Методические рекомендации по подготовке учащихся к олимпиаде по информатике

Содержание

Введение

За годы проведения олимпиад школьников по информатике в печатных изданиях и в Интернете было опубликовано много различных материалов, связанных с олимпиадными задачами и методами их решения. Тем не менее, вопрос, как включиться в олимпиадное движение и лучше подготовиться к олимпиадам по информатике, не перестает быть актуальным и сегодня.

В олимпиадной информатике, а именно во всероссийской олимпиаде школьников, развивается направление, поддерживаемое международным сообществом: алгоритмизация и программирование. Связь этого направления со школьным курсом информатики ограничено лишь несколькими разделами. Для достижения результатов в рамках этого направления знать и уметь требуется много, очень много, и любой школьный учебник не входит в этот необходимый минимум. Но, даже если применить дифференцированный подход к обучению школьников, вряд ли этих часов хватит для подготовки к олимпиадам отдельных школьников с "нуля". Уровень задач олимпиад всех рангов год от года повышается, повышается и уровень подготовки самих участников олимпиад. Поэтому для участников олимпиад необходимы дополнительные знания и умения, которых они не получают в стандартной школьной программе.

Для успешного выступления на областной олимпиаде участник, по крайней мере, должен знать и уметь следующее:

- знать и настраивать программную среду, используемого им языка;
- уметь работать с текстовыми файлами, уметь вводить и выводить информацию в текстовые файлы по заданному формату;
- уметь использовать процедуры и функции;
- знать, что такое рекурсия, и уметь применять ее в программировании;

Очень хорошо, если участник олимпиады областного уровня и выше:

- знаком с основой теорией графов
- умеет писать процедуры сложения, вычитания и умножения для работы с многоразрядными числами («длинная» арифметика);
- знает, что такое динамическое программирование и умеет применять его при решении нестандартных задач;
- умеет решать задачи по вычислительной геометрии на плоскости (нахождение точки пересечения двух отрезков на плоскости и т. п.).

Методика решения олимпиадных задач

Если задача задана, то тут же возникает вопрос: с чего начать ее решение? Можно выделить следующие этапы, которые присущи процессу решения большинства олимпиадных задач:

- ✓ Разбор условия задачи.
- ✓ Формализация условия задачи.
- ✓ Разработка алгоритма решения задачи.
- ✓ Программная реализация алгоритма.

- ✓ Отладка и тестирование программы.
- ✓ Отправка решения на проверку.

Практика показывает, что провести четкие границы между соседними этапами достаточно сложно. Поэтому в одних случаях они могут объединяться, в других могут даже опускаться. Более того, поскольку процесс решения олимпиадной задачи, как и любой творческий процесс, всегда является итерационным, то предполагается также возврат на предыдущие этапы в зависимости от результатов выполнения того или иного этапа.

Тем не менее выделение названных этапов вполне оправданно, так как помогает выстроить четкую последовательность действий при решении олимпиадных задач и обратить внимание на то, что пропуск того или иного этапа может существенно сказаться на результатах решения конкретной задачи и выступления на олимпиаде в целом. Рассмотрим содержание каждого этапа более подробно.

Разбор условия задачи

Важность этого этапа переоценить невозможно, поскольку именно он определяет, какая задача будет в дальнейшем решаться. Внимательное чтение условия необходимо для понимания того, что требуется получить в качестве решения задачи. Неверное понимание условия может привести к тому, что будет решаться совершенно другая задача, а не та, что сформулирована в условии. На этом этапе надо абстрагироваться от оценки задачи: хорошая она или плохая, нравится она или не нравится совершенно не имеет значения. Решать надо именно ее, другой не будет. При чтении условия задачи нужно обращать внимание на каждую фразу, поскольку составители задач тщательно выверяют каждое слово и каждое предложение, часто в них прямо или косвенно содержится важная информация.

Важно отметить, что текст задачи нужно всегда внимательно читать от начала и до конца, поскольку ключевое условие может быть спрятано, например, в формате входных или выходных данных, а также в приведенных примерах файлов входных и выходных данных. Без приведенной там информации условие задачи может быть совершенно другим, но тоже вполне корректным. Необходимо всегда помнить, что ошибки при чтении текста задачи обходятся дорого.

Этот этап важен еще тем, что наряду с тщательным ознакомлением с текстами всех задач каждый участник олимпиады должен для себя определить, какую задачу он начнет решать первой. Какие-либо рекомендации здесь давать очень сложно, поскольку часто кажется, что надо начинать или с задачи, условие которой наиболее понятно, или с задачи, аналогичную которой уже решал. На самом деле может оказаться, что совсем понятная задача является самой сложной с точки зрения ее решения, и лучше затратить время на понимание непонятного после первого прочтения условия другой задачи, но решить ее и оставить время на остальные задачи. Аналогичное может произойти и с задачей, которая очень похожа на условие известной, но реально это совсем другая задача, и ее решение может лежать совсем в другой плоскости.

Формализация условия задачи

От понимания текста задания и словесного описания задачи необходимо перейти к выбору формальной схемы ее решения. Первоначальная попытка формализовать условие задачи часто выявляет все недочеты, допущенные при чтении. В этом случае необходимо опять возвратиться к чтению условия задачи, чтобы обратить внимание еще раз на те фразы, в которых содержится важная информация. Когда условие задачи стало предельно ясным, но идей, как свести ее к математической или алгоритмической формулировке, еще не возникло, необходимо попытаться упростить задачу и начать ее решение с рассмотрения самых простых случаев. Полезным бывает использование входных и выходных данных из условия задачи и ручной способ получения ответа для простых случаев, когда задачу можно решить без использования компьютера.

Сложность этапа формализации заключается в том, что, как правило, существует несколько подходов к решению одной и той же задачи. Главное, увидеть эти подходы или хотя бы один из них и определить, с помощью какой формальной схемы или математического аппарата его можно реализовать.

Таким образом, после формализации задачи мы получаем формальную постановку задачи и можем далее разрабатывать алгоритм ее решения. В силу множественности решений не факт, что мы выбрали правильный или приемлемый путь. Но это можно выяснить только на последующих этапах. Поэтому не исключено, что после неудач в построении алгоритма нам придется опять

возвратиться на этот этап.

Разработка алгоритма решения задачи

На этом этапе уже известно, что делать, осталось ответить на вопрос, как это делать, т. е. найти эффективный алгоритм решения задачи и пути его реализации. Это наиболее творческая часть процесса решения задачи, и здесь главную роль играет опыт решения олимпиадных задач и знания теоретических основ информатики. Никто не скажет, как для конкретной задачи он придумал тот или иной алгоритм решения. Тем не менее, существуют общие рекомендации, которые являются полезными на практике. Рассмотрим некоторые из них.

Начинается этот этап с определения основной идеи искомого алгоритма. Чтобы упростить эту задачу, можно абстрагироваться от того, на каком компьютере в дальнейшем будет исполняться алгоритм и какие ограничения по времени, памяти, в диапазоне переменных и т. п. заданы.

Возникшие идеи требуют предварительной проверки и доведения их до конкретных результатов ручным способом на простейших примерах. Здесь обязательно нужно использовать примеры входных и выходных файлов из условия задачи. Более того, можно придумать и свои тестовые данные, которые могут быть использованы при тестировании своего варианта решения перед отправкой его на проверку жюри. Если возникают трудности при разработке приемлемой идеи решения задачи, то необходимо попытаться декомпозировать ее на более простые. И все сказанное выше последовательно применить к каждой задаче в отдельности. В этом случае наряду с пониманием хода решения задачи можно еще получить и структуру алгоритма, которая впоследствии будет уточняться, и наполняться соответствующим содержанием. Наличие предварительно проработанной идеи позволяет приступить к описанию структуры алгоритма, выделению основных компонентов (процедур и функций) и взаимосвязей между ними. Здесь полезно вспомнить о домашних алгоритмических заготовках, которые могут очень пригодиться. При подготовке к олимпиаде каждый участник достаточно хорошо осваивает определенный набор типовых алгоритмов, встречающихся при решении многих задач. Теперь их можно и нужно использовать как кубики в детском конструкторе для получения окончательного решения.

Следует отметить, что не все задачи решаются «конструированием из кубиков». Решение задачи может базироваться на математических идеях, которые необходимо просто придумать. В этом случае кубиками могут быть типичные процедуры или функции, используемые при решении подобных задач, например отсортировать, осуществить перебор, использовать динамическое программирование и т. д.

Нельзя также исключить, что конструирование из кубиков может помочь быстро придумать решение, которое потом долго и упорно будет реализовываться, тогда как для решения данной задачи существует более эффективный и простой путь, но, для того чтобы это увидеть, надо немного подумать.

После того как выделены алгоритмические единицы (или «кубики») и определены подходы к их конкретизации, необходимо тщательно проработать такие вопросы, как: что собою будет представлять алгоритм в целом и каждый «кубик» в отдельности? Какие массивы и структуры данных будут использоваться? Как будет осуществляться передача параметров? И т. д.

На этой стадии разработки алгоритма иногда оказывается полезным писать фрагменты решения задачи на бумаге. При письме завершается упорядочивание мыслей и происходит фиксация ряда важных фактов, которые помогут избежать ошибок в процессе реализации решения. Такими фактами являются необходимые переменные и используемые их процедуры, начальные значения переменных и т. п. В записях проще обнаружить, все ли условия учтены, где и когда необходимо определить переменные, правильно ли передаются параметры и т. д. Запись решения на бумаге *избавит* от «листания» программы на экране в поисках нужного места при отладке программы и позволит избежать обидных ошибок, особенно таких, как «забыл присвоить начальное значение переменной », что не так редко встречается и у участников международных олимпиад по информатике.

Этап разработки алгоритма решения задачи должен заканчиваться доказательством корректности полученного алгоритма и оценкой его эффективности. Это очень важные компоненты решения олимпиадных задач по информатике, поскольку именно они в значительной степени определяют правильность полученного решения и соответствие его заданным в условии задачи ограничениям.

Доказательство корректности алгоритма необходимо, чтобы быть уверенным, что он действительно решает поставленную задачу. Очень часто интуитивно, кажется, что алгоритм работает правильно, но на самом деле внутренняя структура достаточно сложного алгоритма не так просто осознается человеком, и при более внимательном рассмотрении могут возникать проблемы. Доказательство корректности можно свести к разработке набора тестов, учитывающих важные особенности алгоритма, и затем проверить работу программы на них. Но в этом случае можно столкнуться с еще более сложной задачей, чем доказательство корректности. Опыт лучших участников показывает, что лучше сначала подумать над доказательством корректности алгоритма, чем потом искать маловероятные худшие случаи, которые возникают на любом достаточно большом наборе входных данных. Наличие в условиях практически всех олимпиадных задач по информатике ограничений по памяти и времени работы программы требует от участников олимпиады либо знать такие оценки для используемых ими стандартных алгоритмов, либо уметь их вычислять. При этом необходимо учитывать не только асимптотические оценки, но и игнорируемые в этих оценках константы. Если полученные в процессе разработки алгоритма оценки не удовлетворяют заданным ограничениям, то нужно либо подумать о создании другого, принципиально более эффективного решения всей задачи или отдельных подзадач, либо заняться различными алгоритмическими и программистскими оптимизациями. В любом случае полезно бывает сначала подумать, чем тратить много времени на усовершенствование уже полученного решения. Не исключено, что существует другое, существенно более простое решение, которое сразу снимет многие проблемы и позволит сэкономить время для решения остальных задач.

Оценка эффективности полученного решения важна и с другой точки зрения. Нужно всегда помнить, что на олимпиаде требуется решить задачу не вообще, а только для заданной в условии размерности. Зачем тратить лишнее время на разработку более сложного и, как следствие, более трудоемкого решения, если можно обойтись более простым? Нередко бывает, что участник пытается разработать точное решение для поставленной задачи, долго его ищет и так и не находит. А этого решения в принципе не существует — и простой перебор достаточен для достижения поставленной цели.

В заключение следует отметить, что многие участники совмещают этап разработки алгоритма решения задачи с этапом его программной реализации, поскольку очень трудно удержаться на соревновании от соблазна сразу начать писать программу, особенно когда вокруг все «стучат по клавишам». Иногда это оправдано, например, при достаточном опыте, интуиции и уверенности, что задача простая или известная, либо при критическом недостатке времени в конце тура, когда необходимо получить хоть какое-либо решение и успеть отправить его на проверку. Однако вряд ли этот подход можно рекомендовать повсеместно, особенно для тех, кто только начинает свой путь в олимпиадной информатике.

Программная реализация алгоритма

Когда в том или ином виде алгоритм решения задачи получен, возникает проблема написания собственно программы. Естественно, сначала надо выбрать стратегию раз

работки программы: использовать программирование либо «сверху вниз», либо «снизу вверх». Какая стратегия лучше, однозначно сказать сложно. Иногда предпочтительнее программирование «сверху вниз», иногда «снизу вверх», возможна также их комбинация.

Первый подход используется в том случае, когда существует целостное понимание алгоритма решения задачи. Тогда пишется основная часть программы, а функции и процедуры не реализуются, только согласовывается их интерфейс. Такой подход приводит к хорошим результатам, если разбиение на подпрограммы логично связано со структурой разработанного алгоритма. В этом случае также упрощается отладка и доработка решения до окончательного вида.

Программирование «снизу вверх» используется тогда, когда в силу дефицита времени требуется все-таки писать программу, но не очень понятно, что именно. В этом случае можно сначала написать то, что потребуется в любом случае, например ввод входных данных, какие-то стандартные процедуры или функции и т. п. Во время выполнения таких рутинных операций работа в голове над алгоритмом не прекращается, и не исключено, что возникнут какие-то новые идеи, которые уже можно будет материализовать в виде программных компонентов.

На практике наиболее часто используются все-таки смешанные подходы. Но в любом случае надо писать программы так, чтобы потом было легко в них разбираться. Необходимо также не забывать

о комментариях. Лучше потратить минуту на их написание, чем потом при отладке, особенно в конце тура, искать, что за значения хранятся в каждой из переменных, что возвращает та или иная функция и т. п.

При разработке программы следует также обратить особое внимание на описание формата входных и выходных данных, приведенное в условии задачи. Разрабатываемая программа должна читать входные данные из входного файла в описанном формате, решать задачу и выводить результат в выходной файл. Имена входного и выходного файлов также описаны в условии задачи, и неправильное их написание в программе считается ошибкой.

Необходимо внимательно перечитать условие задачи и тщательно разобраться с приведенными там примерами входных и выходных файлов. Нужно четко следовать условию задачи, так как любая неточность приводит к плачевному результату, несмотря на то что во всем остальном решение задачи является абсолютно правильным.

При написании программы очень часто у участников олимпиады возникает вопрос о проверке входных данных на корректкость. Раньше это являлось неотъемлемой ча-стью программы, хотя в условии задачи об этом ничего не говорилось. На всероссийских и международных олимпиадах по информатике последних лет осуществлять формальную проверку входных данных не требуется, если не оговорено иное в условии задачи. Считается, что при тестировании всегда входные данные полностью соответствуют описанному формату и удовлетворяют всем указанным ограничениям. Такая ситуация вызвана тем, что со временем на передний план при решении олимпиадных задач вышли не технические, а содержательные вопросы программной реализации алгоритмов. Гораздо важнее проверить творческие способности участников, а не уровень освоения рутинных программистских навыков, хотя и это важно при разработке программ.

И последнее, о чем необходимо помнить при написании программы, — это сохранение редактируемых файлов во время тура. Всегда не исключены какие-либо сбои в работе компьютеров или нарушения в электроснабжении, которые могут привести к потере существенной части уже набранного программного кода. Поэтому в начале тура необходимо в используемой среде программирования обязательно включить режим автосохранения редактируемых файлов и, помимо этого, своевременно сохранять свои файлы и данные на компьютере. Таким образом, если, например, что-то произойдет с компьютером и среду программирования придется запускать заново, то большая часть набранного текста программы или каких-либо данных будет сохранена. Казалось бы, простое правило, но очень часто участники олимпиады об этом забывают, и нередко им за это приходится расплачиваться.

Отладка и тестирование программы

Отладка и тестирование программы являются важным и ответственным этапом не только при решении олимпиадных задач по информатике, но и при разработке любого программного обеспечения. Никому не нужна программа, которая либо не работает, либо работает с ошибками.

Отладка программы — это процесс, направленный на обнаружение и исправление ошибок в программе путем ее исполнения, в то время как тестирование — это процесс выполнения программы на некотором наборе данных, для которого заранее известен правильный результат ее работы или известны правила ее поведения. Указанный набор данных называется тестовым или просто тестом. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть определена ошибка в программе, поиска места ошибки в программе и редактирования текста программы с целью устранения обнаруженной ошибки.

Очень часто по разным причинам многие участники олимпиады считают, что если программа компилируется и работает правильно на одном или двух простейших наборах входных данных, то получено правильное решение и его можно послать на проверку. На самом деле это совсем не значит, что полученная программа будет соответствовать заданной размерности входных данных и удовлетворять ограничениям на память и время работы, заданные в условии задачи. Причин может быть много, начиная с мелких ошибок в именах переменных, задании размеров массивов, в формулах и т. п. и кончая тем, что разработанный алгоритм является принципиально неправильным или не эффективным.

Для начала необходимо добиться, чтобы программа компилировалась. Это можно делать с использованием встроенных в среду программирования отладчиков, и овладение этими навыками

является неотъемлемой частью подготовки к олимпиадам по информатике.
Добившись того, что программа компилируется, можно переходить непосредственно к процессу тестирования программы. Понятно, что за минуту до окончания тура тестирование уже невозможно, поэтому если программа компилируется, то ее следует отправить на проверку жюри и надеяться только на удачу.

Программу можно тестировать как по частям, так и в целом. В обоих случаях помогает встраивание в программу элементов автоматической проверки. Здесь будет полезна процедура assert или ее аналоги. Не менее полезно добавление в программу многочисленных исчерпывающих проверок. Если программа достаточно сложная и включает другие процедуры, такие меры предосторожности наверняка окупятся. Всегда следует помнить, что во время соревнований, когда присутствует фактор времени и многие участники находятся в стрессовом состоянии, никто не застрахован от ошибок даже в простейших алгоритмах, которые использовались при подготовке к олимпиаде много раз.

Тестирование программы в первую очередь следует начать с использования теста из примера в условии задачи. Каким бы простым он ни был, все равно у многих участников он сразу не проходит.

Использование тестов из условия задачи позволяет найти существенный процент ошибок. Во многих случаях сразу понятно, где ошибка и как ее можно исправить, не вдаваясь в детали. Если возникают трудности, то необходимо внимательно изучить, что же происходит в программе на этих входных данных, найти ошибки и способы их исправления.
При поиске ошибок в программе всегда приходится решать, что лучше — отлаживать готовую программу или проверять логику ее работы. Если в программе есть небольшая часть, вызывающая большие сомнения, то лучше подумать над ней и переписать ее еще раз. Используя отладку, нужно помнить, что этот процесс может оказаться долгим и утомительным в силу наличия ошибок в разных частях программы. Необходимо разумно использовать эти подходы, соблюдая определенный баланс.

Еще одно важное обстоятельство следует выделить особо: исправление одних ошибок в программе часто приводит к возникновению новых. Чтобы этот процесс не стал бесконечным, необходимо всегда помнить об этом и тщательно подходить к исправлению ошибок в программе. Перед внесением исправлений, в необходимости которых нет уверенности, нужно сделать резервную копию всего решения. Обнаружив впоследствии действительно серьезную ошибку, можно будет затем вернуться к этой копии. Тестами из условия задачи не должен заканчиваться процесс отладки и тестирования программы. Далее необходимо придумать свои тесты. Разработка тестов не менее сложный и творческий вопрос, чем разработка самого алгоритма решения задачи. Простым количеством тестов задачу отладки программы не решить. Следует разрабатывать содержательные тесты, ориентированные на проверку логики работы алгоритма. Такой тест обычно вскрывает очень много различных ошибок, хотя может не повезти и ошибка не будет обнаружена.

Еще один совет: не удаляйте тест, однажды введенный в компьютер. На его основе можно создать другой тест. Поэтому лучше иметь две копии одного и того лее теста, чем потом его опять перенабирать. При тестировании внимательно анализируйте, какой ответ выдает программа на конкретном тесте.

В общем случае необходимо использовать различные типы тестов: простейшие, средней сложности и тесты максимальной сложности. Как правило, тест максимальной сложности — это полностью случайный тест большой размерности с максимальными ограничениями. Для его генерации пишется специальная программа. На практике она может быть встроена в решение. Сгенерированный тест не во всех задачах даст худшие результаты по времени работы программы, и при разработке тестов необходимо понимать и учитывать это.

Тем не менее, никакая специальная подготовка не гарантирует успешного выступления школьника на олимпиаде, ибо олимпиада – это, в первую очередь, соревнование на умение находить нестандартные решения для новых, оригинальных задач.

Примерная программа по олимпиадной информатике

Раздел 1. Математические основы программирования

Раздел 2. Техника программирования

1. Основы языка программирования (Паскаль, Си)

Переменные и простейшие типы данных, размеры типов. Линейные программы. Условные операторы. Циклы. Процедуры и функции. Сложные типы данных (массивы, строки, записи, указатели, файлы).

2. Массивы

Одномерные массивы. Двумерные массивы (матрицы). Многомерные массивы.

3. Строки. Элементы лексического и синтаксического разбора

Операции над строками. Лексемы, подсчет лексем различных типов. Выделение чисел из строки.

4. Работа с файлами

Чтение и запись в текстовый файл. Преобразование полученных из файла данных в удобную структуру. Работа с типизированными файлами. Нетипизированные файлы. Буферизация ввода.

5. Рекурсия

Математические функции, задаваемые рекурсивно. Примеры рекурсивных подпрограмм. Проблема остановки рекурсии. Замена рекурсии итерацией.

6. "Длинная" арифметика

Хранения в программе чисел, которые не вмещаются в стандартные типы. Арифметические операции над "длинными" числами. "Длинные" числа с десятичной частью. Извлечение корня с заданной точностью.

7. Хранение информации в динамической памяти.

Хранение набора данных в линейных списках. Вставка в список, удаление из списка, поиск элемента в списке. Двусвязные списки. Понятия структур данных стека, кольца, очереди, дека; реализация их с помощью динамической памяти. Двоичные деревья. Деревья с неопределенным числом потомков. Хранение больших массивов.

Раздел 3. Алгоритмы, методы и принципы решения задач

1. Понятие сложности алгоритма.

Определение сложности. Классы задач P и NP. NP-полные задачи.

2. Алгоритмы поиска и сортировки

Поиск элемента в неупорядоченном массиве. Двоичный поиск по ключу в упорядоченном массиве (дихотомия). Поиск методом Фибоначчи. Поиск в упорядоченном n-мерном массиве. Поиск k-го по величине элемента массива. Простые методы сортировки ("пузырек", "выборка", "вставка", "подсчет"). Быстрые методы ("быстрая", "слиянием", "пирамидальная"), балансировка двоичных деревьев. Сортировка методом черпака.

3. Решение задач методом перебора вариантов

Применение рекурсии для перебора. Генерация сочетаний, размещений, перестановок и булева множества. Полный перебор. Перебор с возвратом (общая схема, задача о расстановке ферзей, задача о шахматном коне, задача о лабиринте, задача о парламенте, задача о рюкзаке, задача о коммивояжере). Отсечение вариантов (эвристики). Метод ветвей и границ. Метод решета (Решето Эратосфена, Быки и коровы).

4. Вычислительная геометрия и численные методы

Длина отрезка. Уравнение прямой. Скалярное и векторное произведение. Точка пересечения отрезков. Принадлежность точки фигуре на плоскости (например: треугольнику). Площадь выпуклого многоугольника. Выпуклая оболочка множества точек: алгоритмы Грэхема, Джарвиса, "разделяй и властвуй". Ближайшая пара точек. Метод Гаусса для решения системы линейных уравнений. Нахождение решения уравнения.

5. Принцип динамического программирования

Понятие, применимость. Сравнение с перебором. Задача о Черепашке, Треугольник, Степень числа, Автозаправка, Алгоритм Нудельмана-Вунша, Разбиение выпуклого N-угольника, Задача о рюкзаке, Задача о паркете, «Канадские авиалинии»,

6. Жадные алгоритмы

Понятие, применимость. Сравнение с перебором и динамическим программированием.

7. Теория графов. Алгоритмы на графах

Понятие графа. Определения теории графов. Структуры данных для представления графа в программе. Алгоритмы обхода графа (поиски в ширину и глубину). Лабиринт (метод волны). Эйлеров цикл. Кратчайший путь во взвешенном графе (алгоритмы Дейкстры и Минти). Транзитивное замыкание графа (алгоритм Флойда-Уоршилла). Минимальное остовное дерево (алгоритмы Прима и Краскала). Топологическая сортировка графа. Потоки в сетях (алгоритм Форда-Фалкерсона). Паросочетания в двудольном графе (метод удлиняющей цепочки, потоковое решение). Задача о назначениях, назначения на узкое место (венгерский алгоритм). Игры на графах. Раскраска графа. Уложение графа на плоскости. Сильная связность и двусвязность графа. Изоморфизм графов. K-клика. Гамильтонов цикл.

8. Лексический и синтаксический анализ

Задача "Калькулятор". Синтаксические диаграммы. Формы Бэкуса-Наура. Стековая и рекурсивная модель синтаксического разбора. Конечные автоматы. Грамматики.

9. Задачи с "изюминками"

Памятка решения олимпиадных задач по информатике для учащихся

В самом начале тура полезно набить приведенную ниже универсальную заготовку для решения олимпиадной задачи (она представляет собой работающую программу!), особое внимание следует обратить на директивы компилятора, приведенные в образце. С помощью команды <ctrl> + <o>< o> текущие директивы компилятора можно записать в начале текста программы и исправить те из них, которые не соответствуют приведенным.

```
{$A+,B-,D+,E+,F+,G-,I+,L+,N+,O-,P-,Q+,R+,S+,T+,V+,X+,Y+}
{$M 65520,0,655360}
var
  i, j,k:longint;
procedure readdata;
begin
  assign(input,'');
  reset(input);
end;
procedure outdata;
begin
  assign(output,'');
  rewrite(output);
  close(output)
end;
procedure initial;
begin
  fillchar(i,65500,0);
end;
procedure run;
begin
end;
begin
  readdata;
  initial;
run;
  outdata
end.
```

Далее можно скопировать эту заготовку столько раз, сколько задач предложено на туре, и сразу назвать каждый файл так, как это требуется по условиям олимпиады. В результате вам не придется при переходе от решения одной задачи к другой начинать работу с нуля (а попытка править текст с решением одной задачи для ускорения набора текста другой ведет только к порождению ошибок, на исправление которых будет потрачено гораздо больше времени). В разделе **OPTIONS|enviroment|preferences** среды программирования Турбо Паскаль полезно установить параметр**Auto save [X]Editor Files** (автосохранение редактируемых файлов). Это гарантирует автоматическое сохранение текста программы при каждом ее запуске. Таким образом, если, например, программа зависнет и среду программирования придется запускать заново, то результат последнего редактирования будет сохранен (к сожалению, во время тура школьники зачастую забывают самостоятельно время от времени запоминать сделанные ими изменения в тексте программ).

Затем следует очень внимательно прочитать условия всех задач и постараться правильно (**!**) понять, в чем заключается каждая задача. Если что-то непонятно, в том числе и в формате ввода и вывода, то лучше задать вопрос.

Нужно формально подходить к тексту условия задачи, то есть понимать условие буквально, а не так, как покажется при его поверхностном чтении. Если же с точки зрения формальной логики условие все же можно трактовать неоднозначно, то тогда и следует задавать вопросы.

Если вы приступили к решению конкретной задачи и основная структура данных для нее вам ясна, то можно описать основные глобальные переменные и набить процедуру **readdata** ввода данных, чтобы она считывала все параметры задачи так, как это указано в условии. Если не оговорено иное, то делать формальную проверку считанных данных, то есть проверять соответствие введенных значений переменных условию задачи не нужно (**!**). Объясняется это тем, что, по правилам проведения большинства олимпиад последних лет, считается, что все входные данные при тестировании будут корректны. Кроме того, заметим, что при считывании из файла чисел обычно следует использовать только процедуру **read** (а не **readln**), для случаев же считывания символов и строк (тип **string**) это не так. Если количество чисел во входном файле неизвестно, то нужно использовать функцию **seekeof** вместо **eof** для проверки условия окончания считывания чисел. Для файлов, содержащих произвольный текст, это опять же уже не так.

В процедуре **initial** следует обнулить или присвоить соответствующие начальные значения всем (**!**) глобальным переменным, за исключением тех, которые будут использоваться в качестве параметров циклов. Затем запрограммировать вывод результата в процедуре **outdata** так, как это требуется в условии задачи. Это поможет дальнейшей отладке программы, и в дальнейшем не потребуется "простой" вывод переделывать в "правильный". Таким образом, к этому моменту у вас уже должна быть "работающая" программа. Она, по крайней мере, должна компилироваться, считывать данные и выводить результат, пусть и нулевой, но в нужном формате.

Еще одна типичная ошибка из данного класса — задание временных имен для входных и выходных файлов, и как результат, не работающая с точки зрения автоматической системы проверки программа.

5) При выполнении пунктов 3, 4 данной памятки или после следует подумать над подходами к решению задачи, для чего ответить себе на ряд вопросов и проделать ряд действий.

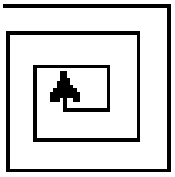
- ✓ Проверить данные на фактическую корректность, то есть определить, всегда ли задача имеет решение для введенного набора данных, например, связен ли граф, нет ли деления на 0 и т. п., если только в условии не сказано, что все данные и в этом смысле корректны.
- ✓ Определить, относится ли данная задача к знакомому вам классу, или решение придется искать "с нуля".
- ✓ Попытаться найти на бумаге (**!**) точное решение, возможно, только для малых размерностей. Такой подход зачастую позволяет обнаружить закономерности, которые затем можно попытаться распространить и на общий случай. Отобразить на бумаге принципиально различные случаи, в том числе и вырожденные. Это поможет при составлении тестов для самопроверки написанной программы.

Примеры олимпиадных задач с решением

Задачи

Задача 1. Спиральный массив

Расположить первые n^2 натуральных чисел в массиве размерностью nXn следующим образом



т.е. по спирали.

Формат входных данных.

Ввод числа n с клавиатуры.

Формат выходных данных.

Вывод матрицы на экран.

Разбор задачи

Внешние и внутренние фигуры подобны, т.е. цикл остается таким же, только изменяются значения начальной и конечной координат и причем оба на единицу. Необходимо только прописать все 4 хода с помощью форовского цикла.

Код программы:

```
program z1;

uses
  crt;

var
  a: array[1..100, 1..100] of integer;
  i, j, k, n, x1, x2, x3, x4, s: integer;

begin
  clrscr;
  writeln('Размерность массива:');
  readln(n);
  k := 1;x1 := 1;x2 := n;x3 := 2;x4 := 6;i := 1;
  while k <= sqr(n) do
  begin
    for j := x1 to x2 do
    begin
      a[i, j] := k;
      k := k + 1;
    end;
    for i := x3 to x2 do
    begin
      a[i, j] := k;
      k := k + 1;
    end;
    for j := x4 downto x1 do
    begin
      a[i, j] := k;
      k := k + 1;
    end;
    for i := x4 downto x3 do
    begin
      a[i, j] := k;
      k := k + 1;
    end;
    x1 := x1 + 1;
    x2 := x2 - 1;
    x3 := x3 + 1;
    x4 := x4 - 1;
  end;
  for i := 1 to n do
  begin
```

```
    for j := 1 to n do
      write(a[i, j]:3);
    writeln;
  end;
  readkey;
end.
```

Задача 2. Счастливые билеты

Создать программу определения числа билетов с 6-значными номерами, у которых сумма первых 3 десятичных цифр равна сумме 3 последних десятичных цифр.

Формат выходных данных.

Вывести количество счастливых билетов на экран.

Разбор задачи

Просматриваем числа от 0 до 999999. Делим число на 2 части: первые 3 цифры и последние 3 цифры, находим сумму цифр каждой из частей, сравниваем результат.

Код программы:

```
program z1;

function summ(x: longint): byte;{возвращает сумму цифр числа}
var
  k, l: byte;
  y: longint;
begin
  y := x;
  l := 0;
  while (y <> 0) do
  begin
    k := y mod 10;
    y := y div 10;
    l := l + k
  end;
  summ := l;
end;

var
  w1, w2, i, j, count: longint;
  n, m: byte;

begin
  count := 0;
  for j := 0 to 999999 do
  begin
    w1 := j div 1000;
    w2 := j mod 1000;
    if summ(w1) = summ(w2)
    then begin
      count := count + 1;
      writeln(j, ' --> ', count);
    end;
  end;
end.
```

Задача 3. Сапер

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	2 секунды

На прямоугольном поле размером 2 на N (N<=10000) в нижней строке случайным образом расставлено некоторое количество мин, не видимых сапёру, а в верхней строке в каждой клетке написаны числа от 0 до 3, которые совпадают с количеством мин в полях нижней строки, соседних с этой клеткой (расположены слева, под ней и справа). Требуется написать программу, которая находит все возможные расположения мин.

Формат входных данных.

Входной файл INPUT.TXT содержит в первой строке число N, а во второй - числа из верхней строки, записанные через пробел.

Формат выходных данных.

В первую строку выходного файла OUTPUT.TXT вывести количество возможных расположений мин (0, если такое невозможно). В следующих строках записать по одному найденному расположению мин (1 – есть мина, 0 – нет, числа разделить одним пробелом).

Пример

input.txt	output.txt
2 2 2	1 1 1

Разбор задачи

Пусть в левой нижней клетке есть мина, тогда, используя заданное значение в левой верхней клетке, можно узнать есть ли мина во второй слева нижней клетке (правда, при этом может получиться и недопустимое значение, но об этом далее). После этого, используя значение во второй верхней клетке, определяем наличие мины в третьей нижней клетке. Эти действия повторяем до тех пор, пока не произойдет одно из событий:

- найденные значения удовлетворяют всем равенствам,
- получено недопустимое значение.

Тогда в первом случае найдена расстановка мин, а во втором получили, что такой расстановки мин не существует. Проделав тоже самое в предположении, что в левом поле нет мины, мы либо найдем еще одну расстановку мин, либо определим, что таковой не существует.

Код программы:

```
program Mines;

var
  f, g: text;
  a, b: array[1..10000] of integer;
  n, i, a0, b0, c0, a1, b1, c1, c: integer;
  t0, t1: boolean;

begin
  assign(f, 'input.txt'); reset(f);
  readln(f, n);
  t0 := true; t1 := true;
  a0 := 0; b0 := 0; a[1] := b0;
  a1 := 0; b1 := 1; b[1] := b1;
  i := 1;
  while (n > i) and (t0 or t1) do
  begin
    i := i + 1;
    read(f, c);
    c0 := c - a0 - b0; a[i] := c0; a0 := b0; b0 := c0;
    t0 := t0 and ((c0 = 0) or (c0 = 1));
    c1 := c - a1 - b1; b[i] := c1; a1 := b1; b1 := c1;
    t1 := t1 and ((c1 = 0) or (c1 = 1))
  end;
```

```
  read(f, c);
  assign(g, 'output.txt'); rewrite(g);
  t0 := t0 and (a0 + b0 = c);
  t1 := t1 and (a1 + b1 = c);
  if t0 and t1 then write(g, 2)
  else
  if t0 or t1 then write(g, 1)
  else write(g, 0);
  if t0 then
  begin
    writeln(g);
    for i := 1 to n - 1 do write(g, a[i], ' ');
    write(g, a[n])
  end;
  if t1 then
  begin
    writeln(g);
    for i := 1 to n - 1 do write(g, b[i], ' ');
    write(g, b[n])
  end;
  close(g)
end.
```

Задача 4. Интересное число

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	2 секунды

Для заданного числа n найдите наименьшее положительное целое число с суммой цифр n, которое делится на n.

Формат входных данных.

Во входном файле содержатся целое число n ($1 \le n \le 1000$).

Формат выходных данных.

Выходной файл должен содержать искомое число. Ведущие нули выводить не разрешается.

Пример

input.txt	output.txt
1	1
10	190

Код программы:

```
program Number;

const
  max = 1 shl 12;
  inf = 1 shl 30;
  y = 10;

var
  a: array[0..max - 1, 0..max] of boolean;
  b, c: array[0..max - 1, 0..max] of integer;
  red: array[0..max - 1, 0..y - 1] of integer;
  qx, qy, s: array[0..max * (max + 1) - 1] of integer;
  front, rear, n, u, v, k, i, j: integer;
begin
  reset(input, 'input.txt');
  rewrite(output, 'output.txt');
  read(n);
  for i := 0 to n - 1 do
    for j := 0 to y - 1 do
      red[i][j] := (i * y + j) mod n;
  for i := 0 to n - 1 do
```

```
    for j := 0 to n do
      a[i][j] := false;
  a[0][0] := true;
  qx[0] := 0;
  qy[0] := 0;
  front := 0;
  rear := 1;
  while front < rear do
  begin
    i := qx[front];
    j := qy[front];
    inc(front);
    if (i = 0) and (j = n) then
    begin
      i := 0;
      j := n;
      k := 0;
      while j > 0 do
      begin
        u := b[i][j];
        v := c[i][j];
        s[k] := j - v;
        inc(k);
        i := u;
        j := v;
      end;
      for i := k - 1 downto 0 do
        write(s[i]);
      writeln;
      exit
    end;
    for k := 0 to y - 1 do
    begin
      u := red[i][k];
      v := j + k;
      if v > n then
        break;
      if not a[u][v] then
      begin
        a[u][v] := true;
        b[u][v] := i;
        c[u][v] := j;
        qx[rear] := u;
        qy[rear] := v;
        inc(rear);
      end;
    end;
  end;
  assert(false);
end.
```

Задача 5. Варианты решения

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	2 секунды

Входной файл INPUT.TXT содержит строку типа **a?b?c?f?e=zz** где a,b,c,....,z = числа типа byte, количество не превышает 20, zz = контрольный результат, умещается в типе Longint (как и все промежуточные результаты).

Требуется определить, что из множества **(+,*,-,div**) требуется подставить вместо знака **?**, чтобы получить верный пример. (Именно верный, с учётом приоритета операций). Вывести все возможные варианты.

Формат входных данных.

Входной файл INPUT.TXT содержит строку типа **a?b?c?f?e=zz**

Формат выходных данных.

В файл OUTPUT.TXT необходимо вывести все возможные варианты. Если вариантов не существует – записать в файл «No variants found».

Пример

input.txt	output.txt
3?2?1=6	3+2+1=6 3*2*1=6 3*2 div 1=6
3?2?1=22	No variants found

Разбор задачи

Задача решается путем перебора. Подробней читайте в комментариях кода программы.

Код программы:

```
Program Variants;
const
  OP_ADD = 1; // +
  OP_SUB = 2; // -
  OP_MUL = 3; // *
  OP_DIV = 4; // div

  opchars: array[1..4] of string = (' + ', ' - ', ' * ', ' div ');

type
  intarray = array[1..128] of integer;

var
  numbers, operations: intarray;
  res, numcount, opcount, varcount: integer;
  s: string;

procedure PrepareString;
var
  i: integer;
  t: string;
begin
  t := ''; numcount := 1; opcount := 1;

  for i := 1 to length(s) do
  begin
    if (s[i] = '?') or (s[i] = '=') then // если встретили ? или =
                                         // то начинается новое число
    begin
      numbers[numcount] := StrToInt(t);
                // записываем старое число в массив
      Inc(numcount);
      Inc(opcount);
      t := '';
    end else t := t + s[i]; //
  end;

  res := StrToInt(t); //то что осталось после = - это то что должно получится
  Dec(numcount);
  Dec(opcount, 2);
end;

function CalcString: integer;
var
  i, sum: integer;
```

```
  oldnumbers, oldoperations: intarray;
begin
  {
       Вычисление значения строки с учетом порядка действий.
       Основная идея в том, чтобы преобразовать последовательность  любых
       действий в последовательность только сложений, такое можно достичь
       следующим образом: a+b*c = a+0+b*c.
       Например, 2+3*4 = 2+0+(3*4) = 2+0+12 = 14
       Т.е. часть выражения 3*4 мы преобразовали в сумму 0+12
  }
  oldnumbers := numbers;       // сохранение старых данных нужно, т.к.
  oldoperations := operations; // числа и действия могут измениться

  for i := 1 to opcount do
  begin
    if operations[i] = OP_SUB then // преобразование вычитания в сложение
                                        // a-b = a+(-b)
    begin
      operations[i] := OP_ADD;
      numbers[i + 1] := -numbers[i + 1];
    end;

    if operations[i] > OP_SUB then  // если встретили умножение или деление
    begin
      if operations[i] = OP_MUL then
        numbers[i + 1] := numbers[i] * numbers[i + 1] else
        numbers[i + 1] := numbers[i] div numbers[i + 1];
                         // то преобразовываем в сложение
      operations[i] := OP_ADD;
      numbers[i] := 0;
    end;
  end;

     // подсчет суммы полученной последовательности чисел
  sum := 0;
  for i := 1 to numcount do Inc(sum, numbers[i]);

  numbers := oldnumbers;       // восстановление исходных данных
  operations := oldoperations;

  CalcString := sum;
end;

procedure ChooseNextOp(k: integer);
var
  i, r: integer;
begin
  if k > opcount then
  begin
          // если выбрали все действия, то подсчитываем результат
          // и сравниваем с нужным
    r := CalcString;

    if r = res then
    begin
      Inc(varcount);
               // распечатка полученной цепочки действий
      write(varcount);
      for i := 1 to opcount do
        write(opchars[operations[i]]);
```

```
      writeln(numbers[numcount], ' = ', r);
    end;

  end else
  begin
        // выбираем дейсвие
    for i := OP_ADD to OP_DIV do
    begin
      operations[k] := i;
      ChooseNextOp(k + 1); // и переходим к выбору след. действия
    end;
  end;
end;

begin
  assign(input, 'input.txt'); reset(input);
  readln(input, s);
  assign(output, 'output.txt');
  rewrite(output);
  PrepareString; // подготовка данных
  varcount := 0;
  ChooseNextOp(1); // выбираем действия
  // если вариантов нет выводим сообщение
  if varcount = 0 then writeln('No variants found'); // Варианты не найдены
  close(input);
  close(output);
end.
```

Задача 6. Города и дороги

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	5 секунд

В галактике "Milky Way" на планете "Neptune" есть N городов, некоторые из которых соединены дорогами. Император "Maximus" галактики "Milky Way" решил провести инвентаризацию дорог на планете "Neptune". Но, как оказалось, он не силен в математике, поэтому он просит вас сосчитать количество дорог.

Формат входных данных.

В файле INPUT.TXT записано число N (0 <= N <= 100). В следующих N строках записано по N чисел, каждое из которых является единичкой или ноликом. Причем, если в позиции (i,j) квадратной матрицы стоит единичка, то i-ый и j-ый города соединены дорогами, а если нолик, то не соединены.

Формат выходных данных.

В файл OUTPUT.TXT вывести одно число - количество дорог на планете "Neptune".

Пример

input.txt	output.txt
5 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0	3

Разбор задачи

В задаче необходимо посчитать количество ребер в графе, вершинами которого являются города, а ребрами - дороги. Во входном файле дана матрица смежности этого графа. Заметим, что каждому ребру соответсвуют ровно две единицы в матрице смежности, симметричные относительно главной диагонали. Значит, ответом на поставленную задачу будет количество единиц в матрице смежности, деленное на 2.

Код программы:

```
program cities_and_roads;
  const MaxN = 50;
INF = 1000000000; //"бесконечность"

type Matrix = array[1..MaxN,1..MaxN] of longint; //тип матрицы смежности.
M[i,j] = true, если существует ребро, идущее от вершины i к j

var
A : Matrix; N: integer;
input: text;

procedure Input_Table(var A : Matrix; N : longint; var T : Text); //процедура
ввода матрицы смежности A(N, N) из текстового файла T
var i, j : longint;
begin
    for i := 1 to N do
    begin
        for j := 1 to N do
        begin
            read(T, A[i, j]);
            if (a[i,j] = 0) and (i <> j) then a[i,j] := INF; //вершины,
которые не связаны ребром, будем обозначать "бесконечностью" ввиду
ограничения на вес рёбер
        end;
        readln(T);
    end;
end;
```

```
procedure Vse_Rebra(var A : Matrix; N : longint); //процедура ввода матрицы смежности A(N, N) из текстового файла T
var i, j, k: longint;//объявлем переменные цикла i, j и переменную счетчика вершин k
begin
    k := 0;//устанавливаем значение счетчика равным нулю
    for i := 1 to N do //запускам цикл для поиска в строке
    begin
        for j := 1 to N do //запускаем цикл для поиска в столбцах
        begin
            if (a[i,j] = 1) then k:= k+1;//Находим вершины ребра и увеличиваем счетчик k
        end;
    end;
    k := k div 2;//делим на 2 найденые вершины для получения общего количества ребер
    writeln(k);//выводим найденое количество ребер на экран
end;

BEGIN
    assign(input,'input.txt');//подключаем файл input.txt для считывания данных
    reset(input);//обнуляем переменную файла для начала считывания с файла
    readln(input, N);//из первой строки считываем N - количество строк матрицы смежности
    Input_Table(A, N, input);//передаем N и файл в процедуру для заполнения матрицы и получаем переменную A - заполненую матрицу
    close(input);//закрываем файл
    Vse_Rebra(A, N);//передаем количество строк N и заполненую матрицу A в процедуру для нахождения количества ребер
END.
```

Задача 7. Светофорчики

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	5 секунд

В подземелье M тоннелей и N перекрестков, каждый тоннель соединяет какие-то два перекрестка. Мышиный король решил поставить по светофору в каждом тоннеле перед каждым перекрестком. Напишите программу, которая посчитает, сколько светофоров должно быть установлено на каждом из перекрестков. Перекрестки пронумерованы числами от 1 до N.

Формат входных данных

В файле INPUT.TXT записано два числа N и M (0 < N <= 100, 0 <= M <= N*(N-1)/2). В следующих M строках записаны по два числа i и j (1 <= i,j <= N), которые означают, что перекрестки i и j соединены тоннелем.

Формат выходных данных

В файл OUTPUT.TXT вывести N чисел: k-ое число означает количество светофоров на k-ом перекрестке.

Примечание Можно считать, что любые два перекрестка соединены не более, чем одним тоннелем. Нет тоннелей от перекрестка i до него самого.

Пример

input.txt	output.txt
7 10 5 1 3 2 7 1 5 2 7 4	3 3 2 2 5 2 3

6 5 6 4 7 5 2 1 5 3	

Разбор задачи

Нам требуется посчитать степени всех вершин. Эта задача не должна вызывать больших трудностей. Попробуйте при реализации этой задачи построить в памяти матрицу смежности графа (хотя задачу можно легко решить и без этого, но умение преобразовывать список ребер к матрице смежности и обратно нам пригодится в дальнейшем).

Код программы:

```
program Svetofor;
const
  MaxN = 50;
  INF = 1000000000;//"бесконечность"

type
  Matrix = array[1..MaxN] of longint;//тип матрицы смежности. M[i,j] = true,
если существует ребро, идущее от вершины i к j

var
  A: Matrix; N, M: integer;
  input: text;

procedure Input_Table(var A: Matrix; N, M: longint; var T: Text);//процедура
ввода матрицы смежности A(N, N) из текстового файла T
var
  i, j, rs, re: integer;
begin
  for i := 1 to N do A[i] := 0;
  for j := 1 to M do
  begin
    read(T, rs, re);
    for i := 1 to N do
    begin
      if (i = rs) or (i = re) then A[i] := A[i] + 1;
    end;
  end;
  for i := 1 to N do write(A[i], ' ');
end;

begin
  assign(input, 'input.txt');
  reset(input);
  readln(input, N, M);
  Input_Table(A, N, M, input);
  close(input);
end.
```

Задача 8. Цветной дождь

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	5 секунд

В Банановой республике очень много холмов, соединенных мостами. На химическом заводе произошла авария, в результате чего испарилось экспериментальное удобрение "зован". На следующий день выпал цветной дождь, причем он прошел только над холмами. В некоторых местах падали красные

капли, в некоторых - синие, а в остальных - зеленые, в результате чего холмы стали соответствующего цвета. Президенту Банановой республики это понравилось, но ему захотелось покрасить мосты между вершинами холмов так, чтобы мосты были покрашены в цвет холмов, которые они соединяют. К сожалению, если холмы разного цвета, то покрасить мост таким образом не удастся. Посчитайте количество таких "плохих" мостов.

Формат входных данных

В файле INPUT.TXT в первой строке записано N (0 < N <= 100) - число холмов. Далее идет матрица смежности, описывающая наличие мостов между холмами (1-мост есть, 0-нет). В последней строке записано N чисел, обозначающих цвет холмов: 1 - красный; 2 - синий; 3 - зеленый.

Формат выходных данных

В файл OUTPUT.TXT вывести количество "плохих" мостов.

Пример

input.txt	output.txt
7 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 3 3	4

Разбор задачи

Переберем все пары холмов. Если данная пара холмов соединена мостом и покрашена в разные цвета, увеличиваем ответ на 1.

Код программы:

```
program Color_Hills;
const
  MaxN = 100;
  INF = 1000000000;//"бесконечность"

type
  Matrix = array[1..MaxN, 1..MaxN] of longint;//тип матрицы смежности. M[i,j]
= true, если существует ребро, идущее от вершины i к j

type
  Color = array[1..MaxN] of longint;//в данном массиве зависывается цвет
холма (вершины)

var
  Hills: Matrix; N: integer;
  ColorHill: Color;
  input: text;

procedure Input_Table(var Hills: Matrix; N: longint; var T: Text);//процедура
ввода матрицы смежности A(N, N) из текстового файла T
var
  i, j: longint;
begin
  for i := 1 to N do
  begin
    for j := 1 to N do
    begin
      read(T, Hills[i, j]);
```

```
      if (Hills[i, j] = 0) and (i <> j) then Hills[i, j] := INF; //вершины,
которые не связаны ребром, будем обзначать "бесконечностью" ввиду ограничения
на вес рёбер
    end;
    readln(T);
  end;
  for i := 1 to N do
  begin
    read(T, ColorHill[i]);
    write(ColorHill[i], ' ');
  end;
  writeln;
end;

procedure Plohie_mosti(var Hills: Matrix; N: longint);//процедура ввода
матрицы смежности A(N, N) из текстового файла T
var
  i, j: longint;
  m, k: integer;
begin
  k := 0;
  for i := 1 to N do
  begin
    m := Hills[i, i];
    for j := 1 to N do
    begin
      if (Hills[i, j] = 1) and (ColorHill[i] <> ColorHill[j])  then k := k +
1
    end;
  end;
  k := k div 2;
  writeln(k);
end;

begin
  assign(input, 'input.txt');
  reset(input);
  readln(input, N);
  Input_Table(Hills, N, input);
  close(input);
  Plohie_mosti(Hills, N);
end.
```

Задача 9. Издевательство

Имя входного файла	input.txt
Имя выходного файла	output.txt
Максимальное время работы на одном тесте:	5 секунд

В городе N площадей. Любые две площади соединены между собой ровно одной дорогой с двусторонним движением. В этом городе живет Штирлиц. У Штирлица есть хобби - он любит воскресным утром выйти из дома, сесть в машину, выбрать какой-нибудь кольцевой маршрут, проходящий ровно по трем площадям (то есть сначала он едет с какой-то площади на какую-то другую, потом - на третью, затем возвращается на начальную, и опять едет по этому маршруту). Он воображает, что где-то на этом пути стоит Борман. И так вот ездит Штирлиц все воскресенье, пока голова не закружится, и радуется...

Естественно, что Штирлицу хочется проезжать мимо точки, в которой, как он воображает, стоит Борман, как можно чаще. Для этого, естественно, выбранный Штирлицем маршрут должен быть как можно короче. Напишите программу, которая выберет оптимальный для Штирлица маршрут.

Формат входных данных

Во входном файле INPUT.TXT записано сначала число N (3 <= N <= 100), а затем матрица NxN расстояний между площадями (число в позиции i,j обозначает длину дороги, соединяющей i-ую и j-ую

площади). Все числа в матрице (кроме стоящих на главной диагонали) - натуральные, не превышающие 1000. Матрица симметрична относительно главной диагонали, на главной диагонали стоят 0.

Формат выходных данных

В выходной файл OUTPUT.TXT выведите номера площадей в оптимальном маршруте. Если маршрутов несколько, выведите любой из них.

Пример

input.txt	**output.txt**
5 0 20 10 30 40 20 0 30 1 2 10 30 0 40 1000 30 1 40 0 21 40 2 1000 21 0	4 5 2

Разбор задачи

Нам требуется найти цикл длины 3 минимального веса в полном взвешенном графе. Переберем всевозможные тройки вершин (это можно сделать, например, тремя вложенными циклами - каждая из переменных цикла соответствует какой-то из трех искомых вершин). Любая тройка однозначно задает цикл длины три. Выберем тройку, для которой вес соответсвующего цикла минимален.

Код программы:

```
program Izdevatelsvo;
const
  MaxN = 100;
  INF = 1000000000;//"бесконечность"

type
  Matrix = array[1..MaxN, 1..MaxN] of longint;//тип матрицы смежности. M[i,j]
= true, если существует ребро, идущее от вершины i к j

var
  Road: Matrix; N: integer;
  input: text;

procedure Input_Table(var Road: Matrix; N: longint; var T: Text);//процедура
ввода матрицы смежности A(N, N) из текстового файла T
var
  i, j: longint;
begin
  for i := 1 to N do
  begin
    for j := 1 to N do
    begin
      read(T, Road[i, j]);
      if (Road[i, j] = 0) and (i <> j) then Road[i, j] := INF; //вершины,
которые не связаны ребром, будем обзначать "бесконечностью" ввиду ограничения
на вес рёбер
    end;
    readln(T);
  end;
end;

procedure Three_Road(var Road: Matrix; N: longint);//процедура ввода матрицы
смежности A(N, N) из текстового файла T
var
  i, j, a, b, c, l, x, y, z: longint;
begin
  l := maxint;
  for a := 1 to N - 2 do
```

```
    for b := a + 1 to N - 1 do
      for c := b + 1 to N do
        if Road[a, b] + Road[b, c] + Road[a, c] < l then
        begin
          l := Road[a, b] + Road[b, c] + Road[a, c];
          x := a;
          y := b;
          z := c;
        end;
  writeln(x, ' ', y, ' ', z);
end;

begin
  assign(input, 'input.txt');
  reset(input);
  readln(input, N);
  Input_Table(Road, N, input);
  close(input);
  Three_Road(Road, N);
end.
```

Задача 9. Метрополитен (поиск в ширину)

В некотором городе Н-ске имеется свой метрополитен. Администрация города решила построить новую кольцевую линию. "Центром" этого кольца должна стать одна станций, при этом "радиус" такого кольца - минимальное кол-во станций, которое надо проехать, чтобы достичь кольцевой линии. То есть если радиус равен R - то кольцевая линия должна строиться на тех станциях, до которых можно добраться как минимум через R станций. Кольцо должно иметь как минимум две станции. Программа должна вывести все станции, через которые надо провести новую линию, или 0, если кольцо нельзя постоить.

Формат входных данных

Первая строка содержит число N - количество станций метро (1<=N<=150) и число K. В следующих K строках содержится информация, описывающая схему. Сначала в строке идёт D - номер станции, затем число M, а далее - M чисел, которые показывают, с какими станциями соединена станция D.

В предпоследней строке - станция, которая будет "центром" станции, а в последней – радиус кольца.

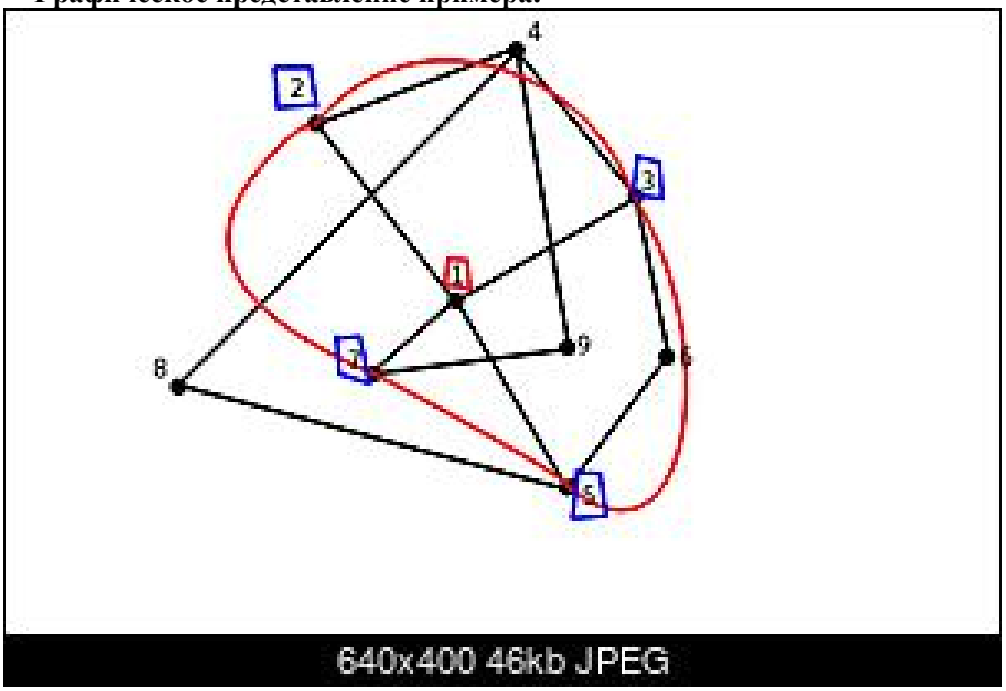
Формат выходных данных

На экран вывести номера искомых станций (в любом порядке). Если кольцо нельзя построить - вывести 0.

Пример:

input.txt	output.txt
9 5 3 3 1 4 6 2 2 1 4 7 2 1 9 5 3 1 8 6 4 2 8 9 1 1	2 3 5 7

Графическое представление примера:



Решение: при помощи поиска в ширину (BFS) на k-ой итерации цикла мы будем находить все станции, до которых можно добраться как минимум через k станций. Сделав ограничение по R и записывая "уровень" каждой станции, мы определим все такие станции.

Код программы:

```
const MaxN = 100;
type Matrix = array[1..MaxN,1..MaxN] of boolean; //тип матрицы смежности. M[i,j] = true, если станции i и j соединены.

var
A : Matrix;
i, j, N, k, aa, bb, S, R, c: integer;

procedure BFS(A : Matrix; N, V, R : integer); //обход в ширину (V - корневая вершина)
var i, ps, pe : integer;
visited : array [1..MaxN] of boolean; //массив посещённости вершин
q : array [1..MaxN] of integer; //"очередь" вершин
ql : array [1..MaxN] of integer; //и их уровней.
begin //в качестве списка очередных вершин будем использовать структуру "очередь"
    ps := 1; //начало очереди
    pe := 1; //конец очереди
    q[pe] := v; //в очередь помещается исходная вершина. На каждом шаге в очередь будем заносить всех преемников данной вершины (назовём их 1-го уровня, т.е. до которых можно дойти напрямую). Затем просматриваем в очереди занесённые ранее вершины 1-го уровня и добавляем в конец очереди вершины 2-го уровня и так далее до опустошения очереди.
    visited[v] := TRUE; //вершина V посещена
    ql[ps] := 0; //сама вершина - нулевой уровень
    while (ps <= pe) and (ql[ps] < R) do //пока очередь не пуста или не достигли необходимого уровня R
    begin
        for i := 1 to n do if (A[v, i]) and (not visited[i]) then //перебираем все связные с V вершины
        begin
            inc(pe); //и добавляем в очередь
            q[pe] := i;
            ql[pe] := ql[ps] + 1; //отмечаем новый уровень преемника.
```

```
            visited[i] := true; //отмечаем вершину I пройденной
        end;

        inc(ps); //переходим к следующей вершине в очереди
        v := q[ps]; //и делаем её корневой
    end;
    while (ql[ps] = R) and (ps <= pe) do //после работы цикла в очереди
останутся только вершины, имеющие уровень R и более.
    begin
        write(q[ps],' ');
        inc(ps);
    end;
end;

BEGIN
    assign(input,'input.txt');
    reset(input);
    readln(input, N, c);
    for i:=1 to c do
    begin
        read(input, aa, k);
        for j:=1 to k do
        begin
            read(input, bb);
            A[aa, bb] := true;
            A[bb, aa] := true;
        end;
        readln(input);
    end;
    readln(input, S);
    readln(input, R);
    close(input);
    BFS(A, N, S, R);
END.
```

Полезные ссылки

e-maxx - сайт с множеством алгоритмов.
algolist.manual.ru - неплохой сайт с алгоритмами
habrahabr.ru - интересные статьи
olimp.bstu.by - контесты(можно задачи решать)
informatics.mccme.ru - очень большой архив задач, также есть теория
dl.gsu.by - очень большой архив задач
brtrg.com - регулярные соревнования и в блоге теорию можно почитать
cppreference.com - описание функций и библиотек C++
cplusplus.com - описание функций и библиотек C++
codeforces.ru - очень хороший сайт по программированию
g6prog.narod.ru - сайт с разборами некоторых задач
acm.timus.ru - сайт с большим кол-вом задач
uva.onlinejudge.org - хороший архив задач, но по большей части для АСМщиков
spoj.pl – архив олимпиадных задач
e-olimp.com - архив задач и регулярно проводятся олимпиады.
http://algolist.manual.ru/olimp (сайт «Олимпиадные задачи по программированию»);
http://www.olympiads.ru/moscow (сайт московских олимпиад по информатике);
http://neerc.ifmo.ru/school (сайт «Олимпиады по информатике. Санкт-Петербург, Россия»);
http://contest.ur.ru (сайт Уральских олимпиад по информатике);
http://www.olympiads.ru (сайт по олимпиадной информатике);
http://www.olympiads.nnov.ru (сайт «Олимпиадная информатика в Нижнем Новгороде»);
http://acmp.ru или http://acm.dvpion.ru (сайт «Школа программиста» для школьников Красноярского края);
http://acmu.ru (сайт «Олимпиады по информатике для школьников Ханты-Мансийского автономного округа»);
http://yadi.sk/d/sqIyOreAJ6Ce8 (сайт, содержащий сборник заданий муниципального этапа в МАО);
http://imcs.dvgu.ru/works/school.html (сайт школьных олимпиад, проводимых в Приморском крае);
http://imcs.dvgu.ru/ru/event/jpa/2010/ai.html (сайт ДВФУ с описанием системы для проведения соревнований по игровому ИИ для школьников);
http://imcs.dvgu.ru/works/work?wid=12124 (сайт ДВФУ с описанием системы для проведения олимпиад по информатике для младших школьников);
http://olymp.karelia.ru/pract.htm (сайт школьных олимпиад Республики Карелия);
http://school.sgu.ru (сайт по алгоритмизации и программированию Саратовского государственного университета);
http://www.olympiads.ru/moscow/2009/79/archive/index.shtml (сайт с задачами московской олимпиады школьников по программированию для 7 – 9 классов).

Список книг

Алгоритмы.Построение и анализ. Кормен - подробно описываются алгоритмы
Сборник статей вычислительная геометрия на плоскости Андреевой - по геометрии
Вычислительные машины и труднорешаемые задачи. Гэри - книга по алгоритмам
Сборник лекций Михаила Густокашина - хорошо описаны структуры данных и некоторые алгоритмы(только C++)
Искусство программирования Кнут - великое произведение по теории алгоритмов
Комбинаторика для программистов Липский - книга по комбинаторике
Комбинаторная оптимизация. Алгоритмы и сложность Стайглиц
Конкретная математика Грэхэм - очень хорошая книга по математике
Перечисление графов Харари - книга по графам
Программирование в алгоритмах Окулов
Решение сложных олимпиадных задач по программированию Долинский - разбор различных задач.
Фундаментальные алгоритмы C++ Седжвик - хорошая книга по алгоритмам
Строки, деревья и последовательности в алгоритмах Гэсфилд

Список использованной литературы:

Кирюхин В.М. Информатика: всероссийские олимпиады. Выпуск 1/ В.М. Кирюхин – М.: Просвещение, 2008 .

Кирюхин В.М., Цветкова М.С. Всероссийская олимпиада школьников по информатике в 2006 году / Науч. ред. Э.М. Никитин. – М.: АПКиППРО, 2006.

http://acmp.ru - "Школа программиста"

Информатика. Международные олимпиады. Вып. 1, - М.: Просвещение, 2009 г., 240 стр.

Методика решения задач по информатике. Международные олимпиады. Школа (в соавторстве с Окуловым С.М.). - Издательство: Бином. Лаборатория знаний, 2007.

http://www.informatics.ru/

http://comp-science.narod.ru/

http://algolist.manual.ru/

http://comp-science.narod.ru/links.html

http://ulm.uni.udm.ru/~pvv/

http://comp-science.narod.ru/olimp.html

http://www.olympiads.ru/

http://acm.timus.ru/

http://byoi.narod.ru/

http://www.olymp.vinnica.ua/

http://contest.ur.ru/

http://inf777.narod.ru - сайт "Информатика в школе"