бит и выполнить (A).

Если $r = 0$, то

а) присвоить единицу 4-му биту *;

б) если в * не соседствуют нулевые биты и * не совпадает с кодом одной из вершин, то выполнить(A);

в) для $i = 1,2,3$ выполнить:

если в результате присвоения единицы i-у биту в * не соседствуют нулевые биты, и * не совпадает с кодом одной из вершин, то присвоить единицу i-му биту в * и выполнить (A).

3. Вершине v_j присвоить знак «+».

Конец алгоритма построения графа.

При достижении вершины с нулевым кодом произвольный путь, соединяющий с ним исходный пункт, и является описанием процесса переправы.

1.1.7. Допустимый поток

Пусть в транспортной сети наряду с пропускными способностями дуг $c(e)$ заданы и нижние потоковые границы $l(e)$. Выяснить, существует ли допустимый поток f, т.е. поток f, удовлетворяющий условиям: $l(e) \leq f(e) \leq c(e)$ [7].

Суть решения заключается в сведении к задаче существования в некоторой другой сети максимального потока определенной величины (см. алгоритм пункта 1.6).

Новая сеть строится следующим образом. Вводится новый источник u, новый сток w, дуги (u, y) и (x, w) пропускной способности $l(x, y)$ для каждой дуги (x, y) прежней сети; пропускная способность $c(x, y)$ при этом заменяется величиной $c(x, y) - l(x, y)$. И наконец, прежний сток соединяется с прежним источником дугой бесконечной пропускной способности. Новая сеть построена.

Обозначим через z суммарную пропускную способность дуг, оканчивающихся в новом стоке w. В прежней сети существует допустимый поток тогда и только тогда, когда величина максимального потока в новой сети равна z.

1.1.8. Связные компоненты

В заданном графе $\mathrm{G} = (V, E)$ выделить связные компоненты за время $O(|E|)$. По завершении алгоритма вершины i-й связной компоненты должны быть помечены i $(i = 1,2, \ldots)$ [1].

Предположим, что граф $G = (V, E)$ не имеет изолированных вершин и представлен списками смежностей $A(v), v \in V$. В алгоритме используется множество R для хранения всех непомеченных вершин и множество $Q \subseteq R$ для хранения непомеченных вершин, смежных с уже помеченными.

$R \leftarrow V; i = 0;$

пока $R \neq \emptyset$

{

выбрать произвольную вершину $v_1 \in R; i = i + 1; Q \leftarrow v_1;$

пока $Q \neq \emptyset$

{

пометить символом i произвольную вершину $v \in Q$ и удалить v из Q и из R;

для всех $v' \in A(v)$: если v' не помечена, то добавить v' к Q;

}

}

В зависимости от способа удаления элемента из Q различают поиск в глубину (использование принципа стека) и поиск в ширину (использования принципа очереди).

1.1.9. Кратчайшее остовное дерево

Для связного графа $G = (V, E)$ заданы расстояния d_{ij} между всеми парами вершин v_i, v_j из V. Найти кратчайшее остовное дерево (V, T) в графе G [1].

Алгоритм, приводимый ниже, решает задачу за время $O(|V|^2)$.

$U := v_1$; $T := \emptyset$;

для всех $v \in V - \{v_1\}$ выполнить $bl[v] = v_1$;

пока $U \neq V$

{

$min := \infty$;

для всех $v \in V - U$ выполнить

если $d(v, bl[v]) < min$, то

{

$min := d(v, bl[v])$;

$sled := v$;

$U := U \cup \{sled\}$;

$T := T \cup \{[sled,\ bl[sled]]\}$;

};

для всех $v \in V - U$ выполнить

{

если $d(v, bl[v]) > d(v, sled)$ то $bl[v] := sled$;

}

}

1.1.10. Размен купюры

Имеется денежная купюра достоинства k. Требуется выяснить, можно ли разменять ее, используя купюры достоинством $c_1, c_2, \dots, c_n$ [1]. Другими словами, существуют ли такие целые неотрицательные числа $x_1, x_2, \dots, x_n$, что $\sum_{j=1}^{n} c_j x_j = k$?

Алгоритм состоит из двух шагов.

1 шаг. Построим следующий орграф $G(c_1, \dots, c_n; k) = (V, A); V = \{v_0, v_1, \dots, v_k)$, $A = \{(v_m, v_k): 0 \le m < k \le K$ и $k - m = c_j$ для некоторого $j \le n\}$.

Граф G можно построить за время $O(nK)$ и имеет $K + 1$ вершину и $O(nK)$ дуг.

Справедливо следующее утверждение: размен купюры возможен тогда и только тогда, когда в графе $G(c_1, \dots, c_n; k)$ имеется путь из v_0 в v_k.

2 шаг. Существование пути из v_0 в v_k можно выяснить за время $O(nK)$:

1. пометить вершину v_0;
2. пока имеются непомеченные вершины, выполнить:
для каждой помеченной вершины пометить смежные непомеченные вершины;
3. заключить, что имеется путь из v_0 в v_k в том, только в том случае, если v_k помечен.

1.1.11. Брачный маклер

Пусть имеются n мужчин $x_1, x_2, \dots, x_n, n$ женщин $y_1, y_2, \dots, y_n$ и m брачных маклеров $z_1, z_2, \dots, z_m$. Каждый маклер m_j имеет список некоторых из этих мужчин (X_j) и женщин (Y_j), являющихся его клиентами, и может устроить брак между любым мужчиной и любой женщиной из этого списка. Определить наибольшее число возможных браков при дополнительном условии, что число браков, которое может устроить маклер m_j, не превосходит b_j.

Алгоритм использует подпрограмму вычисления максимального потока в сети (см. алгоритм пункта 1.6).

Введем «источник» S, «сток» T, вершины

$x_1, x_2, \dots, x_n$;

$z_1', z_1'', z_2', z_2'', \dots, z_m', z_m''$;

$y_1, y_2, \dots, y_n$

и построим сеть, где пропускные способности определяются следующим образом:

$p(s, x_i) = 1;$

$$p(x_i, z_j') = \begin{cases} 1 \text{ при } x_i \in X_j, \\ 0 \text{ при } x_i \notin X_j; \end{cases}$$

$p(z_j', z_j'') = b_j;$

$$p(z_j'', y_k) = \begin{cases} 1 \text{ при } y_k \in Y_j, \\ 0 \text{ при } y_k \notin Y_j; \end{cases}$$

$p(y_k, T) = 1.$

Наибольшее число возможных браков равно величине максимального потока построенной сети.

§1.2. Алгоритмы комбинаторики

1.2.1. Множество пар

Пусть дается некоторое множество A неориентированных пар вида (a_i, b_j), где числа a_i и b_j принадлежат множеству $N = \{1,2, \dots, n\}$, причем известно, что ни одно число из N не встречается в парах из A более 4 раз. Разбить множество A на подмножество A_1 и A_2 так, чтобы ни в одном из них никакое число из N не встречалось более двух раз.

Построим неориентированный граф $G = (V, E)$, где

$V = \{v_1, \dots, v_n\}$ и $(v_i, v_j) \in E$ тогда и только тогда, когда $(i, j) \in A$.

Без ограничения общности граф G будем считать связным.

Достроим G до графа $G^* = (V, E^*)$ с четными локальными степенями 2 или 4, соединяя фиктивными ребрами вершины с нечетными степенями. Это возможно, поскольку в любом графе количество вершин с нечетными степенями четно.

Построим эйлеров цикл φ графа G и пометим последовательные ребра φ по следующему правилу:

В качестве начального ребра для помечивания выбирается произвольное беспетельное ребро (u, v), инцидентное некоторой вершине v степени 4, и помечивая его меткой 1, «входим» в исходную вершину степени 4.

Замечание. Если все ребра, инцидентные вершине степени 4 петельные, то связная компонента графа состоит из единственной вершины с двумя петлями, одну из которых мы помечиваем 1, другую – 2; если же каждая вершина имеет степень, равную 2, то одно из искомых подмножеств для данной связной компоненты полагаем пустым.

Далее помечивание продолжается следующим образом. Если в промежуточную вершину v степени 4 мы вошли, помечая ребро (u, v) цифрой $i, i \in \{1,2\}$, то выходящее из вершины v непомеченное ребро помечиваем цифрой $3 - i$; ребро же, выходящее из вершины v степени 2, помечиваем той же цифрой, по которой вошли в v.

После завершения помечивания каждой вершине степени 4 окажутся инцидентны ровно два ребра с меткой $i;\ i = 1, 2.$ Для промежуточных вершин утверждение очевидно. Докажем его для исходной вершины. Рассмотрим «обобщенный» граф, удаляя каждую вершину v степени 2 и рассматривая пару инцидентных вершине v ребер в качестве одного обобщенного ребра. В полученном однородном графе степени 4 количество ребер четно: складывая степени вершин, получаем число, кратное 4, причем каждое ребро вошло в эту сумму ровно 2 раза. Справедливость утверждения для исходной вершины вытекает из того, что последовательные ребра «обобщенного» графа оказались помеченными чередующимися метками 1 и 2.

Нефиктивные ребра, помеченные цифрой i, образуют множество $A_i,\ i = 1,2$.

1.2.2. Единицы в пустых клетках

В матрице $A\ m * n$ некоторые клетки зачеркнуты, некоторые пусты. Требуется проставить в пустые клетки максимальное число единиц так, чтобы в любом ряду (ряд – строка или столбец) было не более одной единицы [7].

Выберем какое-либо допустимое размещение единиц. Решение получается последовательными итерациями. Опишем шаг итерации.

Начало. Пометим, скажем, черточкой, все строки, не содержащие единиц. Затем выберем какую-либо помеченную строку, окажем, i-ю, и непомеченные столбцы (вначале все столбцы не помечены), соответствующие пустым клеткам этой строки, пометим номером этой строки i. Будем повторять это до тех пор, пока не окажутся просмотрены все помеченные строки, причем столбец, получивший пометку, далее не помечается.

Теперь выберем произвольный помеченный столбец j и просмотрим его, отыскивая единицу: если она будет найдена, пометим строку, в которой стоит эта единица, номером j просматриваемого столбца. Снова выберем произвольный помеченный непросмотренный столбец и повторим эту операцию, причем строка, получившая пометку, далее не помечается. После того, как будут просмотрена

все помеченные столбцы, возвращаемся к просмотру строк, выбирая помеченную цифрой непросмотренную строку и т.д. В этом процессе попеременного просмотра строк и столбцов встретится одна из следующих ситуаций:

1. Столбец, не содержащий единиц, пометить не удалось и больше никаких пометок не удается приписать. В этом случае имеющееся размещение единиц максимально. Алгоритм завершается.

2. Помечен столбец, не содержащий единицы.

Во втором случае общее число единиц в таблице A может быть увеличено следующим образом. В столбце, не содержащем единицы и только что помеченном, поставим единицу в клетку, указываемой пометкой этого столбца; затем в строке, в которой стоит эта единица, перейдем к клетке, указываемой пометкой этой строки и уберем единицу из этой клетки; затем в столбце, в котором находится эта клетка, перейдем к клетке, указываемой его пометкой и поставим в неё единицу и т.д.

В конце концов мы придем к одной из первоначально отмеченных черточкой отрок. На этом перемещения закончатся, и общее число единиц в таблице увеличится на единицу.

Конец.

Сотрем пометки и повторим шаг итерации.

1.2.3. Разбиение на три подмножества

Для заданного множества {a[1], ..., a[n]}, где все a[i] - натуральные, требуется:
(а) выяснить, существует ли разбиение на три подмножества с равными суммами элементов;
(б) в случае существования разбиения сгенерировать соответствующее разбиение [2].
1. Если сумма s=a[1]+...+a[n] не кратна трем, ответ отрицателен; пусть s=3B, где B - натуральное число. Если найдется a[i]>B, то ответ также отрицателен; пусть все a[i] мажорируются значением B.

2. Заполним ячейки t[i,j,k] массива t[1..n, 0..B, 0..B] значениями истинности высказывания «в множестве {a[1], …, a[i]} найдутся два непересекающихся подмножества с суммами j и k»:
всем ячейкам массива t присвоить false;
в множестве t[1,j,k] присвоить трем ячейкам значение true:
t[1, a[1], 0]= t[1,0,0]= t[1, 0, a[1]]=true;
для i=1 массив t заполнен.

Выполним для i=2, …, n, j=0, …, B, k=0,…, B:

t[i,j,k]= t[i-1,j,k] ||
((j-a[i]>=0) && t[i-1, j-a[i], k]) ||
((k-a[i]>=0) && t[i-1,j,k-a[i]];
смысл операций && и || соответствует C#.
Массив t полностью заполнен.

3. Если t[n,B,B] равно false, то завершить алгоритм с отрицательным ответом.
Присвоить J=K=B.
Пока J+K отлично от нуля, выполнить:
{
найти наименьшее I: t[I,J,K] равно true;
если (J-a[I]>=0) && t[I-1, J-a[I],K], то {J=J-a[I] и включить I в список s1}
иначе
{K=K-a[I] и включить I в список s2};
}
Остается вывести искомые подмножества, используя списки s1 и s2.

1.2.4. Минимальный набор линий с нулями

Требуется найти минимальный набор строк и столбцов матрицы C $m * n$, содержащих нули [9].

Рассмотрим строку с наименьшим числом нулей. Отметим точкой один из нулей этой строки и зачеркнем все остальные нули этой строки и того столбца, в котором находится этот нуль.

Аналогичные операции последовательно проводим для всех строк. Отметим точкой:

а) все строки, в которых не имеется ни одного отмеченного точкой нуля;

б) все столбцы, содержащие перечеркнутый нуль хотя бы в одной из отмеченных точкой строк;

в) все строки, содержащие отмеченные точкой нули, хотя бы в одном из отмеченных точкой столбцов.

Действия б) и в) повторять поочередно до тех пор, пока есть, что отмечать. После этого необходимо зачеркнуть каждую непомеченную строку и каждый помеченный столбец.

1.2.5. Рюкзак

Даны целые числа $(w_1, \dots, w_n; c_1, \dots, c_n; K)$. Требуется максимизировать

$$\sum_{j=1}^{n} c_j x_j$$

при условии $\sum_{j=1}^{n} w_j x_j \leq K, x_j = 0,1$. [1], [2]

Не теряя общности, будем считать, что все $w_j \leq K$. Следующий алгоритм решает задачу за время $O(n^2 c)$, где c – значение оптимальной стоимости.

Обозначим $M_0 = \{(\emptyset, 0)\}$.
Для j от 1 до n
{
обозначим $M_j = \emptyset$;
для каждого элемента (S, c) из M_{j-1} добавить к M_j элемент (S, c), а также элемент $(S \cup \{j\}, c + c_j)$, если $\sum_{i \epsilon S} w_i + w_j \leq K$;
Найти в M_j пары элементов (S, c) и (T, c) с одной и той же второй компонентой. Для каждой такой пары удалить (T, c), если $\sum_{i \epsilon T} w_i \geq \sum_{i \epsilon S} w_i$, и удалить (S, c) в противном случае.
}
В качестве оптимального решения выбрать S, где (S, c) – элемент из M_n с наибольшей второй компонентой.

§1.3. Алгоритмы для действий с числами

1.3.1. Извлечение квадратного корня

Составьте алгоритм для составления программы извлечения квадратного корня из больших натуральных чисел, содержащих, например, несколько сотен цифр. Числа рассматривать как цепочечные списки десятичных цифр.

Подготовительные действия

Разбить число справа налево на группы по две цифры в каждой (первая группа может состоять и из одной цифры). Число цифр без учета первой группы обозначить через k. Из числа, образованного первой группой, извлечь целочисленный корень kor «с недостатком» и вычесть квадрат kor^2. Полученную разность, рассматриваемую как цепочечный список цифр, обозначить ap, само число kor напечатать, его удвоение $2 * kor$ обозначить al.

Шаг итерации

В качестве i выбрать наибольшую цифру такую, что $(al || i) * i \leq ap$; здесь $al || i$ – конкатенация al и i. Присвоить ap значение $ap - (al||i) * i$, al – значение $(al||i) + i$. Напечатать i в качестве очередной цифры корня и уменьшить k на 2.

Если значение k равно нулю, завершить процесс, в противном случае повторить шаг итерации.

1.3.2. Возрастающая подпоследовательность

В заданном векторе $a = (a_1 a_2, \dots, a_n)$ попарно различных чисел найти возрастающую последовательность с наибольшим количеством членов [8].

Для каждого момента времени $i = 0, 1, \dots, n-1$ обозначим

1) $S = \{a_0, a_1, \dots, a_i\}$, где a_0 – пустой элемент;
2) v_j – возрастающая последовательность с наибольшим количеством членов, оканчивающаяся в a_j, все остальные элементы которой принадлежат множеству S; $j = 1,2, \dots, n$;
3) t_j – количество элементов в v_j;
4) u_j – позиция, занимаемая в векторе a предпоследнем элементом v_j.

$u := 0; t := 1; v_j := a_j,\ j = 1,2, \dots, n.$

для i от 1 до $n-1$ выполнить

{ для j от 1 до n выполнить

если $a_j < a_j$ и $t_i = t_j$ то $\{t_j := t_j + 1; u_j := i;\}$

}

Пусть $m = t_k = \max\limits_{1 \le j \le n} t_j$.

Выделить память для вектора $b = (b_1, b_2, \dots, b_m)$.

Пока $u_k \neq 0$ выполнить

{

$b_m := a_k; m := m - 1; k := u_k;$

}

Напечатать вектор b.

1.3.3. Местонахождение точки

Для n-угольника, заданного координатами последовательных вершин $A_1(x_1, y_1), \dots, A_n(x_n, y_n)$, и для точки $M(s, t)$, не принадлежащей контуру, определить, располагается ли точка M внутри n-угольника.

1. Выбрать новые оси координат с началом в точке M и найти координаты точек A_j в новой системе координат по формулам:

$x_н = x_c - s, y_н = y_c - t.$

2. Увеличивать k от 0 с шагом 1 до тех пор, пока прямая с угловым коэффициентом k, проходящая через M, не станет свободной от точек A_j, т.е. пока для k можно указать i такое, что $y_{iн} = k * x_{iн}$.

3. Выбрать точку $L(x_0, y_0)$, где $x_0 = \max_i\{x_{iн}\} + 1, y_0 = kx_0$.

4. Определить число p пересечений отрезка LM с отрезками

$$A_1A_2, A_2A_3, \dots, A_{n-1}A_n, A_nA_1.$$

Если p четно, то M находится вне n-угольника, в противном случае – внутри n-угольника.

1.3.4. Игра Ним

Игра заключается в поочередном взятии двумя игроками камней с одной из трех куч. Определить, кто выиграет и сформулировать выигрышную стратегию игры [4].

Пусть кучи содержат a, b и c камней. Переведем эти числа в двоичную систему и выровняем их двоичные представления, дописывая слева нули в случае надобности:

$a_n \dots a_2a_1$,

$b_n \dots b_2b_1$,

$c_n \dots c_2c_1$.

Сумму элементов j-го столбца обозначим через S_j, $j = 1,2, \dots, n$. Если среди S_j имеется хотя бы одно нечетное, т.е. 1 или 3 (и только тогда), начинающий выигрывает игру при правильной стратегии.

Пусть ситуация выигрышная для начинающего, $S_j \in \{1,3\}$ и двоичный разряд в i-й позиции $j -$ го столбца равен 1. Каким должен быть правильный ход начинающего, т.е. такое взятие с одной кучи, после которого для полученных трех куч все n сумм в столбцах четные? Во-первых, он должен взять камни из i-й строки. Чтобы ответить на вопрос: сколько камней он должен взять? достаточно сказать, каким будет двоичный код i-й строки после взятия.

Для $k = j, j - 1, \dots, 1$: если S_k нечетное, в позицию k i-й строки запишем 1, в противном случае – 0.

1.3.5. Знаки операций

В выражении $j_1 \times (j_2 \times (j_3 \times (j_4 \times (j_5 \times j_6)))) = j_7$, где $j_1, j_2, \dots, j_7 \in Z^+$, заменить каждый крестик на один из знаков: + (сложение), - (вычитание), * (умножение), / (деление нацело) так, чтобы получилось верное равенство.

Рассмотрим функцию F:

$$F(k,m,n) = \begin{cases} m+n, & \text{если } k=1, \\ m-n, & \text{если } k=2, \\ m*n, & \text{если } k=3, \\ m/n, & \text{если } k=4. \end{cases}$$

и массив S из четырех элементов символьного типа: $S(1) = '+'$, $S(2) = '-'$, $S(3) = '*'$, $S(4) = '/'$.

Тогда решение дается следующим кратным циклом:

для a от 1 до 4

{для b от 1 до 4

{ для c от 1 до 4

{ для d от 1 до 4

{ для e от 1 до 4

{

если $F\left(a, j_1, F\left(b, j_2, F\left(c, j_3, F\left(d, j_4, F(e, j_5, j_6)\right)\right)\right)\right) = j_7$,

то вывести $S(a)$, $S(b)$, $S(c)$, $S(d)$, $S(e)$ и завершить алгоритм.

} } } } }

1.3.6. k-й наименьший элемент

Пусть заданы $k, n, 1 \le k \le n$ и множество S из n чисел. За $O(n)$ время найти k-й наименьший элемент множества S [3].

Алгоритм носит характер рекурсивной функции. Основное достоинство заключается в достижении оценки $O(n)$, поскольку, скажем, за $O(n^2)$ (или даже $O(nlogn)$) время можно упорядочить множество и выбрать k-й элемент с конца.

Рекурсивная функция MyMin (k, S):

Если $|S| < 50$, то упорядочить S и возвратить k-й наименьший;

разбить S на $\left\lfloor \frac{|S|}{5} \right\rfloor$ упорядоченных подмножеств по 5 элементов в каждой;

обозначить через M последовательность «середин» этих подмножеств, «середину» 5-элементных множеств можно найти за 7 сравнений;

m: = MyMin $\left(\left\lceil\frac{|M|}{2}\right\rceil, M\right)$;

Обозначить через S_1, S_2, S_3 подмножества элементов из S соответственно меньших, равных и больших, чем m;

если $|S_1| \geq k$, то возвратить MyMin (k, S_1)

иначе

{

если $|S_1| + |S_2| \geq k$, то возвратить m

иначе возвратить MyMin $(k - |S_1| - |S_2|, S_3)$

};

Конец описания функции.

Через ⌊ ⌋ и ⌈ ⌉ обозначены соответственно нижняя и верхняя целая часть числа.

1.3.7. Сортировка

Отсортировать массив из n элементов за время $O(n\,logn)$, т.е. переставить элементы в неубывающем порядке [3].

Сформулируем идею алгоритма. Легко слить два отсортированных массива $(a_1, \ldots, a_n)$ и $(b_1, \ldots, b_m)$ в единый отсортированный массив за время $O(n + m)$. Поэтому, если первая и вторая половины массива отсортированы, то легко отсортировать сам массив. Для сортировки этих двух половин можно использовать рекурсию, причем без ограничения общности можно полагать, что $n = 2^k, k \in Z^+$.

1.3.8. Алгоритм банкира

Пусть в текущей момент времени операционная система выделила однотипный ресурс каждой программе множества I в количестве $a_i, i \in I$. Каждая программа p_i заранее сообщила значение b_i ресурса, необходимое ей для завершения. Причем программа p_i возвращает ресурс лишь после получения ею всего количества a_i. Известно, что суммарное значение ресурса s меньше, чем $\sum_{i \in I} b_i$.

Спрашивается, может ли операционная система удовлетворить очередной запрос c_j, поступивший от программы p_j, без риска попасть в тупиковую ситуацию? [5]

Чтобы ответить на этот вопрос, операционная система

1. Предполагает, что запрос c_j выполнен;
2. $a_j := a_j + c_j; s := s - c_j;$
3. находит $k: \min\limits_{q \in I}(b_q - a_q) = b_k - a_k;$
4. если $s < b_k - a_k$, то алгоритм завершается отклонением запроса;

если $s \geq b_k - a_k$, то

{

$s \coloneqq s + b_k$; $I \coloneqq I \backslash \{k\}$;
если $I \neq \emptyset$, то переход к пункту 3;
если $I = \emptyset$, то запрос удовлетворяется.
}

Глава II. Алгоритмы решения некоторых классических головоломок

Материал данной главы подтверждает высказанную в [11, с. 51] идею, что изложение головоломки в терминах графов обычно позволяет свести ее решение к поиску алгоритма построения пути из «начального» узла к некоторому «терминальному» узлу графа. В качестве узлов графа, соответствующего задаче, выбираются некоторые состояния, возникающие в задаче; а ребра/дуги служат для обозначения допустимых переходов между состояниями.

В §2.1 приводится решение известной головоломки о перевозке обезьян-каннибалов, в §2.2 дается решение о переправе «ревнивых мужей», в §2.3 построен алгоритм решения головоломки о переливаниях, §2.4 посвящен задаче о числе 153. Подчеркнем, что во всех случаях задачи исследуются в «оптимизационной» формулировке, т.е. существенно более содержательно, нежели они ранее приводились в литературе.

§2.1. Построение графа для задачи о перевозке обезьян-каннибалов

Рассмотрим задачу в «оптимизационной» постановке.

Задача 1. Три человека, одна большая и две маленькие обезьяны должны переправиться с левого берега реки на правый. Имеется только одна лодка, и она вмещает не более двоих, к тому же грести умеют только люди и большая обезьяна. Как переправить всех шестерых через реку за наименьшее число пересечений реки и с соблюдением «условия безопасности»: запрещается, чтобы вместе с людьми оказалось больше обезьян, чем людей.

Введем обозначения: «ч» – человек, «О» – большая обезьяна, «о» – маленькая обезьяна. *Мобильная* группа, способная самостоятельно уплыть на лодке, должна включать гребца, поэтому мобильными являются группы, задаваемые

списками (здесь и далее элементы списка перечислены без разделительных знаков):

m[1]=«ч», m[2]=«О», m[2]= «чч», m[2]= «чо», m[2]= «чО», m[6]= «Оо»,

а группы «о» и «оо», например, не являются таковыми, как и группы «Ооо», «ччоо» и другие, где количество букв более двух. Состояние, т.е. местонахождение обезьян и людей, а также пришвартованной к берегу лодки, будем рассматривать как узел v_k связного ориентированного ациклического графа G; информация об узле v_k задается меткой $R(v_k)$ – списком букв, образованном теми персонажами, которые в данный момент времени находятся на правом берегу; метке $R(v_k)$ будем приписывать знак $Z(v_k)$: если в рассматриваемый момент времени лодка находится на правом (левом) берегу, то $Z(v_k) = « - »$ (соответственно, $Z(v_k) = « + »$).

Проверка условия безопасности для метки узла. Условие безопасности легко проверить в следующей формулировке: «в метке узла либо нет букв «ч», либо букв «ч» ровно три, либо букв «ч» столько же, сколько и букв «о», «О».

Терминальный узел. Узел с меткой «чччООо» и знаком «-« будем называть *терминальным*. Терминальный узел соответствует состоянию, когда все благополучно переправились на правый берег.

Зеркальные узлы и пути. Два узла графа v_i и v_j будем называть *зеркальными,* если их метки дополняют друг друга до списка U=«чччООо» и имеют разные знаки (запись: R(v_i) = R(v_j)) ; например, «чо» и «-ччОо». Пусть v_i и v_j – зеркальные узлы, соединенные некоторым путем: $v_i \to v_j$. Путь

… , $v_b, v_a, v_i \to v_j, v_A, v_B$, …

будем называть (v_i, v_j)-зеркальным, если зеркальны узлы v_a и v_A, v_b и v_B и т.д.

В следующем утверждении через $s(u, w)$ обозначено расстояние между узлами u и w.

Утверждение. Если в построенной части графа G присутствуют зеркальные узлы v_i и v_j, j$> i$, то максимальный (v_i, v_j)-зеркальный путь с началом в v_1 является решением. Это решение оптимально и имеет длину $2s(v_1, v_i) + s(v_i, v_j)$.

Сформулируем алгоритм.

Инициализация. Вначале G содержит лишь один узел v_1 с пустой меткой $R(v_1) =$ " " (ведь правый берег пуст); $Z(v_1) =$ « + »; узел v_1 имеет статус «не рассмотрен»; текущее число узлов $n = 1$; множество узлов V={v_1}; множество дуг E пусто; множество меток S содержит лишь пустую метку $R(v_1)$.

Начало итерации. Если все узлы графа G рассмотрены, завершить алгоритм с сообщением об отсутствии решения.

Выбрать нерассмотренный узел u с наименьшим номером.

Для i=1,…,6:

{

Если Z(u)=’-’ и m[i] содержится в R(u), то r:=R(u) - m[i].

Если Z(u)=’+’ и m[i] содержится в U-R(u), то r:=R(u) + m[i].

Если для r выполнено условие безопасности и r отсутствует в множестве меток S узлов графа G, то

{

$n := n + 1$; создать узел v_n; $V := V + v_n$; $E := E + (u, v_n)$; $R(v_n) := r$; $S := S + r$;

если $Z(u) =$ " $-$ ", то $Z(v_n) :=$ " $+$ ", иначе $Z(v_n) :=$ " $-$ ";

статус v_n:=”не рассмотрен”;

если узел v_n и некоторый узел v_m из V являются зеркальными, то вывести (v_m, v_n)-зеркальный путь и завершить алгоритм;

если узел v_n – терминальный, то вывести решение – путь из v_1 в v_n и завершить алгоритм;

}

}

статус u:= «рассмотрен»;

Перейти к следующей итерации.

Конец итерации.

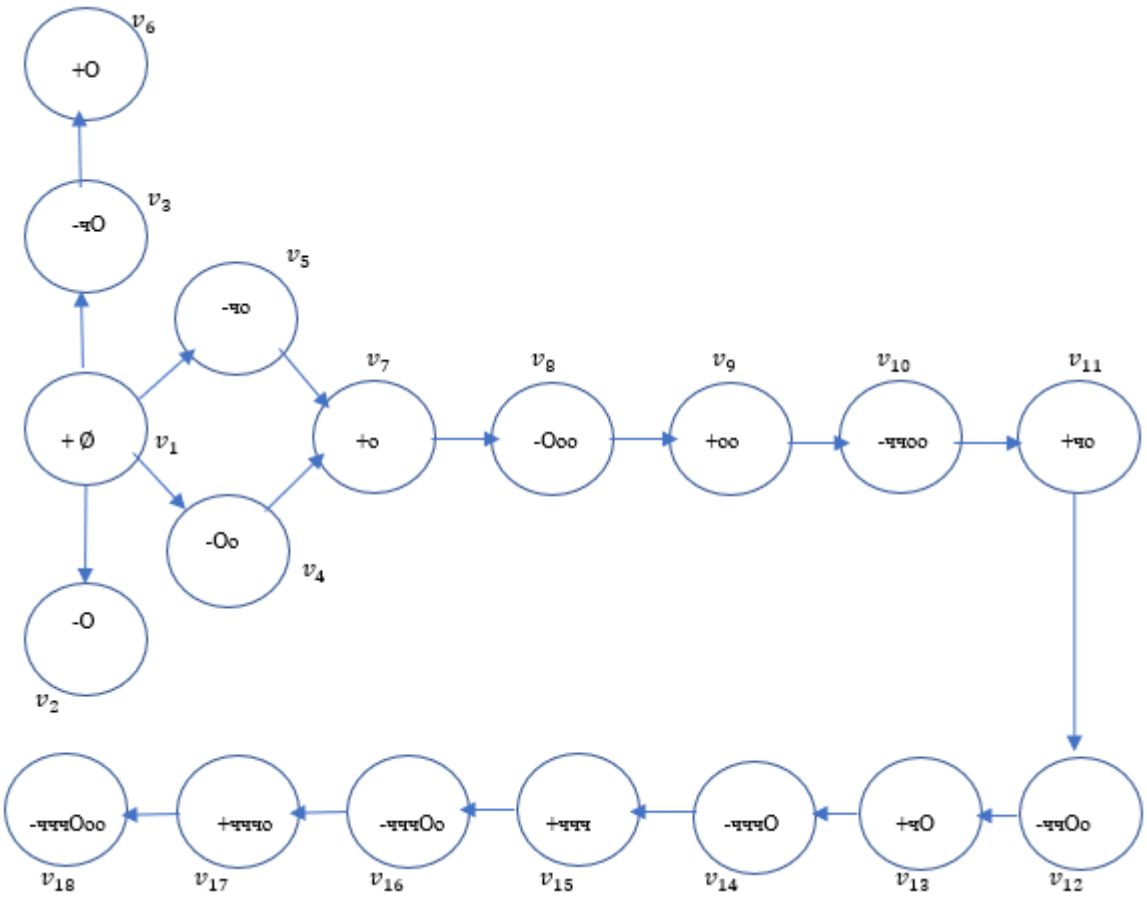

Рис. 1. Из графа для задачи о перевозке обезьян-каннибалов видно, что для решения необходимо и достаточно переплыть реку 13 раз.

Узлы v_{11} и v_{12} – зеркальные (рис. 1), и поэтому алгоритм построения графа завершился сразу после создания узлов v_1, v_4, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, достройкой (v_{11}, v_{12})-зеркального пути добавлением продолжения: v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}; метки узлов данного продолжения указаны в соответствии с определением зеркального пути (рис. 1) и определяют оптимальное решение с 13 перевозками.

§2.2. Алгоритм решения задачи "Ревнивые мужья"

Задачу "Ревнивые мужья" из [11, с. 52] рассмотрим в следующей «оптимизационной» постановке.

Задача 2. Три супружеские пары должны переправиться с левого берега реки на правый. Имеется только одна лодка, которая может выдержать одновременно только двоих. Как переправить всех шестерых, если никакой муж не оставит жену в присутствии других мужчин? При этом требуется минимизировать количество переплытий реки лодкой.

Примем буквенные обозначения: A, a, B, b, C, c, где прописная буква соответствует мужу, а одноименная строчная буква – его супруге.

Перед каждым отплытием (и после каждого прибытия) лодки состояние на берегах реки будем рассматривать как узел ориентированного графа G, а информацию о местонахождении людей задавать структурированной меткой узла – подмножествами L и R, образованными элементами множества $\{A, a, B, b, C, c\}$, которые находятся на левом (L) и правом (R) берегу соответственно; если в рассматриваемый момент времени лодка находится на левом берегу, будем добавлять знак «+» в множество L, в противном случае знак «+» будем добавлять в R. Понятно, что в смысле информативности одно из L и R избыточно, но из соображений наглядности удобно выписывать оба множества.

Вначале G содержит лишь один узел v_1 с двухстрочной меткой

+ABCabc

∅,

узел v_1 считается «не рассмотренным». Два узла графа G будем называть *эквивалентными*, если при замене местами каких-либо двух букв (отдельно – их прописных вариантов и отдельно – их строчных вариантов) в метке одного из этих узлов получится метка другого из этих узлов.

Пример эквивалентных узлов:

В самом деле, если в метке узла слева выполнить замены $B - C$ и $b - c$, то получим метку узла, расположенного справа.

Два узла графа G будем называть *зеркальными,* если при замене местами строк в метке одного из этих узлов получится узел, эквивалентный второму из этих узлов. Примеры зеркальных узлов:

1) 

2) 

Выше была описана метка, характеризующая начальный узел v_1 графа G. Как создаются новые узлы графа (и их метки) и как они соединяются?

На очередном шаге по построению ориентированного графа G:

1) выберем из нерассмотренных узлов графа G тот узел u, который имеет наименьший номер, т.е. построен раньше других;

2) каждый узел w, такой, что

а) w достижим из u выполнением одной-единственной переправы с соблюдением условия ревности - «если в лодке или на каком-либо берегу присутствуют

строчная буква x и какая-либо прописная буква, то обязательно присутствие прописной буквы X»,

б) w не эквивалентен ни одному из ранее построенных узлов графа G,

добавим к графу G вместе с дугой (u, w) в качестве нерассмотренного узла с присвоением очередного номера. После этой процедуры узел u считается рассмотренным.

Примечание. Запрет на добавление в граф узла w, соответствующего состоянию, достижимому из рассматриваемого узла u, в случае, когда граф G уже содержит эквивалентный w узел, призван исключить зацикливание.

Построение графа G завершается либо при создании «терминального» узла с меткой

∅

+ABCabc

(этот узел соответствует состоянию, когда все супружеские пары перевезены на правый берег), либо при возникновении ситуации, когда терминальный узел не создан, и все узлы графа G рассмотрены (последнее равносильно отсутствию решения).

Из рис. 2 видно, что алгоритм построения графа завершается созданием терминального узла v_{14}. Любой путь из узла v_1 до узла v_{14} является решением. Таких путей – четыре, длина каждого из них равна 11. Таким образом, наименьшее количество пересечений реки лодкой для достижения решения равно 11. Выпишем один из этих решений:

$$v_1, v_2, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{14}.$$

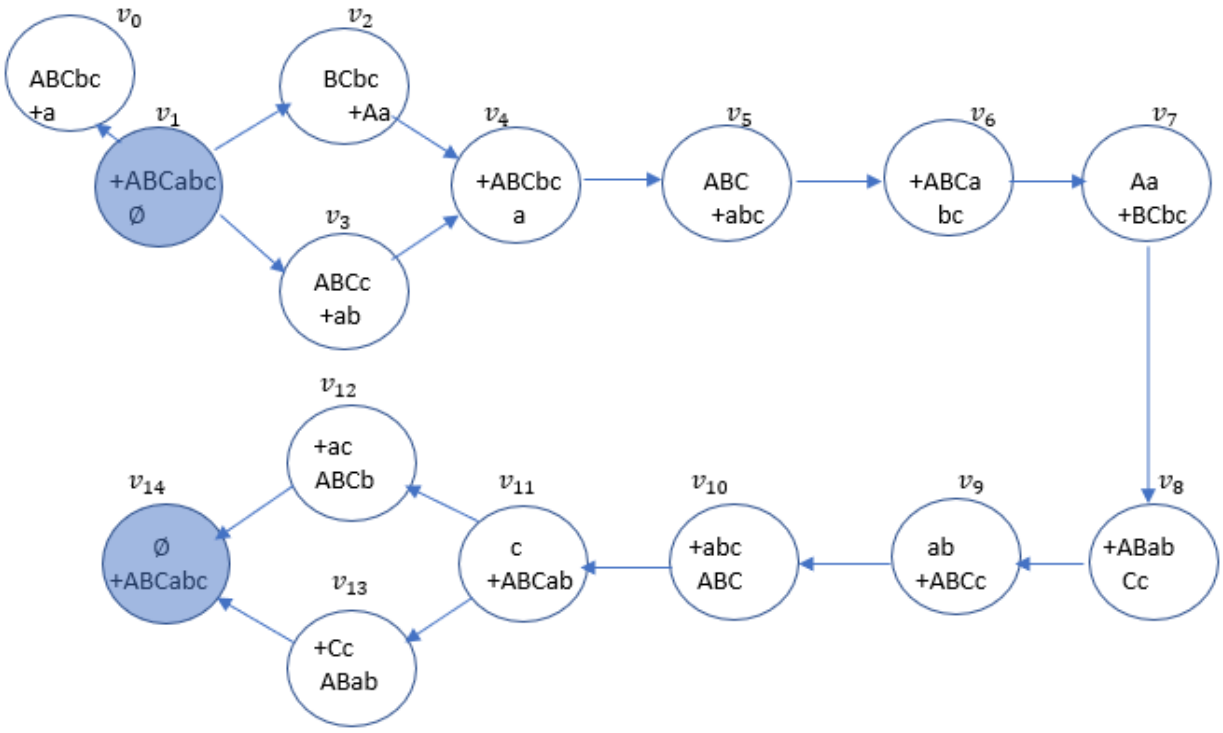

Рис. 2. Граф для задачи о ревнивых мужьях.

В следующем утверждении через $s(u, w)$ обозначено расстояние между узлами u и w.

Утверждение. Если в построенной части графа G присутствуют зеркальные узлы v_m и v_n, $n > m$, то в графе G существует терминальный узел v_k и

$$s(v_1, v_k) \leq 2s(v_1, v_m) + s(v_m, v_n).$$

Очевидно, вершины v_7 и v_8 (см. рис. 2) являются зеркальными; s(v_1, v_7)=5, $s(v_7, v_8) = 1$. В соответствии с Утверждением существует терминальный узел (в нашем случае v_{14}) и $s(v_1, v_{14}) \leq 2s(v_1, v_7) + s(v_7, v_8) = 10 + 1 = 11$.

§2.3. Построение графа для задачи о переливаниях

Многочисленные задачи о переливаниях при внешнем сходстве формулировок принципиально различаются по сути (см., например, [12]). Из обширной литературы, посвященной данной тематике, укажем еще на [13].

Тривиальным и наиболее известным примером задач такого рода является вопрос о том, как, имея пустые сосуды в 3 и 5 литров и полный 8-литровый сосуд, отмерить ровно 4 литра жидкости. В данном разделе мы предложим алгоритм решения задачи с тремя сосудами о переливаниях в общем случае, причем – за наименьшее число переливаний.

Задача 3. Даны три сосуда вместимости $A[1]$, $A[2]$ и $A[3]$, содержащие $a[1], a[2]$ и $a[3]$ литров жидкости соответственно. Для заданного d требуется получить d литров в каком-либо из этих сосудов, выполнив наименьшее число переливаний. Все рассматриваемые числа – натуральные.

Под *состоянием сосудов* будем понимать упорядоченный набор из трех элементов – текущих значений объемов жидкости, содержащейся в текущий момент времени в 1-м, 2-м и 3-м сосудах соответственно. Суть предлагаемого алгоритма заключается в построении связного ациклического ориентированного графа G, узлы которого соответствуют состояниям, и от одного узла («родителя») к другому узлу («потомку») проводится дуга стоимости 1 тогда и только тогда, когда переход между соответствующими состояниями может быть осуществлен выполнением одного переливания.

Построение графа G начинается с источника – узла, соответствующего исходному состоянию ($a[1]$, $a[2]$, $a[3]$) и имеющего статус «не рассмотрен»; сначала будут построены узлы, удаленные от источника на расстояние 1 (его потомки), затем – удаленные на расстояние 2, и т.д. Построение графа завершается лишь когда будет построен узел, соответствующий состоянию, хотя бы один элемент которого равен значению d (такой узел будем называть *терминальным*), или – все узлы графа рассмотрены, но терминальный узел не найден (последнее означает отсутствие решения). Если терминальный узел обнаружен, то решением задачи служит кратчайший путь от источника до него.

Рассмотрим действие по построению потомков для нерассмотренного узла $x = (a[1], a[2], a[3])$. Пусть i, j, k – произвольная перестановка элементов 1, 2 и 3 – номеров сосудов. Очевидно, что объем жидкости, переливаемого из i-го сосуда в j-й, равен меньшему из двух значений – содержащегося в сосуде i и недостающего для наполнения сосуда j: $\alpha_{ij} = min\,(a[i], A[j] - a[j])$. Если α_{ij} >0, то операция переливания из сосуда i в сосуд j корректна; если граф не содержит узел – результат данной операции:

($a[i]$- α_{ij}, $a[j]$+ α_{ij} , $a[k]$),

то создается такой узел со статусом «не рассмотрен»; аналогично определяются и остальные потомки узла x и к каждому из этих потомков проводится дуга из x (количество этих потомков, очевидно, не превышает 6). После описанной процедуры узел x считается рассмотренным.

На основе описанного алгоритма создано программное обеспечение [14]. В качестве примера его применения приведем решение для случая:

A[1]=25, A[2]=18, A[3]=7, a[1]=0, a[2]=18, a[3]=7, d=9.

На рис. 2 римскими цифрами I, II и III обозначены номера сосудов; стрелка показывает, из какого сосуда (и в какой сосуд) выполняется переливание, а число рядом со стрелкой – сколько литров переливается; в последних трех ячейках той же строки указано, сколько литров в результате данного переливания окажется (слева направо) в 1-м, 2-м и 3-м сосудах соответственно. Для решения потребовалось 23 переливания.

				0	18	7		12)	II	7=>	III	17	1	7
01)	I	<= 7	III	7	18	0		13)	I	<= 7	III	24	1	0
02)	II	7=>	III	7	11	7		14)	II	7=>	III	24	0	1
03)	I	<= 7	III	14	11	0		15)	I	18=>	II	6	18	1
04)	II	7=>	III	14	4	7		16)	II	6=>	III	6	12	7
05)	I	<= 7	III	21	4	0		17)	I	<= 7	III	13	12	0
06)	II	4=>	III	21	0	4		18)	II	7=>	III	13	5	7
07)	I	18=>	II	3	18	4		19)	I	<= 7	III	20	5	0
08)	II	3=>	III	3	15	7		20)	II	5=>	III	20	0	5
09)	I	<=7	III	10	15	0		21)	I	18=>	II	2	18	5
10)	II	7=>	III	10	8	7		22)	II	2=>	III	2	16	7
11)	I	<= 7	III	17	8	0		23)	I	<= 9	III	9	16	0

Рис. 3. Решение, предложенное компьютерной программой для d=9.

§2.4. «Компьютерное» доказательство существования терминального узла

Для натурального числа a обозначим через $f(a)$ сумму кубов цифр в десятичном представлении числа a. Для натурального a_1, кратного 3, обозначим через $F(a_1)$ последовательность, построенную по следующему правилу: первый элемент последовательности равен a_1; $a_{i+1} = f(a_i)$, $i = 1, 2, \ldots$

Примеры:

1) 3, 27, 351, 153, …

2) 6, 216, 225, 141, 66, 432, 99, 1458, 702, 351, 153, …

3) 99, 1458, 702, 351, 153, …

Сопоставим последовательности $F(a_1)$ ориентированный граф G, узлы которого расположены в точках $a_1, a_2, \ldots$ действительной оси, и от каждого a_i проведем дугу к a_{i+1}; наконец, начальным и терминальным узлами назовем узлы, соответствующие числам a_1 и 153. Требование головоломки, сформулированной в [15, с. 30] в несколько иных терминах (см. также [15], с. 117 и 188), заключающееся в

доказательстве присутствия в графе G терминального узла, мы дополним требованием оценить длину пути от начального узла до терминального.

Лемма 1. Каждый элемент последовательности $F(a_1)$ кратен 3.

Доказательство. Достаточно доказать, что для любого натурального числа a, кратного 3, и с десятичной записью $a = b_1 b_2 \dots b_x$ сумма $b_1^3 + b_2^3 + \cdots + b_x^3$ кратна 3.Согласно признаку делимости на 3, сумма $b_1 + b_2 + \cdots + b_x$ кратна 3, поэтому значение

$$(b_1 + b_2 + \cdots + b_x)^3 \qquad (1)$$

также кратно 3.

Для каждой тройки попарно различных индексов i, j, k из множества $\{1, 2, \dots, x\}$ выражение (1) содержит каждое слагаемое вида b_i^3 один раз, каждое слагаемое вида $b_i b_i b_j$ – три раза, а каждое слагаемое вида $b_i b_j b_k$ – шесть раз:

$$(b_1 + b_2 + \cdots + b_x)^3 = b_1^3 + b_2^3 + \cdots + b_x^3 + 3 \cdot A + 6 \cdot B, \qquad (2)$$

где A и B – некоторые целые числа.

Т.к. левая часть равенства (2) кратна 3, то $b_1^3 + b_2^3 + \cdots + b_x^3$ кратно 3, что т.д.

Лемма 2. Для всякого натурального a, такого, что a кратно 3 и $a \geq 1902$, справедливо неравенство $f(a) < a$.

Доказательство. Количество цифр в десятичной записи числа a обозначим через x: $a = b_1 b_2 \dots b_x$. Очевидно,

$$10^{x-1} \leq a. \qquad (3)$$

$$f(a) = b_1^3 + b_2^3 + \cdots + b_x^3 \leq 9^3 + 9^3 + \cdots + 9^3 = 729 \cdot x. \qquad (4)$$

Легко показать, что при $x \geq 5$

$$729x \leq 10^{x-1}. \qquad (5)$$

Из (3), (4), (5) следует утверждение леммы для $a > 9999$. В справедливости утверждения леммы для a=1092, 1095, …, 9996, 9999 легко убедиться с помощью элементарной компьютерной программы:

738<1902, 855<1905, 1242<1908, …, 2214<9993, 2403<9996, 2916<9999.

Лемма доказана.

Из Леммы 2 следует, что для натурального a_1, кратного 3 и $a_1 \geq 1902$, найдется элемент a_k последовательности $F(a_1)$, такой, что $a_k < 1902$; наименьшее из таких k обозначим через $ind_{1902}(a_1)$.

В справедливости следующей леммы также легко убедиться при помощи компьютерной программы.

Лемма 3. При каждом $a_1 = 3, 6, \ldots, 1902$ в каждой последовательности $F(a_1)$ элемент a_{11} равен 153.

Из лемм 1-3 немедленно вытекает следующая теорема.

Теорема. Для любого натурального a_1, кратного 3, узел графа G с номером 10+ $ind_{1902}(a_1)$ является терминальным.

Литература

1. Х. Пападимитриу, К. Стайглиц, Комбинаторная оптимизация. -М.: Мир, 1985. – 510 с.

2. М. Гэри, Д. Джонсон. Вычислительные машины и труднорешаемые задачи. -М.: Мир, 1982. – 416 с.

3. А. Ахо, Дж. Хопкрофт, Дж. Ульман. Построение и анализ вычислительных алгоритмов. -М.: Мир, 1979. – 535 с.

4. С. А. Абрамов. Математические построения и программирование. -М.: Наука, 1978. – 192 с.

5. П. Кейлингерт. Элементы операционных система. -М.: Мир, 1985.

6. О. Оре теория графов. -М.: Наука, 1980.

7. Л. Р. Форд, Д. Р. Фалкерсон. Потоки в сетях. -М.: Мир, 1966. – 277 с.

8. Основы информатики и вычислительной техники. -М.: Просвещение, 1986.

9. Е. М. Кудрявцев. Исследование операций в задачах, алгоритмах и программах. -М.: Радио и связь, 1984. – 184 с.

10. Kaufmann A. Graphs, dynamic programming and finite fames, Academic Press, New York. 1967.

11. О. Оре. Теория графов. – 2-е изд. – М.: Наука, Главная редакция физико-математической литературы, 1980. – 336 с.

12. С.Б. Гашков. Системы счисления и их применение. – М.: Издательство МЦНМО, 2012. – 68 с.

13. И.Ф. Шарыгин. Математический винегрет. - М.: Агентство "ОРИОН", 1991.

14. А.М. Магомедов, Т.И. Шарапудинов. Программа решения задачи об оптимальных переливаниях с ограничениями для случая трех сосудов // Свидетельство о государственной регистрации программы для ЭВМ № 2019664676 от 12.11.2019.

15. Ж. Арсак. Программирование игр и головоломок. – М.: Издательство «Наука», 1990. – 224 с.