Дорогие ребята и учителя!

Оргкомитет Олимпиады по дискретной математике и теоретической информатике приветствует участников и организаторов Олимпиады. Олимпиада ДМиТИ поможет расширить представления о математике и информатике, показать важные связи между ними, научить работать с понятиями, которыми должен владеть специалист в области информационных технологий.

Часть понятий, которые будут использоваться в задачах, уже известны вам из курсов математики и информатики, другие наверняка встречались в окружающем вас мире, но, возможно, в иной форме. В Олимпиаде эти понятия предстанут перед вами в виде компьютерных моделей-манипуляторов, с помощью которых вы будете конструировать решения, правильность которых можно сразу же проверить экспериментально.

Расскажем немного о тех понятиях, с которыми вы встретитесь, и разберём задачи, представленные для тренировки.

Перед началом решения Олимпиадных задач советуем:

1) **проверить самостоятельно все представленные ниже решения**, чтобы познакомиться с интерфейсом манипуляторов. Это позволит вам во время прохождения туров сосредоточиться на решении задач и экономить время на адаптации к интерфейсам;

- **прочитать разборы задач прошлых лет**, расположенных на сайте Олимпиады:
  2019-2020, заключительный тур
  2018-2019 заключительный тур
  2017-2018, очный тур
  2016-2017, очный тур
  2016-2017, заочный тур
  2015-2016, очный тур
  2015-2016, заочный тур
  2014-2015, очный тур
  2014-2015, очный тур

2) **посетить страницы Википедии**, относящиеся к важным понятиям теоретической информатики:

https://ru.wikipedia.org/wiki/Регулярные_выражения
(Раздел "В теории формальных языков")
https://ru.wikipedia.org/wiki/Граф_(математика)
https://ru.wikipedia.org/wiki/Логические_элементы
https://ru.wikipedia.org/wiki/Конечный_автомат
https://ru.wikipedia.org/wiki/Машина_Тьюринга
https://ru.wikipedia.org/wiki/Квантор

Регулярные выражения

В теоретической информатике важную роль играют формальные языки. Языки состоят из *множества слов* (цепочек), которые строятся на основе некоторого заданного *множества символов* — алфавита языка. Например, алфавит русского языка состоит из строчных и заглавных букв от «а» до «я», знаков препинания, пробела. Иногда в текст вставляют другие символы (например, цифры) и тогда алфавит будет шире, но алфавит всегда конечен.

Мы будем работать с языками, используя операции над множествами. Из школьного курса математики известна такая важная для нас операция для работы с множествами как объединение множеств: A B. Однако только с помощью операций объединения построить слово из букв алфавита не получится — у нас всегда будет получаться какой-то набор не связанных между собой букв. Поэтому в теории формальных языков вводится операция умножения множеств. Например, если обозначить множество приставок за П={при; у; …}, множество корней слов за К={ехал; шёл; …}, то произведение множеств П·К будет состоять из слов {приехал; уехал; пришёл; ушёл; …}. Иногда удобно добавить так называемый пустой символ , например, если его добавить к множеству приставок: П'={ ; при; у; …}, то произведение П'·К будет включать слова без приставок: {ехал; приехал; уехал; шёл; пришёл; ушёл; …}

Используя эти две операции (объединение и умножение множеств) мы можем построить на базе заданного алфавита различные *конечные множества слов*.

Однако интересные языки содержат *бесконечное множество слов*. Для того, чтобы из конечного множества получить бесконечное, вводится ещё одна операция — *итерация*. Она означает умножение множества на себя любое количество раз и объединение получающихся слов. Иными словами, если мы применяем к множеству операцию итерации, то получаем все слова любой длины из букв этого множества (включая пустое слово). Например если эту операцию применить к алфавиту из двух символов {0; 1}, то мы получим все двоичные наборы. Эта операция обозначается звёздочкой:

{0; 1}*={ ; 0; 1; 00; 01; 10; 11; 000; 001;…}.

Для записи таких множеств используют формулы, которые называются *регулярными выражениями*:

для объединения множеств используют знак *сложения*,

для умножения множеств — знак *умножения (по традиции его опускают и вместо a·b пишут ab)*,

выше был введён знак *итерации*.

Пример выше теперь запишется такой формулой: (0+1)* или (1+0)*.

Решим простую <u>задачу</u>: как с помощью этих обозначений записать множество всех двоичных чисел?

<u>Ответ</u>: 1(0+1)*+0.

<u>Решение</u>. Двоичные числа не могут начинаться с нуля (мы не пишем 0101, а пропуская первые нули, записываем 101), поэтому мы умножаем выражение, для всех двоичных наборов на множество из одного элемента {1}, слева. При этом мы теряем одно число, которое может начинаться с нуля — 0. Поэтому мы добавляем этот элемент (множество из одного элемента {0}) к произведению.

Можно придумать много разных регулярных выражения, для записи одного и того же множества. Например, предыдущий пример может быть переписан и так: (10+11)(+0+1)*+0+1.

Однако ответ 1*0* будет неверен по нескольким причинам, главная из них в том, что в множестве двоичных наборов, заданных этой формулой сначала идут единицы, а потом нули, тем самым, числа с чередующимися цифрами в него не входят. Также заметим, что формула 1* задаёт множество с пустым символом: { ; 1; 11; 111…}, поэтому формула будет задавать и наборы из одних нулей.

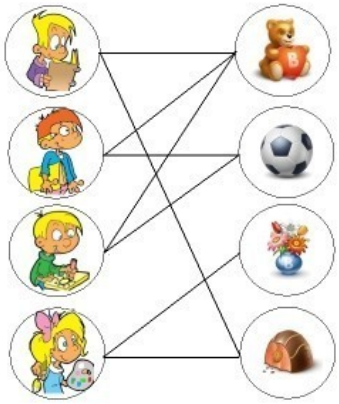
Графы

В этом учебном году в Олимпиаде будут предлагаться различные задачи на *двудольные графы*.

Множество вершин такого графа можно условно разделить на два подмножества так, что рёбра графа будут соединять только вершины одного подмножества с вершинами другого.

Приведём простой пример. Пусть элементы одного подмножества интерпретируются как люди, а элементы второго подмножества как подарки.

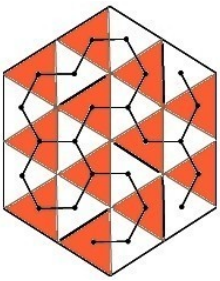
Ребра графа соединяют конкретных людей с подарками и показывают что можно подарить отдельному человеку, чтобы он остался доволен.



Одной из задач, которую можно поставить на графах является задача наибольшего паросочетания.

Если мы выберем рёбра так, чтобы каждому человеку достался ровно один подарок и на один подарок не претендовало несколько человек, то такой набор рёбер называется *паросочетанием двудольного графа*.

Тогда задача состоит в том, чтобы выбрать паросочетание с наибольшим количеством ребер. Это означает, что таким распределении подарков будет удовлетворено максимальное число человек.



Обратите внимание, что двудольные графы могут быть изображены так что разделение множества вершин на два подмножества с указанными свойствами не будет столь очевидным, как в рассмотренной задаче.

На рисунке показан пример двудольного графа. Для доказательства двудольности этого графа достаточно раскрасить вершины в два цвета так, чтобы рёбра соединяли только вершины разных цветов.

Конечные автоматы

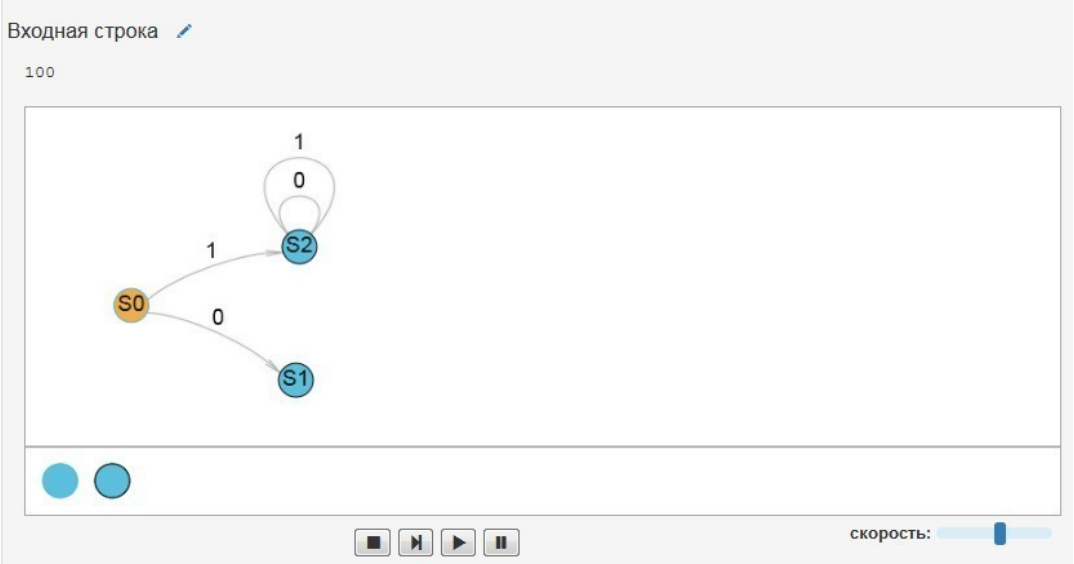
Решению задач с регулярными выражениями может помочь понятие конечного автомата. Несмотря на то, что это понятие математическое, объяснить его можно на примере работы автомата для продажи кофе или устройства интерфейса мобильного телефона.

Важной составляющей конечного автомата являются состояния, в которых может находиться автомат во время работы (на английском языке автомат называют *машиной*

состояний). Например, кофе-автомат при его подключении находится в режиме ожидания, далее, после того, как в него начинают поступать монеты, он переходит в новые состояния, пока не поступит нужная сумма, в этом состоянии он уже готов выдать чашку с напитком.

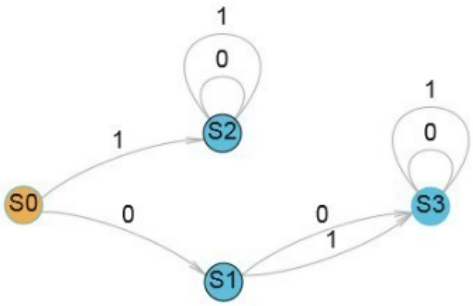
Используемый в задачах манипулятор конечного автомата «выдавать» ничего не будет. Такие автоматы называются *распознающими*: при вводе символов они просто переходят из одного состояния в другое. В приведенном примере в первой части сеанса покупки кофе кофе-автомат работает как распознающий — распознаёт сумму опущенных в него монет.
Процесс распознавания введённой суммы заканчивается тем, что автомат переходит в состояние, когда сумма оказывается достаточной для оплаты напитка. Такие состояния в распознающем автомате называются *конечными*. Если после ввода последовательности символов автомат оказывается в конечном состоянии, то последовательность называется *правильной*. Заметим, что автомат не останавливается в конечном состоянии, а продолжает работать, если на вход подаются новые символы последовательности.

Для наглядного изображения работы автомата используется *граф переходов*. Вершины графа соответствуют состояниям автомата, а рёбра с указанными над ними символами показывают как меняются состояния автомата при вводе этих символов. Вот как выглядит (в манипуляторе «Конечный автомат») граф переходов для распознавания чисел в двоичной системе счисления.

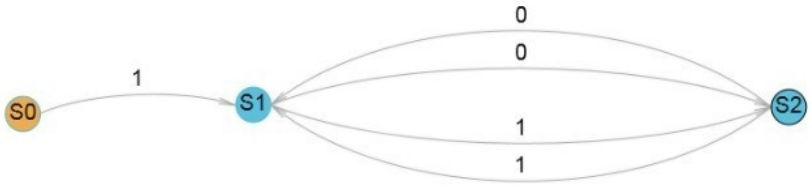


Здесь S0 обозначает начальное состояние, из которого автомат начинает свою работу, а окаймленные кружки показывают конечные состояния. Петли показывают, что при вводе соответствующих символов автомат не меняет своего состояния. Если автомат заканчивает работу в них, то введенная последовательность символов является правильной. Видно, что ввести последовательность, начинающуюся с нуля, например 010, в автомат не удастся.

Можно немного изменить автомат (см. следующий рисунок) так, что в него можно будет ввести любой двоичный набор, но мы окажемся в одном из конечных состояний S1 или S2, только если этот набор является правильной записью двоичного числа (не начинается с 0, кроме случая, когда число равно нулю). В остальных случаях мы попадём в состояние S3, которое конечным не является.



Следующий рисунок показывает пример другого автомата, у которого есть *промежуточное* (не конечное) состояние — S1. Этот автомат распознает все двоичные числа из чётного числа цифр.



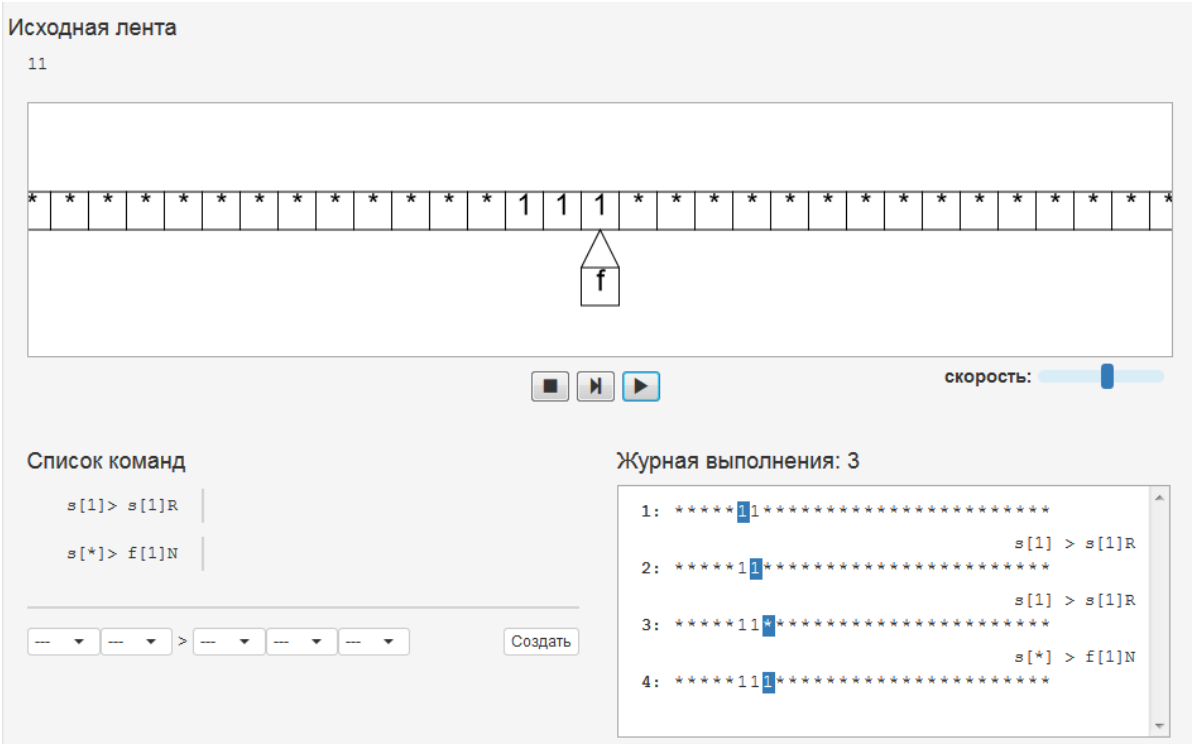
Машина Тьюринга

После знакомства с конечными автоматами легко предположить, что для любого алгоритма можно сконструировать автомат, который этот алгоритм будет выполнять. Однако это не так. Проблема в том, что состояния конечного автомата можно рассматривать как его память, и эта память оказывается конечной. Можно возразить, что и наша память тоже конечна, но тем не менее мы можем выполнять различные алгоритмы, например, умножать и делить числа с любым числом цифр. Это оказывается возможным, так как для вычислений мы используем средства *«внешней» памяти*, которые можно наращивать по мере надобности. Например, даже перемножая длинные (например, 100-значные) числа столбиком, можно представить, что мы будем делать вычисления на больших листах бумаги и сможем записать все промежуточные результаты.

Оказывается, если конечный автомат снабдить неограниченной внешней памятью, то на таком устройстве можно будет смоделировать работу любого алгоритма. Такую конструкцию — *конечный автомат*, к которому подключена *считающе-пишущая головка* и *лента неограниченной длины* для ввода данных, записи промежуточных и конечных результатов — предложил математик Алан Тьюринг ещё до появления электронно-вычислительных машин. Эта конструкция использовалась для теоретических целей, чтобы доказать, что для некоторых задач нельзя сконструировать алгоритм, который её решит (в современных терминах можно сказать, что существуют задачи неподвластные ни одному, даже самому лучшему, программисту) — такие задачи называются *алгоритмически неразрешимыми*, а сама теоретическая конструкция — *машиной Тьюринга.*

Поскольку в основе машина Тьюринга лежит конечный автомат, можно было бы изображать её в виде графа, однако на рёбрах пришлось бы писать не только очередной входной символ (символ, который в данный момент считывает головка машины), но и то, какой символ нужно записать на ленту и куда переместить считывающую головку.

Так же как пользователи компьютера предпочитают видеть то, что создает на экране компьютерная программа, но при этом не питают интереса к её изучению, так и демонстрация работы машины Тьюринга будет гораздо интереснее, если показать, что происходит на её ленте в процессе работы алгоритма.



При этом описание машины Тьюринга можно привести в «программистском» стиле как набор команд.

Например, машина Тьюринга, добавляющая к последовательности единиц на ленте ещё одну (это можно интерпретировать как добавление единицы к натуральному числу в единичной системе счисления) запишется так:

s [1]-> s [1] R
s [*] -> f [1] N

В левой части «команд» указывается
- в каком состоянии находится машина и
- на какой символ указывает считывающая (и записывающая) головка. Справа указывается
- в какое состояние переходит машина,
- какой символ записывается на ленту (символы машина не стирает, а просто всегда пишет на ленту какой-то символ, поэтому чтобы символ сохранился надо его как бы заново перезаписывать на ленту) и
- куда перемещается головка (R – вправо, L – влево, N – остается на месте).

В начале работы машина всегда находится в начальном состоянии s, а если она перейдет в конечное состояние f, то дальше работать не будет (в отличие от конечного распознающего автомата).

Важно отметить, что «команды» машины Тьюринга выполняются не по очереди. На каждом шаге выбирается та команда, левая часть которая описывает состояние, в котором в этот момент находится машина и символ, на который смотрит её головка. Поэтому машина остановится и в том случае, если не найдётся ни одной команды с требуемой левой частью.

В нашем примере первая команда будет выполняться и продвигать считывающую головку вправо пока она указывает на 1, как только считывающая головка дойдёт до первой звездочки, выполнится вторая команда, машина запишет вместо звёздочки 1 и остановится.

Замечание. Обычно договариваются, что машина начинает и заканчивает свою работу в положении, когда головка указывает на первый содержательный символ входной строки (звёздочку здесь можно интерпретировать как пустой символ).

Нетрудно нашу программу доработать, чтобы она вернула головку в начало (проверьте её работу!):
s [1]-> s [1] R
s [*]-> q1 [1] L
q1 [1]-> q1 [1] L
q1 [*] -> f [*] R

Логические схемы

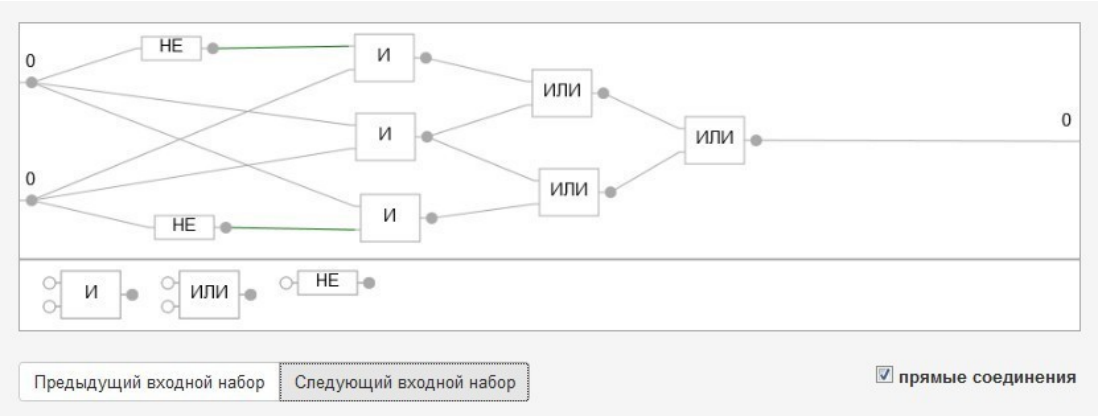
На уроках информатики вы познакомились с такими логическими связками как НЕ, И. ИЛИ и возможно, менее известными, которые называются «исключающее или», «импликация», «эквиваленция». Эти логические связки возникли в языке для создания сложных утверждений из более простых, а потом превращены английским математиком Джорджем Булем в аппарат логических операций, который позволил говорить об *алгебре логики*.

Такой взгляд на логику и сейчас очень важен, например, для автоматизации доказательств, создания экспертных систем и систем искусственного интеллекта. Однако с появлением вычислительной техники оказалось, что все микропроцессоры и другие цифровые устройства оперируют сигналами, которые можно кодировать нулем и единицей и элементы, из которых создаются такие схемы, являются технической реализацией известных логических операций. Поэтому такие схемы можно назвать логическими.

Задача построения микросхем с заданными параметрами функционирования породило несколько математических задач, например,
- как по таблице вход-выход из заданного набора элементов построить схему, которая будет работать по этой таблице и
- как уменьшить число элементов из которых состоит схема.

Например, на рисунке ниже представлена схема, которая может быть описана логической формулой: (x И y) ИЛИ (НЕ x И y) ИЛИ (x И НЕ y)

Эту схему можно упростить так: x И y. Это не очень очевидно, но можно с помощью манипулятора построить обе схемы и проверить результаты на всех комбинациях значений x и y, то есть на всех входных наборах (0;0), (0;1), (1;0), (1;1).



Мир Тарского

Логические связки, о которых говорилось выше, не всегда достаточны для описания суждений, особенно, когда они имеют общий характер.

Например, если мы говорим, «СУММА УГЛОВ ТРЕУГОЛЬНИКА РАВНА 180 ГРАДУСАМ», то полагаем, что это утверждение верно не для одного какого-то треугольника, а для любого. Правильная логическая запись была бы такая: «У ЛЮБОГО ТРЕУГОЛЬНИКА СУММА УГЛОВ РАВНА 180 ГРАДУСАМ»

В то же время, с утверждением «ДЛЯ ЛЮБЫХ ПОЛОЖИТЕЛЬНЫХ ЧИСЕЛ a, b и c, СУЩЕСТВУЕТ ТРЕУГОЛЬНИК, ДЛИНЫ СТОРОН КОТОРОГО РАВНЫ ЭТИМ ЧИСЛАМ» согласиться трудно, достаточно взять одно число очень большим, например a=10, b=1, c=1. В этом случае правильным будет другое утверждение: «СУЩЕСТВУЮТ ТАКИЕ ПОЛОЖИТЕЛЬНЫЕ ЧИСЛА a, b и c, ЧТО НЕВОЗМОЖНО ПОСТРОИТЬ ТРЕУГОЛЬНИК, ДЛИНЫ СТОРОН КОТОРОГО РАВНЫ ЭТИМ ЧИСЛАМ».

В этих высказываниях появились *предметные переменные* a, b и c, которые не являются логическими (то есть, не принимают значения ИСТИНА или ЛОЖЬ), в данном случае это любые положительные числа.

Кроме того, появились новые словесные обороты, которые нельзя описать известными логическими связками:

ДЛЯ ЛЮБЫХ …
СУЩЕСТВУЮТ …

Первое словосочетание называется *квантором всеобщности* и обозначается ∀
Второе — *квантором существования* и обозначается ∃

Если попытаться отвлечься от существа математических фактов, а сосредоточиться только на логических связях, эти фразы можно формализовать следующим образом.

Обозначим T(a;b;c) – утверждение о том, что треугольник со сторонами a, b и c можно построить. Например, T(10;1;1)=0 (ЛОЖЬ), а T(3;4;5)=1 (ИСТИНА).

Тогда первое (неправильное) утверждение запишется:

ДЛЯ ЛЮБЫХ a, b, c (T(a;b;c)) или в формальной записи
∀a, b, c (T(a;b;c)), что иногда записывают подробно ∀a ∀b ∀c (T(a;b;c)) Правильное

утверждение будет выглядеть так:

СУЩЕСТВУЮТ a, b, c (НЕ T(a;b;c)) или в формальной записи
∃a ∃b ∃c (T(a;b;c))