ЗАОЧНАЯ ШКОЛА СОВРЕМЕННОГО ПРОГРАММИРОВАНИЯ

Обучение искусству программирования требует изучения таких дисциплин, как логика, дискретная математика, прикладная теория алгоритмов и др., которым в школьном курсе уделяется совсем немного времени, а во многих школах их не изучают вовсе. Школа современного программирования ставит своей целью восполнить этот пробел, а также приобщить школьников, интересующихся профессией программиста, к решению содержательных задач по программированию. Предполагается, что в школе будут обучаться как те ребята, которые владеют теми или иными языками программирования, так и те, которые не имеют о них пока никакого представления.

В соответствии с этим, в школу принимаются ребята 5-11 классов, а предлагаемые задачи имеют два уровня (***Уровень 1*** не требует умения программировать). Каждый обучающийся может выбрать тот уровень, который окажется ему по силам.

Обучение ведется на модульной основе, поэтому прием в школу осуществляется непрерывно. Учащиеся получают материалы всех занятий текущего календарного года.

Все подписчики журнала найдут условия задач в очередном номере журнала, начиная с первого. Тем, кто не является подписчиком, задания будут высылаться по электронной или обычной почте.

Решения принимаются на дискетах и в письменном виде (по адресу: 191025, Санкт-Петербург, Марата 25, Заочная школа современного программирования, телефон для справок (812) 164-13-55) и по электронной почте (school@aec.neva.ru). Обращаем внимание, что участниками школы могут быть как отдельные школьники, так и коллективы (классы, кружки и пр.).

УСЛОВИЯ, КОТОРЫМ ДОЛЖНЫ УДОВЛЕТВОРЯТЬ ПРИСЫЛАЕМЫЕ РЕШЕНИЯ.

Программы (***Уровень 2***) проверяются автоматической системой тестирования, поэтому необходимо точно следовать указанным в условиях форматам ввода/вывода, а также указанным ниже правилам.

Все программы должны быть скомпилированы под DOS и должны использовать стандартный ввод/вывод. Для Паскаля это процедуры read/readln и write/writeln, для Си - функции scanf и printf, для Си++ - объекты cin и cout, для Бейсика - операторы INPUT и PRINT. Нельзя использовать модули или библиотеки, которые перенаправляют стандартный ввод/вывод (например, для Паскаля - модуль CRT). При одинаковых входных данных программа должна выдавать одинаковый результат.

Названия исполнимых файлов с программой:

<первые три буквы фамилии>_<задача>.exe

Например, pet_2.exe, sid_3.exe.

Все тексты должны быть набраны в формате ASCII.

Тексты программ будут читаться проверяющими, поэтому следует использовать поясняющие комментарии и уделять внимание структуре программы, а также названиям идентификаторов.

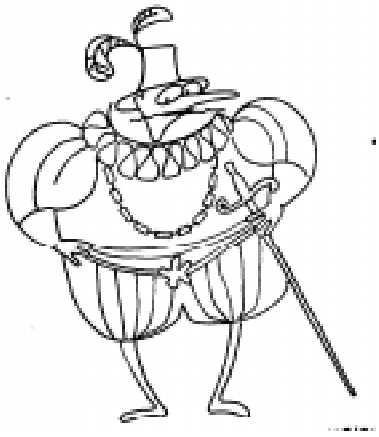
Примечание: те, кто не может прислать решения в электронном виде, присылают подробно и аккуратно оформленные решения на бумаге.

Черкасова Полина Геннадьевна

ЗАНЯТИЕ 5. КОМПЬЮТЕР И ГРАФЫ

1. Основные определения.

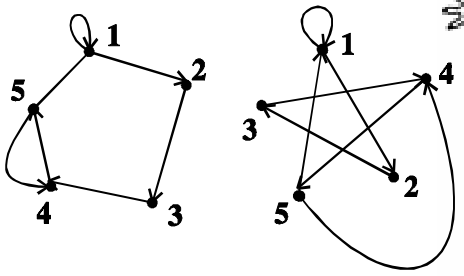
Граф – очень полезная математическая структура. А в программировании она применяется на каждом шагу – всюду, где нужно исследовать *связи* между объектами. Графы помогают в решении разнообразных задач, внешне вроде бы совсем не похожих друг на друга.



Граф – очень полезная математическая структура

Граф состоит из элементов двух типов – вершин и дуг. Вся зависимость между этими элементами заключается в том, что каждая дуга соединяет две вершины, которые называются ее *началом* и *концом*. Таким образом, вершины – это объекты, а дуги – связи между ними. Если у дуги начало и конец совпадают, то она называется *петлей*.

Графы удобно рисовать, «изображать графически», потому они так и называются. На рисунке вершины изображаются точками, а дуги – направленными линиями произвольной формы. Дуга соединяет свои начало и конец и направлена от начала к концу. Вершины на плоскости располагаются так, как нам удобно – их местоположение не несет никакой информации. Следующие два рисунка изображают один и тот же граф:



Для того чтобы в этом убедиться, достаточно правильно пронумеровать вершины и проверить, что в картинках наборы дуг получились идентичными.

Если направление дуг в конкретной задаче не имеет значения, то стрелочки на рисунке опускаются.

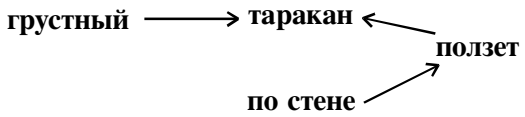
Полным графом назовем граф без петель, в котором любые две различные вершины соединены ровно одной дугой.

Поищем графы вокруг нас.

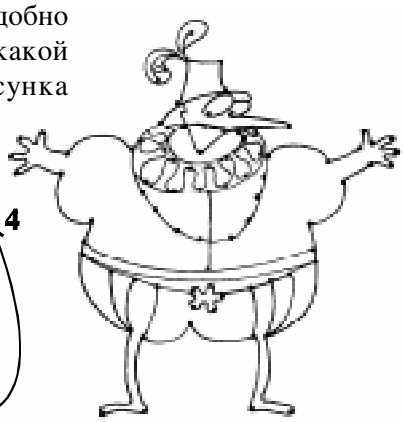
Слова в предложении и их взаимосвязи образуют граф. Например, в предложении «По стене ползет грустный таракан» имеются связи:

По стене – ползет
ползет – таракан
грустный – таракан.

Вот вам и граф:

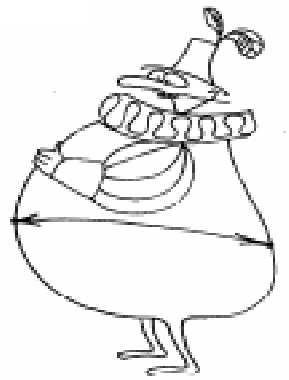


В предложении «На небе появилась долгожданная луна» смысл совсем другой, но связи те же самые. Значит, и знаки препинания ставятся так же (здесь их нет, но ведь мы взяли очень простой случай). Наверняка текстовые редакторы при проверке орфографии используют именно такой механизм.



Граф состоит из... вершин и дуг.

Обратимся к биологии. Соединим биологический



Полным графом назовем граф без петель...

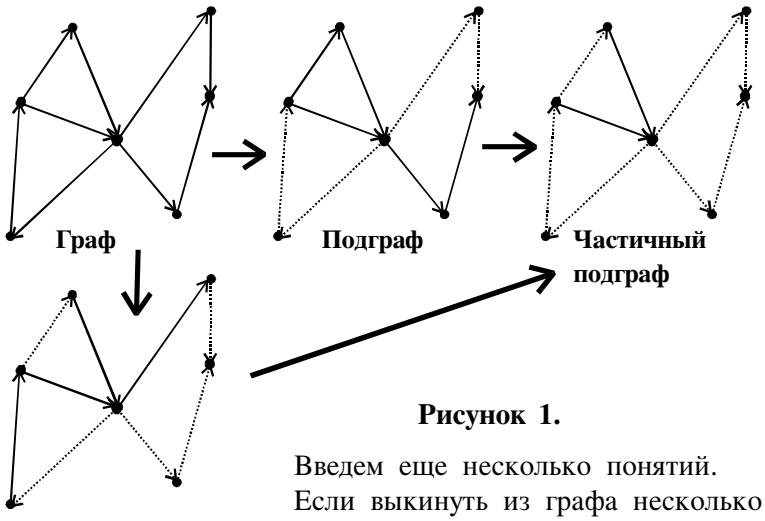
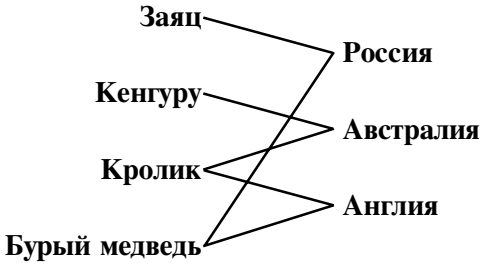


Рисунок 1.

вид дугами с теми государствами, в которых он обитает. Вот маленький подграф того большого графа, который мог бы получиться и наверняка существует в каком-нибудь биологическом учреждении:



Родственные связи русских царей тоже образуют граф:

Введем еще несколько понятий.

Если выкинуть из графа несколько дуг, тогда то, что останется, будет называться *частичным графом*.

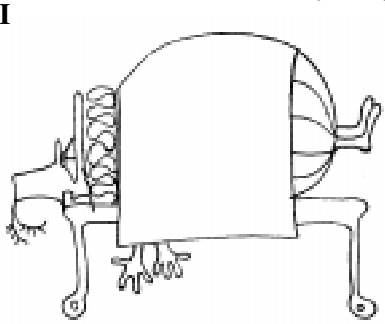
Теперь выкинем не дуги, а вершины. Некоторые дуги «повиснут», оставшись без начала или без конца. Их нужно отрезать, и тогда получим *подграф*. А если мы уберем еще несколько ребер, получим частичный подграф.

На рисунке 1 пунктиром нарисованы ребра, которых на самом деле уже нет в графе, – мы их выкинули. А нарисованы они для того, чтобы вам было понятнее.

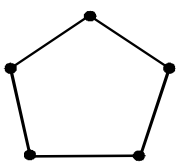
Все дуги, входящие в вершину, образуют *входящую звезду*, выходящие – *исходящую звезду*, а все вместе – просто *звезду* вершины.

Степенью вершины в графе называется количество дуг в ее звезде.

Предположим, что в графе n вершин, и нам известны их степени p_i, где i – номер вершины. Сосчитаем количество дуг. Если мы просто сложим все степени, то каждая дуга будет учтена по два раза, – ведь она выходит из одной вершины и входит в другую. Поэтому полученную сумму следует поделить пополам:



И в этом графе действительно пять дуг.



В графе «Пятиугольник» каждая вершина имеет степень 2. Так как (2*5) / 2 = 5, то в графе должно быть 5 дуг. И в этом графе действительно пять дуг.

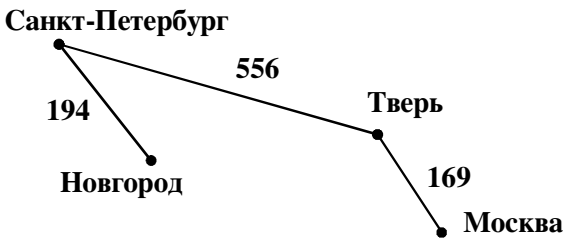
Из этой формулы следует интересный вывод, – *сумма степеней всех вершин в графе должна делиться на 2, иначе количество ребер получилось бы дробным.*

Вершина графа называется *четной*, если она имеет четную степень, иначе она называется нечетной. В графе не может быть нечетного числа нечетных вершин, иначе сумма степеней будет нечетной. Вот «практический вывод»: пять человек не смогут созвониться так, чтобы каждый поговорил с тремя другими.

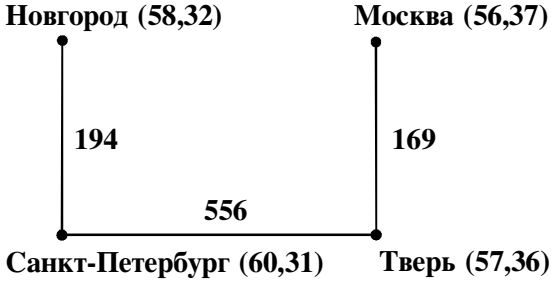
2. Графы разные важны.

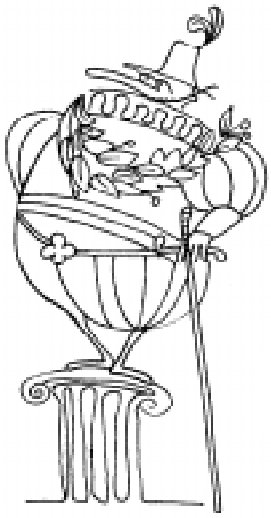
Обычно, когда графы используют для решения задач по программированию, то вершины и дуги представляют в виде структур данных. Как правило, объекты, которые мы подразумеваем под дугами или вершинами, несут в себе существенную для нас информацию, которая и хранится в таких структурах.

Возьмем в качестве вершин города. Соединим два города дугой, если между ними имеется железнодорожное сообщение. Тогда вершина будет содержать информацию о названии города, а может быть, и еще какую-нибудь, – например, количество жителей или наличие топлива для поездов. В каждую дугу, кроме номеров ее вершин, можно записать расстояние между городами или время движения поезда по этому отрезку. Нет, разные поезда проходят участок с разной скоростью, поэтому время мы записывать не будем. Записывать надо, например, предельные скорости для поездов разных типов или габаритные ограничения.



На картинке вершины расположены приблизительно так, как они располагаются на географической карте. Однако, если мы их сдвинем, граф не потеряет никакой информации. Ведь в памяти компьютера граф все равно не будет храниться в виде картинки.



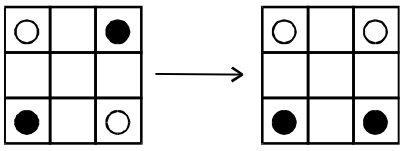


Решение многих задач на порядок упрощается, если использовать графы.

Если вдруг нам важно знать реальное расположение объектов, которые взяты в качестве вершин, то нужно записать в каждую вершину в качестве дополнительной информации еще и координаты объекта. В данном случае это широта и долгота.

Решение многих задач на порядок упрощается, если использовать графы.

Задача. В углах доски 3×3 стоят 4 шахматных коня – два белых и два черных через один. Требуется переставить их так, как показано на картинке (естественно, кони должны ходить, как положено в шахматах), или доказать, что это невозможно.

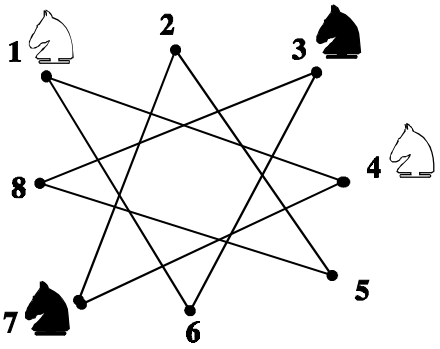


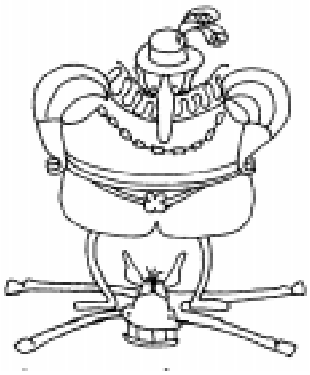
Если вы немножко поколдуете над этой задачей, то увидите, что кони не хотят становиться так, как требуется в задаче. Давайте попробуем доказать, что это невозможно.

Пронумеруем клетки, по которым могут ходить кони, и возьмем их в качестве вершин графа.

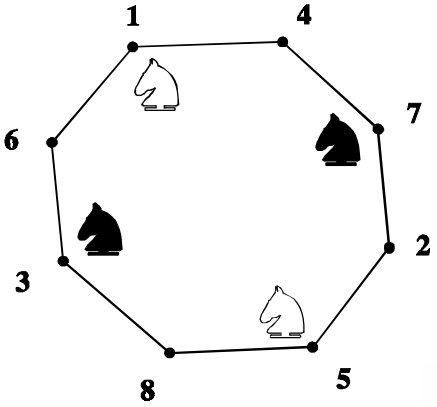
1	2	3
8		4
7	6	5

Вершины соединим дугой, если конь может попасть из одной вершины в другую одним ходом. После этого «развернем» граф, то есть расположим вершины так, чтобы граф выглядел наглядно и в нем можно было бы разобраться.





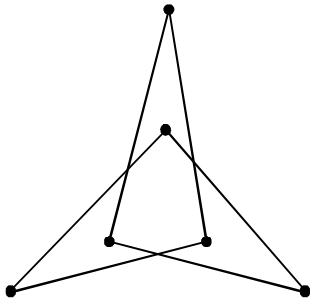
...кони не хотят становиться так, как требуется в задаче.



Получилась очень удачная картинка. Теперь видно, что если кони не встанут на одну клетку, то им друг друга не обойти.

3. Четность вершин.

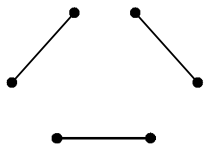
В одной из задач по языку ПостСкрипт (4 занятие, задача 4) предлагалось нарисовать шестизвенную замкнутую ломаную, каждое звено которой пересекается ровно с одним другим звеном. Ее непросто придумать, но, тем не менее, она существует. Вот она:



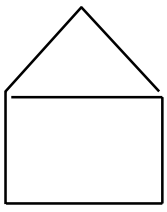
Существует ли пятизвенная ломаная с тем же свойством? Здесь опять понадобится граф. В качестве вершин мы возьмем звенья ломаной. Вершины соединим дугой, если соответствующие звенья пересекаются друг с другом.

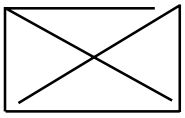
Теперь посмотрим, что от нас требуется в задаче. Нужно, чтобы из каждой вершины выходило ровно одно ребро. Другими словами нужно, чтобы степень каждой вершины равнялась единице. А вершин ровно пять. Сумма степеней получается нечетной. Значит, такой ломаной не бывает.

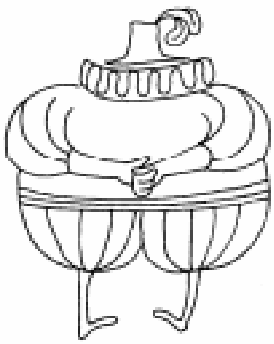
А вот какой граф получился для шестизвенной ломаной:



Четность вершин играет решающую роль в задаче о складывании графа из одного куска проволоки. Например, «домик» можно сложить из куска проволоки, а «конверт» уже нельзя.

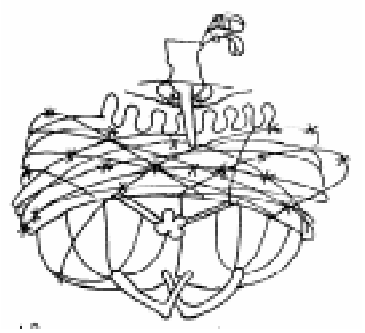






...«развернем» граф, то есть расположим... так, чтобы граф выглядел наглядно...

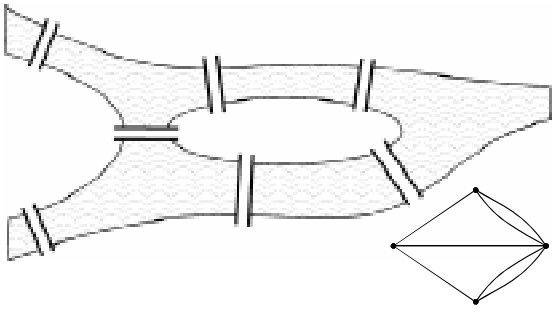
Проследим за четностью вершин в графе, сложенном из проволоки. Если проволока входит в вершину, то она обязательно выходит из нее, либо в ней кончается. Таким образом, она присоединяет к вершине, через которую проходит, сразу две дуги. Следовательно, все вершины должны быть четными, за исключением двух – начало и конец проволоки. А если проволока кончилась в той же вершине, в которой началась, то нечетных вершин в графе нет совсем.



...«домик» можно сложить из куска проволоки, а «конверт» уже нельзя.

Вывод: граф можно сложить из проволоки, если в нем не более двух нечетных вершин. Такой граф называется *эйлеровым графом*.

Дело в том, что, хотя во времена Леонарда Эйлера (L. Euler) теория графов еще не существовала, Эйлер ею уже занимался. Он решал знаменитую задачу о Кенигсбергских мостах: можно ли так спланировать прогулку по этому городу, ныне называемому Калининградом, чтобы каждый из семи городских мостов проходился бы ровно один раз.



Если мы участки суши возьмем в качестве вершин, а мосты – в качестве дуг, то получим граф, в котором все четыре вершины нечетные. Следовательно, не существует пути, который проходит через каждый мост ровно один раз.

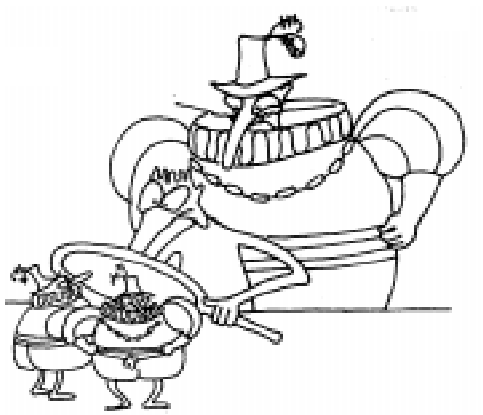
Говорят, что Льюис Кэррол очень любил задавать маленьким девочкам головоломку: нарисовать, не отрывая карандаша от бумаги, линию, изображенную на рисунке слева. Когда девочка с легкостью решала эту задачу, Кэррол предлагал ей сделать то же самое, но так, чтобы получившаяся линия была без самопересечений. Вот здесь-то девочка и задумывалась.

Подобные задачи быстро решаются с помощью следующего метода: нужно раскрасить чертеж в шахматном порядке (это возможно, если граф – эйлеров) и разъединить его в некоторых вершинах.

4. Граф и компьютер.

До этого мы рассматривали маленькие, симпатичные графы, с которыми приятно работать. Но в жизни приходится иметь дело с большими и громоздкими графами, где много вершин. Их нужно как-то обрабатывать и хранить в памяти компьютера.

Я знаю два основных способа хранения и использую их в зависимости от обстоятельств. Каждый из них имеет свои преимущества и недостатки.



До этого мы рассматривали маленькие, симпатичные графы, с которыми приятно работать.

Предположим, нам нужно хранить информацию об авиалиниях. Вершинами графа являются населенные пункты. Две вершины соединены дугой, если существует рейс из одного пункта в другой.

Для хранения этого графа в памяти компьютера заведем таблицу, в которой первой строкой и первым столбцом будут являться населенные пункты. Обозначим их буквами «A» – «Z». В ячейке таблицы будем писать 0, если рейса нет, и 1, если рейс есть.

	A	B	C	D	E	F
A	0	0	0	1	0	0
B	0	0	1	0	0	0
C	0	1	0	0	0	0
D	1	0	0	0	1	1
E	0	0	0	1	0	1
F	0	0	0	1	1	0

Посмотрите, таблица состоит из совершенно одинаковых двух треугольников. Действительно, если есть рейс из A в D, то есть рейс и из D в A.

Нам еще нужно где-то хранить дополнительную информацию. Например, номер и время отправления рейса и названия населенных пунктов. Номер и время можно записать в соответствующие ячейки таблицы. А для названий пунктов придется завести отдельный массив с индексацией от A до F.

Такой способ красив и нагляден, его удобно применять в олимпиадных задачах по программированию, но в серьезных задачах он не используется.

Предположим, что у нас 100 населенных пунктов, и 200 авиарейсов между ними. Тогда ради двухсот рейсов нам придется хранить 10000 табличных ячеек. Память будет использована на 2%.

В этой ситуации есть такой выход: хранить каждую дугу в виде отдельного объекта, в котором записаны две вершины – начальная и конечная. На языке Паскаль это можно оформить в виде записи или объекта:

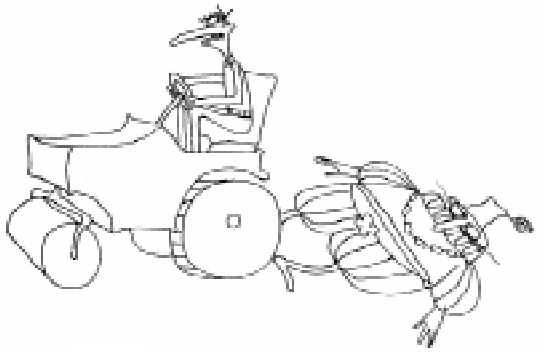
```
type
Tedge = record
 V1, v2: integer; {вершины,
соединяемые ребром}
 .. . {дополнительная информация}
end;
```

Переменные этого типа будем хранить в массиве. В нашем случае получается такая таблица:

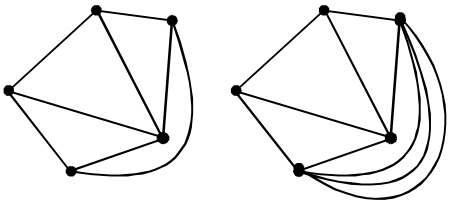
Номер дуги	1	2	3	4	5	6	7	8	9	10
Начало	A	B	C	D	D	D	E	E	F	F
Конец	D	C	B	A	F	E	F	D	E	D

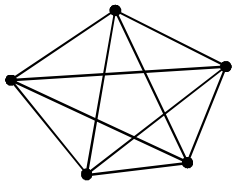
5. Плоские графы.

Планарным, или *плоским*, называется связный граф, который можно нарисовать на плоскости так, чтобы дуги не пересекались. Такой рисунок назовем правильным изображением плоского графа.

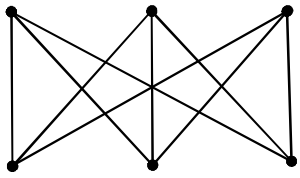


...плоским, называется связный граф, который можно нарисовать на плоскости так, чтобы дуги не пересекались.





Полный граф с пятью вершинами

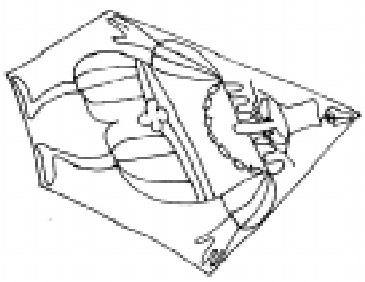


Три дома - три колодца

Рисунок 2.

Возникает естественный вопрос, – как выяснить, является ли граф плоским?

С точки зрения планарности не имеет смысла рассматривать графы, в которых две вершины соединяются больше, чем одной дугой. Ведь если мы смогли на правильном изображении нарисовать одну дугу, то мы можем ее «размножить» и, таким образом, получить нужное количество. Поэтому, если мы хотим проверить граф на планарность, нужно сразу выкинуть лишние дуги (рисунок 2).



Любой выпуклый многогранник является плоским графом...

Нетрудно увидеть, что любой граф, в котором менее пяти вершин, является плоским. Для этого достаточно нарисовать правильные изображения полных графов, у которых 1, 2, 3 и 4 вершины.

Есть два замечательных графа, которые не являются планарными. Первый – полный граф с пятью вершинами. Второй – так называемый граф «Три дома – три колодца». Дома и колодцы – вершины, и от каждого дома ведет дуга к каждому колодцу.

А *теорема Понтрягина-Куратовского* дает ответ на наш вопрос: граф планарен тогда и только тогда, когда он не содержит частичного подграфа, который составлял бы полный граф с пятью вершинами или граф «три дома – три колодца».

Для планарных графов существует очень приятное соотношение.

Плоский граф разбивает плоскость на некоторое количество кусков. Обозначим число кусков через P, число дуг – E, число вершин – V и попробуем определить зависимость этих величин друг от друга.

Предположим, в графе найдется хотя бы один простой цикл. Удалим из него одну дугу. Тогда число кусков тоже уменьшится на один. Таким образом, разность E – P не изменится.

Этот шаг будем повторять до тех пор, пока в графе имеется хотя бы один цикл. Как только циклы кончились, граф превратился в дерево. В дереве $E = V - 1$ и $P = 1$. Поэтому $E - P = V - 2$. При этом ни левая, ни правая части равенства в процессе удаления циклов не изменялись.

Таким образом, мы получили, что *в любом плоском графе* $E - P = V - 2$.

То же самое получил и Леонард Эйлер в 1758 году, поэтому теорема носит его имя.

Любой выпуклый многогранник является плоским графом, следовательно, для многогранников справедлива теорема Эйлера – число ребер многогранника на два меньше суммы числа его вершин и граней.

ЗАДАЧИ

Задача 1.

Уровень 1. Какое максимальное число листьев может быть у дерева из n вершин? А минимальное?

Уровень 2. Напишите программу, которая вводит дерево и считает количество листьев в нем.

Формат ввода:

Число вершин N (< 100)

Начало первой дуги, пробел, конец первой дуги

...

Начало (N – 1) дуги, пробел, конец (N – 1) дуги

Формат вывода:

Количество листьев

Пример:

Ввод:

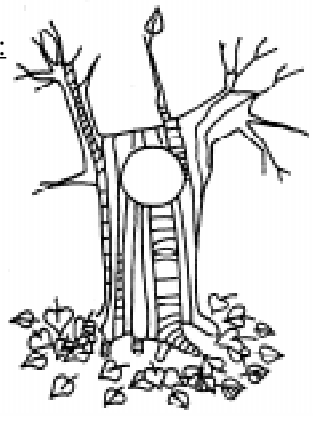
3

1 2

1 3

Вывод:

2



Задача 2.

Уровень 1. Могут ли в дереве две вершины соединяться двумя разными путями? А цепями?

Уровень 2. Напишите программу, которая определяет, является граф деревом или нет.

Формат ввода:

Число вершин N (< 100)

Число дуг M

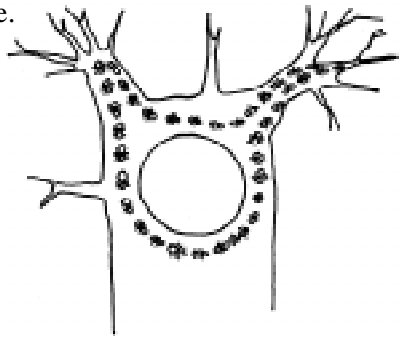
Начало первой дуги, пробел, конец первой дуги

...

Начало дуги с номером M, пробел, конец дуги с номером M

Формат вывода:

Tree. / Not tree.



Пример:

Ввод:

4
3
1 2
2 2
2 3

Вывод:

Not tree.

Задача 3.

Уровень 1. Нарисуйте дерево, в котором одна и та же вершина является и листом и корнем.

Уровень 2. Напишите программу, которая определяет, является ли дерево ориентированным.

Формат ввода:

Число вершин N (< 100)

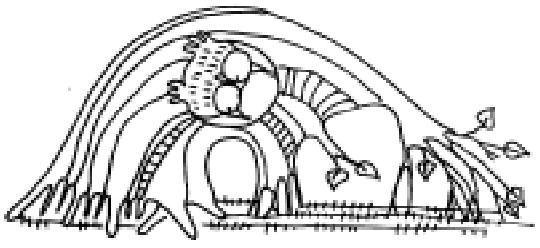
Начало первой дуги, пробел, конец первой дуги

...

Начало (N – 1) дуги, пробел, конец (N – 1) дуги

Формат вывода:

Oriented tree. / Not oriented tree.



Пример:

Ввод:

4
1 2
3 2
3 4

Вывод:

Not oriented tree.

Задача 4.

Уровень 1. Какое получится дерево поиска, если названия месяцев брать с конца – начинать декабрем, а заканчивать январем?

В каком порядке следует добавлять названия месяцев в дерево поиска, чтобы высота дерева получилась минимальной? А максимальной?

Уровень 2. Напишите программу, которая строит дерево поиска из введенного ряда чисел и выводит листья в порядке возрастания. Если корень тоже является листом, то его выводить не нужно.

Формат ввода:

Количество чисел

Числа через пробел

Формат вывода:

Количество листьев

Листья через пробел

Пример:

Ввод:

5
2 16 4 8 10

Вывод:

1
10

Задача 5.

Уровень 1. Докажите, что в ориентированном дереве у каждой вершины не более одного родителя.

Уровень 2. Напишите программу, которая находит корень ориентированного дерева.

Формат ввода:

Число вершин N (< 100)

Начало первой дуги, пробел, конец первой дуги

...

Начало ($N-1$) дуги, пробел, конец ($N-1$) дуги

Формат вывода:

Номер вершины, которая является корнем.

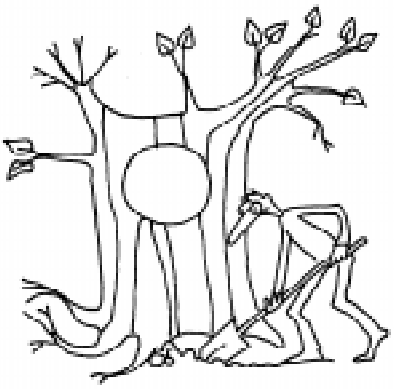
Пример

Ввод:

5
3 2
1 3
2 4
2 5

Вывод:

1



Задача 6.

Уровень 1. Куб является плоским графом. Изобразите его правильный рисунок.

Уровень 2. Напишите программу, которая определяет, является ли введенный граф из восьми вершин и двенадцати дуг графом куба.

Формат ввода:

Начало первого ребра, пробел, конец первого ребра
...
Начало 12-го ребра, пробел, конец 12-го ребра

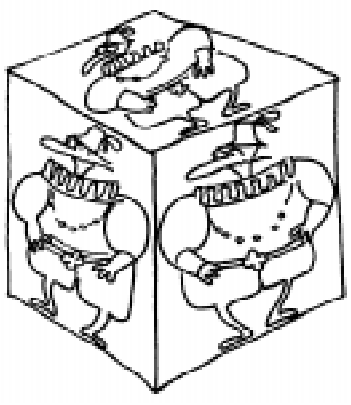
Формат вывода:

Cube. / Not cube.

Пример:

Ввод:

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



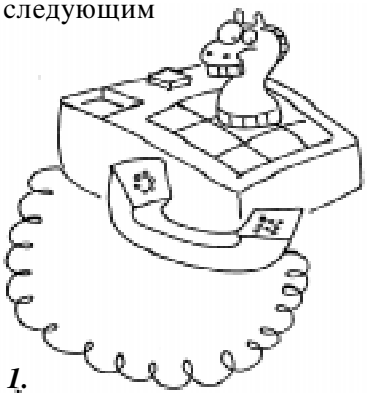
Вывод:

Cube.

Задача 7.

Сколько существует способов на кнопочном телефоне набрать номер длины N ($1 < N < 100$) ходом коня? Кнопки расположены следующим образом:

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



Уровень 1.

Изложите алгоритм решения задачи.

Уровень 2. Напишите программу.

Формат ввода:

N

Формат вывода:

Количество телефонных номеров.

Пример:

Ввод:

2

Вывод:

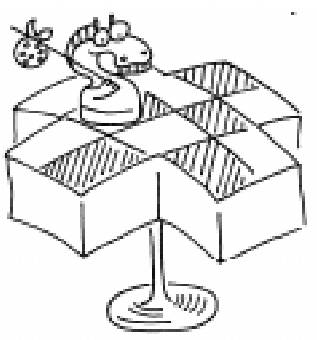
18

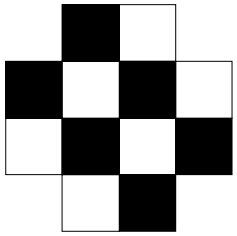
Задача 8.

Уровень 1. Требуется обойти шахматным конем крестовидную доску, побывав на каждой клетке ровно один раз.

Объясните, почему нельзя этого сделать на доске 5x5?

Существует ли путь коня на крестовидной доске, в котором встречается ровно один раз каждый существующий на этой доске ход коня?





Задача 9.

Даны две точки на поверхности единичного куба. Найдите длину кратчайшего пути между этими двумя точками, проходящего по поверхности куба.

Уровень 1. Решите задачу для точек (0.2, 0.3, 1) и (0.5, 0.5, 0).

Сформулируйте алгоритм общего решения.

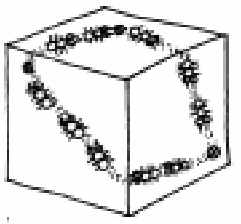
Уровень 2. Напишите программу. Одна из вершин куба имеет координаты (0,0,0), противоположная – (1,1,1), а ребра параллельны координатным осям.

Формат ввода:

Координаты точки A через пробел
Координаты точки B через пробел

Формат вывода:

Длина пути



Пример:

Ввод:

0 0 0
1 1 0.5

Вывод:

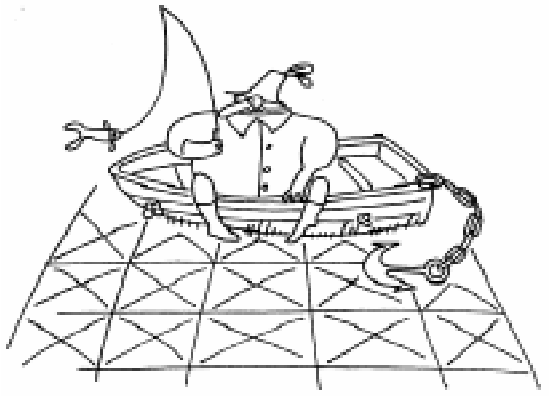
1.802775637731994

Задача 10.

На клетчатом поле M (< 100) x N (< 100) нарисована морская карта. Если в клетке стоит крестик, значит, там суша, иначе – море. По краям карты крестиков нет. Острова состоят из клеток суши, которые соприкасаются сторонами.

Рыболовы хотят расположиться на острове, у которого самая длинная береговая линия. Помогите им выбрать остров.

Уровень 1. Изложите алгоритм решения задачи. Как определить количество островов и как найти длину их границы?



Уровень 2. Напишите программу, которая вводит карту и выводит координаты одной из клеток острова с самой длинной границей и ее длину. На карте клетки моря обозначены символом ‘0’, а клетки суши символом ‘#’.

Формат ввода:

M, пробел, *N*
Первая строка карты (длиной *M*)
...
N-ая строка карты

Формат вывода:

Количество островов.
Координаты клетки через пробел
Длина границы

Пример:

Ввод:

7 6
0000000
0####00
0#000#0
0000##0
00000#0
0000000

Вывод:

2
6 5
10

Задача 11.

Уровень 1. Можно ли сделать плоский граф из полного графа на пяти вершинах, удалив из него одну дугу? Нарисуйте его правильное изображение.

А из графа «Три дома – три колодца»?

Уровень 2. Напишите программу, которая вводит граф и определяет, является ли он плоским.

Формат ввода:

Число вершин N (< 100)
Число дуг M
Начало первой дуги, пробел, конец первой дуги
...
Начало дуги с номером M, пробел, конец дуги с номером M

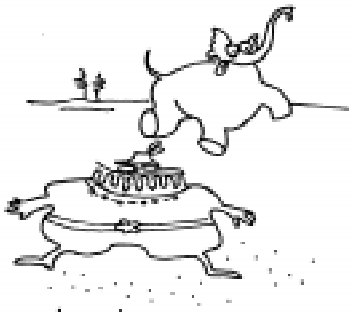
Формат вывода:

Planar graph. / Not planar graph.

Пример:

Ввод:

4
5
1 3
3 4
4 2
4 2
3 1



Вывод:

Planar graph.

Задача 12.

Тра-ля-ля и Тру-ля-ля праздновали рождество и готовились съесть плитку шоколада. Но перед этим заглянули под елку и нашли там погремушку. Последний раз они дрались из-за погремушки совсем недавно, синяки и шишки еще давали о себе знать, поэтому погремушку решено было делить мирным путем.

– Давай, одному достанется самый сладкий кусок шоколадки, а другому – погремушка, – сказал Тра-ля-ля.

– Давай, – согласился Тру-ля-ля, – а какой кусок шоколадки самый сладкий?

– Конечно, последний. Я возьму погремушку, а тебе, так и быть, отдам последний кусок.

– Нет уж, погремушку возьму я.

Перспектива драки снова замаячила впереди. Но Тру-ля-ля нашел выход:

– Давай отламывать шоколадку по очереди. Одному достанется последний кусок, а другому – погремушка.

Итак, правила следующие: двое игроков по очереди отламывают полоски от клетчатой шоколадки *M* x *N*. Ломать можно только по линии углубления. Проигрывает тот, который вынужден взять последнюю клеточку.

Уровень 1. Опишите выигрышную стратегию.

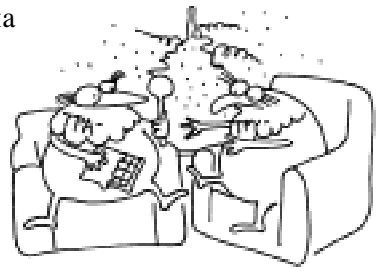
Уровень 2. Напишите программу, которая вводит размеры шоколадки и выводит новые размеры шоколадки после выигрышного хода первого игрока, либо пишет, что такого хода не существует.

Формат ввода:

M
N

Формат вывода:

Win move exist. / Win move not exist.
Одна сторона шоколадки
Другая сторона шоколадки



Пример:

Ввод:

1
2

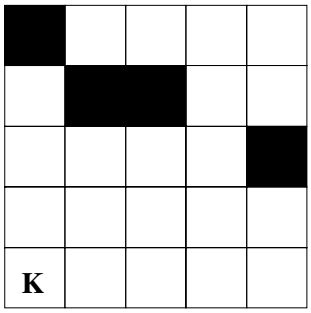
Вывод:

Win move exist.
1
1

Задача 13.

Одинокий король научился ходить по диагонали вправо-вверх.

Уровень 1. Выясните, сможет ли выиграть первый игрок на таком поле:



На какие клетки следует ходить, чтобы выиграть?

Уровень 2. Напишите программу, которая вводит игровое поле и выводит его же, но с помеченными белыми клетками: плюсом, если на клетку следует ходить, чтобы выиграть, и минусом, если на нее не следует ходить. При вводе белая клетка обозначается символом ‘0’, а черная символом ‘#’.

Формат ввода:

M, пробел, *N*
Первая строка поля (длиной *M*)
...
N-ая строка поля

Формат вывода:

Первая строка поля (длиной *M*)
...
N-ая строка поля

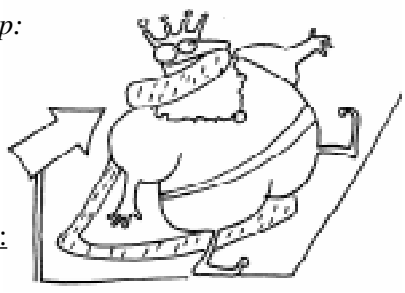
Пример:

Ввод:

```
3 3
000
00#
000
```

Вывод:

```
+-+
—#
+-+
```



Задача 14.

Имеется парк – клеточное поле *N* x *N*. На некоторых клетках растут дубы. По парку бегают Волк и Заяц. Волк, естественно, хочет догнать Зайца, а Заяц убегает. Они могут бегать по всем клеткам, кроме тех, на которых стоят дубы.

Игроки ходят по очереди. За один ход и Волк, и Заяц могут переместиться в соседнюю клетку – на одну клетку в сторону или по диагонали. Сможет ли Волк догнать Зайца (то есть оказаться с ним на одной клетке), даже если Заяц очень умный?

Пример игровой ситуации:

	Д	**Д**	
	Д	**З**	
В			**Д**

В начале игры дано игровое поле – расположение дубов – и позиции Волка и Зайца. Волк ходит первым.

Уровень 1. Существует ли игровая ситуация, в которой у Волка нет выигрышной стратегии на таком поле:

	Д	

Уровень 2. Напишите программу, которая вводит игровое поле, координаты Волка и Зайца и сообщает, сможет Волк догнать Зайца или нет. (Нумерация клеток поля начинается с единицы)

Формат ввода:

Сторона игрового поля (≤ 5)
Количество дубов
Координаты первого дуба через пробел
...
Координаты последнего дуба через пробел
Координаты Волка через пробел
Координаты Зайца через пробел

Формат вывода:

Win. / No win.

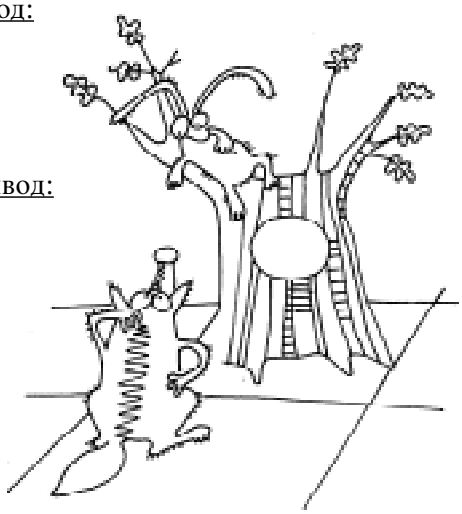
Пример:

Ввод:

```
3
0
1 1
3 3
```

Вывод:

```
Win.
```



Черкасова Полина Геннадьевна,
методист заочной школы
современного программирования.