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 к решению содержательных задач по программированию.

В соответствии с этим, в школу принимаются ребята 5-11 классов, а предлагаемые задачи имеют два уровня (***Уровень 1*** не требует умения программировать).

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.

Предложенные задачи не обязывают вас использовать при решении материал соответствующего занятия. У задачи может быть множество разных хороших решений. Могут предлагаться также задачи не по теме – на общую сообразительность.

Теоретические решения (***Уровень 1***) требуют не просто голого ответа, а объяснений, обоснований и доказательств.

Если вы знаете о том, что такое время работы алгоритма, то при решении задач лучше его указывать. На одном из занятий мы обязательно расскажем, что это такое.

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, вводить информацию из файла in.txt и выводить результат в файл out.txt. Никаких других файлов программа использовать не должна, если это особо не оговорено в условии. При одинаковых входных данных программа должна выдавать одинаковый результат.

Названия исполнимых файлов с программой:

<первые три буквы фамилии><двузначный номер занятия> <двузначный номер задачи>.exe

Например, pet0212.exe (Двеннадцатая задача второго занятия), sid0102.exe (Вторая задача первого занятия).

Все тексты должны быть набраны в формате ASCII.

Тексты программ, возможно, 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. Если вы пишете программу на бумаге, то перед этим следует словами описать алгоритм решения.

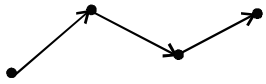
Черкасова Полина Геннадьевна

ЗАНЯТИЕ 6. КОМПЬЮТЕР И ГРАФЫ
ЧАСТЬ II

1. Связность.

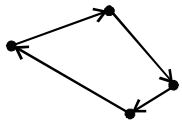
Путь в графе – это такая последовательность дуг, что каждая дуга, кроме первой, начинается в той же вершине, в которой закончилась предыдущая.

Простой путь – это путь, в котором каждая дуга встречается ровно один раз.

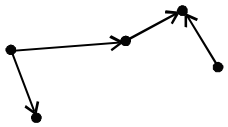


Путь

Контуром называется путь, в котором начало первой дуги совпадает с концом последней. Если в этом определении перед словом «путь» добавить слово «простой», то получим *простой контур*.

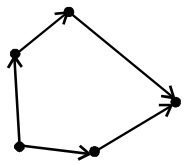


Контур



Цепь

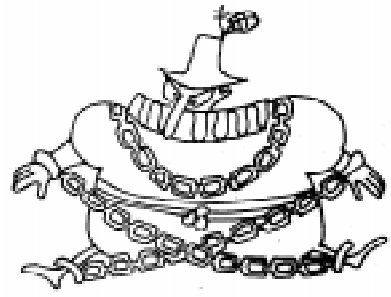
С другой стороны можно сказать, что контур – это подграф, в котором каждая вершина имеет одну входящую и одну исходящую дугу, и можно перенумеровать дуги так, чтобы конец каждой (кроме последней) совпадал с началом следующей, а конец последней дуги – с началом первой.



Цикл

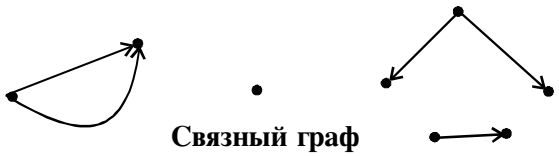
Если в последовательности различных дуг некоторые можно перевернуть (поменять местами начало и конец) так, чтобы получился путь, то такая последовательность называется цепью.

Цепь, у которой начало и конец совпадают, является *циклом*.



Связный граф — это граф, в котором любые две различных вершины связаны цепью...

Связный граф – это граф, в котором любые две различных вершины связаны цепью.



Связный граф

Связный граф **Несвязный граф**

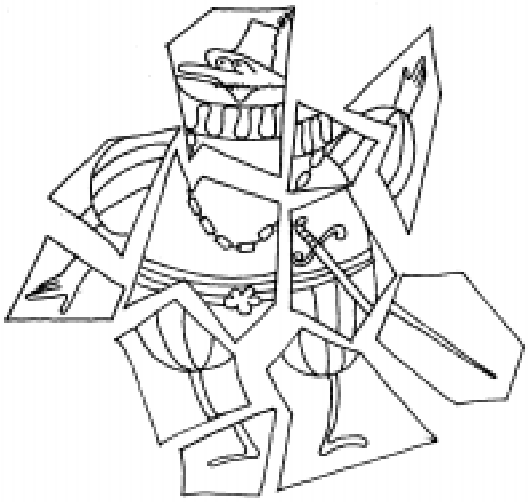
Хотелось бы организовать хранение данных для нашего примера с авиарейсами таким образом, чтобы можно было быстро определить, существует ли авиасообщение между двумя пунктами.

Если граф несвязный, то он распадется на несколько кусков – компонент связности. Говорят, что подграф S графа G является компонентой связности графа G, если он связен и все дуги исходного графа, входящие или выходящие из вершин S, принадлежат ему.

В любом связном графе ровно одна компонента связности, и каждая компонента связности является связным графом.

В несвязном графе, изображенном на картинке, две компоненты связности. Существует классический алгоритм поиска компонент связности в графе. Метод заключается в том, чтобы раскрасить вершины в каждой компоненте в свой цвет. И тогда количество цветов будет совпадать с количеством компонент, а взглянув на окраску двух вершин, мы сможем определить, принадлежат они одной компоненте связности или нет.

Раскрасим каждую вершину в свой индивидуальный цвет и начнем



Если граф несвязный, то он распадется на несколько кусков...

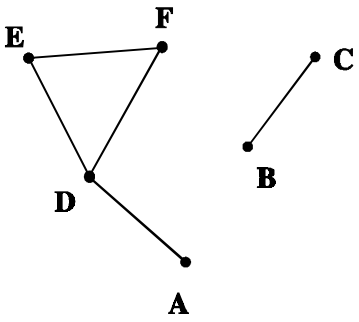
перебирать дуги. Если дуга соединяет две вершины, которые имеют цвета *a* и *b*, тогда каждую вершину цвета *a* следует выкрасить в цвет *b*.

Посмотрим, что произойдет с нашим графом из авиалиний. В ячейках таблицы указаны цвета вершин после рассмотрения соответствующей дуги.

Ребро	A	B	C	D	E	F
	1	2	3	4	5	6
AD	1	2	3	1	5	6
BC	1	2	2	1	5	6
DE	1	2	2	1	1	6
DF	1	2	2	1	1	1
EF	1	2	2	1	1	1

Смотрите, вершины B и C выкрашены в один цвет, а все остальные в другой.

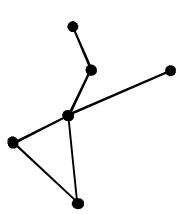
Пришла пора нарисовать этот граф:



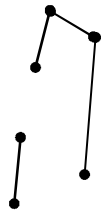
2. Деревья.

Дерево, или древовидный граф, – это связный граф без циклов.

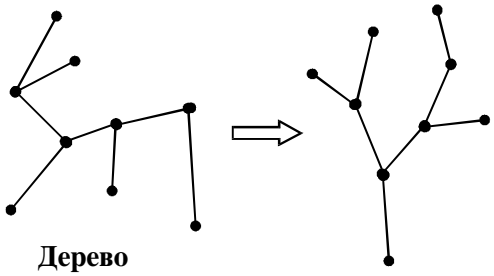
Такое название дано потому, что любой древовидный граф можно нарисовать так, что он будет похож на дерево-растение.



Не дерево



Не дерево

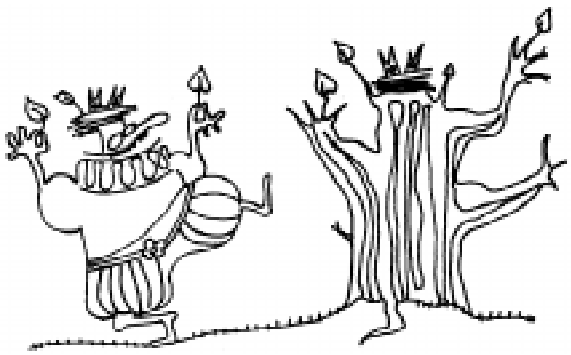


Дерево

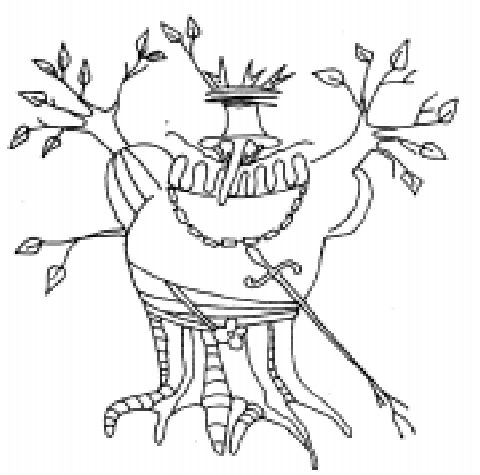
Листьями в дереве называются вершины, соединенные с графом только одной дугой.

Осталось заметить, что в деревьях число вершин всегда на единицу больше, чем число дуг. Действительно, если мы будем отстригать у дерева листья вместе с дугами, на которых они висят, то разность между числом вершин и числом дуг меняться не будет. В конце стрижки останется одинокая вершина. Следовательно, эта разность равна единице.

Если в дереве есть вершина, из которой существует путь в любую другую



...любой древовидный граф ... похож на дерево-растение...

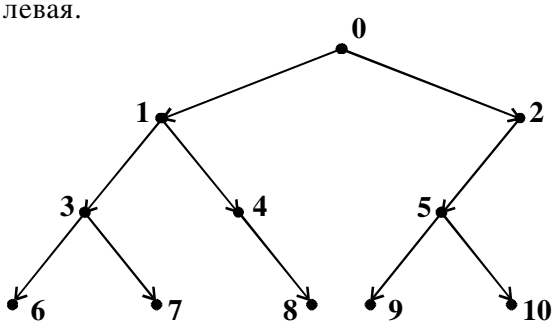


Листьями в дереве называются вершины, соединенные с графом только одной дугой...

вершину графа, то дерево называется ориентированным, а вершина – корнем. Деревья, а особенно ориентированные деревья, играют очень важную роль в информатике. Их часто используют для хранения данных.

В ориентированном дереве выходящие дуги связывают вершину с дочерними вершинами, а входящие – с родительскими.

Особую популярность имеют двоичные деревья, в которых у вершины может быть не более двух детей – правая ветвь и левая.



Чтобы работать с такой структурой, ее нужно как-то хранить в памяти компьютера. Будем хранить ее так, чтобы каждая вершина содержала ссылку на своих детей и своего родителя. В таблице 1 содержится вся информация о дереве.

Номер	0	1	2	3	4	5	6	7	8	9	10
Родитель		0	0	1	1	2	3	3	4	5	5
Левая ветвь	1	3	5	6		9					
Правая ветвь	2	4		7	8	10					
Рабочая информация											

Таблица 1.

Рабочая информация – это данные об объектах, которые мы обозначили вершинами дерева. Ведь не зря же мы его создавали.

На языке Паскаль эта структура оформляется так:

```
type
TNode = record
  parent, left, right: integer;
  info: string;
end;
var
Tree: array[0..100] of TNode;
```

Если дерево не двоичное, и детей может быть больше, чем два, то их следует держать в массиве и хранить их количество.

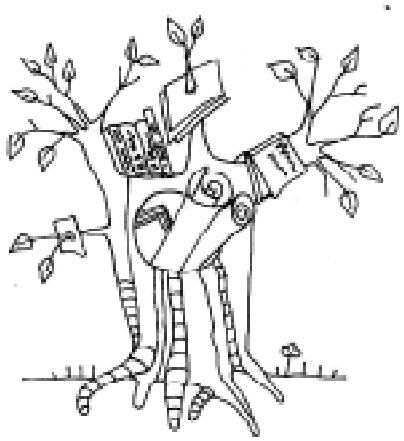
```
type
TNode = record
  parent: integer;
 Children: array[1..10] of integer;
  ChildrenCount: integer;
  info: string;
end;
var
Tree: array[0..100] of TNode;
```

Но на самом деле деревья нужно организовывать с помощью указателей. Если вы хотите овладеть этими приемами, посмотрите задачник Пильщикова по языку Pascal.

Приведем несколько примеров использования древовидных графов.

I. Первое, что приходит в голову, – это древовидная файловая система, или дерево директорий. Корнем является физическое устройство (дисковод, винчестер, лазерный диск и т. д.), узлами – директории (они же каталоги, они же папки), а листьями – файлы.

II. Для хранения информации иногда очень удобно использовать дерево поиска.

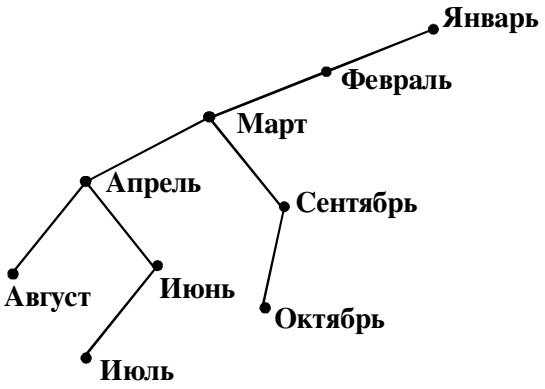


Деревья ... часто используют для хранения данных...

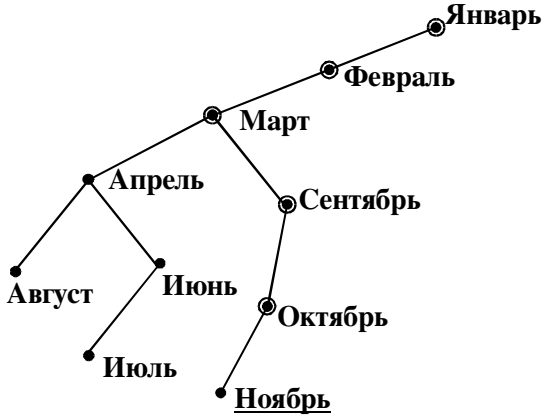
Предположим, нам нужно хранить сведения о членах клуба так, чтобы иметь возможность быстро находить людей по фамилии. Хранить их в массиве в алфавитном порядке неудобно, потому что тогда для каждого нового человека придется раздвигать массив, чтобы вставить имя на нужное место.

Создадим двоичное дерево поиска.

В дереве поиска все потомки с левой стороны всегда меньше родителя, а с правой не меньше. В качестве примера возьмем названия месяцев.



Добавим еще одно название в дерево поиска. Следующий месяц, ноябрь, сначала следует сравнить с корневым элементом – с январем. Ноябрь по алфавиту меньше января, поэтому его следует направить по левой ветке. Левая ветвь тоже, в свою очередь, является деревом поиска с корнем «Февраль». Поэтому новый элемент следует сначала сравнить с корнем, а затем направить в нужном направлении – направо или налево. И так до тех пор, пока не наткнемся на свободное место.



В таком дереве число шагов, необходимое для поиска нужной информации, не будет превышать высоту дерева.

III. Если мы хотим организовать иерархическую организацию данных в компьютере, то ничего удобнее деревьев придумать нельзя.

Например: планета – часть света – страна – населенный пункт – улица.

IV. В памяти компьютера часто требуется хранить математические формулы и выражения, а также оперировать с ними. Например, суммировать или перемножать.

Двоичные деревья – очень удобный для этого инструмент.

Граф устроен следующим образом. Узлами являются математические операции – умножение, деление, возведение в степень и др. А в качестве листьев выступают объекты, над которыми совершаются операции – переменные или числа.

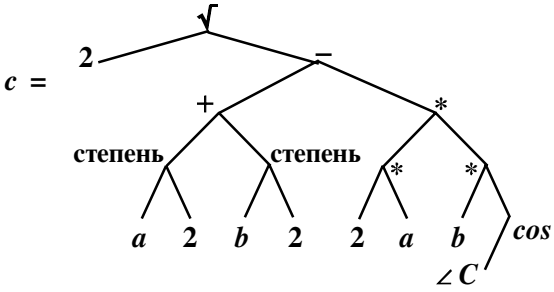
Информацию в дереве следует трактовать так.

Если в дереве одна вершина, то выражение тождественно равно переменной или числу, которое содержится в вершине.

В обратном случае выражение вычисляется как результат операции, записанной в корне дерева, над дочерним выражением слева и дочерним выражением справа.

Если операция унарная, то есть требующая только один операнд, тогда присутствует только одна дочерняя ветвь, – например, левая.

Вот какое дерево получилось для теоремы косинусов:



3. Игры и графы.

С помощью графов можно смоделировать множество игр и просчитать выигрышную стратегию для игроков.

Разберем для примера игру «Одинокий король».

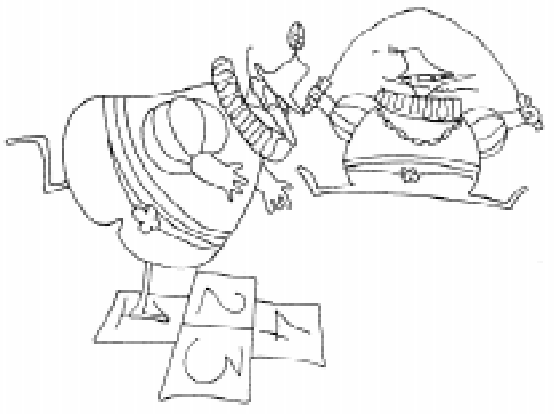
Имеется доска N×N, состоящая из черных и белых клеток (не обязательно в шахматном порядке). В нижнем левом ее углу стоит белый король, который умеет ходить на одну клетку вправо или вверх, но только по белым клеткам. Двое игроков ходят этим королем по очереди. Проигрывает тот, кому некуда ходить

Решим эту задачу на доске 5×5, которая изображена на рисунке.

Возьмем клетки доски в качестве вершин графа. Две вершины соединены дугами, если с одной на другую можно перейти одним ходом нашего короля. Тогда партия игры будет представлять собой путь в этом графе.

Будем ставить на клетку плюс, если на нее следует ходить, чтобы выиграть, и минус – в противном случае.

Очевидно, что вершины, не имеющие исходящих ребер, являются выигрышными, – если игрок сделает ход на такую клетку, то его противнику будет некуда ходить. Поэтому на соответствующие клетки сразу поставим плюсы.

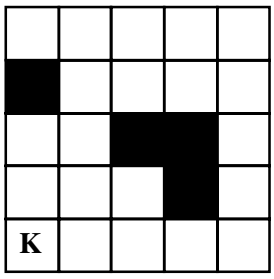


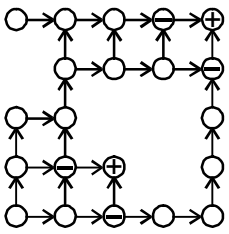
С помощью графов можно смоделировать множество игр...

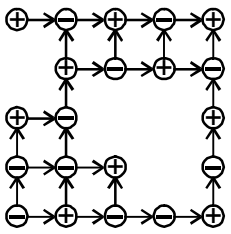
Дальше будем действовать следующим образом:

Понятно, что если игрок походит на клетку, из которой ведет дуга в клетку, помеченную плюсом, то он предоставит своему противнику возможность сделать выигрышный ход. Поэтому все клетки, из которых существует ход в выигрышную клетку, являются проигрышными. Поставим в такие клетки минусы.

Если из вершины любая исходящая дуга ведет в проигрышную позицию, то, сделав ход на эту клетку, игрок заставляет противника сделать проигрышный ход. Поэтому на соответствующие клетки нужно поставить плюсы.







Теперь видно, что выигрывает игрок, который ходит первым. Для этого ему нужно сделать первый ход направо. А если он по ошибке сделает ход наверх, то второй игрок сможет перехватить инициативу и завладеть выигрышной стратегией.

4. Опорные деревья.

В связи с понятием пути в графе возникает совершенно естественная жизненная задача, имеющая совершенно естественные классические решения.

Предположим, что в некотором государстве правительство хочет построить сеть железных дорог, которая связывала бы наиболее важные города. И при этом правительство хочет потратить как можно меньше средств налогоплательщиков.

Давайте формализуем задачу. Имеется граф, в котором каждая вершина соответствует какому-то городу, а дуга, связывающая два города, – железнодорожному полотну между ними. Введем для каждой дуги понятие веса. В данном случае это будет стоимость соответствующего отрезка железной дороги.

Требуется выделить подграф, имеющий минимальный суммарный вес по всем дугам. Этот подграф должен быть связным, чтобы из любого города можно было попасть в любой другой.

Теперь я утверждаю, что этот подграф должен быть еще и деревом. Действительно, если он не дерево, то в нем есть цикл. Если в нем есть цикл, то из негу можно удалить одну дугу, и при этом подграф останется связным, а суммарный вес его дуг станет меньше. Значит, связный подграф с минимальным суммарным весом дуг обязан быть деревом.

Если подграф является деревом, то он называется *опорным деревом*.

А в нашей задаче требуется найти опорное дерево с минимальным суммарным весом дуг.

Существуют два классических решения этой задачи.

Первый алгоритм – алгоритм Прима (Prim).

Образно говоря, этот алгоритм на белом графе выращивает из одной черной вершины черное дерево, постепенно присоединяя к нему все вершины графа.

Раскрасим все вершины в белый цвет, а одну – в черный (рисунок 1). В процессе работы алгоритма будем строить дерево и все вершины, которые в него попали, перекрашивать в черный цвет. Сначала дерево состоит из одной черной вершины.

Назовем белыми дуги, соединяющие белые вершины, черными – дуги, соединяющие черные вершины, а все остальные дуги – разноцветными.

На каждом шаге алгоритма нужно выбрать из всех разноцветных дуг дугу с наименьшим весом и добавить ее в наше дерево. А вершины, которые соединяет эта дуга, выкрасить в черный цвет.

Алгоритм заканчивает работу, когда все вершины стали черными. Теперь дерево соединяет все вершины и, следовательно, стало опорным.

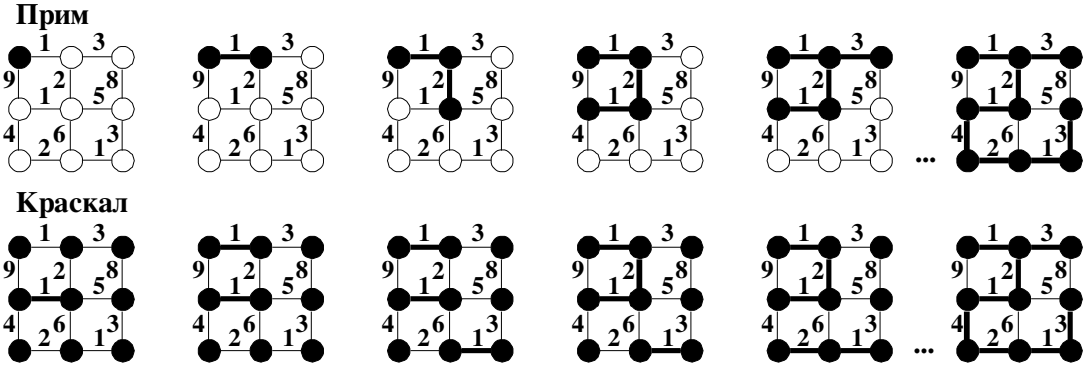
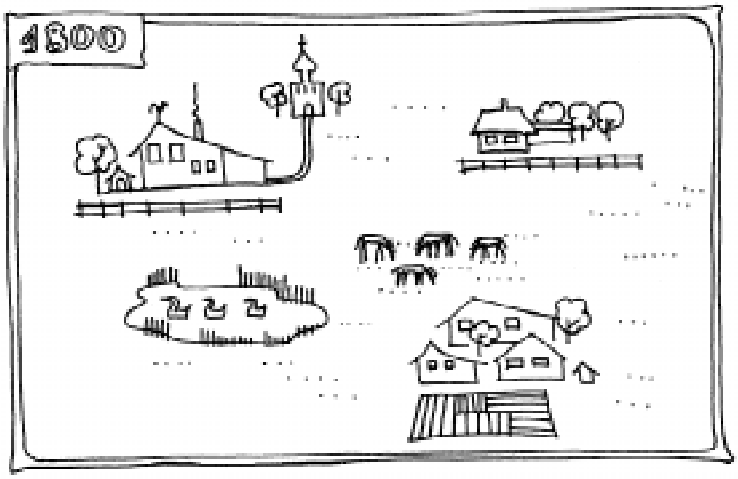


Рисунок 1.

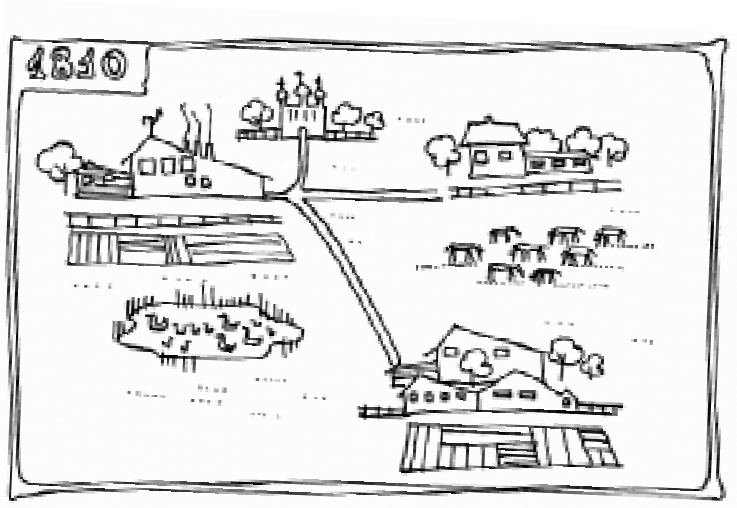


Второй алгоритм – алгоритм Краскала (Kruskal).

Здесь много маленьких деревьев, постепенно срастаясь, превращаются в одно большое опорное дерево (но с минимальным суммарным весом дуг).

Будем строить подграф, состоящий из деревьев. Сначала он содержит все вершины (как и любой подграф) и не содержит ребер. Будем считать, что вершины в разных деревьях выкрашены в разный цвет.

На каждом шаге алгоритма будем выбирать из всех разноцветных дуг дугу с наименьшим весом и добавлять его в наш подграф. С каждой добавленной дугой будет происходить соединение двух деревьев в одно.



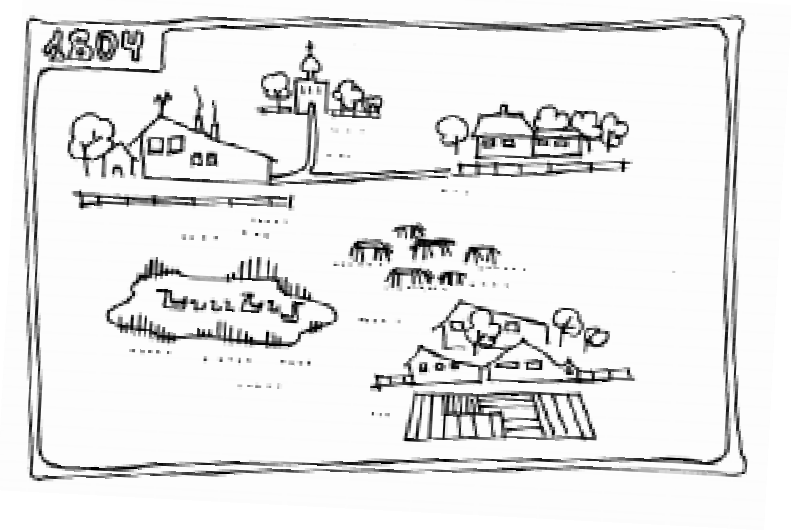
Когда подграф станет связным, алгоритм прекращает работу.

5. Поиск кратчайшего пути.

Ну вот, мы решили задачу правительства о постройке железных дорог.

Теперь возникает задача путешественника, которому нужно добраться из одного пункта в другой. Пользоваться ему построенной железной дорогой или в данном случае проще проехать на велосипеде?

В Санкт-Петербурге самый быстрый транспорт – метро, и, скорее всего, его схема представляет собой минимальное опорное дерево, но иногда от одной станции метро до другой быстрее проехать наземным транспортом.



Итак, дан граф, и для каждой дуги известен ее вес. Весом может быть стоимость проезда или время, затрачиваемое на этот участок дороги. А может быть и комбинация этих величин. Требуется найти кратчайший путь от вершины i до вершины j.

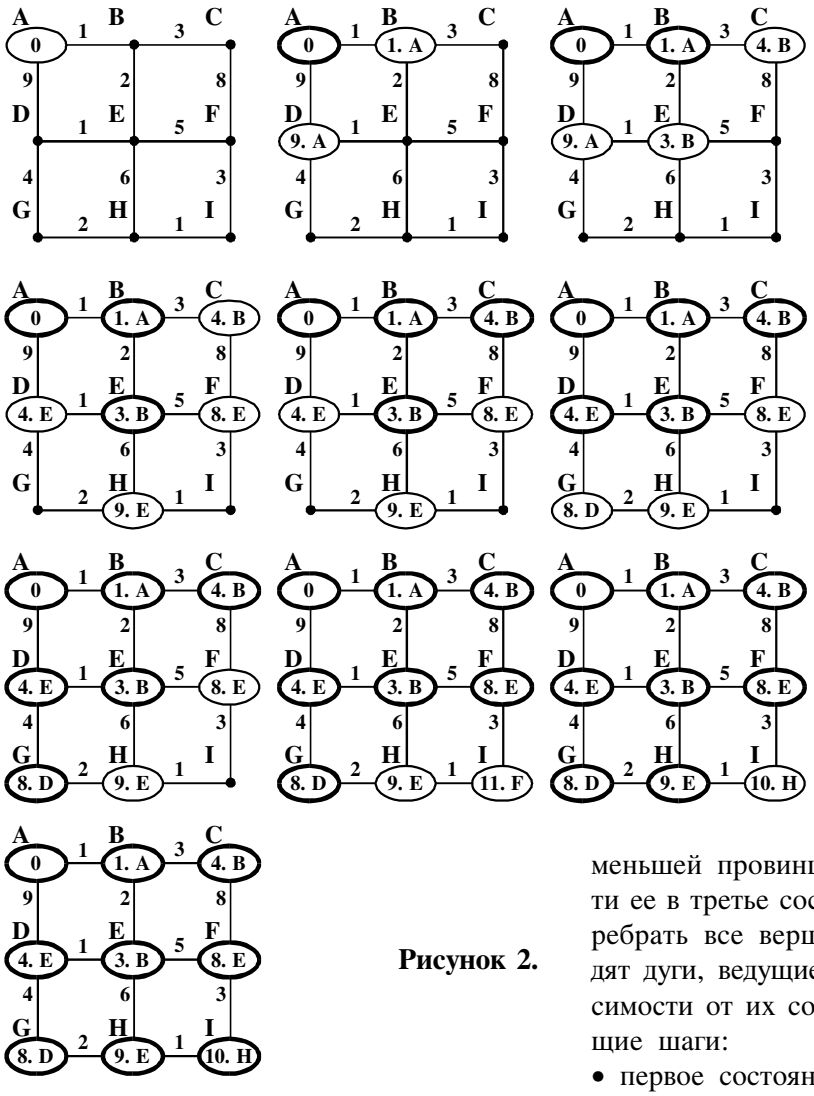


Рисунок 2.

Эту задачу решает алгоритм Дейкстры (Dijkstra E. W.) для поиска кратчайших путей. В процессе его работы вычисляются кратчайшие пути от всех вершин графа до вершины *j*. Это удобно, потому что пути в графе опираются друг на друга, и, зная один из них, мы можем быстро сосчитать похожие.

Во время вычислений каждая вершина побывает в трех состояниях. Первое – мы про нее еще ничего не знаем. Второе – мы нашли путь из данной вершины в вершину *j*, который может оказаться кратчайшим. Вершины во втором состоянии на схеме будем обводить в кружочек, а в кружочке записывать конец первой дуги предполагаемого пути и его длину (рисунок 2). Длину возможного пути назовем провинциальностью вершины (полагая вершину *j* стольным городом), а конец первой дуги – важным соседом. Третье состояние вершины – известен кратчайший путь от нее до вершины *j*. Такие вершины будем обводить жирным кружочком.

Вначале все вершины находятся в первом состоянии, а вершина *j* – во втором. Путь от вершины к ней же равен нулю.

На каждом шаге алгоритма нужно сделать следующее. Из всех вершин во втором состоянии следует выбрать вершину *k* с наименьшей провинциальностью и перевести ее в третье состояние. Затем нужно перебрать все вершины, из которых выходят дуги, ведущие в вершину *k* и, в зависимости от их состояния, сделать следующие шаги:

- первое состояние – перевести вершину во второе состояние, ее важным соседом сделать вершину *k*, а провинциальностью – сумму провинциальности *k* и длины дуги, ведущей в *k*.
- второе состояние – вычислить сумму провинциальности *k* и длины дуги, ведущей в *k*, и, если эта величина будет меньше текущей провинциальности данной вершины, присвоить ей новое значение провинциальности и важным соседом сделать вершину *k*.
- третье состояние – никаких действий выполнять не нужно.

Как только не осталось вершин во втором состоянии, алгоритм заканчивает работу.

Теперь в каждой вершине записано, как быстрее добраться до вершины *i*.

ЗАДАЧИ

Уважаемые читатели, при публикации задач 5-го занятия Заочной школы современного программирования была допущена ошибка. Редакция приносит свои извинения и просит считать задачи № 6, 7, 8, 9, 11 («Компьютерные инструменты в образовании» № 5), 15, 16, 17, 18, 19 задачами к занятию № 5, а задачи № 1, 2, 3, 4, 5, 10, 12, 13, 14 («Компьютерные инструменты в образовании» № 5), 20 задачами к занятию № 6.

Задача 15.

Уровень 1. В редакции имеется три списка: список авторов, список произведений и список выпущенных книг. Взаимосвязи между ними занесены в две таблицы, в каждой из которых два столбца. В первой таблице – порядковый номер автора и порядковый номер произведения, во второй – номер произведения и номер книги.

Одно произведение может быть написано несколькими авторами, и книга может содержать больше одного произведения.

Автор	Произведение
1	3
1	2
1	4
2	4
3	2
3	1
1	1
2	5
2	6
3	5

Произведение	Книга
2	1
3	1
5	1
1	2
5	2
1	3
4	4
2	4
3	4
1	4
6	4

Каждое издание нужно подарить всем авторам, участвовавшим в его написании. Сколько книг будет подарено авторам?

Уровень 2. Напишите программу, которая вводит две таблицы и выводит количество подаренных книг. В каждом из трех списков не более 20 наименований.

Формат ввода:

Число строк в первой таблице
Первая строка
...
Последняя строка
Число строк во второй таблице
Первая строка
...
Последняя строка

Формат вывода:

Количество книг

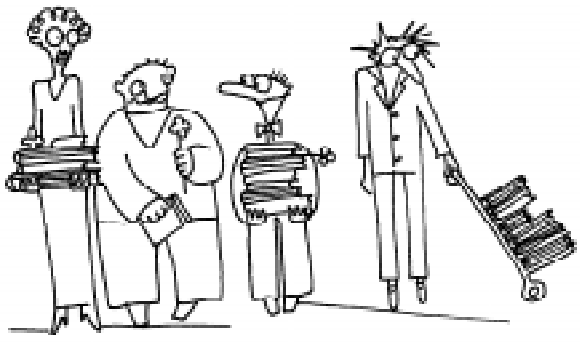
Пример:

Ввод:

```
5
1 1
1 2
2 2
2 3
3 3
3
1 1
2 1
3 2
```

Вывод:

```
4
```



Задача 16.

У султана много жен, и каждой он обещал посвятить песню, но придворный поэт внезапно заболел меланхолией, а султан сумел сочинить только две. Каждая жена расскажет о посвященной ей песне только своим подругам, которые не будут передавать информацию дальше. Сможет ли султан спеть каждой жене песню так, чтобы она не узнала о том, что еще кому-то посвящена такая же?

Уровень 1. У султана 8 жен, и дружат они такими парами:

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

Каким женам надо посвятить одну песню, а каким – другую?

Уровень 2. Напишите программу, которая выясняет, – есть ли у султана такая возможность? Количество жен не превышает сотни.

Формат ввода:

Количество жен.
Количество дружеских пар.
Первая пара.
...
Последняя пара.

Формат вывода:

Yes./No.

Пример:

Ввод:

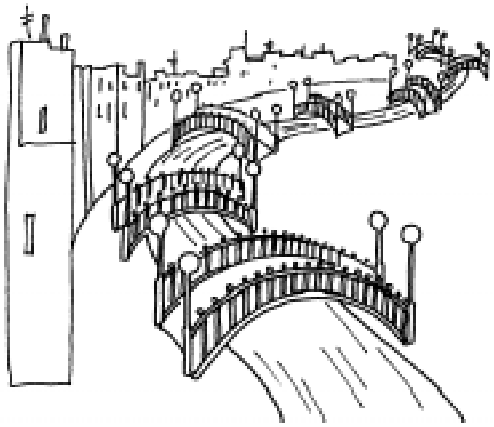
```
3
3
1 2
2 3
3 1
```

Вывод:

```
No.
```

Задача 17.

Уровень 1. Сколько мостов нужно построить, чтобы у задачи о Кенигсбергских мостах появилось решение?



Уровень 2. Напишите программу, которая вводит план города и определяет, можно ли по нему прогуляться, проходя по каждому мосту ровно один раз.

В плане города мост записывается как номера участков суши, которые он соединяет.

Формат ввода:

Количество участков суши
Количество мостов
Первый мост
...
Последний мост

Формат вывода:

Yes./No.

Пример:

Ввод:

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

Вывод:

```
Yes.
```

Задача 18.

Уровень 1. Нарисуйте правильный рисунок полного графа с четырьмя вершинами. Сколько дуг в полном графе с N вершинами?

Уровень 2. Напишите программу,

которая определяет, – является ли введенный граф полным.

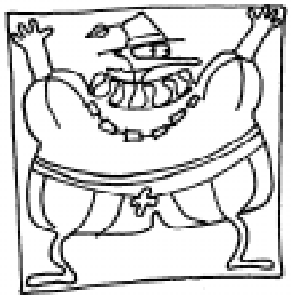
Формат ввода:

Число вершин *N* (< 100)
Число дуг *M*
Начало первой дуги, пробел, конец первой дуги
...
Начало *M*-й дуги, пробел, конец *M*-й дуги

Формат вывода:

Complete graph. / Not complete graph.

Пример:



Ввод:

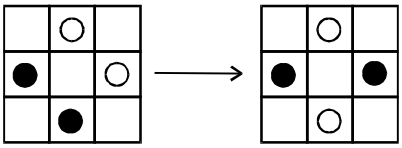
3
4
1 3
3 2
2 3
1 2

Вывод:

Not complete graph.

Задача 19.

Уровень 1. Можно ли, сделав несколько ходов конями, изменить их расположение так, как показано на картинке?



Уровень 2. Напишите программу, которая вводит начальную и конечную позицию на доске 3x3 и определяет, можно ли, по правилам переставляя коней, перейти из одной в другую.

Позиция вводится тремя строчками, из трех символов каждая. Символ обозначает состояние соответствующей клетки доски.

«W» – белый конь
«B» – черный конь
«-» – пустая клетка

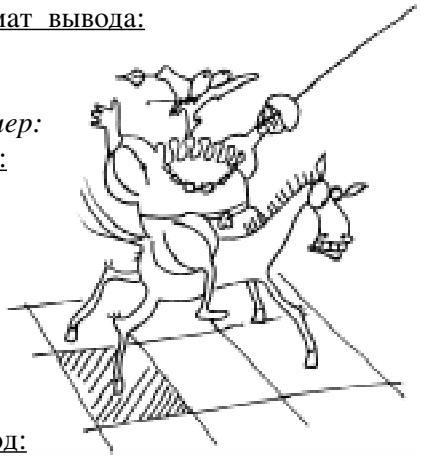
Формат ввода:

Начальная позиция
Пустая строка
Конечная позиция

Формат вывода:

Yes./No.

Пример:



Ввод:

W – –
– B –
– – –

B – –
– W –
– – –

Вывод:

No.

Задача 20.

Уровень 1. Какой кратчайший путь от вершины H до вершины A в графе из примера? Укажите последовательность вершин, через которые он проходит.

Что произойдет, если запустить алгоритм Дейкстры на несвязном графе?

Черкасова Полина Геннадьевна, методист Заочной школы современного программирования.