Московский государственный технический университет
имени Н.Э. Баумана

И.П. Иванов, А.Ю. Голубков,
С.Ю. Скоробогатов

СБОРНИК ЗАДАЧ ПО КУРСУ «ДИСКРЕТНАЯ МАТЕМАТИКА»

Методические указания

Москва
Издательство МГТУ им. Н.Э. Баумана
2013

УДК 519
ББК 22.176
И20

Рецензент *П.Г. Ключарёв*

Иванов И. П.

И20 Сборник задач по курсу «Дискретная математика» : метод. указания / И. П. Иванов, А. Ю. Голубков, С. Ю. Скоробогатов. — М. : Изд-во МГТУ им. Н.Э. Баумана, 2013. — 31, [1] с. : ил.

ISBN 978-5-7038-3682-8

Приведены задачи по курсу «Дискретная математика», относящиеся к теории графов и теории автоматов.

Для студентов, обучающихся по направлению подготовки бакалавров «Прикладная математика и информатика».

Рекомендовано методической комиссией факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана.

УДК 519
ББК 22.176

Учебное издание

Иванов Игорь Потапович
Голубков Артем Юрьевич
Скоробогатов Сергей Юрьевич

СБОРНИК ЗАДАЧ ПО КУРСУ «ДИСКРЕТНАЯ МАТЕМАТИКА»

Редактор *С.А. Серебрякова*
Корректор *Р.В. Царева*
Компьютерная верстка *В.И. Товстоног*

Подписано в печать 08.05.2013. Формат 60×84/16.
Усл. печ. л. 1,86. Тираж 100 экз. Изд. № 49.
Заказ

Издательство МГТУ им. Н.Э. Баумана.
Типография МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.

ISBN 978-5-7038-3682-8

ВВЕДЕНИЕ

Курс «Дискретная математика» является логическим продолжением курса «Алгоритмы и структуры данных». Он посвящен изучению таких математических объектов, как графы и автоматы, способов их представления в памяти компьютера и алгоритмов для решения связанных с ними задач.

В отличие от курса алгоритмов, в котором для решения задач предлагается использовать язык C, курс дискретной математики рассчитан на применение языка Go, в котором автоматическое управление памятью сочетается с отсутствием жесткой объектной ориентации, характерной для языков Java и C#.

Сборник содержит 28 задач, в которых рассмотрена бóльшая часть алгоритмов, изучаемых в курсе «Дискретная математика».

Курс разбит на три модуля. Домашнее задание по модулю «Программирование на языке Go» состоит из восьми задач, приведенных в разделе 1 данного сборника. Раздел 2 состоит из 12 задач, которые нужно решить в модуле «Графы». И, наконец, раздел 3 содержит восемь задач домашнего задания по модулю «Автоматы».

1. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ GO

1.1. Быстрая сортировка

Реализуйте алгоритм быстрой сортировки произвольных данных в функции

```
func qsort(n int,
           less func(i, j int) bool,
           swap func(i, j int)) {
    ...
}
```

Параметрами функции qsort являются:

n — число сортируемых записей,

less — функция сравнения i-й и j-й записей;

swap — функция обмена i-й и j-й записей.

Составьте программу qsort.go, демонстрирующую работоспособность функцию qsort.

1.2. Кодирование и раскодирование текста в кодировке UTF-8

Реализуйте алгоритмы перевода текста из кодировки UTF-32 в UTF-8 и обратно. Алгоритмы должны быть оформлены в виде двух функций:

```
func encode(utf32 []rune) []byte {
    ...
}

func decode(utf8 []byte) []rune {
    ...
}
```

Параметром функции encode служит текст в виде массива кодовых точек, функция возвращает образ этого текста в кодировке

UTF-8. Функция decode выполняет обратное преобразование. Правила кодирования текста в кодировке UTF-8 приведены в табл. 1.

Таблица 1

Кодирование символов стандарта Unicode в кодировке UTF-8

Код символа в Unicode	1	2	3	4
0 0000 0000 0000 0xxx xxxx	0xxx xxxx	–	–	–
0 0000 0000 0yyy yyxx xxxx	110y yyyy	10xx xxxx	–	–
0 0000 zzzz yyyy yyxx xxxx	1110 zzzz	10yy yyyy	10xx xxxx	–
u uuzz zzzz yyyy yyxx xxxx	1111 0uuu	10zz zzzz	10yy yyyy	10xx xxxx

Составьте программу utf8.go, демонстрирующую работоспособность функций decode и encode. Проверьте правильность результатов работы функций с помощью встроенных средств Go.

1.3. «Длинное» сложение

Реализуйте алгоритм сложения натуральных чисел, представленных в виде массива цифр в системе счисления по основанию p, где $1 < p < 2^{30}$. Порядок цифр в массиве – little-endian (младшая цифра располагается в нулевом элементе массива):

```
func add(a, b []int32, p int) []int32 {
    ...
}
```

Составьте программу add.go, демонстрирующую работоспособность функции add.

1.4. Решение систем линейных алгебраических уравнений в рациональных числах

Составьте программу gauss.go, выполняющую решение системы линейных алгебраических уравнений в рациональных числах методом Гаусса.

Формат входных данных. Программа должна считывать из входного потока число N уравнений ($1 \le N \le 5$) и матрицу системы размером $N \times (N+1)$. Матрица содержит целые числа от -9 до 9.

Формат результата работы программы. Если система не имеет решения, следует выводить фразу «No solution».

Если решение существует, программа должна выводить в выходной поток N значений переменных, каждое из которых представляет собой нормализованное рациональное число, записанное в виде n/d, где n – числитель дроби, d – знаменатель дроби.

Пример. Если во входном потоке программы находятся значения

```
3
-4  -1   8   2
 7  -7   7   3
 5  -1  -4   7
```

то на выходе мы получаем

```
377/21
214/7
274/21
```

1.5. Вычисление выражений в польской записи

Польская запись — это форма записи арифметических, логических и алгебраических выражений, в которой операция располагается слева от операндов. Выражения в польской записи могут обходиться без скобок, однако мы оставим скобки для наглядности.

Например, выражение $5 \cdot (3 + 4)$ в польской записи выглядит как

```
(* 5 (+ 3 4))
```

Пусть в нашем случае выражения состоят из чисел от 0 до 9, круглых скобок и трех знаков операций: плюс, минус и звездочка (умножить). Составьте программу polish.go, вычисляющую значение выражения.

1.6. Экономное вычисление выражений в польской записи

Пусть выражения в польской записи состоят из имен переменных (от a до z), круглых скобок и трех знаков операций: #, $ и @ (смысл операций мы определять не будем).

Выражения могут содержать повторяющиеся подвыражения. Экономное вычисление таких выражений подразумевает, что повторяющиеся подвыражения вычисляются только один раз.

Составьте программу econom.go, вычисляющую число операций, которые нужно выполнить для экономного вычисления выражения. Примеры работы программы приведены в табл. 2.

Таблица 2

Набор тестов для программы экономного вычисления выражений в польской записи

Выражение	Число операций
x	0
($xy)	1
($(@ab)c)	2
(#i($jk))	2
(#($ab)($ab))	2
(@(#ab)($ab))	3
(#($a($b($cd)))(@($b($cd))($a($b($cd)))))	5
(#($(#xy)($(#ab)(#ab)))(@z($(#ab)(#ab))))	6

1.7. Лексический анализ

Пусть *идентификатор* — это последовательность латинских букв и цифр, начинающаяся с буквы.

Известно, что в некоторой строке записаны идентификаторы, разделенные произвольным числом пробелов. При этом строка может начинаться и заканчиваться произвольным числом пробелов. Назовем такую строку *предложением*.

Лексический анализ предложения заключается в выделении из него последовательности записанных в нем идентификаторов. В результате лексического анализа получают массив целых чисел, каждое из которых соответствует одному из идентификаторов. Целые числа назначают идентификаторам произвольно, но так, чтобы разным идентификаторам соответствовали разные числа, а равным идентификаторам — одинаковые числа.

Пример. Пусть дано предложение

```
alpha x1 beta alpha x1 y
```

Тогда на выходе лексического анализатора может получиться последовательность чисел

```
1 2 3 1 2 4
```

Здесь число 1 соответствует идентификатору alpha, число 2 — идентификатору x1, число 3 — идентификатору beta, а число 4 — идентификатору y.

Разработайте функцию lex, выполняющую лексический анализ предложения

```
func lex(sentence string, array AssocArray)
[]int {
    ...
}
```

Первым параметром функции lex является предложение, второй параметр задает ассоциативный массив, который должен быть использован внутри функции для хранения соответствия идентификаторов и целых чисел.

Ассоциативный массив, который можно передавать в функцию lex, должен реализовывать интерфейс AssocArray:

```
type AssocArray interface {
    Assign(s string, x int)
    Lookup(s string) (x int, exists bool)
}
```

Метод Assign добавляет в ассоциативный массив словарную пару.

Метод Lookup выполняет поиск словарной пары по ключу и возвращает два значения: x и exists. Если словарной пары с указанным ключом в массиве нет, то exists принимает значение false. В противном случае exists равен true, а x — связанное с ключом значение.

Составьте программу lex.go, демонстрирующую работоспособность функции lex. В качестве ассоциативных массивов для тестирования функции lex нужно использовать список с пропусками и АВЛ-дерево.

1.8. Арифметическое выражение

Составьте программу arith.go, вычисляющую значение скобочного арифметического выражения.

Арифметическое выражение считывается из аргументов командной строки и представляет собой строку, содержащую целые числа от 0 до 2^{31}, знаки четырех арифметических операций, имена переменных, круглые скобки и пробелы. Имена переменных представляют собой последовательности латинских букв и цифр, начинающиеся с буквы. Арифметические операции — целочисленные, т. е. 5/2 дает 2, а не 2.5. Пробелы не несут никакого смысла и должны игнорироваться программой.

Примеры выражений, которые могут быть поданы на вход программе:

```
-x * (x+10) * (128/x-5)
Length * Height * Depth
```

Грамматика для арифметических выражений записывается как

```
<E>  ::= <E> + <T> | <E> - <T> | <T>.
<T>  ::= <T> * <F> | <T> / <F> | <F>.
<F>  ::= <number> | <var> | ( <E> ) | - <F>.
```

Здесь number и var обозначают множества терминальных символов, соответствующих числам и именам переменных.

После удаления левой рекурсии получается LL(1)-грамматика

```
<E>  ::= <T> <E'>.
<E'> ::= + <T> <E'> | - <T> <E'> | .
<T>  ::= <F> <T'>.
<T'> ::= * <F> <T'> | / <F> <T'> | .
<F>  ::= <number> | <var> | ( <E> ) | - <F>.
```

Алгоритм вычисления значения выражения должен быть разбит на две части: лексический анализатор и синтаксический анализатор.

Задачей лексического анализатора является разбиение арифметического выражения на лексемы, т. е. выделение в нем числовых констант, имен переменных, знаков операций и круглых скобок. Пробелы во время лексического анализа пропускают. Каждую лексему представляют в виде экземпляра структуры Lexem:

```
type Lexem struct {
    Tag
    Image string
}
```

Здесь Image — подстрока арифметического выражения, соответствующая распознанной лексеме, Tag — целочисленный код, задающий тип лексемы:

```
type Tag int

const (
  ERROR Tag = 1 << iota // @Неправильная лексема@
  NUMBER                // @Целое число@
  VAR                   // @Имя переменной@
  PLUS                  // @Знак +@
  MINUS                 // @Знак -@
  MUL                   // @Знак *@
  DIV                   // @Знак /@
  LPAREN               // @Левая круглая скобка@
  RPAREN                // @Правая круглая скобка@
)
```

Значения кодов лексем образуют степени числа 2, и это позволяет проверять принадлежность текущей лексемы некоторому множеству лексем. Например, если lx — переменная, в которой хранится текущая лексема, то проверить, является ли она аддитивной арифметической операцией, можно следующим образом:

```
if lx.Tag & (PLUS | MINUS) != 0 {
    ...
}
```

Лексический анализатор должен выполняться в отдельной go-программе, получающей на вход арифметическое выражение и канал лексем и отправляющей распознанные лексемы в этот канал:

```
func lexer(expr string, lexems chan Lexem) {
    ...
}
```

Синтаксический анализатор должен быть составлен методом рекурсивного спуска по приведенной LL(1)-грамматике.

Программа должна запрашивать у пользователя значения переменных, входящих в выражение, и выводить в стандартный поток вывода значение выражения или сообщение «error», если выражение содержит синтаксическую ошибку. Если некоторая переменная входит в выражение многократно, ее значение все равно должно запрашиваться только один раз.

2. ГРАФЫ

2.1. Граф делителей

Составьте программу dividers.go, выполняющую построение графа делителей натурального числа x.

Вершинами графа делителей являются все делители числа x. Ребро соединяет вершины u и v в том случае, если u делится на v, и не существует такого w, что u делится на w, и w делится на v. Пример графа делителей изображен на рис. 1.

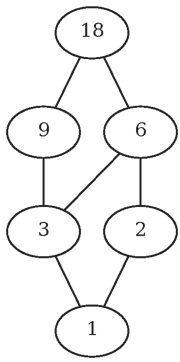

Рис. 1. Граф делителей числа 18

Число x должно считываться со стандартного потока ввода. Граф делителей, описанный на языке DOT, должен выводиться в стандартный поток вывода.

2.2. Подготовка экспедиции на Марс

Пусть N кандидатов готовятся к двум космическим экспедициям на Марс. Поскольку экспедиции будут продолжаться несколько лет, а их участники окажутся в замкнутом пространстве небольшого объема, важное значение приобретает психологическая совместимость членов экипажа. Путем тестирования были установлены пары кандидатов, присутствие которых в одной и той же экспедиции было бы нежелательным. Результаты тестирования отражены в таблице размера $N \times N$. Если на пересечении i-й строки и j-го

столбца таблицы находится знак «+», то участие i-го и j-го кандидатов в одной экспедиции нежелательно.

Составьте программу mars.go, разделяющую кандидатов на две группы для участия в экспедициях. Если такое разделение невозможно, программа должна выводить сообщение «No solution». В противном случае программа должна выводить номера кандидатов, принадлежащих первой группе. Первой группой мы будем считать группу, в которой меньше кандидатов.

Естественно, в результате работы хорошо написанной программы размеры групп не должны очень сильно различаться. Поэтому если возможно несколько разбиений на группы, программа должна выбирать разбиение с минимальной разницей количеств кандидатов в группах. При этом в случае, если разбиений с минимальной разницей все равно получается несколько, для определенности выбирают разбиение, в котором первая группа лексикографически меньше, чем первые группы остальных разбиений.

Программа должна считывать со стандартного потока ввода число кандидатов и матрицу размера $N \times N$. Например, для входных данных

```
8
- - + - - - - -
- - - + - - - -
+ - - - - - - +
- + - - - + - -
- - - - - - - -
- - - + - - - -
- - - - - - - +
- - + - - - + -
```

программа должна выводить номера

```
1 2 6 8
```

2.3. Компоненты связности

Составьте программу components.go, определяющую число компонент связности в неориентированном простом графе.

Программа должна считывать со стандартного потока ввода число вершин графа N, число ребер M и данные о ребрах графа. При этом каждое ребро кодируется номерами инцидентных ему вершин u и v такими, что $0 \leq u, v < N - 1$.

Например, для входных данных

```
7
8
0 1
0 5
1 5
1 4
5 4
2 3
3 6
```

программа должна выводить число 2.

Программа должна использовать представление графа в виде списка инцидентности.

2.4. Прямой порядок вершин

Составьте программу preorder.go, перечисляющую вершины связного неориентированного простого графа в прямом порядке относительно обхода в глубину из заданной вершины v.

Прямой порядок вершин получается, если при обходе графа в глубину выводить номера вершин, которые мы посещаем в первый раз.

Программа должна считывать со стандартного потока ввода число вершин графа N, число ребер M, номер вершины v, с которой начинается обход в глубину, и данные о ребрах графа. При этом каждое ребро кодируется номерами инцидентных ему вершин u и v такими, что $0 \leq u, v < N - 1$.

Чтобы гарантировать уникальность ответа, условимся, что алгоритм обхода в глубину, находясь в некоторой вершине x, должен посещать смежные с x вершины в порядке возрастания их номеров.

Например, для входных данных

```
10
10
4
0 3
3 9
3 2
2 4
4 7
```

```
7 5
5 2
4 6
4 8
1 5
```

программа должна выводить данные

```
4 2 3 0 9 5 1 7 6 8
```

Программа должна использовать представление графа в виде матрицы смежности.

2.5. Маршрут, содержащий все ребра мультиграфа

Составьте программу route.go, вычисляющую маршрут, по которому проходит алгоритм обхода связного неориентированного мультиграфа из заданной вершины v. Этот маршрут, очевидно, должен содержать все ребра мультиграфа.

Программа должна считывать со стандартного потока ввода число вершин графа N, число ребер M, номер вершины v, с которой начинается обход в глубину, и данные о ребрах графа. При этом каждое ребро кодируется номерами инцидентных ему вершин u и v такими, что $0 \leq u, v < N - 1$, и уникальным строковым атрибутом s, который представляет собой последовательность латинских букв.

В качестве ответа программа должна выводить последовательность атрибутов ребер маршрута.

Чтобы гарантировать уникальность ответа, условимся, что алгоритм обхода в глубину, находясь в некоторой вершине x, должен должен посещать смежные с x вершины в порядке возрастания их номеров.

Например, для входных данных

```
5
9
0
0 1 a
0 2 b
2 1 d
1 1 c
1 3 e
2 4 g
3 4 h
```

```
2 3 f
2 3 i
```

программа должна выводить последовательность

```
a c d b b f e e i g h
```

Программа должна использовать представление графа в виде списка инцидентности.

2.6. Дорожки в парке

Составьте программу kruskal.go, реализующую алгоритм Крускала для вычисления минимальной суммарной длины дорожек в парке аттракционов. Дорожки должны быть проложены таким образом, чтобы между любыми двумя аттракционами существовал маршрут.

Программа должна считывать со стандартного потока ввода число аттракционов и их координаты. При этом координаты каждого аттракциона задаются парой целых чисел (в декартовой системе).

Программа должна выводить в стандартный поток вывода минимальную суммарную длину дорожек с точностью до двух знаков после запятой.

Например, для входных данных

```
12
2 4
2 5
3 4
3 5
6 5
6 6
7 5
7 6
5 1
5 2
6 1
6 2
```

программа должна выводить число 14,83.

2.7. Телефонные линии

На строительном участке нужно создать телефонную сеть, соединяющую все бытовки. Для того чтобы телефонные линии не мешали строительству, их решили проводить вдоль дорог. Составьте

программу prim.go, реализующую алгоритм Прима для вычисления минимальной общей длины телефонных линий для указанной конфигурации участка. Граф конфигурации участка должен быть представлен в программе в виде списка инцидентности.

Программа должна считывать со стандартного потока ввода число бытовок N, число дорог M, соединяющих бытовки, и информацию об этих дорогах. При этом каждая дорога задается тремя целыми числами u, v и len, где u и v — номера соединяемых дорогой бытовок ($0 \leq u, v < N$), а len — длина дороги.

Программа должна выводить в стандартный поток вывода минимальную общую длину телефонных линий.

Например, для входных данных

```
7
10
0 1 200
1 2 150
0 3 100
1 4 170
1 5 180
2 5 100
3 4 240
3 6 380
4 6 210
5 6 260
```

программа должна выводить число 930.

2.8. Компоненты сильной связности

Составьте программу tarjan.go, определяющую число компонент сильной связности в орграфе.

Программа должна считывать со стандартного потока ввода число вершин графа N, число дуг M и данные о дугах графа. При этом каждая дуга кодируется парой чисел u и v, где u — номер вершины, из которой дуга исходит, а v — номер вершины, в которую дуга входит. Вершины нумеруются, начиная с нуля.

Например, для входных данных

```
6
7
0 1
1 2
2 0
```

```
3 4
4 5
5 3
1 3
```

программа должна выводить число 2.

Программа должна использовать представление графа в виде списка инцидентности.

2.9. Конденсация орграфа

Пусть $S_1, S_2, \ldots, S_k$ — компоненты сильной связности орграфа G. *Конденсацией* орграфа G называется орграф $G^\star = \langle V^\star, E^\star \rangle$, множеством вершин $V^\star$ которого служит множество $\{S_1, S_2, \ldots, S_k\}$, а дуга $\langle S_i, S_j \rangle$ является элементом множества $E^\star$, если в орграфе G есть по крайней мере одна дуга, исходящая из некоторой вершины компоненты S_i и входящая в одну из вершин компоненты S_j.

Составьте программу condense.go, выполняющую построение конденсации заданного орграфа.

Программа должна считывать со стандартного потока ввода число вершин орграфа N, число дуг M и данные о дугах орграфа. При этом каждая дуга кодируется парой чисел u и v, где u — номер вершины, из которой дуга исходит, а v — номер вершины, в которую дуга входит. Вершины нумеруются, начиная с нуля.

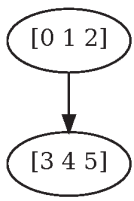

Рис. 2. Пример конденсации орграфа

Построенная конденсация орграфа, описанная на языке DOT, должна выводиться в стандартный поток вывода. При этом каждая компонента сильной связности в вершине конденсации должна быть помечена номерами вершин орграфа, которые в нее входят.

Например, для входных данных

```
6
7
0 1
1 2
2 0
3 4
```

```
4 5
5 3
1 3
```

программа должна порождать граф, изображенный на рис. 2.

2.10. База орграфа

База — это подмножество вершин орграфа, обладающее следующими свойствами:

1) каждая вершина орграфа достижима по крайней мере из одной вершины базы;

2) в базе нет вершин, достижимых из других вершин базы.

Очевидно, что в базе не может быть двух вершин, принадлежащих одной и той же компоненте сильной связности.

Также нетрудно доказать, что в ациклическом орграфе существует только одна база. Она состоит из всех вершин с полустепенью захода, равной нулю.

С учетом вышесказанного поиск баз в орграфе можно проводить в следующем порядке:

1) найти все компоненты сильной связности орграфа;

2) построить его конденсацию;

3) найти базу конденсации;

4) из каждой компоненты сильной связности, образующей вершину базы конденсации, взять по одной вершине.

Составьте программу base.go, вычисляющую базу заданного орграфа.

Программа должна считывать со стандартного потока ввода число вершин орграфа N, число дуг M и данные о дугах орграфа. При этом каждая дуга кодируется парой чисел u и v, где u – номер вершины, из которой дуга исходит, а v — номер вершины, в которую дуга входит. Вершины нумеруются, начиная с нуля.

Для обеспечения уникальности ответа из компоненты сильной связности, образующей вершину базы конденсации, следует брать вершину с минимальным номером.

Программа должна выводить в стандартный поток вывода номера вершин базы, отсортированные в порядке возрастания.

Например, для входных данных

```
22
33
 0  8     1  3     1 10     2 11     2 13     3 14
 4  6     4 16     5 17     6 19     8  1     8  9
 9  0     9  2    10  1    10  4    11 12    12  2
12  4    13  5    13 12    14 15    15  3    15  6
16  4    16  7    17  7    17 18    18  5    19  6
20 21    21 18    21 20
```

программа должна выводить номера

```
0 20
```

2.11. Минимальный путь на карте

Карта представляет собой целочисленную матрицу размера $N \times N$.

Путь на карте — это последовательность элементов карты с координатами (x_1, y_1), (x_2, y_2), …, (x_k, y_k) такая, что для любых двух соседних элементов последовательности справедливо одно утверждение из двух:

x-координаты элементов совпадают, а y-координаты различаются на единицу;

y-координаты элементов совпадают, а x-координаты различаются на единицу.

Длина пути — это сумма его элементов.

Например, пусть дана карта:

```
2 1 3
5 8 2
7 1 6
```

Тогда $(1,1)$, $(1,2)$, $(1,3)$, $(2,3)$, $(3,3)$ — путь, имеющий длину 14.

Требуется составить программу maproute.go, вычисляющую минимальную длину пути, соединяющего точки $(1,1)$ и (N,N).

Программа должна считывать из входного потока размер N карты ($1 \le N \le 500$) и саму карту. Карта содержит целые числа от 0 до 9.

Программа должна выводить в выходной поток целое число — минимальную длину пути.

Например, для входных данных

```
5
1 7 7 9 1
8 5 0 6 0
```

```
4 1 2 9 8
4 1 5 7 6
5 6 8 8 7
```

программа должна выводить число 40.

2.12. Идеальный путь в лабиринте

В парке аттракционов построили лабиринт. Лабиринт состоит из n комнат, соединенных с помощью m проходов. Каждый проход покрашен в некоторый цвет c_i. Посетителей лабиринта забрасывают с вертолета в комнату номер 1, и их задача — добраться до выхода из лабиринта, расположенного в комнате номер n.

Владельцы лабиринта планируют устроить соревнование. Несколько участников будут заброшены в комнату номер 1. Они должны будут добраться до комнаты номер n, записывая цвета переходов, через которые им придется проходить. Участник с самой короткой последовательностью цветов выиграет соревнование. Если у нескольких участников длина последовательности окажется одинакова, то победит тот из них, чей путь будет идеальным. Путь считается *идеальным*, если соответствующая ему последовательность цветов — лексикографически меньшая среди всех возможных кратчайших путей.

Андрей готовится к участию в соревновании. Он пролетел на вертолете над парком аттракционов и зарисовал лабиринт. Необходимо составить программу ideal.go, которая поможет ему найти идеальный путь из комнаты номер 1 в комнату номер n, чтобы выиграть соревнование.

Формат входных данных. Сначала программа считывает со стандартного потока ввода два целых числа n и m, содержащие число комнат и переходов, соответственно ($2 \le n \le 100\,000$, $1 \le m \le 200\,000$). Далее программа должна считать m описаний переходов, представляющих собой тройки чисел: a_i, b_i и c_i. Каждая тройка задает номера комнат, соединяемых переходом, и цвет перехода ($1 \le a_i, b_i \le n$, $1 \le c_i \le 109$). Каждый переход можно проходить в любом направлении. Две комнаты могут соединяться несколькими переходами. Кроме того, могут существовать переходы, ведущие из комнаты в нее же. Гарантируется, что комната номер n достижима из комнаты номер 1.

Формат результата работы программы. Программа должна выводить в стандартный поток вывода длину k идеального пути и последовательность из k цветов переходов, получаемую в процессе прохода по идеальному пути.

Пример. Для входных данных

```
4 6
1 2 1
1 3 2
3 4 3
2 3 1
2 4 4
3 1 1
```

программа должна выводить результаты

```
2
1 3
```

3. АВТОМАТЫ

3.1. Визуализация автомата Мили

Составьте программу vismealy.go, выполняющую визуализацию заданного автомата Мили через graphviz. При этом входной алфавит автомата содержит сигналы a, b и c, пронумерованные числами 0, 1 и 2 соответственно, а выходной алфавит — сигналы x и y.

Программа должна считывать из стандартного потока ввода число состояний автомата N, матрицу переходов и матрицу выходов. Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по сигналу j, причем состояния нумеруются, начиная с нуля. Элемент Φ_{ij} матрицы выходов содержит выходной сигнал, который порождается автоматом при переходе из состояния i по сигналу j.

Описанние автомата на языке DOT должно выводиться в стандартный поток вывода.

Например, для входных данных

```
4
1 3 3
1 1 2
2 2 2
```

```
1 2 3
x y y
y y x
x x x
x y y
```

программа должна выводить описанние автомата

```
digraph {
    rankdir = LR
    0 -> 1 [label = "a(x)"]
    0 -> 3 [label = "b(y)"]
    0 -> 3 [label = "c(y)"]
    1 -> 1 [label = "a(y)"]
    1 -> 1 [label = "b(y)"]
    1 -> 2 [label = "c(x)"]
    2 -> 2 [label = "a(x)"]
    2 -> 2 [label = "b(x)"]
    2 -> 2 [label = "c(x)"]
    3 -> 1 [label = "a(x)"]
    3 -> 2 [label = "b(y)"]
    3 -> 3 [label = "c(y)"]
}
```

3.2. Язык автомата Мили

Пусть дан входной алфавит $X = \{a, b\}$ и выходной алфавит $Y = \{\lambda, x, y, z\}$.

Назовем *языком* инициального автомата Мили $A = \langle Q, X, Y, \delta, \varphi, q_0\rangle$ множество всех конечных последовательностей выходных сигналов, порождаемых автоматом из начального состояния q_0. При этом выходной сигнал λ в эти последовательности не входит.

В качестве примера рассмотрим автомат Мили, изображенный на рис. 3. Для последовательности входных сигналов $\langle b, a, a, b, b, b\rangle$ автомат за первые шесть тактов своей работы выдаст последовательность выходных сигналов $\langle \lambda, x, \lambda, \lambda, x, y\rangle$ (здесь первый сигнал λ присутствует в последовательности, потому что на первом такте, когда поступает входной сигнал b, на выходе автомата сигнала нет). Убрав все сигналы λ из этой последовательности, получим $\langle x, x, y\rangle$ — *слово* в языке нашего автомата.

Составьте программу langmealy.go, вычисляющую множество слов автомата Мили, длина которых не превышает числа M. На-

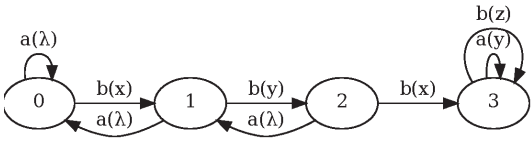

Рис. 3. Пример автомата Мили

пример, для вышеприведенного автомата множество слов, длина которых не превышает 4, выглядит как

```
x xx xxx xxy xy xyx xyy
```

Программа должна считывать со стандартного потока ввода число состояний автомата N, матрицу переходов, матрицу выходов, номер начального состояния q_0 и число M. Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по сигналу j, причем состояния нумеруются, начиная с нуля. Элемент Φ_{ij} матрицы выходов содержит выходной сигнал, который порождается автоматом при переходе из состояния i по сигналу j. При этом сигнал λ обозначается знаком «$-$».

Например, для входных данных

```
4

0 1
0 2
1 3
3 3

- x
- y
- x
y z

0
4
```

программа должна выводить множества

```
x xx xxx xxxx xxxy xxy xxyx xxyy
xy xyx xyxx xyxy xyxz xyy xyyx xyyy
```

3.3. Минимизация автомата Мили

Пусть дан входной алфавит $X = \{a, b, c\}$ и выходной алфавит $Y = \{x, y\}$. Составьте программу minmealy.go, выполняющую минимизацию заданного автомата Мили.

Программа должна считывать со стандартного потока ввода число состояний автомата N, матрицу переходов и матрицу выходов. Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по сигналу j, причем состояния нумеруются, начиная с нуля. Элемент Φ_{ij} матрицы выходов содержит выходной сигнал, который порождается автоматом при переходе из состояния i по сигналу j.

Программа должна выводить в стандартный поток вывода описание минимизированного автомата, эквивалентного заданному, в формате DOT.

Например, для входных данных

```
5
1 2 3
3 4 1
3 4 2
3 0 4
4 4 3
x x y
y x x
y x x
x x y
x y x
```

программа должна выводить описание автомата Мили, изображенного на рис. 4.

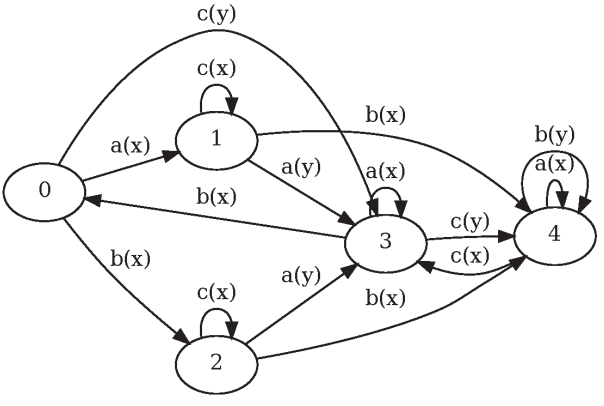

Рис. 4. Минимизированный автомат Мили

3.4. Преобразование автомата Мили в автомат Мура

Составьте программу mealy2moore.go, выполняющую построение автомата Мура по автомату Мили, не имеющему переходящих состояний.

Программа должна считывать со стандартного потока ввода входной и выходной алфавиты, число состояний N, матрицу переходов и матрицу выходов автомата Мили.

Входной и выходной алфавиты заданы в стандартном потоке ввода следующим образом: сначала идет размер алфавита K, а затем следует K не содержащих пробелов строк, представляющих сигналы алфавита.

Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по сигналу j, причем состояния нумеруются, начиная с нуля. Элемент Φ_{ij} матрицы выходов содержит выходной сигнал, который порождается автоматом при переходе из состояния i по сигналу j.

Программа должна выводить в стандартный поток вывода описание автомата Мура в формате DOT.

Например, для входных данных

```
4
00 01 10 11
2
0 1
2
0 0 0 1
0 1 1 1
0 1 1 0
1 0 0 1
```

программа должна выводить описание автомата Мура, изображенного на рис. 5.

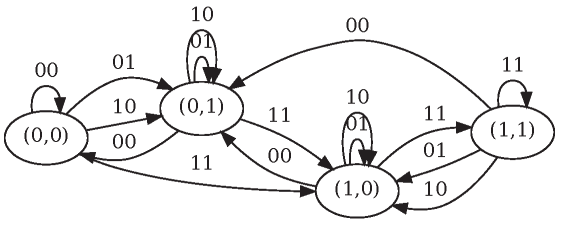

Рис. 5. Автомат Мура

3.5. Визуализация распознающего автомата

Составьте программу visrec.go, выполняющую визуализацию заданного распознающего автомата через программу graphviz.

Программа должна считывать со стандартного потока ввода алфавит, число состояний N, матрицу переходов и массив принимающих состояний распознающего автомата, а также номер начального состояния.

Алфавит представлен в стандартном потоке ввода следующим образом: сначала идет размер алфавита K, а затем следует K не содержащих пробелов строк, представляющих символы алфавита.

Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по символу j, причем состояния нумеруются, начиная с нуля. Элемент $Final_i$ массива принимающих состояний содержит единицу, если i-е состояние принимающее, и нуль в противном случае.

Описанние автомата на языке DOT должно выводиться в стандартный поток вывода.

Например, для входных данных

```
2
a b
4
1 3
1 2
3 3
3 3
0 0 1 0
0
```

программа должна выводить описанние автомата

```
digraph {
        rankdir = LR
        dummy [label = "", shape = none]
        0 [shape = circle]
        1 [shape = circle]
        2 [shape = doublecircle]
        3 [shape = circle]
        dummy -> 0
        0 -> 1 [label = "a"]
        0 -> 3 [label = "b"]
        1 -> 1 [label = "a"]
        1 -> 2 [label = "b"]
```

```
        2 -> 3 [label = "a, b"]
        3 -> 3 [label = "a, b"]
}
```

3.6. Минимизация распознающего автомата

Составьте программу minrec.go, выполняющую минимизацию заданного распознающего автомата.

Программа должна считывать со стандартного потока ввода алфавит, число состояний N, матрицу переходов и массив принимающих состояний распознающего автомата, а также номер начального состояния.

Алфавит представлен в стандартном потоке ввода следующим образом: сначала идет размер алфавита K, а затем следует K не содержащих пробелов строк, представляющих символы алфавита.

Элемент Δ_{ij} матрицы переходов содержит номер состояния, в которое автомат переходит из состояния i по символу j, причем состояния нумеруются, начиная с нуля. Элемент $Final_i$ массива принимающих состояний содержит единицу, если i-е состояние принимающее, и нуль в противном случае.

Описанние минимизированного автомата на языке DOT должно выводиться в стандартный поток вывода.

Например, для входных данных

```
3
a b c
8
1 1 2
3 4 6
5 5 6
4 4 7
3 3 7
5 5 7
6 6 6
5 3 3
0 0 0 0 0 0 1 1
0
```

программа должна выводить описание распознающего автомата, изображенного на рис. 6.

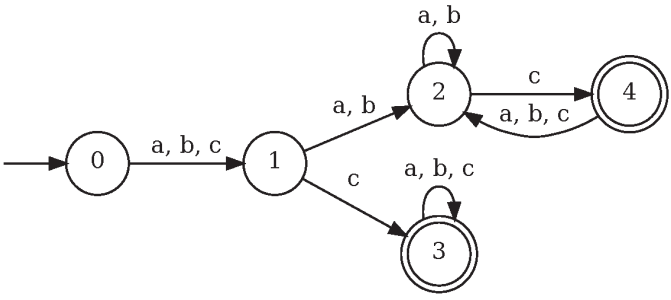

Рис. 6. Минимизированный распознающий автомат

3.7. Визуализация недетерминированного распознающего автомата

Составьте программу visndrec.go, выполняющую визуализацию заданного недетерминированного распознающего автомата через программу graphviz.

Программа должна считывать со стандартного потока ввода число состояний N, число переходов M, затем M описаний переходов, массив принимающих состояний, а также номер начального состояния.

Каждый из M переходов описывается следующим образом: сначала идет номер исходного состояния, затем — номер целевого состояния, а за ними следует символ, которым помечен переход. Состояния нумеруются с нуля. Символ задается в виде строки, причем λ-переход помечается строкой «lambda».

Элемент $Final_i$ массива принимающих состояний содержит единицу, если i-е состояние принимающее, и нуль в противном случае.

Описание автомата на языке DOT должно выводиться в стандартный поток вывода.

Например, для входных данных

```
5
8
0 1 a
0 2 a
1 3 b
3 1 lambda
```

```
3 3 a
2 4 c
4 4 a
4 4 b
0 0 0 1 1
0
```

программа должна выводить описанние автомата

```
digraph {
        rankdir = LR
        dummy [label = "", shape = none]
        0 [shape = circle]
        1 [shape = circle]
        2 [shape = circle]
        3 [shape = doublecircle]
        4 [shape = doublecircle]
        dummy -> 0
        0 -> 1 [label = "a"]
        0 -> 2 [label = "a"]
        1 -> 3 [label = "b"]
        2 -> 4 [label = "c"]
        3 -> 3 [label = "a"]
        3 -> 1 [label = "@$\lambda$@"]
        4 -> 4 [label = "a, b"]
}
```

3.8. Детерминизация распознающего автомата

Составьте программу detrec.go, выполняющую детерминизацию заданного недетерминированного распознающего автомата.

Программа должна считывать со стандартного потока ввода число состояний N, число переходов M, затем M описаний переходов, массив принимающих состояний, а также номер начального состояния.

Каждый из M переходов описывается следующим образом: сначала идет номер исходного состояния, затем — номер целевого состояния, а за ними следует символ, которым помечен переход. Состояния нумеруются с нуля. Символ задается в виде строки, причем λ-переход помечается строкой «lambda».

Элемент $Final_i$ массива принимающих состояний содержит единицу, если i-е состояние принимающее, и нуль в противном случае.

Описанние детерминированного автомата на языке DOT должно выводиться в стандартный поток вывода.

Например, для входных данных

```
6
10
0 5 x
0 5 y
0 1 lambda
1 2 lambda
1 3 lambda
1 4 lambda
3 3 x
4 4 y
5 5 z
5 1 lambda
0 0 1 1 1 0
0
```

программа должна выводить описание распознающего автомата, изображенного на рис. 7.

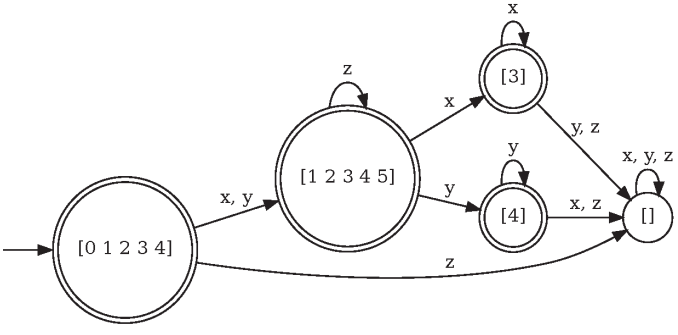

Рис. 7. Детерминированный распознающий автомат

ОГЛАВЛЕНИЕ