# Глава 1. ИНФОРМАТИКА - КЛЮЧЕВОЙ ПРЕДМЕТ СОВРЕМЕННОЙ ШКОЛЫ

## 1.1. Программирование

> Врач, инженер-строитель и программист спорили о том, чья профессия древнее. Врач заметил: «В Библии сказано, что Бог сотворил Еву из ребра Адама. Такая операция может быть проведена только хирургом, поэтому я по праву могу утверждать, что моя профессия самая древняя в мире». Тут вмешался строитель и сказал: «Но еще раньше Бог сотворил небо и землю из хаоса. Это первое и, несомненно, наиболее выдающееся применение строительной инженерии. Поэтому, дорогой доктор, Вы не правы. Моя профессия самая древняя в мире». Программист при этих словах откинулся в кресле, загадочно улыбнулся и произнес: «Да, но кто, как вы думаете, сотворил хаос?»
>
> Бруч Г.Объектно - ориентированное проектирование с примерами применения. - М.: Конкорд, 1992.

*Вводные замечания*. Вопросы: что такое программирование, кто такой программист? Уточним вопрос. Является ли человек, умеющий разрабатывать программы средней сложности, например на языке программирования Бейсик, программистом? В работе [16] определено: «Основными видами человеческой интеллектуальной деятельности, изучаемой в информатике, являются:

- математическое моделирование (фиксация результатов познавательного процесса в виде математической модели);
- алгоритмизация (реализация причинно-следственных связей и других закономерностей в виде направленного процесса обработки информации по формальным правилам);
- программирование (реализация алгоритма на ЭВМ);
- выполнение вычислительного эксперимента (получение нового знания об изучаемом явлении или объекте с помощью вычисления на ЭВМ);
- решение конкретных задач, относящихся к кругу объектов и явлений, описываемых исходной моделью».

Академик А.П. Ершов говорил «о программировании, как второй грамотности», а вслед за ним практически все авторы современных учебников по информатике вводят разделы программирования. Следовательно, ответ положительный - «да». Итак, получается, что школьник, уверенно пишущий программы на одном, а может быть, и нескольких языках программирования, является программистом. Мы не будем говорить об уровне профессионализма, суть от этого не меняется (главное - ответ положительный). Хочется сказать «нет»и еще раз «нет». Знание математики в объеме школьной программы не говорит о том, что человек является математиком. Знание основ программирования не делает из человека программиста. В этом случае для чего необходимо изучение основ программирования? Может быть, справедлива точка зрения [3], что «изложенная модель постепенного растворения информатики (а у нас все программисты) в других предметах представляется наиболее перспективной», или утверждение о том, что нашему обществу не требуется такое количество программистов, или сведение информатики к изучению только очередных «букетов» стандартного прикладного программного обеспечения. Попытаемся опровергнуть эти точки зрения, и, так как образовательная информатика является определенным временным «срезом» информатики - как науки и, главное, информатики - как отрасли производства, без исторического экскурса не обойтись.

Программным проектам присуща сложность, что является отнюдь не случайным свойством, а скорее необходимым. Всю историю развития технологии программирования можно рассматривать под углом преодоления трудностей (хаоса) на пути создания сложных систем. Это развитие, как любое развитие, шло и идет по спирали (более детально об этом в следующем параграфе «Парадигмы программирования»). Каждый виток определен своими принципами декомпозиции, абстрагирования и иерархии. Определимся с тем, какие по сложности бывают программы. Согласно [ 10]:

| Программа | Количество операторов исходного текста (до) |
|---|---|
| Простая | 1000 |
| Средней сложности | 10 000 |
| Сложная | 100 000 |
| Сверхсложная | 1 000 000 |
| Гиперсложная | 10 000 000 и более |

Вопрос. Есть ли школьники или студенты, написавшие за время обучения программу (хотя бы одну) средней сложности? Есть, но их количество несопоставимо с числом обучаемых. Назовем ли мы человека столяром только по его умению изготавливать ножки для стульев и столов различного типа или по его умению забивать гвозди молотками различной тяжести? Вряд ли. И вряд ли мы доверим свою жизнь летчику, умеющему только поднимать самолет в воздух.

В 70-х годах считалось [11], что программный проект должен пройти через следующую цепочку специалистов:

•конечный пользователь (автор исходной задачи и в ряде случаев заказчик соответствующего программного проекта);

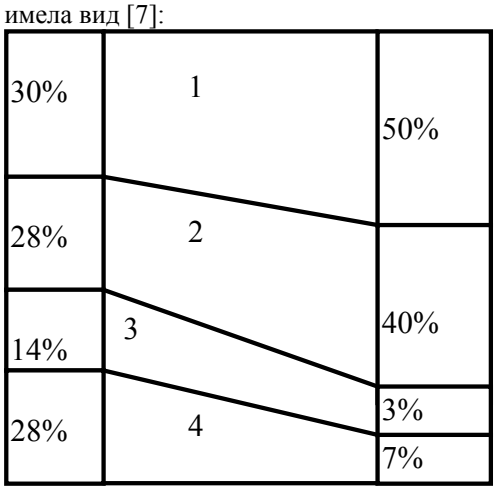
•системный аналитик (преобразует цели, назначение, технические характеристики и условия правильности решения в общие формальные требования на программный проект);

•системный аналитик-программист (преобразует общие формальные требования в детальные спецификации на отдельные программы, участвует в разработке логической структуры и т. д.);

•прикладной программист (преобразует спецификации на программу в логическую структуру программных модулей, а затем в программный код);

•системный программист (ведет сопровождение операционной системы, разрабатывает инструментальные средства для поддержки процесса разработки программ, обеспечивает сопряжение с операционной системой драйверов нестандартных внешних устройств и т. д.).

Динамика структуры трудозатрат по фазам реализации программного проекта в 1970 -1980 гг. имела вид [7]:




1970-1975 гг. 1980-1985 гг.

1 -сопровождение и поддержка;
2 - анализ приложений и постановка задачи
3- программирование и автономная отладка
4- комплексная отладка и приемосдаточные испытания

В настоящее время [7] половина общей численности мирового корпуса программистов - системные аналитики, которые нередко вообще не пишут ни одной строчки программного кода. В таком случае кто такой программист? Если человек, ответственный за все фазы прохождения программного проекта (кроме, разумеется, фазы конечного пользователя), то только 3% своего времени он занимается той деятельностью, которая согласно [16] называется программированием. Итак, видимо, следует считать программистом и специалиста, связанного с реализацией программных проектов, но не знающего языков программирования (утрированный случай с целью подчеркнуть данную мысль).

Программирование как интеллектуальная деятельность, как технология в своем развитии прошло различные этапы.

Первый этап. Программисты работают на уровне регистров и ячеек памяти ЭВМ (отдельных деталей), программируются, в основном, математические формулы, решаются задачи численного анализа.

Второй этап. Рост сложности программ. Невозможность работать по-старому. Развитие языков высокого уровня, структурное проектирование, длительные усилия специалистов по формализации. Например, в [13] сказано, что «программирование как научная дисциплина является частью прикладной математики, а как вид деятельности - разновидностью математической практики, и по своей природе основания программирования являются логико-математическими». Усилиями классиков программирования Э. Дейкстрой [8], Ч. Хоаром[20], Д. Кнутом, Э. Йоданом [10] и др. разрабатываются формальные аспекты науки программирования. Предложена технологическая цепочка, рассмотренная выше. Однако после 10 - 15 лет значительных усилий осознано, что цена этого слишком велика. Невозможно на бумаге создать сложные информационные конструкции, усилиями мысли заставлять их при этом взаимодействовать с реальным миром (во всяком случае так, как этот мир понимается). Процент задач, для которых принципиально можно разработать формально безупречные спецификации, в общем объеме прикладного программного обеспечения снижается с расширением сфер применения ЭВМ. И вслед за Г.Р. Громовым можно сказать, что «было бы столь же трудно ожидать, что воспринимаемые буквально математические приемы, иллюстративные рекомендации и метафорические научные прогнозы классиков программирования могли бы оказаться практически более полезны, чем, например, использование в клинической практике ярких медицинских зарисовок А.П. Чехова». К программированию, по мнению Г. Л. Смоляна [18], «вполне применим тезис Д. Гильберта, утверждавшего, что всякая физическая или математическая теория проходит три фазы развития: наивную, формальную и критическую». Программирование как инженерная деятельность, как отрасль производства находится в последней фазе. Образовательная информатика пытается разродиться от формализма. Явная видимость его недостаточности и отсутствие, в первом приближении, достойной замены, приводит к отчаянным «броскам», например, в направлении чисто прикладного программного обеспечения.

Третий этап. Дальнейшее развитие программирования происходило в двух направлениях, связанных между собой: персональные вычисления и разработка инструментальных сред для создания программных проектов, реализующих на новом витке спирали принципы декомпозиции, абстракции и иерархии. Во-

первых, это разработка объектно-ориентированного подхода, от его первых версий, например в Турбо - Паскале 5.5, до Object Pascal среды Delphi, который определяет новый уровень развития перечисленных принципов. Во-вторых, создание визуальных сред быстрой разработки проектов (Delphi), основанных на компонентной (элементе иерархии) архитектуре. Разработка, основанная на компонентах, оказывается следующим эволюционным шагом в развитии методов программирования. Delphi по сравнению с другими существующими средами программирования дает возможность исследовать различные варианты программного проекта, делать ошибки прежде, чем будет принято определенное решение. Другими словами, может быть, впервые у программистов появилась возможность начинать разработку не с традиционных попыток составления формальных спецификаций, а с разработки макета. Макет программы - информационный объект, единственное назначение которого дать возможность пользователю и программисту выработать единое понимание того, что должна делать программа, ибо «определить, что хочет пользователь, в отличие от того, что он говорит, что он хочет, - это и есть настоящее искусство». Книга Д. Пойа «Математическое открытие» начинается со следующего утверждения Лейбница: «Метод решения хорош, если с самого начала мы можем предвидеть - и далее подтвердить это, - что, следуя этому методу, мы достигнем цели». В технологии программирования создание макетов средствами визуального программирования - единственный на сегодня метод, удовлетворяющий критерию Лейбница.

Программист - это специалист, своей деятельностью охватывающий все циклы (фазы) прохождения программных проектов. При этом доля (процент) времени, когда он занимается кодированием и автономной (программированием) отладкой программ, минимальна (может совсем отсутствовать). Обучение написанию простых программ на одном из языков программирования является не более чем пропедевтикой программирования. Знание языков программирования не делает из человека программиста.

## 1.2. Парадигмы программирования

Стиль - это отражение мышления.
Шопенгауэр А.

Парадигма (греч. paradeigma - пример, образец) - схема, модель постановки проблем и их решения, методы исследования, господствующие в течение определенного исторического периода в научном обществе. Характерные идеи и методы программирования и соответствующий образ мышления образуют так называемую парадигму программирования. Между парадигмами и языками программирования высокого уровня прямая связь, хочется сказать, изоморфная (противники этой точки зрения будут утверждать что и на Бейсике можно создать «шедевр»...). В большинстве учебников по информатике подразумевается, что суть программирования заключается в знании алгоритмов и определений языка, описанных в них. Но правила конкретного языка программирования можно изучить за несколько часов, соответствующие парадигмы требуют гораздо больше времени как для того, чтобы научиться им, так и для того, чтобы отучиться от них .

Примечание. У Йодана Э.[10] приводится следующая притча о классике программирования Э. Дейкстре (элемент фольклора в computer science). Известно, что Э. Дейкстра был мало заинтересован в приеме на старшие курсы университета, где он работал, студентов со знанием Фортрана по той причине, что вместе с этим знанием могли привиться дурные привычки программирования. От него пошло также высказывание, что если знание Фортрана можно сравнить с младенческим расстройством, то уж ПЛ/1 - определенно роковая болезнь. Язык программирования ПЛ/1 ушел в прошлое, сегодня, видимо, о Бейсике можно говорить как о таком типе болезни.

*Операциональное программирование* (языки программирования типа ассемблеров, Бейсика, Фортрана). Программа «собирается» из мелких деталей, отдельных операций и имеет достаточно простую структуру: область глобальных данных и подпрограммы. Уровень абстрагирования - отдельное действие, принципы декомпозиции задачи отсутствуют, во всяком случае, о них не говорят.

*Нисходящая технология конструирования программ*. Суть нисходящего конструирования программ в разбивке большой задачи на меньшие подзадачи, которые могут рассматриваться отдельно. Основными правилами для успешного применения данной технологии являются:

- формализованное и строгое описание программистом входов функций и выходов всех модулей программы и системы;
- согласованная разработка структур данных и алгоритмов;
- ограничение на размер модулей.

Нисходящая технология разработки программ не есть свод жестких правил, скорее это основной принцип, допускающий вариации в соответствии с конкретными особенностями решаемой задачи. В свое время в обширной литературе по этому поводу говорилось и о *восходящей* технологии. В этом случае решение (программа) как бы «складывалось из отдельных кирпичиков», из известных решений подзадач. Таким образом, данной технологией оговаривается определенный принцип декомпозиции и иерархическая структура программы. Если проводить черту, то суть этого этапа развития технологии программирования в том, что «мы не знаем, как этого достичь, у нас нет инструментария для обеспечения процесса правильной разработки программы, но так должно быть».

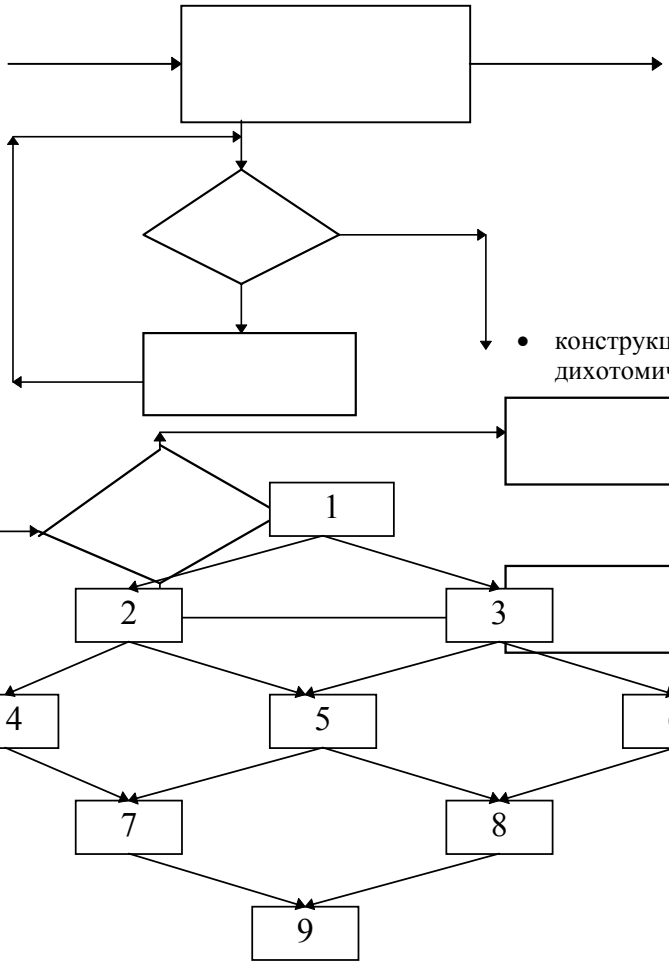
*Структурное программирование* (языки программирования Паскаль, Модула-2). Профессор Э. Дейкстра был первым инициатором структурного программирования. В 1965 г. он высказал предположение

о том, что оператор GO TO мог бы быть исключен из языков программирования. Разумеется, структурное программирование представляет собой нечто большее, чем один лишь отказ от оператора GO TO. Структурное программирование - это некоторые принципы написания программ. Теоретическими основаниями структурного программирования являются:

- формальные системы теории вычислимости (общие рекурсивные функции, системы Поста, алгоритмы Маркова, лямбда исчисление Чёрча);
- анализ программ по нисходящей схеме, декомпозиция, основанная на разбивке задач по уровням 0, 1, ..., k. В классической работе Бома и Джакопини [22] показано, что такая структура (иерархическая, разбитая на уровни) может быть реализована в языке, включающем только две управляющие конструкции.

Работа [22] имеет фундаментальное значение. По Бому и Джакопини для реализации программ требуется три основных составляющих блока:

- функциональный блок (или конструкция следования);



- конструкция обобщенного цикла;
- конструкция принятия двоичного или дихотомического решения.

Пример неструктурированной программы.




Характерные черты структурного стиля программирования:

- простота и ясность (программа легко читается и анализируется, достаточное комментирование);
- использование только базовых конструкций;
- отсутствие сетевых структур в программе;
- отсутствие многоцелевых функциональных блоков;
- отсутствие неоправданно сложных арифметических и логических конструкций;
- расположение в строке программы не более одного оператора языка программирования;
- содержательность имен переменных.

При этом процесс нисходящей разработки программы может продолжаться до тех пор, пока не будет достигнут уровень «атомарных» блоков, т. е. базовых конструкций (присвоения, if-then-else, do-while).

Итак, если формулировать суть в сжатом виде, то в структурном программировании уточнен принцип декомпозиции задачи (в основном ее алгоритмического аспекта, т.е. действий, однако уровень интеграции действий и данных «на совести» разработчика) и сделана попытка его строгой формализации.

*Модульное программирование*. Достаточно независимые фрагменты задачи оформляются как модули. Создаются библиотеки модулей, разрабатывается механизм включения модулей в разрабатываемую программу. Модуль должен иметь строго определенный интерфейс и скрытую часть, одну точку входа и одну точку выхода. Из фольклора computer science - «модульность в программировании подобна честности в политике: каждый утверждает, что она - одно из его достоинств, но кажется, никто не знает, что она собой

представляет, как ее привить, обрести или добиться». Очередной этап развития принципа декомпозиции задачи и абстрагирования.

*Объектно-ориентированное программирование* (языки программирования Турбо Паскаль, начиная с версии 5.5, Смоллток, C++). Характеризуется тремя основополагающими идеями: инкапсуляцией, наследованием, полиморфизмом. *Инкапсуляция*. Сочетание данных с допустимыми действиями над этими данными приводит к «рождению» нового элемента в конструировании программы - объекта. «Рожденный ползать - летать не может» - и наш объект действует только так, как это в нем заложено и только над тем, что в нем описано. Обращение к данным объекта не через его действия недопустимо. *Наследование*. Программист для решения определенного класса задач строит иерархию объектов, в которой, и это самое главное, каждый следующий производный объект имеет доступ (наследует) к данным и действиям всех своих предшественников («прародителей»). Характер связей между объектами вертикальный. *Полиморфизм*. Выделение некоторого действия, т.е. действие должно иметь имя и создание средств использования действия объектами иерархии. Причем каждый объект реализует это действие так, как оно для него подходит. Пример: есть множество геометрических фигур, образующих иерархию. Действие - перемещение по экрану. Мы видим «скачок» в технологии программирования, впервые действия и данные образуют нечто единое - новый уровень абстрагирования.

*Визуальная технология конструирования программ* (система программирования Delphi). Во-первых, полностью поддерживается объектно-ориентированная технология, во-вторых, идеи модульного программирования получают логическое завершение, в-третьих, и это принципиально новое в данной технологии (создан соответствующий инструментарий), программирование реакции на события (любая программа в процессе своей работы с чем-то или кем-то взаимодействует) автоматизировано.

*Обзор. О спирали развития*. Первый виток - операциональный, 1954 -1965 годы (Разбивка по годам достаточно приблизительна, отражает лишь точку зрения авторов.) Языки программирования: FORTRAN I и II, ALGOL 58, 60, COBOL, LISP и др. Нарабатываемые идеи: подпрограмма (подпрограммы возникли до 1950 года, но рассматривались не как элемент абстрагирования, а как средство, упрощающее работу); типы данных и их описание, раздельная компиляция, блочная структура, обработка списков, указатели и т.д. Второй виток - структурный (объединяем этим термином и нисходящее проектирование, и модульное). 1966 - 1985 годы. Языки программирования ПЛ/1, ALGOL 68, Pascal, Simula, C, Ada (наследник ALGOL 68, Pascal, Simula), Clos, C++(возникший в результате слияния C и Simula) и т.д. В 70-е годы созданы тысячи языков и диалектов. Нарабатываемые идеи: подпрограммы как элемент абстрагирования (разработаны механизмы: передачи параметров; вложенности подпрограмм; локальных и глобальных переменных; теория типирования; развитие модулей от группы логически связанных подпрограмм до раздельно компилируемых фрагментов со строго определенным интерфейсом). Третий виток - объектно-ориентированный (включая визуальную технологию), с 1986г. по настоящее время. Языки программирования: Smalltalk, Object Pascal, C++. Основным элементом конструирования программы является модуль, составленный из логически связанных объектов.

*Вывод*. Каждый новый виток, «вбирая» все от предыдущего, решает основную проблему технологии (разработка надежного и эффективного программного проекта с минимальными затратами) новым, более совершенным инструментарием (принципы декомпозиции и абстрагирования, приемы анализа и синтеза и т. д.). Мера дезорганизации (энтропия) программных проектов (имеется в виду весь их «жизненный цикл») как сложных систем уменьшается.

Примечание. В данном материале не рассматривалась декларативная ветвь развития программирования(описываются ключевые абстракции задачи, что необходимо сделать, какими свойствами должен обладать результат, но не описывают, каким способом это результат будет получен) . Имеются в виду языки программирования типа Пролога (логическое программирование), Лиспа (функциональное программирование). В 70-е, а особенно, в связи с проектом ЭВМ пятого поколения, в 80-е годы они достаточно интенсивно развивались. Однако полнокровная реализация идей этих систем программирования требует не фон Неймановской архитектуры ЭВМ и на сегодня это не так актуально (не исключено, что очень непродолжительное время), как казалось бы по огромному валу публикаций на эту тематику.

## 1.3. Основные идеи образовательной информатики по Сеймуру Пейперту

Кажется почти чудом, что современные методы обучения еще не совсем удушили святую любознательность.

Эйнштейн А.

У С. Пейперта[14] есть следующее эмоциональное высказывание: «...большая часть того, что теперь делается под именем «технология образования» или «компьютеры в образовании», все еще представляет стадию простого смешения старых методов обучения с новыми технологиями... Консерватизм мира образования превратился в самовоспроизводящий социальный феномен». Под старыми методами обучения понимается традиционное обучение, которое имеет три составляющих метода: показ, объяснение и контроль и является процессом взаимодействия учителя и ученика при достижения определенных целей образования.

Действительно, до появления компьютеров общество не обладало инструментарием, способным придать процессу обучения естественный, неформализованный характер. Поэтому о кардинальном изменении концепции обучения речи быть не могло. Однако и эйфория первоначального этапа компьютеризации образования сменилась некоторой апатией. Время так называемых автоматизированных обучающих курсов, просто обучающих программ, суть которых сводилась к элементарному показу и игре в вопросы и ответы, кануло. Что это было? Видимо, простая попытка распространить традиционные методы обучения на новый инструментальный аппарат. Другого и быть не могло, ибо этот инструмент является принципиально новым. Хочется вслед за Р. Брэдбери сказать: «Сами машины - это пустые перчатки, но их надевает человеческая рука, которая может быть хорошей или плохой». Что же взамен? Идет новый виток не понимания сути, а именно превращения компьютера в цель обучения. Поток учебных материалов в рамках этого направления возрастает с каждым годом. А не «выплескиваем ли мы с водой ребенка»? Тем более что продуктивные разработки другой направленности известны, и известны давно. В отличие от многих специалистов С.Пейперт рассматривает **компьютер** лишь как **инструмент**, с помощью которого обучение (а точнее говоря, учение) может стать более интересным, быстрым, простым, а получаемые знания и навыки - более глубокими и обобщенными. В традиционном использовании компьютера предполагается, что происходит «обучение с помощью компьютера» или, по - другому, - «компьютер обучает ребенка». Закачивая эту логическую цепочку, можно сказать и так: «Компьютер используется, чтобы программировать ребенка». Это не что иное, как перенос традиционного программируемого обучения (из дидактики) на новые средства, и является наглядным примером консервативности системы и одной из причин того, что первый виток внедрения компьютера в образование не дал ожидаемых результатов. Точка зрения С. Пейперта противоположна, **ребенок должен программировать компьютер** и, делая это, «ребенок не только овладевает частичкой самой современной техники, но и приобщается к некоторым из самых глубоких идей естествознания, математики, а также к искусству интеллектуального моделирования». Резюмируя, можно сказать, что компьютер должен быть в первую очередь одним из инструментом развития интеллектуальных способностей ребенка.

Как по С. Пейперту реализуется на практике это положение? Через использование среды ЛОГО, специально разработанной для достижения выше выделенных целей. При разработке среды ЛОГО С. Пейперт опирался на ряд фундаментальных исследований в области психологии.

Во-первых, на работы Ж. Пиаже, следуя которым ребенок является как бы зодчим, возводящим структуры собственного интеллекта. Дети, по-видимому, от рождения одарены способностью к учению и задолго до школы осваивают огромный объем знаний благодаря процессу «научения без обучения». Любой зодчий нуждается в материалах, из которых он будет строить. Окружающая культура является источником этих материалов. В одних случаях культура поставляет их в изобилии, облегчая конструктивное учение по Пиаже, в других - нет. Например, сам факт, что многие из предметов обихода (ножи и вилки, мамы и папы, ботинки и носки) соотносятся друг с другом, становится «материалом» для построения интуитивного понятия число. Но в отличие от Ж. Пиаже, который объясняет более медленное формирование отдельных понятий их большой сложностью или абстрактностью, С. Пейперт считает, что решающим фактором в этом случае является плохая представленность в культуре тех материалов, которые могли бы превратить данные понятия в простые и доступные. ЛОГО как раз тот материал нашей современной культуры, который позволяет облегчить ребенку процесс освоения математических понятий, его работу зодчего в этом процессе.

Во-вторых, на понимание того, что является «синтонным я» по З. Фрейду (из клинической психологии). Это термин использовался З. Фрейдом для описания инстинктов или представлений, приемлемых «я», т.е. совместимых с целостностью «я» и с его требованиями. «Синтонное я» означает созвучность представлениям детей о себе как людях с определенными целями, намерениями, желаниями, симпатиями и антипатиями. Работа с Черепашкой в среде ЛОГО соответствует уровню развития ребенка, нет элементов «тыканья лицом в грязь». Абстрактный совет Д. Пойа[15], «чтобы решить задачу, поищите, на что она похожа, и тогда вы ее поймете», превращается в геометрии Черепашки в конкретное действие: поиграй в Черепашку, чтобы понять, как бы ты это сделал сам. «Геометрия Черепашки пригодна для учения именно потому, что она синтонна. И она является средством изучения других вещей, поскольку поощряет ребенка к их сознательному, продуманному использованию при решении проблем».

Работа в среде ЛОГО разрешает ошибаться. Можно сказать сильнее - любая работа за компьютером разрешает ошибаться. Программа как модель некоторой проблемы в представлении ребенка должна работать так-то и так. Компьютер «говорит», что она работает по-другому, есть ошибка или результат не соответствует предполагаемому - противоречие. Позволяя ошибаться, разрешая ошибаться, создавая возможности ошибаться, мы даем возможность познавать через противоречия, ибо ошибка - это источник противоречия. Система образования, по мнению С. Пейперта, отвергает «ошибочные теории» детей, а значит, отвергает путь, которым дети учатся. Однако Ж. Пиаже показал, что ошибочные теории детей являются существенной частью процесса овладения мышлением. Суть образовательного процесса многими педагогами видится в том, чтобы выявлять и исправлять ошибочные взгляды детей, в усиленном «пичканье» правильными теориями, прежде чем дети оказываются готовыми их воспринять.

Среда ЛОГО создана С.Пейпертом, в первую очередь, для освоения математических понятий. Он безусловно достиг этой цели, сделав этот процесс естественным и удобным для ребенка. Основное отличие нашего подхода в том, что **информатика сама по себе является базовым инструментом развития интеллектуальных способностей ребенка (его ума), без привязки к какому-либо конкретному школьному предмету, ибо она сама тот самый предмет**. Например, освоение структурной парадигмы мышления или эвристических приемов решения проблем по Д. Пойа необходимо как историку, так и математику, как бизнесмену, так и слесарю - инструментальщику.

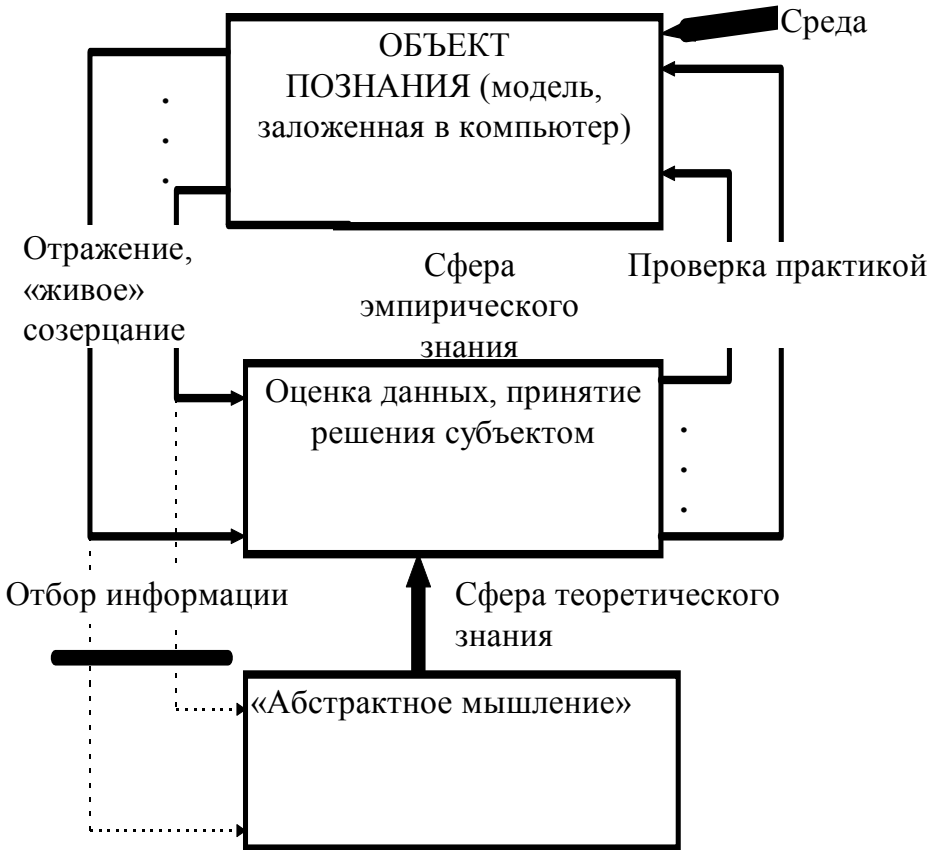
## 1.4. Информатика - ключевой инструмент развития интеллекта школьника

Всякое человеческое познание начинается с созерцания, переходит от них к понятиям и заканчивается идеями
Кант И. Критика чистого разума, Соч., т.3. М.,1964.

Определим понятие: интеллект - ум, рассудок, разум; мыслительная способность человека[17]. Цель наших рассуждений заключается в том, чтобы показать, что информатика, как никакой другой предмет, позволяет развивать ум. Если утверждение будет обосновано, то высказывания о «вспомогательной роли курса» информатики в цикле школьных предметов, об информатике как подспорье математики будут не убедительными.

Понятие ума. Ум - или дар божий, или то, чему можно научить? Если дар божий (или дар природы), «то руки по швам». Есть такая точка зрения. Утверждают, что только 6 процентов людей умны, а остальные принадлежат к категории чистых дураков, или «репродуктивов», то есть способных исключительно на «репродукцию» - работу, заключающуюся в монотонном и однообразном воспроизведении одних и тех же, раз и навсегда заученных операций, правила которых разработаны для них «умными». Не останавливаясь на разборе различных точек зрения, их философском обосновании, обратимся к работе Э.В. Ильенкова[9]. «Ум - умение соотносить некоторые общие, усвоенные в ходе образования, пусть самого элементарного, пусть самого высшего, «истины» с фактически складывающимися в жизни и поэтому каждый раз неповторимыми, каждый раз непредусмотренными, каждый раз неожиданными и индивидуальными стечениями обстоятельств». Философским языком - умение «опосредовать общее с единичным, с индивидуальным, с особенным». **Итак, определим ум как способность выносить суждения о единичном факте с высоты усвоенной человеком культуры.** При этом чем больше запас усвоенных знаний, тем больше простора для обнаружения ума. Но, если нет способности (умения) самостоятельно выносить суждения - ума нет вообще. Есть его отсутствие - глупость, даже при огромном запасе знаний. Недаром говорят, «многознание уму не научает» (Гераклит). Но откуда берется эта способность выносить суждения, способность рассуждать? На основе работ И.А.Соклянского и А.И.Мещярикова со слепоглухими детьми Э.В.Ильенков делает вывод о том, что «ум - это не естественный дар, а результат социально-исторического развития человека...» и «*построение процесса усвоения знаний* должно быть таким, чтобы он одновременно был процессом развития той самой способности, которой эти знания обязаны своим рождением, - способности осмысливать ... объективную реальность...». Итак, вывод - в процессе познания (получения знаний) должна развиваться способность выносить суждения. Чтобы двигаться дальше, нам необходимо понять суть любого познавательного процесса, его основные характеристики и сущность способности рассуждать.

Определим схему процесса познания по Р.Ф.Абдееву [1], она приведена на рисунке.




В основе данного процесса - активность, цикличность и два контура обратной связи. Первый контур обратной связи - область эмпирического знания. Второй контур - отбор и обобщение информации, попытка выявить очередную относительную истину, т.е. область теоретического знания. Другими словами, в первом контуре воспринимается явление, а во втором познается сущность, причем постижение сущности углубляется в ходе осуществления все более целенаправленного воздействия на объект. Схема процесса познания раскрывает суть информационного взаимодействия активного познающего субъекта и исследуемого объекта, отражая механизм движения познания от относительной истины к абсолютной. Каждый текущий результат оценивается на фоне все возрастающего уровня знаний. Это и есть процесс познания, «вечное, бесконечное приближение мышления к объекту», ко все большему соответствию наших представлений объективной природе вещей. Итак, чтобы развился интеллект, т. е. помимо знаний выработалась способность выносить суждения, требуются миллионнократные (миллиарднократные) «прогоны» по этой схеме.

Обратимся к дидактике. Проблемное обучение[12] - одна из схем развивающего обучения. Целью проблемного обучения, в отличие от традиционного, является усвоение не только результатов научного познания и системы знаний, но и поиск самого пути - процесса получения этих результатов, формирование познавательной самостоятельности ученика и развитие его творческих способностей. При традиционном обучении преподаватель излагает «готовые» знания, а учащиеся их *пассивно* усваивают. Затем, чтобы закрепить знания в памяти, они применяют их в процессе решения учебных задач (теоретических, практических). Проблемное обучение реализуется через создание соответствующих ситуаций на занятиях, повышающих активность учащихся и усиливающих степень взаимодействия ученика и учителя и учеников между собой, а также долей решаемых творческих (дидактических, по М.И. Махмутову) задач. Проблемное обучение - это оптимальное сочетание репродуктивной и творческой деятельности по передаче и усвоению системы научных понятий и приемов, способов логического мышления, это дидактический подход, учитывающий психологические закономерности мыслительной деятельности субъектов. Итак, мы видим, что проблемное обучение призвано развивать интеллект учащихся. Но как? Усиливая традиционный контур обратной связи «учитель - ученик» и вводя новый контур - «ученик - ученик», а также путем повышения активности учащихся. Результат почти полностью согласуется со схемой процесса познания. (Выскажем гипотезу: все, что есть в дидактике по развивающему обучению, так или иначе направлено на усиление и развитие традиционного контура обратной связи «учитель - ученик»). Цикличность присутствует, но она «растянута» (периодичностью занятий), нет того, что называют реальным масштабом времени. На уроках информатики есть компьютер. Но это не только компьютер сам по себе, это инструментарий нового контура обратной связи, повышающий активность на порядок, обеспечивающий цикличность в реальном масштабе времени, дающий индивидуализацию и дифференциацию обучения (один из ведущих принципов развивающего обучения). Появляются все условия для полной реализации схемы процесса познания.


***Первый вывод. В школе впервые за годы ее существования появился инструмент для полной реализации схемы процесса познания.***

Продолжим наши рассуждения. Необходимо ответить на второй вопрос - в чем сущность способности к рассуждениям (не в обычном житейском смысле)? Это владение приемами анализа и синтеза; дедукцией и индукцией; использование и соблюдение законов логики и т.д. Ограничимся этими утверждениями и вернемся к информатике. Владение структурной парадигмой мышления (обязательное свойство ума любого профессионала в информатике) требует от учащихся определенных умений и навыков. В частности, необходимо знание технологий нисходящего и восходящего проектирования программ, при этом должно быть полное соответствие по данным между программными компонентами различных уровней - без владения приемами анализа и синтеза это невозможно. В языках программирования третьего поколения любую программу можно логически построить, используя ограниченный набор конструкций и т. д. Главное заключается в том, что овладение структурной парадигмой - это так или иначе развитие способности к суждениям. Причем это овладение происходит только при написании программ, путем миллионнократных (сознательное повторение) действий по схеме процесса познания. И в целом, вспомним спираль развития технологий программирования. Каждый новый виток характеризуется новым уровнем абстрагирования, другими принципами построения иерархических структур. Последовательное «протаскивание» школьника по этой спирали - не есть ли это то самое, что называют развитием мыслительных способностей, ума.

***Второй вывод. Изучение информатики формирует и развивает способность к суждениям.***

***Третий вывод (на основе первых двух). Информатика - инструмент развития мыслительных способностей школьников, т. е. инструмент развития интеллекта.***

Обозначены только контуры обоснования базовой роли информатики в школе, а именно с философской точки зрения - информатика и интеллект. Если структура занятий по информатике более других предметов соответствуют познавательной схеме, разработанной в философии, и развивает интеллект, то курс должен быть основным, ключевым, если хотите. Проблемы методики курса информатики и его места в учебном плане школы - отдельные вопросы. Однако прагматическая ориентация курса, заключающаяся, например, в изучении только прикладных программ, явно не может претендовать на такую роль.

## 1.5. Олимпиадная информатика

> ...Таланты создавать нельзя, но можно создавать культуру, то есть почву, на которой растут и процветают таланты. Чем больше, шире и демократичнее культура, тем чаще появление таланта и гения. Один ученый назвал живопись Ренессанса эпидемией гениальности.
>
> Нейгауз Г.. Об искусстве фортепьянной игры. 1958г.

Олимпиады - это тот «срез» в образовании, который проверяет не только владение предметом, но и формирует тенденции развития этого предмета, определяет требования к школе через этот предмет со стороны общества, то есть то, что называют социальным запросом. Олимпиады по информатике (и школьные, и студенческие), как по содержанию, так и по методике проведения можно считать сформировавшимся явлениям.

*Основное утверждение.* Попытаемся разобраться, ответить на вопрос: кто может победить на олимпиаде по информатике? Напомним, что участник должен для победы за 4 - 5 часов найти алгоритмы решения нескольких, достаточно трудных задач, написать и отладить (проверить с помощью тестов) в общей сложности от 300 до 600 строк программного кода. Мыслимо ли это? Практика показывает, что да. Лет 25-30 тому назад считалось, что 5-10 строк кода в день - это хорошая производительность профессионального программиста. Критерий не однозначный, однако, даже при всем при этом, прогресс впечатляет. Начнем с самого простого условия - скорость набора текста. Она должна быть не менее 90-100 символов в минуту, причем набор должен быть «слепым». Далее - знание компьютера, системы программирования, предпочтительно Турбо Паскаля. На детальном анализе причин выбора именно этой

системы программирования не будем останавливаться. Сошлемся лишь на тот факт, что большинство участников российских и международных олимпиад по информатике работают на Турбо Паскале. Оказывается, что просто знание, даже хорошее знание, даже отличное знание системы программирования, возможностей компьютера не решает проблемы. Школьник ***должен владеть на подсознательном уровне структурным стилем (парадигмой) мышления***, ***плюс - классической алгоритмистикой*** - это залог успеха. Попытаемся раскрыть и обосновать данное утверждение.

> Сложность программного обеспечения - отнюдь не случайное его свойство, скорее необходимое.
>
> Способ управления сложными системами был известен еще в древности: divide et impera (разделяй и властвуй).
>
> Буч Г. Объектно-ориентированное программирование с примерами применения. М., 1992.

*Структурная парадигма и алгоритмистика.* О структурной парадигме мышления речь шла выше. Отметим только, что уровень владения этим стилем должен быть таков, что при возникновении идеи решения задачи, вся остальная работа выполняется практически в автоматическом режиме. Мы не задумываемся над построением предложений, фраз при общении на родном языке. Точно так же, не задумываясь, участник олимпиады выражает свои мысли в системе программирования, причем автоматически создается программный код без ошибок, а тут уже без владения структурной парадигмой мышления никак не обойтись.

Что касается алгоритмистики, то ограничимся перечислением основных разделов: арифметика целых чисел; комбинаторика (подсчет комбинаторных конфигураций, комбинаторика конечных множеств, перечислительные задачи комбинаторного анализа); поиск и сортировка; алгоритмы на графах (связность, кратчайшие пути, циклы, потоки в сетях и т.д.); перебор и методы его сокращения (динамическое программирование, метод ветвей и границ, метод «решета» и т.д.); геометрия (формулы геометрических преобразований на плоскости, скалярное и векторное произведения, уравнение прямой, нормальный вектор, нахождение прямой, проходящей через две точки, уравнения параллельной и перпендикулярной прямых, уравнение окружности по трем точкам, пересечение прямой и окружности, принадлежность точки многоугольнику, выпуклая оболочка и т. д.); элементы теории формальных грамматик и абстрактных автоматов (алгоритмы синтаксического разбора выражения и построения соответствующего дерева, формулы Бэкуса-Наура, понятие лексемы, машины Тьюринга). По каждой теме необходимо решить определенное количество (какое?) задач, довести их до уровня работающих программ. Но этого мало. Мы задумываемся над тем, чему равно 2 умножить на 2? Нет, вероятно, только в первом классе. Без участия нашего сознания правильный ответ откуда-то извлекается. А в нашем случае? Если, например, задача, независимо от ее содержательной «упаковки», сводится после ряда преобразований к алгоритму нахождения кратчайшего пути в графе, то все - она решена. Участник олимпиады (по-другому - профессионал в определенной части информатики), установив этот факт на сознательном уровне, всю остальную работу выполняет почти как автомат, она не должна требовать значительных усилий на сознательном уровне - можно переключаться на следующую задачу. Сколько раз ребенку требуется 2 умножить на 2, чтобы автоматически извлекать ответ? Столько же раз требуется использовать алгоритм Дейкстры для нахождения кратчайшего пути в графе при решении задач, чтобы ответ извлекался откуда-то с таким же количеством усилий, как и при умножении 2 на 2.

> Для того, чтобы изобретать, надо быть в двух лицах. Один образует сочетания, другой выбирает то, что соответствует его желанию и что он считает важным из того, что произвел первый. То, что называют «гением», является не столько заслугой того, кто комбинирует, сколько характеризует способность второго оценивать только что произведенную продукцию и использовать ее.
>
> Поль Валери

> И тут в мой разум грянул блеск с высот,
> Неся свершенье всех его усилий.
> Данте, Рай, Песнь XXXIII. М., 1961.

*О подсознательном.* Механизм формирования высших функций мозга, таких, как сознание, творчество и мышление в целом, представляет собой одну из фундаментальных тайн природы, которая давно привлекает специалистов различных областей знания. Говорить о «подсознательном» - профессиональный удел психологов. Однако попробуем, ибо для этого есть причины. Работая со школьниками, приходится наблюдать у них моменты внезапного озарения, после которых фактически и начинается процесс программной реализации задачи. В зависимости от степени владения системой программирования этот процесс отнимает все силы и время или приводит к успеху (решению задачи или модификации метода решения). Назвать случайными эти «вспышки» озарения нельзя. Им предшествует значительная умственная работа и значительная ее часть выполняется на подсознательном уровне. Как сделать так, чтобы эти моменты открытия, озарения у школьников возникали как можно чаще? Как сделать

так, чтобы успех при реализации задачи был стабильным? Это проблема из проблем для учителя, и без своей точки зрения, без понимания логики взаимодействия сознательных и подсознательных процессов в мозгу человека, без построения модели этих процессов вряд ли можно успешно работать. Итак, мы говорим о бессознательном в двух различных аспектах. Первый: это генерация (назовем это так) идеи (метода) решения задачи. Второй: реализация метода решения задачи на компьютере. Разумеется, говорить о том, что вся эта работа выполняется на подсознательном уровне, нонсенс. Это сложное взаимосплетение, взаимосвязь сознательных и подсознательных процессов. Попытаемся внести некоторую ясность, хотя бы для себя.

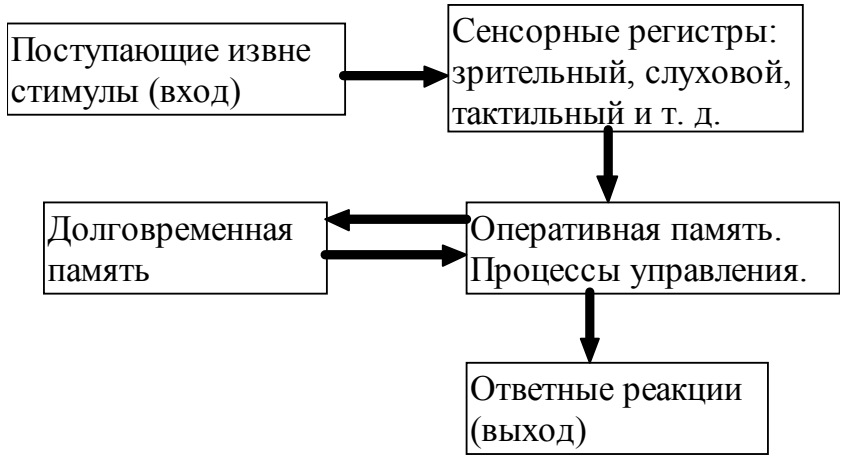
Анри Пуанкаре в докладе «Математическое творчество» (приведен в [2]) отмечает: «Я - подсознательное нисколько не является низшим по отношению к я - сознательному, оно не является чисто автоматическим, оно способно здраво судить, оно имеет чувство меры и чувствительность, оно умеет выбирать и догадываться ... оно преуспевает там, где сознание этого не может». То, что называют внутренним озарением, является результатом длительной неосознанной работы; роль ее несомненна и она плодотворна лишь в том случае, когда ей предшествует и за ней следует сознательная работа. Ж. Адамар [2] выделяет четыре стадии работы над проблемой: подготовка; инкубация; озарение; проверка (завершение). Первая и четвертая относятся к сознательной работе. Во время периода инкубации не заметно никакой сознательной работы ума, но в подсознании, как показал А. Пуанкаре, происходит «прокрутка» огромного количества различных комбинаций, сочетаний, соответствующих проблеме, сравнение их между собой. Творить - это значит не заниматься бесполезными сочетаниями, а исследовать только полезные, которые составляют лишь небольшое меньшинство. Когда и как осуществляется выбор? На стадии озарения и (по А. Пуанкаре) «среди бессознательных идей привилегированными, т. е. способными стать сознательными, становятся те, которые прямо или косвенно наиболее глубоко воздействуют на наши чувства». Действительно, как заметил Р. Фейнман, «истину можно узнать по ее красоте». Кто из специалистов по информатике не испытывал настоящее эстетическое чувство при виде хорошо сделанной программы?

В термине «модель» все же имеется какое-то весьма оригинальное семантическое зерно, заставляющее бесчисленных авторов все же пользоваться этим термином и выставлять на первый план, несмотря на его путаность и противоречивость.

Лосев А.Ф. Введение в общую теорию языковых моделей. М., 1968.

*Модель интеллекта* Предположим, что наше сознание и подсознание - две связанные между собой, но функционально различные системы обработки данных (вычислительные системы). Объем памяти и производительность подсознания как распределенной вычислительной системы с ассоциативными принципами работы на много порядков больше, чем аналогичные характеристики сознания. Основным условием сознательного состояния является способность обозревать свое внутреннее состояние посредством периодической передачи на вход имеющейся в памяти информации [21]. Ее не может быть много, поэтому производительность сознания как системы обработки информации и принятия решений небольшая. Потеря информации происходит по мере того, как она перестает активно использоваться. Повторение является эффективным способом удержания информации в оперативной памяти и важным условием ее перевода в долговременную память. Емкость долговременной памяти обеспечивает хранение всей информации, потенциально полезной для человека. В долговременной памяти подсознания содержатся глубоко усвоенные социальные нормы поведения, *доведенные до автоматизма профессиональные* и бытовые *навыки* и другие знания, выражающие личность и индивидуальность человека. Время хранения информации в долговременной памяти соответствует продолжительности жизни. Известно, что нейроны, в отличие от большинства других клеток организма, никогда не обновляются. Этим обеспечивается сохранение информации и вычислительной структуры мозга на протяжении длительного времени без искажений. Долговременная память имеет иерархическую организацию с различными временами выборки информации. В типичных случаях выборка информации осуществляется вне прямого контроля сознания в соответствии с целью и содержанием конкретной деятельности. В область же сознания пропускается лишь предварительно обработанная информация.

Процесс преобразования информации (по Ричарду Аткинсону - американскому психологу) в системе памяти выглядит следующим образом.




Процессы управления (основные):

- повторение - скрытое (про себя) или открытое (вслух) неоднократное воспроизведение информации;
- кодирование - подлежащая запоминанию информация вводится в виде условной, легко извлекаемой информации (мнемоническая - искусственно запомненная фраза, опорные сигналы В.Ф. Шаталова);
- представление - вербальная (словесная) информация запоминается с помощью зрительного образа;
- принятие решения, организующие схемы, методы решения задач, стратегия извлечения.

Из всей предъявленной (новой) информации человек выбирает «пробную информацию» и помещает ее в оперативную память. Затем из долговременной памяти извлекается и переносится в оперативную «поисковый набор информации, соответствующее подмножество информации, тесно связанное, по мнению человека, с «пробной информацией». Производится сопоставление «пробной информации» с «поисковым набором». Если искомый образ обнаружен, поиск прекращается. Если нет, то человек принимает решение либо о новом цикле поиска, либо о прекращении поиска ввиду бесперспективности.

Рассматриваемая модель интеллекта вполне согласуется с нейробиологическими данными о структуре и клеточных механизмах головного мозга. По данным [21], новая кора переднего мозга содержит 50 млрд. нейронов, организованных в 600 млн. вертикальных миниколонок, функционирующих параллельно. Очевидно, что производительность ассоциативного матричного процессора столь высокой размерности должна быть чудовищной.

*Объяснение некоторых феноменов психической деятельности.* Сознание реализуется в сравнительно небольших областях головного мозга, которые расширяются только при интенсивной рассудочной деятельности. В зонах, которые граничат с областями сознания, имеет место краевое сознание, а в остальных частях мозга идет производительная подсознательная работа.

Область сознания сокращается до минимума во время сна, и мозг освобождается для более производительной работы. Это может быть внутренняя работа, которая необходима в любой системе обработки информации, например упорядочивание массивов информации, доработка и перестройка моделей и др. Во сне могут решаться проблемы, которые были поставлены ранее в процессе взаимодействия сознания и подсознания. По данным нейрофизиологии, интенсивность работы мозга в целом постоянна днем и ночью. Но во время сна снижается до минимума обработка внешней сенсорной информации, управление моторикой и процедура осознания. Следовательно, во время сна интенсивность работы мозга как автономной вычислительной системы возрастает. «Утро вечера мудренее» - богатейший экспериментальный материал, накопленный в процессе разносторонней человеческой деятельности. Примеры появления идей, решения ключевых проблем во время отдыха, прогулок и т. д.

Гипноз. Если предположить существование способов воздействия, расстраивающих специфические процедуры осознания, то создается интеллект - кентавр, состоящий из сознания гипнотизера и подсознания гипнотизируемого...

Интуиция. Видимо, кульминационный момент в интуитивном процессе возникает тогда, когда все элементы ситуации, находившиеся до этого в разрозненном состоянии, замыкаются на неизвестном ранее звене наглядно в единую, целостную структуру, когда эти элементы и их связи становятся обозримыми. Может быть, это появление новой связи (или связей) между элементами во второй вычислительной структуре. Интуиция опирается на множество операций мышления, которые приводят к искомому результату. Однако своеобразие этого творческого процесса состоит в том, что новое знание появляется в голове еще до того, как найдены необходимые способы логического доказательства его истинности. Новое знание выступает при этом в виде некоторого положения (высказывания), которое не только логически не вытекает из существующей системы знаний, но иногда и не вписывается в нее. Знание, полученное посредством интуиции, возникает как «скачок» на основе удивительной «смеси», соединения сознательного и бессознательного.

> Каждая решенная мною задача становилась образом, который служил впоследствии для решения других задач.
> Декарт Р. Избранные произведения, М., 1950

*Структура творческого процесса при решении задачи.* Итак, школьник приступает к решению задачи. Идет ее анализ. Лучше всего анализируются задачи небольшой размерности (вспомните высказывание: «голова пошла кругом» - сознание не справляется со сложностью, размерностью задачи, происходит его отказ как вычислительной системы, не хватает памяти или производительности). Если задача большой размерности, то так или иначе школьник пытается свести ее к более простой (подтверждение низкой производительности сознания) и создать некий «зрительный» образ задачи, другими словами, нарисовать картинку. Затем наступает этап как бы «ручной прокрутки» задачи. Не зная метода решения, делается попытка предсказать, найти ответ для этой простой вариации задачи. Подсознательные процессы запущены - период инкубации. В это время опытный участник олимпиады набрасывает «скелет» программы, делает стандартные процедуры ввода исходных данных из файла, вывода результата, общую схему перебора (при переборной задаче), анализирует следующую задачу и т.д. Он, как профессиональный «рыбак», ждет, ждет настоящую «поклевку», назовем это моментом озарения. У профессионала она обычно наступает как результат того, что заложено в долговременную память, какие структурные связи во второй вычислительной системе установлены на стадии обучения. Менее опытный - обычно сидит и рисует картинки (если собрать черновики после олимпиадного тура, то можно увидеть разнообразие рисунков, никоим образом, казалось бы, не связанных с задачами). Идея найдена, озарение произошло. Разрабатывается программа, точнее, ее решающая часть, все остальное у профессионала уже есть. Участие сознания на этом этапе минимально - структурный стиль, что называется, у него «в крови». На сознательном уровне идет проверка решения. И опять выигрыш по времени, ибо в каждый момент есть работающая версия программы. Вместо несделанных фрагментов стоят «заглушки». Есть возможность контроля за изменением данных, весь процесс их трансформации от исходных до результата на виду, под контролем. Структурный стиль разработки работает в полную силу.

Следует отметить, что описанная структура процесса решения задачи является лишь грубой схемой. В реальной деятельности взаимодействие между сознанием и подсознанием протекает непрерывно. Поэтому выделенные этапы работы мозга весьма условны. Так, например, в информатике известно [5,6], что правильно выбранные структуры данных - половина решения задачи, т. е. выделенные этапы: анализ, инкубация, озарение, проверка - неоднократно «прокручиваются» на первом этапе работы с задачей.

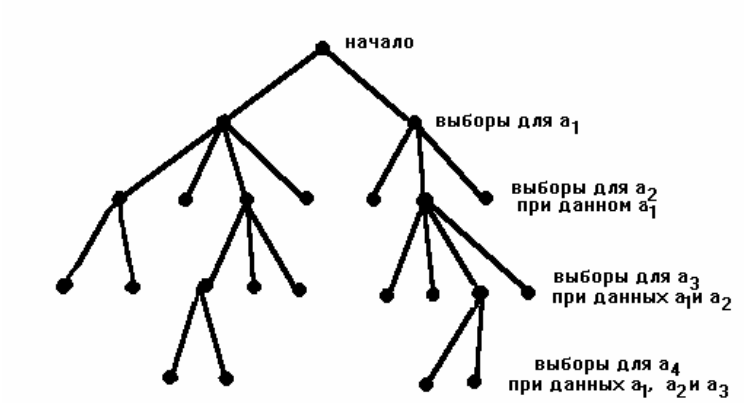
# Глава 2. ПЕРЕБОР И МЕТОДЫ ЕГО СОКРАЩЕНИЯ

## 2.1. Перебор с возвратом

### 2.1.1. Общая схема

Даны N упорядоченных множеств $U_1$, $U_2$, ..., $U_N$ (N - известно), и требуется построить вектор $A=(a_1, a_2, ..., a_N)$, где $a_1 \in U_1$, $a_2 \in U_2$, ..., $a_N \in U_N$, удовлетворяющий заданному множеству условий и ограничений.

В алгоритме перебора вектор A строится покомпонентно слева направо. Предположим, что уже найдены значения первых (k-1) компонент, $A=(a_1, a_2, ..., a_{(k-1)}, ?, ..., ?)$, тогда заданное множество условий ограничивает выбор следующей компоненты $a_k$ некоторым множеством $S_k \subset U_k$. Если $S_k <> [\ ]$ (пустое), мы вправе выбрать в качестве $a_k$ наименьший элемент $S_k$ и перейти к выбору (k+1) компоненты и так далее. Однако если условия таковы, что $S_k$ оказалось пустым, то мы возвращаемся к выбору (k-1) компоненты, отбрасываем $a_{(k-1)}$ и выбираем в качестве нового $a_{(k-1)}$ тот элемент $S_{(k-1)}$, который непосредственно следует за только что отброшенным. Может оказаться, что для нового $a_{(k-1)}$ условия задачи допускают непустое $S_k$, и тогда мы пытаемся снова выбрать элемент $a_k$. Если невозможно выбрать $a_{(k-1)}$, мы возвращаемся еще на шаг назад и выбираем новый элемент $a_{(k-2)}$ и так далее.




Графическое изображение - дерево поиска. Корень дерева (0 уровень) есть пустой вектор. Его сыновья суть множество кандидатов для выбора $a_1$, и, в общем случае, узлы k-го уровня являются кандидатами на выбор $a_k$ при условии, что $a_1$, $a_2$, ..., $a_{(k-1)}$ выбраны так, как указывают предки этих узлов. Вопрос о том, имеет ли задача решение, равносилен вопросу, являются ли какие-нибудь узлы дерева решениями. Разыскивая все решения, мы хотим получить все такие узлы.

Рекурсивная схема реализации алгоритма.

```
procedure Backtrack(вектор,i);
 begin
        if <вектор является решением> then <записать его>
         else begin <вычислить Si>;
                        for a∈Si do Backtrack(вектор|| a,i+1);
                        {|| - добавление к вектору компоненты}
         end;
 end;
```

*Оценка временной сложности алгоритма.* Данная схема реализации перебора приводит к экспоненциальным алгоритмам. Действительно, пусть все решения имеют длину N, тогда исследовать требуется порядка $|S_1| * |S_2| * ... * |S_N|$ узлов дерева. Если значение $S_i$ ограничено некоторой константой C, то получаем порядка $C^N$ узлов.

### 2.1.2. Задача о расстановке ферзей

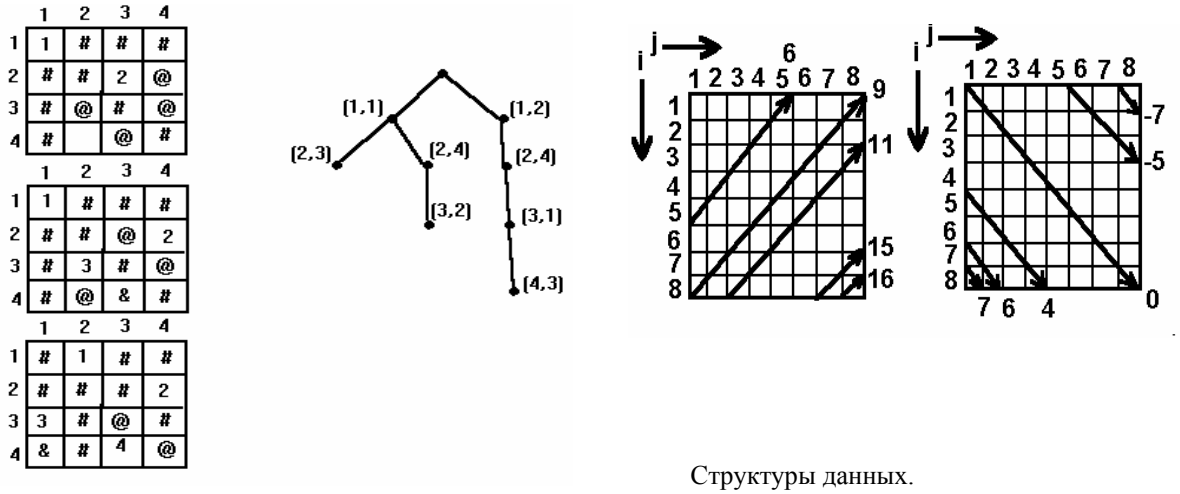
На шахматной доске N*N требуется расставить N ферзей таким образом, чтобы ни один ферзь не атаковал другого.

Примечание. Трудно найти книгу по информатике, в которой не рассматривалась бы эта задача. Нам трудно избежать этой участи, ибо нет лучшей задачи для первоначального ознакомления со схемой решения переборных задач. Приведем краткое описание материала в том виде, в котором он излагается на занятии.

*Отметим следующее.* Все возможные способы расстановки ферзей - $C^N_{N^2}$ (около $4,4*10^9$ для N=8). Каждый столбец содержит самое большее одного ферзя, что дает только $N^N$ расстановок ($1,7*10^7$ для N=8). Никакие два ферзя нельзя поставить в одну строку, а поэтому, для того чтобы вектор $(a_1, a_2, ..., a_N)$ был решением, он должен быть перестановкой элементов (1, 2, ..., N), что дает только N! ($4,0*10^4$ для N=8) возможностей. Никакие два ферзя не могут находиться на одной диагонали, это сокращает число

возможностей еще больше (для N=8 в дереве остается 2056 узлов). Итак, с помощью ряда наблюдений мы исключили из рассмотрения большое число возможных расстановок N ферзей на доске размером N*N. Использование подобного анализа для сокращения процесса перебора называется поиском с ограничениями или отсечением ветвей в связи с тем, что при этом удаляются поддеревья из дерева. Второе. Другим усовершенствованием является слияние, или склеивание, ветвей. Идея состоит в том, чтобы избежать выполнения дважды одной и той же работы: если два или больше поддеревьев данного дерева изоморфны, мы хотим исследовать только одно из них. В задаче о ферзях мы можем использовать склеивание, заметив, что если $a_1>\lceil N/2\rceil$, то найденное решение можно отразить и получить решение, для которого $a_1\le\lceil N/2\rceil$. Следовательно, деревья, соответствующие, например, случаям $a_1=2$ и $a_1=N-1$, изоморфны.

Следующие рисунки иллюстрируют сказанное и поясняют ввод используемых структур данных.




Структуры данных.

```
Up:array[2..16] of boolean;{признак занятости  диагоналей первого типа}
Down:array[-7..7] of boolean;{признак занятости диагоналей второго типа}
Vr:array[1..8] of boolean;{признак занятости вертикали}
X:array[1..8] of integer;{номер вертикали, на которой стоит ферзь на каждой горизонтали}
```

Далее идет объяснение “кирпичиков”, из которых “складывается” решение (технология “снизу вверх”).

```
procedure Hod(i,j:integer); {сделать ход}
 begin
  X[i]:=j;Vr[j]:=false;Up[i+j]:=false;Down[i-j]:=false;
 end;
procedure O_hod(i,j:integer);{отменить ход}
 begin
   Vr[j]:=true;Up[i+j]:=true;Down[i-j]:=true;
  end;
function D_hod(i,j:integer);
{проверка допустимости хода в позицию (i,j)}
 begin
   D_hod:=Vr[j]andUp[i+j]andDown[i-j];
 end;
```

Основная процедура поиска одного варианта расстановки ферзей имеет вид:

```
procedure Solve(i:integer;var q:boolean);
 var j:integer;
 begin
          j:=0;
          repeat
           inc(j);q:=false;{цикл по вертикали}
           if D_hod(i,j) then begin     Hod(i,j);
                                        if i<8 then begin      Solve(i+1,q);
                                                               if not q then O_hod(i,j);
                                                    end else q:=true;{решение найдено}
                                        end;
          until q or (j=8);
 end;
```

Возможные модификации.

Поиск всех решений. Для доски 8*8 ответ 92.

```
Procedure Solve(i:integer);
 var j:integer;
 begin
        if i<=N then begin
         for j:=1 to N do if D_hod(i,j) then begin
                                                Hod(i,j);
                                                Solve(i+1);
                                                O_hod(i,j);
                                                end;
                        end
         else begin
                Inc(S);{счетчик числа решений, глобальная переменная}
                Print;{вывод решения}
                end;
 end;
```

Поиск только не симметричных решений. Для доски 8*8 ответ 12.

Эта модификация требует предварительных разъяснений. Из каждого решения задачи о ферзях можно получить ряд других при помощи вращений доски на $90^o$, $180^o$ и $270^o$ и зеркальных отражений относительно линий , разделяющих доску пополам (система координат фиксирована). Доказано, что в общем случае для доски N*N (N>1) для любой допустимой расстановки N ферзей возможны три ситуации:

- при одном отражении доски возникает новая расстановка ферзей, а при поворотах и других отражениях новых решений не получается;
- новое решение при повороте на $90^o$ и ее отражения дают еще две расстановки;
- три поворота и четыре отражения дают новые расстановки.

Для отсечения симметричных решений на всем множестве решений требуется определить некоторое отношение порядка. Представим решение в виде вектора длиной N, координатами которого являются числа от 1 до N. Для ферзя, стоящего в i-й строке, координатой его столбца является i-я координата вектора. Для того, чтобы не учитывать симметричные решения, будем определять минимальный вектор среди всех векторов, получаемых в результате симметрий. Процедуры Sim1, Sim2, Sim3 выполняют зеркальные отображения вектора решения относительно горизонтальной, вертикальной и одной из диагональных осей. Известно (из геометрии), что композиция этих симметрий дает все определенные выше симметрии шахматной доски, причем не принципиально, в какой последовательности выполняются эти процедуры для каждой конкретной композиции. Проверка «на минимальность» решения производится функцией Cmp, которая возвращает значение true в том случае, когда одно из симметричных решений строго меньше текущего.

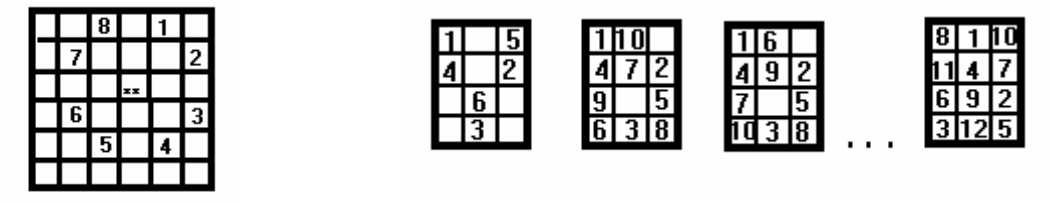
```
{не лучший вариант реализации - отсечение на выводе решений}
type Tarray=array[1..N] of integer;
....
procedure Sim1(var X:Tarray);
 var i:integer;
 begin
        for i:=1 to N do X[i]:=N-X[i]+1;
 end;
procedure Sim2(var X:Tarray);
 var i,r:integer;
 begin
        for i:=1 to N do begin
         r:=X[i]; X[i]:=X[N-i+1];X[N-i+1]:=r;
        end;
 end;
procedure Sim3(var X:Tarray);
 var Y:Tarray;
        i:integer;
 begin
        for i:=1 to N do Y[X[i]]:=i;
        X:=Y;
 end;
function Cmp(X,Y:Tarray):boolean;
 var i:integer;
 begin
```

```
        i:=1;
        while (i<=N) and (Y[i]=X[i]) do Inc(i);
        if i>N then Cmp:=false
         else if Y[i]<X[i] then Cmp:=true else Cmp:=false;
end;
Procedure Solve(i:integer);
 var j:integer;f:boolean;
        Y:Tarray;
begin
        if i<=N then begin
         for j:=1 to N do if D_hod(i,j) then begin
                                                Hod(i,j);
                                                Solve(i+1);
                                                O_hod(i,j);
                                                end;
                        end
         else begin
                f:=true;
                for j:=0 to 7 do begin
                        Y:=X;
                        if j and 1 =0 then Sim1(Y);
                        if j and 2 =0 then Sim2(Y);
                        if j and 4 =0 then Sim3(Y);
                        if Cmp(Y,X) then f:=false;
                end;
                if f then begin
                Inc(S);{счетчик числа решений, глобальная переменная}
                Print;{вывод решения}
                end;
         end;
end;
```

### 2.1.3. Задача о шахматном коне

Существуют способы обойти шахматным конем доску, побывав на каждом поле по одному разу. Составить программу подсчета числа способов обхода.

Разбор задачи начинается с оценки числа 64! - таково общее число способов разметки доски 8*8. Каждую разметку следует оценивать на предмет того, является ли она способом обхода конем доски(решение в “лоб”). Затем оцениваем порядок сокращения перебора исходя из условия - логика выбора очередного хода коня. Будем считать, что поля для хода выбираются по часовой стрелке. Объяснение иллюстрируется следующими рисунками.



Структуры данных.

```
        Const
N= ; M= ;

        Dx:array
[1..8]          of
                integer=(-2,-1,1,2,2,1,-1-2);
        Dy:array[1..8] of integer=(1,2,2,1,-1,-2,-2,-1);
        Var A:array[-1..N+2,-1..M+2] of integer;
```

Основной фрагмент реализации - процедура Solve.

```
procedure Solve(x,y,l:integer);
        var z,i,j:integer;
        begin
                A[x,y]:=l;
                if l=N*M then Inc(t)
                 else for z:=1 to 8 do begin i:=x+Dx[z];j:=y+Dy[z];
                        if A[i,j]=0 then Rec(i,j,l+1)
                        end;
```

```
                A[x,y]:=0;
        end;
...
for i:=-1 to N+2 do for j:=-1 to M do A[i,j]:=-1;
for i:=1 to N do for j:=1 to M do A[i,j]:=0;
t:=0;
for i:=1 to N do for j:=1 to M do Solve(i,j,1);
writeln('число способов обхода конем доски',N,'*',M,'--',t);
....
```

Изменим логику так, чтобы находился только один вариант обхода конем доски. При этом маршрут коня находится с использованием правила Варнсдорфа выбора очередного хода (предложено более 150 лет тому назад). Его суть - при обходе шахматной доски коня следует ставить на поле, из которого он может сделать минимальное количество перемещений на еще не занятые поля, если таких полей несколько, можно выбирать любое из них. В этом случае в первую очередь занимаются угловые поля и количество "возвратов" значительно уменьшается.

Вариант процедуры Solve для этого случая.

```
procedure Solve(x,y,l:integer);
 var    W:array[1..8] of integer;
        xn,yn,i,j,m:integer;
 begin
  A[x,y]:=l;
  if (l<N*N) then begin
                  for i:=1 to 8 do begin{формирование массива W}
                   W[i]:=0;xn:=x+dx[i];yn:=y+dy[i];
                  if (A[xn,yn]=0) the begin
                    for j:=1 to 8 do
                     if (A[xn+dx[j],yn+dy[j]]=0) the Inc(W[i]);
                    end else W[i]:=-1;
                  end;
                  i:=1;
                  while (i<=8) do begin
                   m:=1;{ищем клетку, из которой можно сделать наименьшее число перемещений}
                   for j:=2 to 8 do if W[j]<W[m] then m:=j;
                   if (W[m]>=0) and (W[m]<maxint)
                     then Solve(x+dx[m],y+dy[m],l+1);
                   W[m]:=maxint;{отмечаем использованную в переборе клетку}
                   Inc(i);
                  end;
   end
   else begin <вывод решения>;
                  halt;
        end;
  A[x,y]:=0;
 end;
```

### 2.1.4. Задача о лабиринте

Классическая задача для изучения темы. Как и предыдущие, не обходится без внимания в любой книге по информатике. Формулировка проста. Дано клеточное поле, часть клеток занята препятствиями. Необходимо попасть из некоторой заданной клетки в другую заданную клетку путем последовательного перемещения по клеткам. Изложение задачи опирается на рисунок произвольного лабиринта и две «прорисовки» с использованием простого перебора и метода «волны». Классический перебор выполняется по правилам, предложенным в 1891 г. Э.Люка в "Математических досугах":

- в каждой клетке выбирается еще не исследованный путь;
- если из исследуемой в данный момент клетки нет путей, то возвращаемся на один шаг назад (в предыдущую клетку) и пытаемся выбрать другой путь.

Естественными модификациями задачи поиска всех путей выхода из лабиринта являются:

- поиск одного пути;
- поиск одного пути кратчайшей длины методом «волны».

Решение первой задачи.

```
program Labirint;
        const Nmax=...;
```

```
        dx:array[1..4] of integer=(1,0,-1,0);
        dy:array[1..4] of integer=(0,1,0,-1);
    type MyArray=array[0..Nmax+1,0..Nmax+1] of integer;
    var  A:MyArray;
        xn,yn,xk,yk,N:integer;
procedure Init;
...
 begin
<Ввод лабиринта, координат начальной и конечной клеток. Границы поля отмечаются как препятствия>;
 end;
procedure Print;
....
 begin
 <вывод матрицы A - метками выделен путь выхода из лабиринта>;
 end;
procedure Solve(x,y,k:integer);{k - номер шага, x,y - координаты клетки}
 var i:integer;
 begin
  A[x,y]:=k;
  if (x=xk) and (y=yk) then Print
   else for i:=1 to 4 do
        if A[x+dx[i],y+dy[i]]=0 then Solve(x+dx[i],y+dy[i],k+1);
  A[x,y]:=0;
 end;
begin
    Init;
    Solve(xn,yn,1);
 end.
```

### 2.1.5. Задача о парламенте

На острове Новой Демократии каждый из жителей организовал партию, которую сам и возглавил. Ко всеобщему удивлению, даже в самой малочисленной партии оказалось не менее двух человек. К сожалению, финансовые трудности не позволили создать парламент, куда вошли бы, как предполагалось по Конституции острова, президенты всех партий.

Посовещавшись, островитяне решили, что будет достаточно, если в парламенте будет хотя бы один член каждой партии.

Помогите островитянам организовать такой, как можно более малочисленный парламент, в котором будут представлены члены всех партий.

Исходные данные: каждая партия и ее президент имеют один и тот же порядковый номер от 1 до N ($4 \le N \le 150$). Вам даны списки всех N партий острова Новой Демократии. Выведите предлагаемый Вами парламент в виде списка номеров ее членов. Например, для четырех партий:

| Президенты | Члены партий |
|---|---|
| 1 | 2,3,4 |
| 2 | 3 |
| 3 | 1,4,2 |
| 4 | 2 |

Список членов парламента 2 (состоит из одного члена).

Задача относится к классу NP-полных задач. Ее решение - полный перебор всех вариантов. Покажем, что ряд эвристик позволяет сократить перебор для некоторых наборов исходных данных.

Представим информацию о партиях и их членах с помощью следующего зрительного образа - таблицы. Для примера из формулировки задачи она имеет вид:


Тогда задачу можно переформулировать следующим образом. Найти минимальное число столбцов, таких, что множество единиц из них “покрывают” множество всех строк. Очевидно, что для примера это один столбец - второй. Поговорим о структурах данных.

```
Const Nmax=150;
Type    Nint=0..Nmax+1;
        Sset=Set of 0..Nmax;
        Person=record
                man:Nint;{номер жителя}
                number:Nint;{число партий, которые он представляет}
        part:Sset;{партии}
    end;
```

```
Var A:array[Nint] of Person;
```
Логика формирования исходных данных (фрагмент реализации).
```
function Num(S:Sset):Nint;{подсчитывает количество элементов в множестве}
 var i,ch:Nint;
 begin ch:=0;
        for i:=1 to N do if i in S then Inc(ch);
        Num:=ch;
 end;
procedure Init;{предполагается, что данные корректны и во входном файле они организованы так, как представлены в примере из формулировки задачи}
 ....
 begin
        .....
        readln(N);{число жителей}
        for i:=1 to N do with A[i] do begin man:=i; part:=[i]; end;{каждый житель представляет свою партию}
        for i:=1 to N do begin
         while Not eoln do begin read(t);A[t].part:=A[t].part+[i];{житель t представляет партию с номером i,
или партия с номером i представлена жителем t}
          end;
         readln;
        end;
        for i:=1 to N do A[i].number:=Num(A[i].part);
        ....
 end;
```
Следующим очевидным шагом является сортировка массива A по значению поля number. Становится понятным и необходимость ввода поля man в структуру записи - после сортировки номер элемента массива A не соответствует номеру жителя острова.

Пока не будем задумываться о том, как сократить перебор. Попытаемся набросать общую схему. Используемые переменные достаточно очевидны, они описываются в процессе их ввода, присвоение начальных значений глобальным переменным осуществляется в процедуре Init .
```
procedure Include(k:Nint);{включение в решение}
        begin
         Rwork:=Rwork+[A[k].man];
         Inc(mn);
        end;
 procedure Exclude(k:Nint);{исключить из решения}
        begin
         Rwork:=Rwork-[A[k].man];
         Dec(mn);
        end;

procedure Solve(k:Nint;Res,Rt:Sset);
{k- номер элемента массива A; Res - множество партий, которые представлены в текущем решении; Rt -
множество партий, которые следует “покрыть” решением; min - минимальное количество членов в
парламенте; mn - число членов парламента в текущем решении; Rbest - минимальный парламент; Rwork -
текущий парламент; первый вызов - Solve(1,[],[1..N])}
 var i:Nint;
begin
```
| блок общих отсечений |
|---|

```
                                                    if Rt=[] then begin if nm<min then
                                        begin min:=mn;Rbest:=Rwork end;
                        end
        else begin
                i:=k;
                while i<=N do begin
```
| блок отсечений по i |
|---|

```
                                                                Include(i);
                                        Solve(i+1,Res+A[i].part,Rt-A[i].part);
                                        Exclude(i);
                                        Inc(i);
```

```
                    end;
                end;
end;
```

Итак, а что же дальше? Очевидно, что приведенная схема решения работает только для небольших значений N, особенно если есть ограничения (а они всегда есть) на время тестирования задачи. Предварительная обработка (до первого вызова процедуры Solve), заключающаяся в проверке, есть ли жители, которые представляют все партии (это первый шаг).

```
procedure One(t:Nint;Q:Sset);{проверяет - есть ли среди первых t элементов массива A такой, что A[i].part совпадает с Q}
 var i:Nint;
 begin
        i:=1;
        while (i<=t) and (A[i].part<>Q) do Inc(i);
        if i<=t then begin Rbest:=Rbest+[i]; Rt:=[] end;
end;
```

Первый вызов этой процедуры - One(N,[1..N]), осуществляется в блоке предварительной обработки.

Рассмотрим пример.

| Президенты | Члены партии |
|---|---|
| 1 | 2,3 |
| 2 | 4 |
| 3 | 2 |
| 4 | 1 |


*Идея*. Третий житель состоит в партиях 1 и 3, второй - в 1, 2 и 3. Есть “вхождение”, третий житель менее активный, исключим его. Однако из примера проглядывает и другая идея - появилась строка, в которой только одна единица. Мы получили “карликовую” партию, ее представляет один житель, заметим, что первоначально, по условию задачи, таких партий нет. Житель, представляющий “карликовую” партию должен быть включен в решение, но он же активный и представляет еще другие партии. Значит, эти партии представлять в парламенте нет необходимости - они представлены. Исключаем эти партии (строки таблицы), но после этого возможно появление других “карликовых” партий. Рассмотрим этот процесс на следующем примере: 1 - 8,9; 2 - 10,11; 3 - 12, 13; 4 - 8,9,10; 5 - 11,12,13; 6 - 8,9,10,11; 7 - 9,10,11,12,13;8 - 1,4,6; 9 - 1,4,6,7; 10 - 2,4,6,7; 11 - 2,5,6,7; 12 - 3,5,7; 13 - 3,5,7; 14 - 8; 15 - 9; 16 - 10; 17 - 11; 18 - 12; 19 - 13.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 12 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 13 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Выполняя операцию исключения жителей, «представительство» которых скромнее , чем у оставшихся, получаем.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 12 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 13 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Итак, партии с 14-й по 19-ю «карликовые», их представляют жители с 8-го по 13-й. Мы обязаны включить этих жителей в парламент. Включаем. Формируем множество партий, которые они представляют. Оказывается, что все. Решение найдено без всякого перебора.

Вывод - перебор следует выполнять не по всем жителям и не для всех партий! Если бы это выручало всегда. Сверхактивность жителей сводит на нет этот путь сокращения перебора. Остается надеяться, что кто-то должен и выращивать хлеб, а не только митинговать. Итак, “набросок” общей логики предварительной обработки.

```
...
while <есть вхождения> do begin
        <исключить менее активных жителей>;
        <сжать A>;
        <для “карликовых” партий включить жителей, представляющих их, в состав парламента>;
        <изменить значения величин, описывающих процесс формирования парламента (Res, Rt, mn, Rwork)>;
        <откорректировать A>;
 end;
...
```

Заметим, что необходимо исключить партии, “покрытые” жителями, представляющими карликовые партии из A[i].part оставшихся жителей. Это может привести к тому, что возможно появление жителей, представляющих все оставшиеся партии. Совместим проверку наличия вхождений, исключение части жителей и сжатие массива A в одной функции. Ее вид.

```
function Come(var t:Nint):boolean; {Проверяем - есть ли вхождения? Если есть, то исключаем соответствующих жителей и сжимаем массив A}
 var i,j,l:Nint;
 begin
        for i:=1 to t-1 do
         for j:=i+1 to t do if A[j].part<=A[i].part then begin
                                                          A[j].part:=[];A[j].number:=0;
                                                          end;
        l:=t;
        for i:=1 to t do begin
         if (A[i].part=[]) and (i<=l) then begin for j:=i to l-1 do A[j]:=A[j+1];
                                                 A[l].number:=0;A[l].part:=[];
                                                 Dec(l);
                                                 end;

        end;
        Come:=Not(t=l);
        t:=l;
 end;
```

Вариант построения процедуры исключения «карликовых» партий может быть и таким.

```
procedure Pygmy(t:Nint;var r,p:Sset);{t - количество обрабатываемых элементов массива A; r - множество номеров жителей, включаемых в парламент;  p - множество номеров партий, представляемых жителями, включенных в парламент}
```

```
var i,j:Nint;v:Sset;
begin
        r:=[];p:=[];
        for i:=1 to t do begin
        {определяем множество партий представляемых всеми жителями кроме A[i].man}
         v:=[];
         for j:=1 to t do if i<>j then v:=v+A[j].part;
        {если есть хотя бы одна партия, которую представляет только житель с номером A[i].man, то этого
жителя мы обязаны включить в парламент}
         if A[i].part*v<>A[i].Part then r:=r+[A[i].man];
        end;
        {формируем множество партий, имеющих представительство в данном составе парламента}
        for i:=1 to t do if A[i].man in r then p:=p+A[i].part;
end;
```

Итак, фрагмент предварительной обработки (до перебора).

```
....
t:=N;Rt:=[1..N];Rwork:=[];
One(t,Rt);
while Come(t) and (Rt<>[]) do begin Rg:=[];Rp:=[];
                                    Pygmy(t,Rg,Rp);
                                    Rt:=Rt-Rp;Rwork:=Rwork+Rg;
                                    if Rp<>[] then begin
                                     for i:=1 to t do begin{исключение}
                                      for j:=1 to N do
                                       if (j in Rp) and (j in A[i].part) then
                                A[i]part:=A[i].part-[j];
                                      A[i].number:=Num(A[i].part);
                                     end;
                                    <сортировка A>;
                                    end;
end;
if (Rt<>[]) then One(t,Rt);
```

Блок общих отсечений. Подсчитаем для каждого значения i ($1 \le i \le t$) множество партий, представляемых жителями, номера которых записаны в элементах массива c i по t (массив C:array[1..N] of Sset). Тогда, если Res - текущее решение, а Rt - множество партий, требующих представления, то при $Res+C[i] \le Rt$ решение не может быть получено и эту ветку перебора следует “отсечь”.

Формирование массива C.

```
C[t]:=A[t].part; for i:=t-1 downto 1 do begin
                                        C[i]:=[];C[i]:=A[i].part+C[i+1];
                                        end;
```

Блок отсечений по i. Если при включении элемента с номером i в решение, значение величины Rt не изменяется, то это включение бессмысленно (A[i].part*Rt=[]).

На этом мы закончим обсуждение этой, очень интересной с методической точки зрения, задачи. Заметим, что для любой вновь возникающей идеи по сокращению перебора место для ее реализации в логике определено. А именно, предобработка, общие отсечения, покомпонентные отсечения - другого не дано.

Примечание. Графовая модель задачи (двудольный граф). Каждому жителю соответствует вершина в множестве X, каждой партии - вершина в множестве Y. Ребро (i,j) существует, если житель с номером i представляет партию с номером j. Требуется найти минимальное по мощности множество вершин S, такое, что $S \subseteq X$ и для любой вершины $j \in Y$ существует вершина $i \in S$, из которой выходит ребро в вершину j. Модификация задачи о нахождении минимального доминирующего множества.

## 2.1.6. Задача о рюкзаке (перебор вариантов)

*Постановка задачи*. В рюкзак загружаются предметы n различных типов (количество предметов каждого типа не ограничено). Максимальный вес рюкзака W. Каждый предмет типа i имеет вес $w_i$ и стоимость $v_i$ (i=1,2, ..., n). Требуется определить максимальную стоимость груза, вес которого не превышает W. Обозначим количество предметов типа i через $k_i$, тогда требуется максимизировать $v_1*k_1+v_2*k_2+...+v_n*k_n$ при ограничениях $w_1*k_1+w_2*k_2+...+w_n*k_n \le W$, где $k_i$ - целые ($0 \le k_i \le [W/w_i]$), квадратные скобки означают целую часть числа.

Рассмотрим простой переборный вариант решения задачи, работоспособный только для небольших значений n (определить, для каких?). Итак, данные:

```
Const MaxN=????;
Var     n,w:integer;{количество предметов, максимальный вес}
        Weight,Price:array[1..MaxN] of integer;{вес, стоимость предметов}
        Best,Now:array[1..MaxN] of integer;{наилучшее, текущее решения}
        MaxPrice:LongInt;{наибольшая стоимость}
```

Решение, его основная часть - процедура перебора:

```
Procedure Rec(k,w:integer;st:LongInt);
{k - порядковый номер группы предметов, w - вес, который следует набрать из оставшихся предметов, st - стоимость текущего решения}
 var i:integer;
 begin
        if (k>n) and (st>MaxPrice) then begin {найдено решение}
                                                Best:=Now;MaxPrice:=st; end
         else if k<=n then
                for i:=0 to w div Weight[k] do begin
                        Now[k]:=i;
                        Rec(k+1,W-i*Weight[k],st+i*Price[k]);
                end;
 end;
```

Инициализация переменных, вывод решения и вызывающая часть (Rec(1,w,0)) очевидны. В данной логике отсутствуют блоки предварительной обработки, общих отсечений и отсечений по номеру предмета (смотрите задачу о парламенте). В блоке предварительной обработки целесообразно найти какое-то решение, лучше, если оно будет как можно ближе к оптимальному (наилучший вариант загрузки рюкзака). «Жадная» логика дает первое приближение. Кроме того, разумно выполнить сортировку, например, по значению стоимости предметов или отношению веса предмета к его стоимости. Построение блока общих отсечений аналогично тому, как это сделано в задаче о парламенте, а ответ на вопрос, почему предметы данного типа не стоит складывать в рюкзак, остается открытым.

### 2.1.7. Задача о коммивояжере (перебор вариантов)

*Постановка задачи*. Классическая формулировка задачи известна уже более 200 лет : имеются n городов, расстояния между которыми заданы ; коммивояжеру необходимо выйти из какого-то города, посетить остальные n-1 городов точно по одному разу и вернуться в исходный город. При этом маршрут коммивояжера должен быть минимальной длины (стоимости).

Задача коммивояжера принадлежит классу NP-полных, то есть неизвестны полиномиальные алгоритмы ее решения. В задаче с n городами необходимо рассмотреть (n-1)! маршрутов, чтобы выбрать маршрут минимальной длины. Итак, при больших значениях n невозможно за разумное время получить результат.

*Перебор вариантов*. Оказывается, что и при простом переборе не обязательно просматривать все варианты. Это реализуется в данном классическом методе.

*Структуры данных.*

```
Const Max=100;
Var     A:array[1..Max,1..Max] of integer;{Матрица расстояний между городами}
```

B:array[1..Max,1..Max] of byte;{Вспомогательный массив, элементы каждой строки матрицы сортируются в порядке возрастания, но сами элементы не переставляются, а изменяются в матрице B номера столбцов матрицы A }

```
Way, BestWay:array[1..Max] of byte;{Хранится текущее решение и лучшее решение}
```

Nnew:array[1..Max] of boolean;{Значение элемента массива false говорит о том, что в соответствующем городе коммивояжер уже побывал}

```
BestCost:integer;{Стоимость лучшего решения}
```

*Идея решения*. Пусть мы находимся в городе с номером v. Наши действия.

Шаг 1. Если расстояние (стоимость), пройденное коммивояжером до города с номером v, не меньше стоимости найденного ранее наилучшего решения (BestCost), то следует выйти из данной ветви дерева перебора.

Шаг 2. Если рассматривается последний город маршрута (осталось только вернуться в первый город), то следует сравнить стоимость найденного нового решения со стоимостью лучшего из ранее найденных. Если результат сравнения положительный, то значения BestCost и BestWay следует изменить и выйти из этой ветви дерева перебора .

Шаг 3. Пометить город с номером v как рассмотренный, записать этот номер по значению Count в массив Way .

Шаг 4. Рассмотреть пути коммивояжера из города v в ранее не рассмотренные города. Если такие города есть, то перейти на эту же логику с измененными значениями v, Count, Cost, иначе на следующий шаг.

Шаг 5. Пометить город v как нерассмотренный и выйти из данной ветви дерева перебора.

*Пример.* Ниже приведены матрицы A и B (после сортировки элементов каждой строки матрицы A).

Примечание. Можно использовать любой из методов сортировки, но авторы предпочитают использовать метод Хоара[1] из-за определенного изящества в его реализации. Рекурсивная логика плюс смена направления изменения индекса в одной циклической конструкции.

A

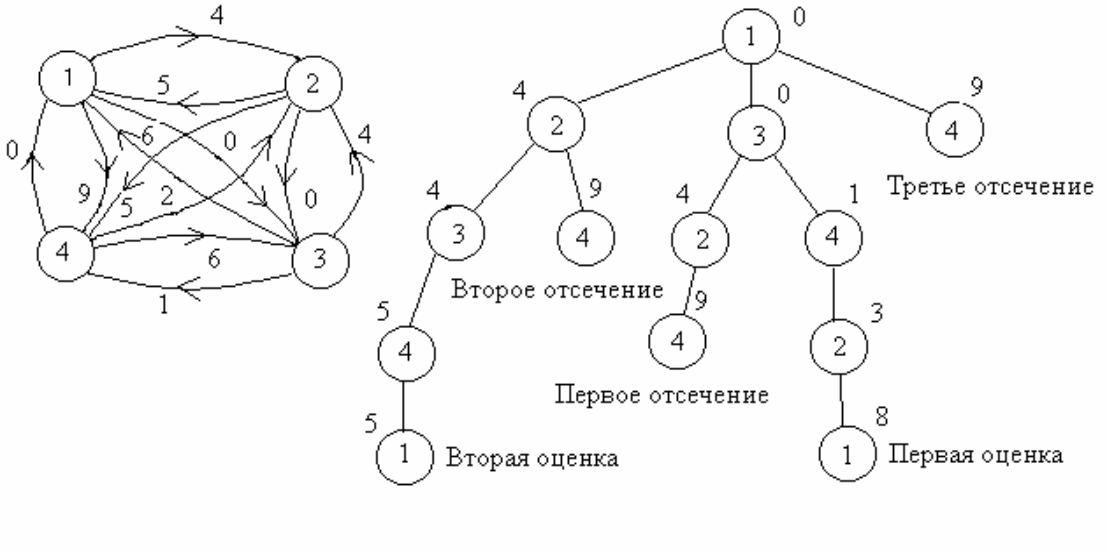
| | | | | | |
|---|---|---|---|---|---|
| @ | 27 | 43 | 16 | 30 | 26 |
| 7 | @ | 16 | 1 | 30 | 25 |
| 20 | 13 | @ | 35 | 5 | 0 |
| 21 | 16 | 25 | @ | 18 | 18 |
| 12 | 46 | 27 | 48 | @ | 5 |
| 23 | 5 | 5 | 9 | 5 | @ |

B

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 2 | 5 | 3 | 1 |
| 4 | 1 | 3 | 6 | 5 | 2 |
| 6 | 5 | 2 | 1 | 4 | 3 |
| 2 | 5 | 6 | 1 | 3 | 4 |
| 6 | 1 | 3 | 2 | 4 | 5 |
| 2 | 3 | 5 | 4 | 1 | 6 |

Примечание. Символом @ обозначено бесконечное расстояние.

Прежде чем перейти к обсуждению логики, целесообразно разобрать этот перебор на примере. На рисунке приведен пример и порядок просмотра городов. В кружках указаны номера городов, рядом с кружками - суммарная стоимость проезда до этого города из первого.




Итак, запись логики.

```
procedure Solve(v,Count:byte;Cost:integer);
{v - номер текущего города; Count - счетчик числа пройденных городов; Cost - стоимость текущего решения}
 var i:integer;
 begin
  if Cost>BestCost then exit;{Стоимость текущего решения превышает стоимость лучшего из ранее полученных }
  if Count=N then begin Cost:=Cost+A[v,1];Way[N]:=v;{Последний город пути. Доформировываем решение и сравниваем его с лучшим из ранее полученных.}
                        if Cost<BestCost then begin
                                                BestCost:=Cost;BestWay:=Way;
                                                end;
                        exit;
                  end;
  Nnew[v]:=false;Way[Count]:=v;{Город с номером v пройден, записываем его номер в путь коммивояжера}
  for i:=1 to N do if Nnew[B[v,i]] then
                        Solve(B[v,i], Count+1,Cost+A[v,B[v,i]]); {Поиск города, в который коммивояжер может пойти из города с номером v}
                        Nnew[v]:=true; {Возвращаем город с номером v в число непройденных}
end;
```

Первый вызов - Solve(1,1,0).

Разработка по данному «шаблону» работоспособной программы - техническая работа. Если Вы решитесь на это, то есть смысл проверить ее работоспособность на следующем примере (матрица A приведена ниже). Решение имеет вид 1 8 9 2 5 10 6 7 4 3 1, его стоимость 158.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| @ | 32 | **19** | 33 | 22 | 41 | 18 | 15 | 16 | 31 |
| 32 | @ | 51 | 58 | 27 | 42 | 35 | 18 | 17 | 34 |
| **9** | 51 | @ | 23 | 35 | 49 | 26 | 34 | 35 | 41 |
| 33 | 58 | 23 | @ | 33 | 37 | 23 | 46 | 46 | 32 |
| 22 | 27 | 35 | 33 | @ | 19 | 10 | 23 | 23 | 9 |
| 41 | 42 | 49 | 37 | 19 | @ | 24 | 42 | 42 | 10 |
| 18 | 35 | 26 | 23 | 10 | 24 | @ | 25 | 25 | 14 |
| 15 | 18 | 34 | 46 | 23 | 42 | 25 | @ | 1 | 32 |
| 16 | 17 | 35 | 46 | 23 | 42 | 25 | 1 | @ | 32 |
| 31 | 34 | 41 | 32 | 9 | 10 | 14 | 32 | 32 | @ |

Оцените время работы программы. Если у Вас мощный компьютер, то создайте матрицу A[1..50,1..50] и попытайтесь найти наилучшее решение с помощью разобранного метода. Заметим, что набор 2500 чисел - утомительное занятие и разумная лень - «двигатель прогресса».

## 2.1.8. Задача о секторах

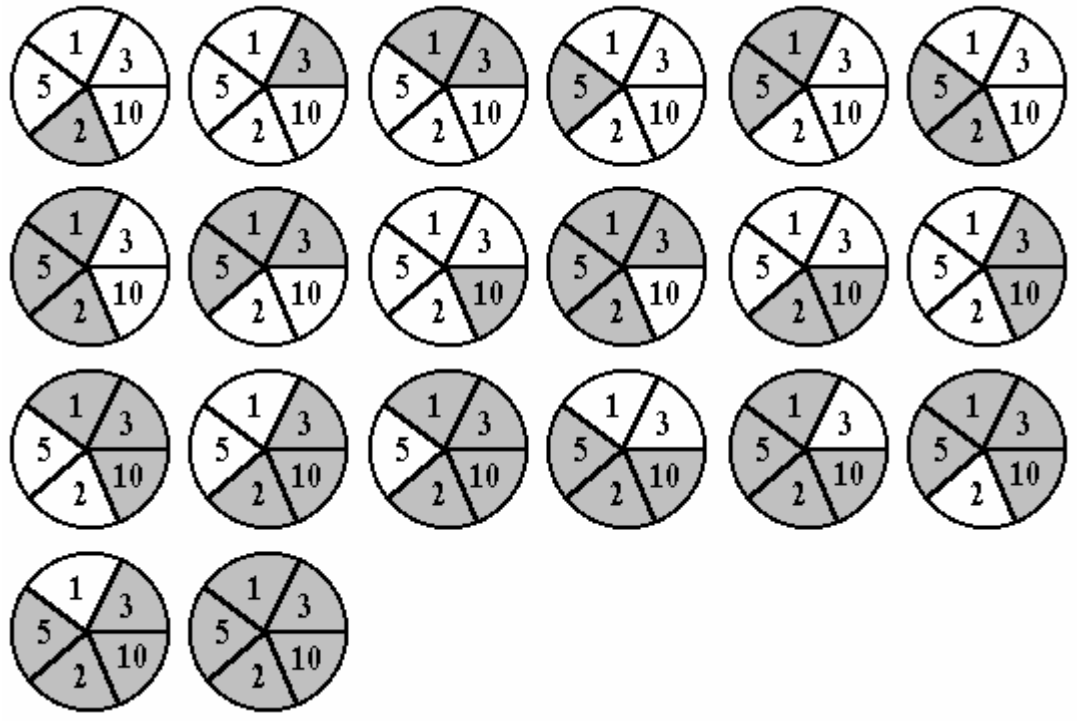
Задан круг, разделенный на секторы. Даны три числа: k (0<=k<=20), n (n<=6) и m (m<=20), где n - количество секторов. В каждый из секторов помещается одно число >= k. Когда секторы заполнены числами, Вы можете получать из них новые числа по одному из следующих правил:

- взять число из одного сектора;
- взять число, равное сумме двух или более чисел в смежных секторах.

Из этих чисел составляется наибольшая последовательность подряд идущих новых чисел, начинающихся с числа m :(m, m+1, m+2, ..., i).

Напишите программу, которая определяет способ расстановки чисел в секторах, максимизирующий длину последовательности.

Пример. На рисунке показано, как получаются все новые числа от 2 до 21 из чисел, записанных в секторах. Серым цветом выделены суммируемые числа.




*Входные и выходные данные*

Исходные данные расположены во входном файле с именем INPUT.TXT, который содержит числа n, m и k. Ниже приведен пример файла исходных данных INPUT.TXT.

```
5
2
1
```

Выходной файл с именем OUTPUT.TXT должен содержать:

- наибольшее число i в неразрывной последовательности, которое может быть получено из чисел в секторах;

- все наборы чисел в секторах, из которых можно получить неразрывную последовательность от m до i. В каждой строке выводится один набор чисел в секторах. Каждый такой набор - список чисел, начинающихся с наименьшего ( которое может быть не единственным).

Например, (2 10 3 1 5) не является решением, так как начинается не с наименьшего числа. Обратите внимание, что (1 3 10 2 5) и (1 5 2 10 3) считаются различными решениями и должны быть оба выведены. Ниже приведен файл OUTPUT.TXT для нашего примера.

```
21
1 3 10 2  5
1 5 2 10  3
2 4 9 3 5
2 5 3 9 4
```

Если наименьшее число встретится несколько раз, вывести все возможные комбинации, например: (1 1 2 3), (1 2 3 1), (1 3 2 1) и (1 1 3 2).

*Рассуждения по поводу задачи*. Очевидно, что в первый сектор может помещаться число из интервала от k до m. Считаем, что минимальное из чисел находится в первом секторе. Если k больше m, то задача не имеет решения. Обозначим через Up верхнюю границу, а через Down - нижнюю границу интервала, из которого могут браться числа для записи в сектора с номерами от 2 до n. Общая схема («набросок»).

```
for l:=k to m do begin
<выбор числа в первый сектор>;
for j:=2 to n do begin (* j - номер сектора *)
 <определение значений верхней и нижней границ для чисел, записываемых в сектор с номером j>;
 for i:=Down to Up do   (* i - записываемое число *)
  begin
   < записать в сектор с номером j число i >
   < откорректировать массив сумм, которые можно составить из чисел, записанных в сектора>
   < выполнить проверку последовательности сумм >
  end;
 end;
end;
```

Эта привлекательная схема перебора мало пригодна для «жизни», ибо время перебора будет весьма не привлекательным. Попробуем на этапе уточнения повысить пригодность схемы. "Откорректировать массив сумм". Пусть в Q (array[1..n,0..n] of byte) будут храниться суммы чисел из секторов круга S (array[1..n] of byte). В нулевом столбце элементы равны 0. В 1-м столбце - суммы, составленные из чисел, взятых из одного сектора, 2-м - из двух и т.д. В n-м столбце подсчитывается только элемент в первой строке.

Массив Q:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 S[1] | Q[1,1]+S[2] | Q[1,2]+S[3] | Q[1,3]+S[4] | Q[1,4]+S[5] | Q[1,5]+S[6] |
| 2 | 0 S[2] | Q[2,1]+S[3] | Q[2,2]+S[4] | Q[2,3]+S[5] | Q[2,4]+S[6] | |
| 3 | 0 S[3] | Q[3,1]+S[4] | Q[3,2]+S[5] | Q[3,3]+S[6] | Q[3,4]+S[1] | |
| 4 | 0 S[4] | Q[4,1]+S[5] | Q[4,2]+S[6] | Q[4,3]+S[1] | Q[4,4]+S[2] | |
| 5 | 0 S[5] | Q[5,1]+S[6] | Q[5,2]+S[1] | Q[5,3]+S[2] | Q[5,4]+S[3] | |
| 6 | 0 S[6] | Q[6,1]+S[1] | Q[6,2]+S[2] | Q[6,3]+S[3] | Q[6,4]+S[4] | |

Итак, если у нас есть массив констант (для простоты) Sq вида

```
1 2 3 4 5 6
2 3 4 5 6 0
3 4 5 6 1 0
5 6 1 2 3 0
6 1 2 3 4 0,
```

то элемент массива Q[i,j] вычисляется следующим образом: Q[i,j]:=Q[i,j-1] + Q[Sq[i,j],1] .

При изменении числа в секторе с номером t необходимо откорректировать часть матрицы Q, показанную на рисунке темным цветом (эта часть больше, чем требуется, но для простоты считаем ее такой).

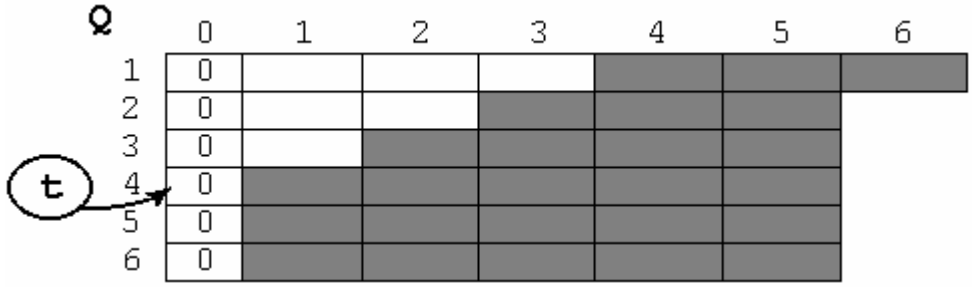
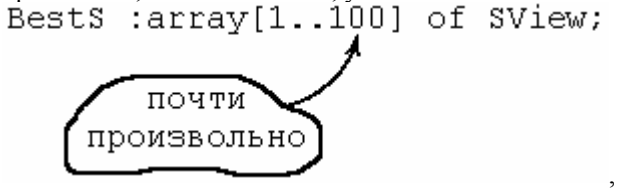



Схема процедуры уточнения сумм:

```
procedure AddSum(t,number :byte);
(* Q, Sq - глобальные переменные *)
```

```
 var z,i,j :integer;
 begin
  Q[t,1]:=number;
  for i:=1 to n do begin
   if t-i+1>1 then z:=t-i+1 else z:=2;
          for j:=z to n-1 do begin
                  Q[i,j]:=0;
                  Q[i,j]:=Q[i,j-1]+Q[Sq[i,j],1];
                  end;
  end;
  Q[1,n]:=Q[1,5] + Q[n,1];
 end;
```

Следующее уточнение. "Выполнить проверку последовательности сумм". Из чего следует исходить? Во-первых, наилучшая последовательность сумм может получиться не из одного варианта заполнения секторов числами. Поэтому необходимо ввести структуру данных - двумерный массив - для их хранения и, соответственно, указатель числа записанных вариантов:




,

где type SView = array[1..nMax] of byte;
NumS :byte;

Во-вторых, для хранения наибольшего числа необходима переменная MaxS. В-третьих, значения сумм лучше представить множественным типом данных SetS :set of byte. Это упростит логику поиска последовательности. Процедура проверки имеет вид:

```
procedure Check;
 (* SetS, BestS, NumS, MaxS, M - глобальные *)
 var i,j :integer;
 begin
  SetS:=[];
  for i:=1 to N do
   for j:=1 to N-1 do SetS:=Sets+[Q[i,j]];
   SetS:=SetS+[Q[1,N]];
  i:=M;
  while i in SetS do Inc(i);
  if i>MaxS then begin
          NumS:=1;
          BestS[NumS]:=S;
          MaxS:=i;(* на единицу больше действительной длины *)
  end
  else if i=MaxS then begin Inc(NumS); BestS[NumS]:=S; end;
 end;
```

Общая схема уточнена. Но если довести ее до программы, то решение, например, для исходных данных n=6, m=1, k=1 за реальное время не будет получено. Необходимо сокращать перебор, искать эвристики, позволяющие сделать это.

1-я эвристика. Пусть у нас есть (n-1) сектор, в которые записаны числа, образующие последовательность m, m+1, ... . количество сумм (арифметическая прогрессия) равно n*(n-1)div2, т. е. это количество различных чисел, получаемое из чисел в заполненных n-1 секторе. Обозначим через X первое число из последовательности m, m+1, ..., которое мы не можем получить. В пустой сектор не имеет смысла записывать число, большее X. Поскольку X не превосходит n(n-1) div 2 + m, то это выражение можно считать значением верхней границы Up чисел, записываемых в сектора. В качестве нижней границы Down следует брать S[1], ибо числа, записываемые в сектора 2, 3, ..., n не меньше этого числа.

2-я эвристика. Пусть m<2*k. Тогда числа m, m+1, ..., 2*k-1 как сумма чисел из нескольких (более одного) секторов. Следовательно, либо, если 2*k-m≤n, все они должны находиться в круге, либо, если 2*k-m>n, в круге должны находиться числа m, m+1, ..., m+n-1.

3-я эвристика. "Исключение симметричных решений". При поиске числа для записи в последний сектор значение Down можно уточнить:

```
   if < сектор последний > then Down:=S[2]
   else Down:=S[1];
```

4-я эвристика. "Отсечение снизу". При поиске числа для последнего сектора нам известна максимальная сумма, которую дают числа, записанные в (N-1) сектор. Это значение Q[1,N-1]. Если сумма Up и Q[i,N-1] меньше ранее полученного наибольшего числа MaxS, то эту "ветку" в переборе вариантов можно "отсечь"(при любом варианте заполнения последнего сектора лучшего решения мы не получим).

Описанных эвристик (чем не эвристическое программирование?) оказывается достаточно для написания работоспособной программы для нижеприведенных тестов. В таблице приведены и правильные результаты.

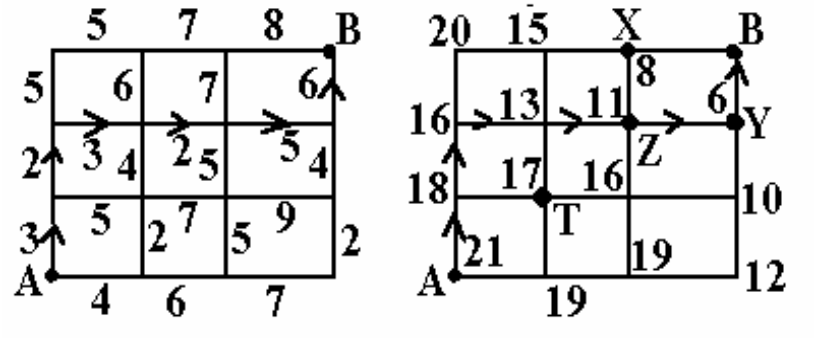
| N | M | K | max | |
|---|---|---|---|---|
| 3 | 1 | 1 | 7 | 1 2 4; 1 4 2 |
| 4 | 2 | 2 | 13 | 2 3 4 6; 2 6 4 3 |
| 5 | 3 | 1 | 22 | 3 5 7 4 6; 3 6 4 7 5 |
| 4 | 16 | 1 | 23 | 1 16 2 20; 1 20 2 16;<br>1 16 4 18; 1 18 4 16;<br>2 16 4 17; 2 17 4 16 |
| 5 | 16 | 12 | 20 | 16 17 18 19 20; 16 20 19 18 17;<br>16 17 18 20 19; 16 19 20 18 17;<br>16 17 19 18 20; 16 20 18 19 17;<br>16 17 19 18 20; 16 20 18 19 17;<br>16 17 19 20 18; 16 18 20 19 17;<br>16 17 20 18 19; 16 19 18 20 17<br>. . . |
| 6 | 1 | 1 | 31 | 1 2 5 4 6 13; 1 13 6 4 5 2;<br>1 2 7 4 12 5; 1 5 12 4 7 2;<br>1 3 2 7 8 10; 1 10 8 7 2 3;<br>1 3 6 2 5 14; 1 14 5 2 6 3;<br>1 7 3 2 4 14; 1 14 4 2 3 7 |
| 6 | 2 | 2 | 29 | 2 3 7 4 9 6; 2 6 9 4 7 3 |
| 6 | 4 | 4 | 31 | 4 4 6 5 7 9; 4 9 7 5 6 4;<br>4 4 9 7 5 6; 4 6 5 7 9 4;<br>4 5 5 6 7 8; 4 8 7 6 5 5 |

## 2.2. Динамическое программирование

Важнейший раздел изучаемой темы, поэтому вполне применим принцип: чем больше задач , тем лучше. Динамическое программирование - один из методов решения задач оптимизации. Слово “программирование” здесь не имеет того значения, которое используется в информатике. В данном случае оно является производным от термина programme mathematique, обозначающего систему неравенств, которые надо решить.

**s**

Черепашке необходимо попасть из пункта A в пункт B. На каждом углу она может поворачивать только на север или только на восток. Время движения по каждой улице указано на рисунке. Требуется найти минимальное время, за которое Черепашка может попасть из пункта A в пункт B.




Путь, показанный на рисунке линиями со стрелкой, требует 21 единицу времени. Отметим, что каждый путь состоит из 6 шагов: трех на север и трех на восток. Количество возможных путей Черепашки 20 ($C_6^3$). Мы можем перебрать все пути и выбрать самый быстрый. Это метод полного перебора (ответ - 21). Для вычисления каждого времени требуется пять операций сложения, а для нахождения ответа 100 операций сложения и 19 операций сравнения. Задача решается вручную. Однако при N, равном 8, у Черепашки уже 12870 путей. Подсчет времени для каждого пути требует 15 операций сложения, а в целом -

193050 сложений и 12869 сравнений, то есть порядка 200000 операций. Компьютер при быстродействии 1000000 операций в секунду найдет решение за 0.2 секунды, а человек? Предположим, что N равно 30, тогда количество различных путей 60!*(30!*30!). Это очень большое число, его порядок $10^{17}$. Количество операций, необходимых для поиска решения, равно $60*10^{17}$. Компьютер с быстродействием 1000000 операций в секунду выполнит за год порядка $3.2*10^{13}$ операций (подсчитайте). А сейчас не трудно прикинуть количество лет, требуемых для решения задачи.

Вернемся к исходной задаче. Начнем строить путь Черепашки от пункта B. Каждому углу присвоим вес, равный минимальному времени движения Черепашки от этого угла до пункта B. Как его находить? От углов X, Y очевидно. Для угла Z время движения Черепашки в пункт B через угол X 15 единиц, а через угол Y 11 единиц. Берем минимальный, то есть вес угла будет равен 11. Продолжим вычисления. Их результаты приведены на рисунке. Путь, отмеченный стрелками, является ответом задачи. Оценим количество вычислений. Для каждого угла необходимо выполнить не более двух операций сложения и одной операции сравнения, то есть три операции. При N, равном 300, количество операций - 3*301*301, это меньше 1000000, и компьютеру потребуется меньше одной секунды. Итак, много лет при N=30 и 1 секунда при N=300.

Идея второго способа решения задачи основана на методе динамического программирования. Заслуга его открытия принадлежит американскому математику Ричарду Беллману. Выделим особенность задачи, которая позволяет решить ее описанным способом. Мы начинаем с угла B, и для каждого угла найденное число является решением задачи меньшей размерности. Поясним эту мысль. Если бы мы решали задачу поиска пути Черепашки из пункта T в пункт B, то найденное число 17 - решение задачи. Для каждого угла найденное значение времени не изменяется и может быть использовано на следующих этапах.

## 2.2.2. Треугольник

На рисунке изображен треугольник из чисел. Напишите программу, которая вычисляет наибольшую сумму чисел, расположенных на пути, начинающемся в верхней точке треугольника и заканчивающемся на основании треугольника.

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

- Каждый шаг на пути может осуществляться вниз по диагонали влево или вниз по диагонали вправо.
- Число строк в треугольнике > 1 и <100.
- Треугольник составлен из целых чисел от 0 до 99.

Рассмотрим идею решения на примере, приведенном в тексте задачи. Входные данные запишем в матрицу D. Она, для примера, имеет вид:

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

Будем вычислять матрицу R:array[1..MaxN,0..MaxN+1] следующим образом, предварительно обнулив ее элементы:

R[1,1]=D[1,1]

R[i,j]=max(D[i,j]+R[i-1,j],D[i,j]+R[i-1,j-1])

для i=2..N; j=1..i.

Ее вид:

```
0  7
0 10 15
0 18 16 15
0 20 25 20 19
0 24 30 27 26 24
```

Осталось найти наибольшее значение в последней строке матрицы R. Итак, в решении задачи используются идеи метода динамического программирования.

## 2.2.3 Степень числа

Даны два натуральных числа n и k. Требуется определить выражение, которое вычисляет $k^n$ . Разрешается использовать операции умножения и возведения в степень, круглыми скобками и переменной с именем k.Умножение считается одной операцией, возведению в степень q соответствует q-1 операция. Найти минимальное количество операций, необходимое для возведения в степень n. Желательно сделать это для как можно больших значений n.

Пример. При n=5 необходимо три операции - $(k*k)^2*k$.

Определим массив Op, его элемент Op[i] предназначен для хранения минимального количества операции при возведении в степень i (Op[1]=0). Для вычисления выражения, дающего n-ю степень числа k, арифметические операции применяют в некоторой последовательности, согласно приоритетам и расставленным скобкам. Рассмотрим последнюю примененную операцию.

Если это было умножение, тогда сомножителями являются натуральные степени числа k, которые в сумме дают n. Степень каждого из сомножителей меньше n, и ранее вычислено, за какое минимальное число операций ее можно получить. Итак:

opn1:=min{по всем p:1≤p<n, Op[p]+Op[n-p]+1}.

Если это возведение в степень:

opn2:=min{ для всех p (≠1) - делителей n, Op[n div p]+p-1}.

Результат - Op[n]=min(opn1,opn2). Фрагмент реализации:

```
.....
Op[1]:=0;
for n:=2 to ???? do begin
 opn:=n;{opn - рабочая переменная}
 for p:=1 to n-1 do begin
  opn:=Min(opn,Op[p]+Op[n-p]+1);{Min - функция поиска минимума двух чисел}
  if (n mod p=0) and (p<>1) then opn:=Min(opn,Op[n div p]+p-1);
 end;
 Op[n]:=opn;
end;
....
```

### 2.2.4. Автозаправка

Вдоль кольцевой дороги расположено m городов, в каждом из которых есть автозаправочная станция. Известна стоимость Z[i] заправки в городе с номером i и стоимость C[i] проезда по дороге, соединяющей i - й и (i+1)-й города, C[m] - стоимость проезда между первым и m-м городами. Для жителей каждого города определить город, в который им необходимо съездить, чтобы заправиться самым дешевым образом, и направление - «по часовой стрелке» или «против часовой стрелки», города пронумерованы по часовой стрелке.

Не будем рассматривать переборный вариант решения задачи, суть которого в проверке всех 2*m вариантов для жителей каждого города, итого - 2*m*m проверок. Введем два дополнительных массива

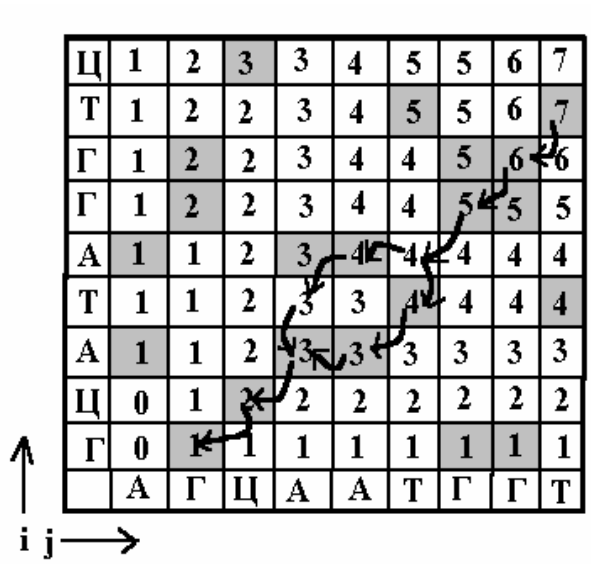
On, Ag: array[1..m] of record wh, qh:integer; end; .

On[i] означает, где следует заправляться (wh) и стоимость заправки (qh) жителям i-го города, если движение разрешено только по часовой стрелке. В этом случае жители города i имеют две альтернативы: либо заправляться у себя в городе, либо ехать по часовой стрелке. Во втором случае жителям города i надо заправляться там же, где и жителям города i+1, или в первом, если i=m. Итак, On[i]=min{Z[i],C[i]+On[i+1].qh}. Откуда известно значение On[i+1].qh? Необходимо найти город j с минимальной стоимостью заправки - On[j]:=(j,Z[j]). После этого можно последовательно вычислять значения On[j-1], On[j-2], ..., On[j+1]. Аналогичные действия необходимо выполнить при формировании массива Ag[i], после этого для жителей каждого города i следует выбрать лучший из On[i].qh и Ag[i].qh вариант заправки.

### 2.2.5. Алгоритм Нудельмана-Вунша

Пример из молекулярной биологии. Молекулы ДНК, содержащие генетическую информацию - это длинные слова из четырех букв (А, Г, Ц, Т). В процессе эволюции, в результате мутаций, последовательности меняются, одна буква может замениться на другую, выпасть, а может добавиться новая. Насколько похожи два фрагмента, каким наименьшим числом превращений можно один из них получить из другого? Формулировка задачи. Даны два слова (длины M и N), состоящие из букв А, Г, Ц, Т. Найти подпоследовательность наибольшей длины, входящую в то и другое слово.

Пример. Слова ГЦАТАГГТЦ и АГЦААТГГТ. Схема решения иллюстрируется следующим рисунком.




На рисунке закрашены клетки, в строке и в столбце которых находятся одинаковые буквы. Принцип заполнения таблицы W следующий: элемент W[i,j] равен наибольшему из чисел W[i-1,j], W[i,j-1], а если клетка <i,j> закрашена, то и W[i-1,j-1]+1. Формирование первой строки и первого столбца выполняется до заполнения таблицы и осуществляется так: единицей отмечается первое совпадение, затем эта единица автоматически заносится во все оставшиеся клетки. Например, W[3,1] - первое совпадение в столбце, затем эта единица идет по первому столбцу. Подпоследовательность формируется при обратном просмотре заполненной таблицы от клетки, помеченной максимальным значением. Путь - это клетки с метками, отличающимися на единицу, буквы из закрашенных клеток выписываются. Последовательность этих букв - ответ задачи. Для нашего примера две

подпоследовательности: ГЦААГГТ и ГЦАТГГТ.

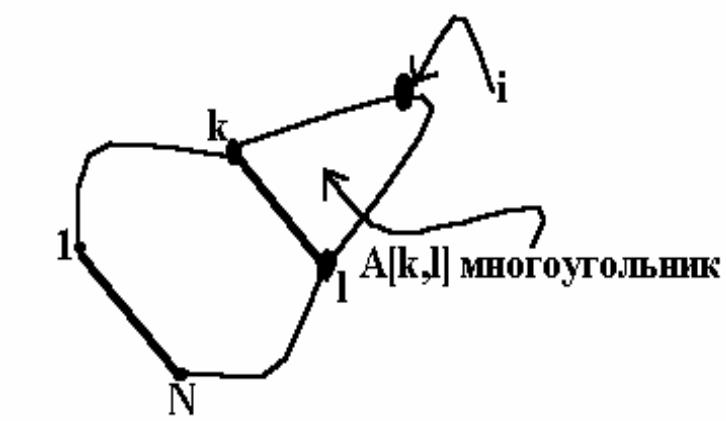
Фрагмент основной логики.

```
...
for i:=1 to Length(S1) do
 for j:=1 to Length(S2) do begin
  A[i,j]:=Max(A[i-1,j],A[i,j-1]);
  if S1[i]=S2[j] then A[i,j]:=Max(A[i,j],A[i-1,j-1]+1);
 end;
Writeln('Ответ: ',A[Length(S1),Length(S2)]);
....
```

### 2.2.6. Разбиение выпуклого N-угольника

Дан выпуклый N-угольник, заданный координатами своих вершин в порядке обхода. Он разрезается N-2 диагоналями на треугольники. Стоимость разрезания определяется суммой длин всех использованных диагоналей. Найти разрез минимальной стоимости.

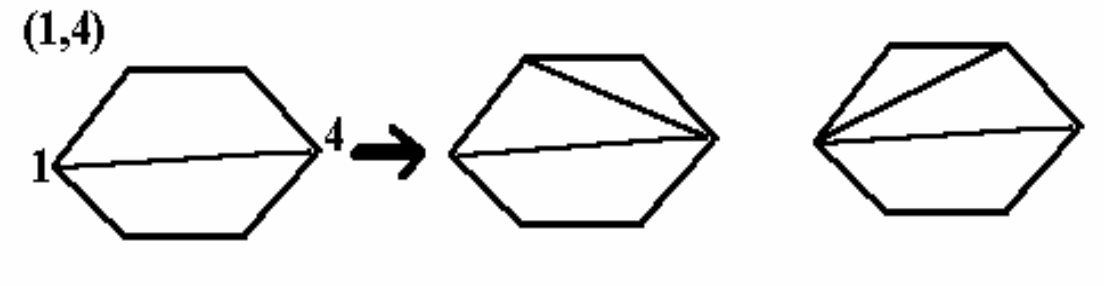
Идея решения разбирается с использованием следующего рисунка.

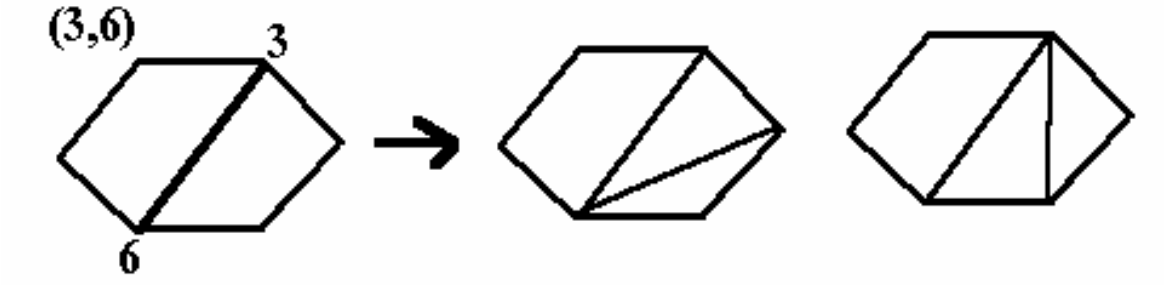


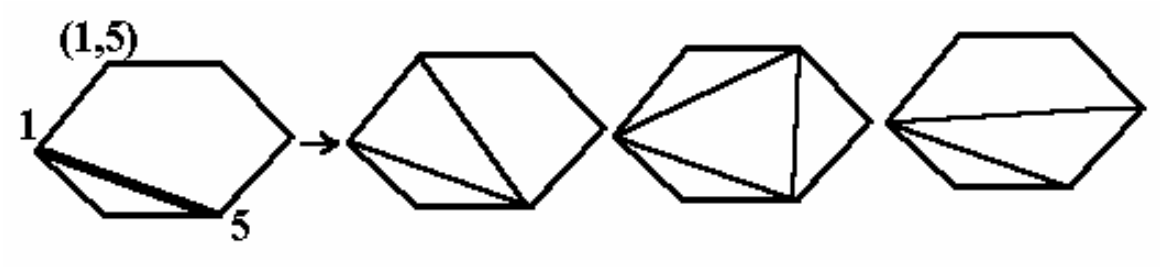

Обозначим через S[k,l] стоимость разрезания многоугольника A[k,l] диагоналями на треугольники. При l=k+1 или k+2 S[k,l]=0, следовательно, l>k+2. Вершина с номером i, i изменяется от k+1 до l-1, определяет какое-то разрезание многоугольника A[k,l]. Стоимость разрезания определим как:

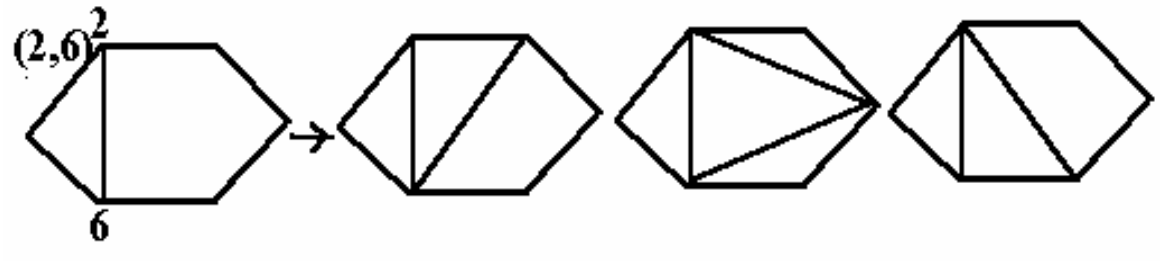
S[k,l]=min{длина диагонали <k,i>+длина диагонали <i,l>+S[k,i]+S[i,l]}. При этом следует учитывать, что при i=k+1 диагональ <k,i> является стороной многоугольника и ее длина считается равной нулю.

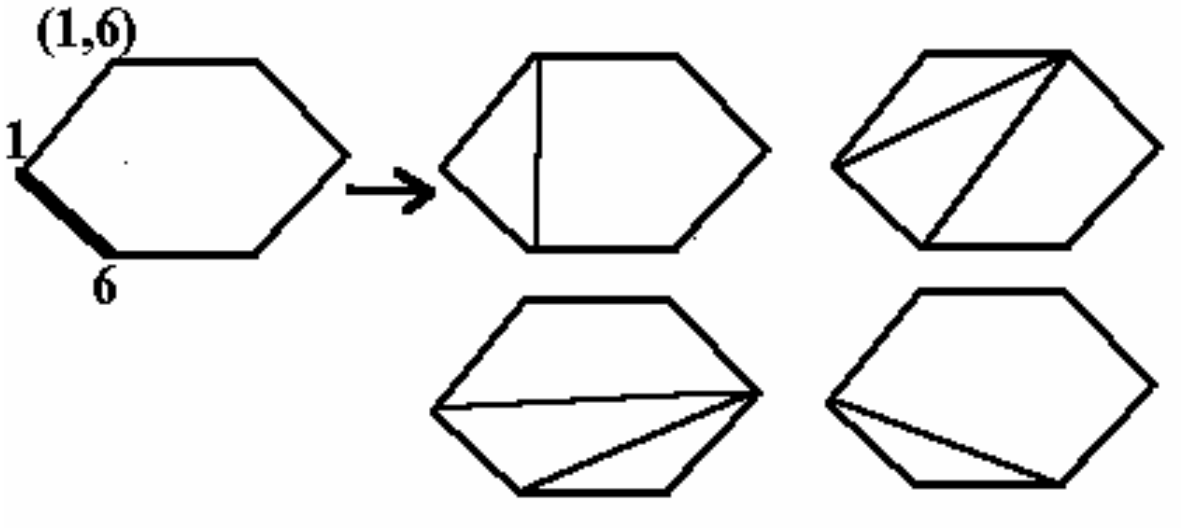
Пример(N=6).






### 2.2.7. Задача о рюкзаке (динамическая схема)

Рассмотрим задачу из пункта 1.1.6. Напомним ее формулировку. В рюкзак загружаются предметы n различных типов (количество предметов каждого типа не ограничено). Максимальный вес рюкзака не превышает W. Каждый предмет типа i имеет вес $w_i$ и стоимость $v_i$ (i=1,2, ..., n). Требуется определить максимальную стоимость груза, вес которого не превышает W. Обозначим количество предметов типа i через $k_i$, тогда требуется максимизировать $v_1*k_1+v_2*k_2+...+v_n*k_n$ при ограничениях $w_1*k_1+w_2*k_2+...+w_n*k_n \le W$, где $k_i$ - целые ($0 \le k_i \le [W/w_i]$), квадратные скобки означают целую часть числа.

Пусть вес рюкзака должен быть равен W. Формализуем задачу следующим образом.

- Шаг i ставится в соответствие типу предмета i=1,2,...,n.
- Состояние $y_i$ на шаге i выражает суммарный вес предметов, решение о загрузке которых принято на шагах 0,1,...,i. При этом $y_n=W$, $y_i=0,1,...,W$ при i=1,2,...,n-1.
- Варианты решения $k_i$ на шаге i описываются количеством предметов типа i, $0 \le k_i \le \lfloor W/w_i \rfloor$.

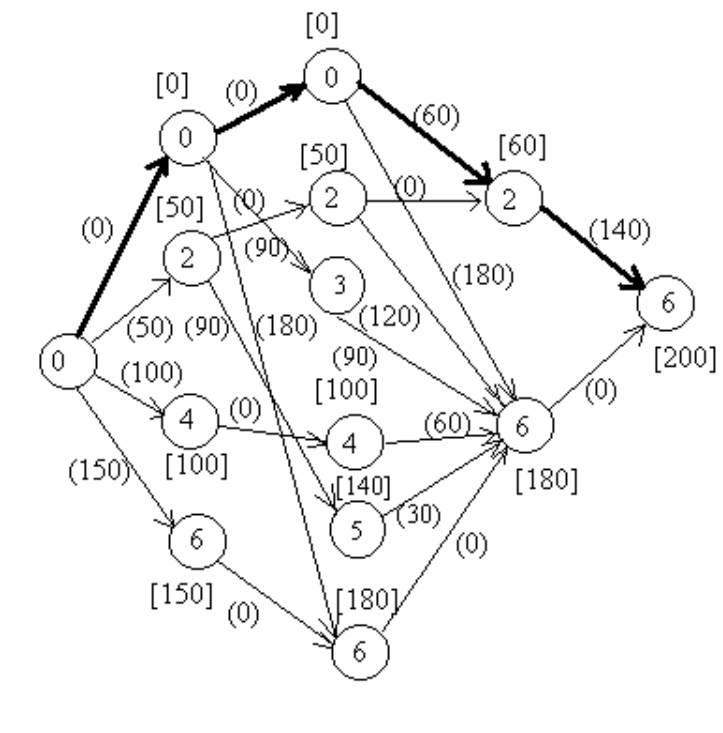
Рассмотрим пример. W=6, и дано четыре предмета

| i | $w_i$ | $v_i$ |
|---|---|---|
| 1 | 2 | 50 |
| 2 | 3 | 90 |
| 3 | 1 | 30 |
| 4 | 4 | 140 |

Схема работы для данного примера приведена на рисунке. В кружочках выделены только достижимые состояния (суммарные веса для каждого шага в соответствии с приведенной выше формализацией) на каждом шаге. В круглых скобках указаны стоимости соответствующих выборов, в квадратных скобках - максимальная стоимость данного заполнения рюкзака. «Жирными» линиями выделен способ наилучшей загрузки рюкзака.

Текст решения.

```
Const MaxN=???;
      MaxK=???;
Type Thing=Record W,V:Word; end;
Var   A:array[1..MaxN,0..MaxK] of word;
      P:array[1..MaxN] of Thing;
```

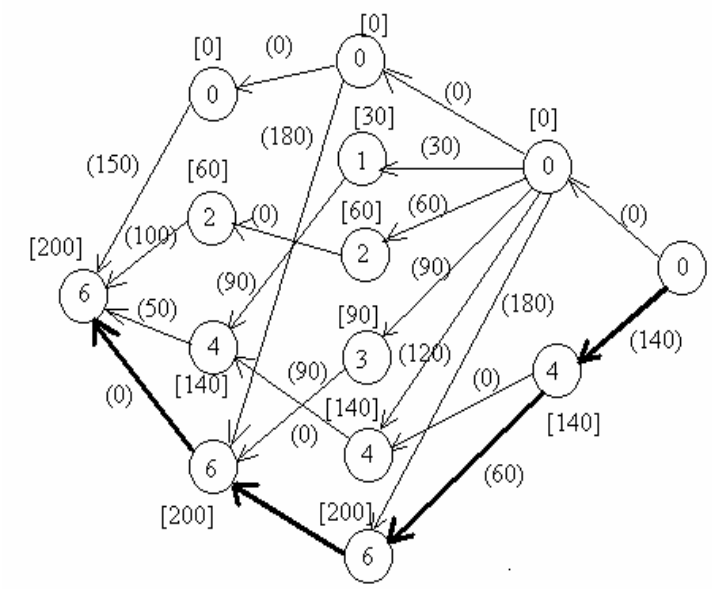



```
                                  Old,NewA:array[
0..MaxK] of longint;
                                        N,W:integer;
...
procedure Solve;
 var k,i,j:integer;
 begin

fillchar(Old,sizeof(Old),0);
 for k:=1 to N do
begin{цикл по шагам}

 fillchar(NewA,sizeof(New
A),0);
  for i:=0 to W do {цикл
по состояниям шага}
   for j:=0 to i div P[k].W
do {цикл по вариантам
решения - количеству
предметов каждого
вида}
    if j*P[k].V+Old[i-
j*P[k].W]>=NewA[i] then
begin
                NewA[i]:=j*P[k].V+Old[i-j*P[k].W];
                                        A[k,i]:=j;{здесь j количество предметов?}
                                    end;
  Old:=NewA;
 end;
end;
```

Вывод наилучшего решения.

```
procedure OutWay(k,l:integer);
 begin
  if k=0 then exit
   else begin
    OutWay(k-1,l-A[k,l]*P[k].V);{а здесь вес}
    Write(A[k,l],' ');
   end;
 end;
```

Первый вызов - OutWay(N,W). Эту схему реализации принято называть «прямой прогонкой». Ее можно изменить. Пусть пункт два формализации задачи звучит следующим образом. Состояние $y_i$ на шаге i выражает суммарный вес предметов, решение о загрузке которых принято на шагах i, i+1, ..., N при этом $y_1=W$ $y_i=0,1,...,W$ при i=2,3, ...,N. В этой формулировке схему реализации называют «обратной прогонкой».



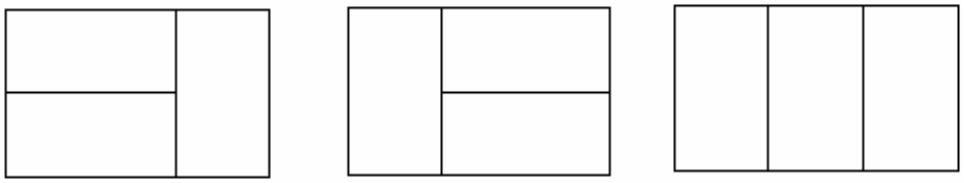

## 2.2.8. Задача о паркете

Комнату размером n*m единиц требуется покрыть одинаковыми плитками паркета размером 2*1 единиц без пропусков и наложений (m≤20, n≤8, m,n -целые). Пол можно покрыть паркетом различными способоми. Например, для m=2, n=3 все возможные способы укладки приведены на рисунке.

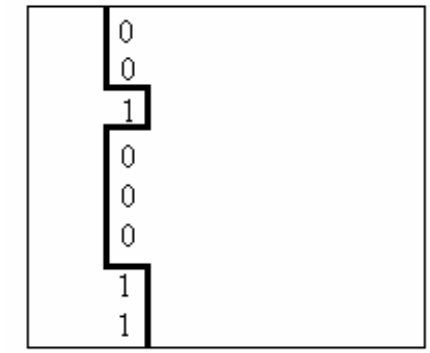
Требуется определить количество всех возможных способов укладки паркета для конкретных значений m≤20, n≤8. Результатом задачи является таблица, содержащая 20 строк и 8 столбцов.

Элементом таблицы является

йденных результатов



азрежем» комнату на одим справа, т.е. при либо нет. При других нумеровываются, ибо ствие обхода - 0. На 0011 (35). Количество




Не будем пока учитывать то, что некоторые из сечений не могут быть получены. Обозначим парой (k,j) комнату с фиксированной длиной i, правый край которой не ровный, а представляет собой сечение с номером k. B[k,j] - количество вариантов укладки паркета в такой комнате. Задача в такой постановке сводится к нахождению B[0,j] - количество укладок паркета в комнате размером (i,j) с ровным правым краем.

Считаем, что B[0,0]=1 при любом i, ибо комната нулевой ширины, нулевого сечения и любой длины укладывается паркетом, при этом не используется ни одной плитки. Кроме этого считаем B[k,0]=0 для всех сечений с номерами k<>0, так как ненулевые сечения при нулевой ширине нельзя реализовать.

Попытаемся найти B[k,j] для фиксированного i. Предположим, что нам известны значения B[l,j-1] для всех сечений с номерами l ($0 \le l \le 2^i-1$). Сечение l считаем совместимым с сечением k, если путем добавления целого числа плиток паркета из первого можно получить второе. Тогда $B[k,j]=\sum B[l,j-1]$, суммирование ведется по всем сечениям l, совместимым с сечением k. Налицо динамическая схема решения задачи.

Оставляя пока в стороне вопрос совместимости сечений, «набросаем» логику решения.

Данные.

```
var     B:array[0..255,0..20] of Comp;
        A:array[1..8,1..20] of Comp;{Результирующая таблица}
function St2(k:integer):integer;{Вычисляем k - ю степень 2}
 begin
  if k<=0 then St2:=1 else St2:=2*St2(k-1);
 end;
procedure Solve;{Основная логика}
 var i,j,k,l,max:integer;
 begin
  for i:=1 to 8 do begin {Цикл по значению длины комнаты}
```

```
  max:=St2(i)-1;
  fillchar(B,sizeof(B),0);
  B[0,0]:=1;
  for j:=1 to 20 do begin {Цикл по значению ширины комнаты}
          for k:=0 to max do {Сечение с номером k}
           for l:=0 to max do{Сечение с номером l}
            if Can(k,l,i) then B[k,j]:=B[k,j]+B[l,j-1];{Совместимость сечений «зарыта» в функции Can(k,l,i)}
           A[i,j]:=B[0,j];
  end;
 end;
end;
```

Остался открытым вопрос о совместимости сечений. В этом случае необходимо различать понятия совместимость сечений в целом и совместимость в отдельном разряде. При анализе последнего входными данными являются значения разрядов сечений и информация о предыстории процесса (до текущего разряда) - целое или нецелое количество плиток уложено (значение переменной b). Выходными данными - признак - целое, нецелое количество плиток, требуемых для перевода сечения l в сечение k, или решение о том, что анализ продолжать не следует - сечения несовместимы. В данном случае этапу написания логики предшествует анализ на бумаге. Пример этой работы приведен в таблице.

Таблица

| l | k | Значе-ние b | Новое значе-ние b или резуль-тат | Примечания |
|---|---|---|---|---|
| 1 | 0 | false | false | До данного разряда уложено целое число плиток. Для перехода от l к k не надо ничего укладывать. Переходим к следующему разряду. |
| 1 | 0 | true | несовмести-мы | До этого разряда уложено нецелое число плиток, а на этом «закрываем доступ». Сечения несовместимы. |
| 0 | 1 | false | false | Укладываем целую плитку, количество уложенных плиток остается целым. |
| 0 | 1 | true | несовмести-мы | Для перехода к новому сечению в этом разряде необходимо уложить целую плитку. Однако, укладывая её мы «перекрываем» доступ к нецелой плитке, уложенной до этого. |
| 0 | 0 | false | true | До этого разряда между сечениями было уложено целое число плиток, а на этом можно уложить только полплитки. |
| 0 | 0 | true | false | Нецелую плитку дополняем до целой. |
| 1 | 1 | false | несовмести-мы | В этом разряде l лежит целая плитка. Необходимо уложить целую плитку для того, чтобы была 1 в этом разряде сечения k, но сделать этого мы не можем. Аналогично и для случая, когда b=true. |

Данный анализ совместимости сечений реализован в функции Can.

```
function Can(k,l,pi:byte):boolean;{k,l -номера сечений, pi - количество анализируемых разрядов сечений}
 var i,d:integer;b:boolean;
 begin
  Can:=false;b:=false;
  for i:=1 to pi do begin
   d:=St2(i);
   if k and d =0 then{определяется значение разряда с номером d для сечения k}
    if l and d =0 then b:=not(b)
     else begin if b then exit end
    else if (l and d<>0) or b then exit
  end;
  Can:=not(b);
  end;
```

Осталось сделать еще несколько замечаний. Во-первых, если произведение n на m нечетно (размеры комнаты), то количество укладок паркетом такой комнаты равно 0. Во-вторых, при m=1 и четном n ответ равен 1. В-третьих, результирующая таблица симметрична относительно главной диагонали. В-четвертых, для комнат размером 2*t достаточно просто выводится следующая рекуррентная формула: A[2,t]=A[2,t-1]+A[2,t-2] (ряд Фибоначчи). Эти замечания реализуются на этапе предварительной обработки и приводят к незначительной модификации логики процедуры Solve. Еще одно замечание касается точности результата.

Для больших значений n и m необходимо организовать вычисления с использованием длинной арифметики. Мы ограничимся маленькой хитростью - просто выпишем результат при выводе результирующей матрицы.

```
...
for i:=1 to 20 do begin
 for j:=1 to 8 do
  if (i=8) and (j=20) then write('3547073578562247994':20)
   else write(A[j,i]:20);
  writeln;
end;
```

## 2.2.9. «Канадские авиалинии»

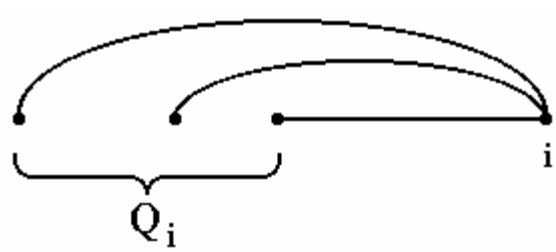
Вы победили в соревновании, организованном Канадскими авиалиниями. Приз - бесплатное путешествие по Канаде. Путешествие начинается с самого западного города, в который летают самолеты, проходит с запада на восток, пока не достигнет самого восточного города, в который летают самолеты. Затем путешествие продолжается обратно с востока на запад, пока не достигнет начального города. Ни один из городов нельзя посещать более одного раза за исключением начального города, который надо посетить ровно дважды ( в начале и в конце путешествия). Вам также нельзя пользоваться авиалиниями других компаний или другими способами передвижения. Задача состоит в следующем: дан список городов и список прямых рейсов между парами городов; найти маршрут, включающий максимальное количество городов и удовлетворяющий вышеназванным условиям.

Сделаем первоначальное упрощение задачи. Пусть нам необходимо попасть из самого западного города в самый восточный, посетив при этом максимальное количество городов. Связи между городами будем записывать с помощью массива West:

$$West[i,j]=\begin{cases}true, & \text{есть авиалиния между городами с номерами } i \text{ и } j;\\ false, & \text{авиалинии нет.}\end{cases}$$

Пусть мы каким-то образом решили задачу для всех городов, которые находятся западнее города с номером i, т.е. для городов с номерами 1..(i-1). Что значит решили? У нас есть маршруты для каждого из этих городов, и нам известно, через сколько городов они проходят. Обозначим через $Q_i$ множество городов западнее i и связанных с городом i авиалиниями. Для этих городов задача решена, т.е. известны значения d[j] ($j \in Q_i$) - количество городов в наилучшем маршруте, заканчивающемся в городе с номером j.

Итак:




```
d[i]:=0;
for j∈Qi do
if d[j]+1>d[i] then d[i]:=d[j]+1;
```

Остается открытым вопрос, а где же маршрут? Ибо значение d дает только количество городов, через которые он проходит. А для этого необходимо запомнить не только значение d[i], но и номер города j, дающего это значение. Возможно использование следующей структуры данных:

```
A:array[1..max] of record
                    d, l:byte;
                  end;
```

И реализация сказанного имеет вид:

```
FillChar(A,SizeOf(A),0);
        A[1].d:=1;
        for i:=2 to n do begin
         for j:=1 to i-1 do if west[j,i] then if A[j].d+1>A[i].d then begin
                                                                      A[i].d:=A[j].d+1;
                                                                      A[i].l:=j;
                                                                      end;
        end;
```

Видим, что это не что иное, как метод динамического программирования в действии. Осталось выполнить обратный просмотр массива A по значениям поля l, начиная с элемента A[n].l и вывести из массива name:array[1..max] of string[15] названия городов.

Этот вывод можно осуществить с помощью следующей простой рекурсивной процедуры.

```
procedure rec(i:byte);
 begin
```

```
    if i<>1 then begin rec(A[i].l); writeln(name[i]);end;
end;
```

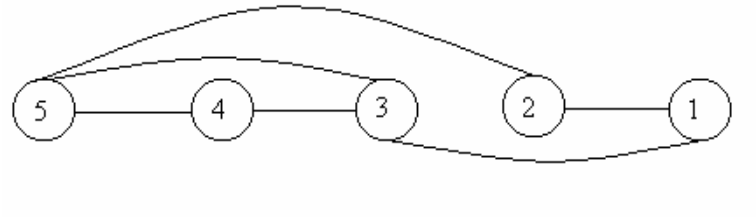
Как обобщить этот набросок решения для первоначальной формулировки задачи? Для удобства перенумеруем города в обратном порядке - с востока на запад. Изменим массив A:

```
A:array[1..max,1..max] of record
                            d,l:byte;
                            end;
```

Под элементом A[i,j].d - путь из города i в город j - понимается максимальное число городов в маршруте, состоящем из двух частей: от i до 1 (на восток) и от 1 до j (на запад). По условию задачи нам необходимо найти наилучший по числу городов путь из n-го города в n-й. Считаем, что A[1,1].d=1 и A[i,j]=A[j,i]. Через $Q_i$ обозначим множество городов, из которых можно попасть в город i. Верны следующие соотношения:

- A[i,j].d=max(A[k,j].d+1), при $k \in Q_i$, k<>j, если j<>1;
- A[i,i].d=max(A[k,i].d), при i>1 и $k \in Q_i$.

Рассмотрим логику формирования элементов массива A на следующем примере.




Основываясь на вышеприведенных соотношениях, выполним трассировку (ручную прокрутку задачи). Результаты приведены в таблице. Прочерк в клетке означает, что действия (вычисления) не выполнялись, знак «~» - сравнение. Присвоение симметричным относительно главной диагонали A элементам происходит одновременно (в таблице не приведены).

| i | j | $Q_i$ | k | Действия |
|---|---|---|---|---|
| 2 | 1 | [1] | 1 | A[2,1].d:=A[1,1].d+1=2 A[2,1].l:=1 |
| 2 | 2 | - | - | - |
| 3 | 1 | [1] | 1 | A[3,1].d:=A[1,1].d+1=2 A[3,1].l:=1 |
| 3 | 2 | [1] | 1 | A[3,2].d:=A[1,2].d+1=3 A[3,2].l:=1 |
| 3 | 3 | - | - | - |
| 4 | 1 | [3] | 3 | A[4,1].d:=A[3,1].d+1=3 A[4,1].l:=3 |
| 4 | 2 | [3] | 3 | A[4,2].d:=A[3,2].d+1=4 A[4,2].l:=3 |
| 4 | 3 | [3] | 3 | - |
| 4 | 4 | - | - | - |
| 5 | 1 | [2,3,4] | 2 | A[5,1].d:=A[2,1].d+1=3 A[5,1].l:=1 |
| 5 | 1 | [2,3,4] | 3 | A[5,1].d~A[3,1].d+1 |
| 5 | 1 | [2,3,4] | 4 | A[5,1].d:=A[4,1].d+1=4 A[5,1].l:=4 |
| 5 | 2 | [2,3,4] | 2 | - |
| 5 | 2 | [2,3,4] | 3 | A[5,2].d:=A[3,2].d+1=4 A[5,2].l:=3 |
| 5 | 2 | [2,3,4] | 4 | A[5,2].d:=A[4,2].d+1=5 A[5,2].l:=4 |
| 5 | 3 | [2,3,4] | 2 | A[5,3].d:=A[2,3].d+1=4 A[5,3].l:=2 |
| 5 | 3 | [2,3,4] | 3 | - |
| 5 | 3 | [2,3,4] | 4 | A[5,3].d~A[4,3].d+1 |
| 5 | 4 | [2,3,4] | 2 | A[5,4].d:=A[2,4].d+1=5 A[5,4].l:=2 |
| 5 | 4 | [2,3,4] | 3 | A[5,4].d~A[3,4].d+1 |
| 5 | 4 | [2,3,4] | 4 | - |
| 5 | 5 | [2,3,4] | 2 | A[5,5].d:=A[2,5].d+1=6 A[5,5].l:=2 |
| 5 | 5 | [2,3,4] | 3 | A[5,5].d~A[3,5].d+1 |
| 5 | 5 | [2,3,4] | 4 | A[5,5].d~A[4,5].d+1 |

После этой прокрутки массив A по полю d имеет вид:

| A[i,j].d | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 3 | 4 |
| 2 | 2 | 1 | 3 | 4 | 5 |
| 3 | 2 | 3 | 1 | 0 | 4 |
| 4 | 3 | 4 | 0 | 1 | 5 |
| 5 | 4 | 5 | 4 | 5 | 6 |

а по полю l («*» обозначено неопределенное значение):

| A[i,j].l | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | * | 1 | 1 | 3 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 3 | 4 |
| 3 | 1 | 1 | 1 | * | 2 |
| 4 | 3 | 3 | * | 3 | 2 |
| 5 | 4 | 4 | 2 | 2 | 2 |

Попробуем по этой таблице вывести путь для нашего примера с помощью следующей процедуры:

```
procedure Way(i,j:integer);
 begin
   if (i=1) and (j=1) then writeln(name[1])
    else if i>=j then begin
                                writeln(name[i]);
                                Way(A[i,j].l,j);
                               end
                    else begin
                                Way(i,A[i,j].l);
                                writeln(name[j]);
                               end;
 end;
```

Получаем цепочку: **Way[5,5]** →*Way[2,5]* →*Way[2,4]* →*Way[2,3]* →**Way[2,1]** →Way[1,1]. Жирным шрифтом выделены процедуры, в которых имя города выводится на входе в рекурсию, курсивом - на выходе из рекурсии. Цепочка имеет вид: 5 2 1 3 4 5 (индексы городов, напомним, что нумерация изменена для удобства на противоположную).

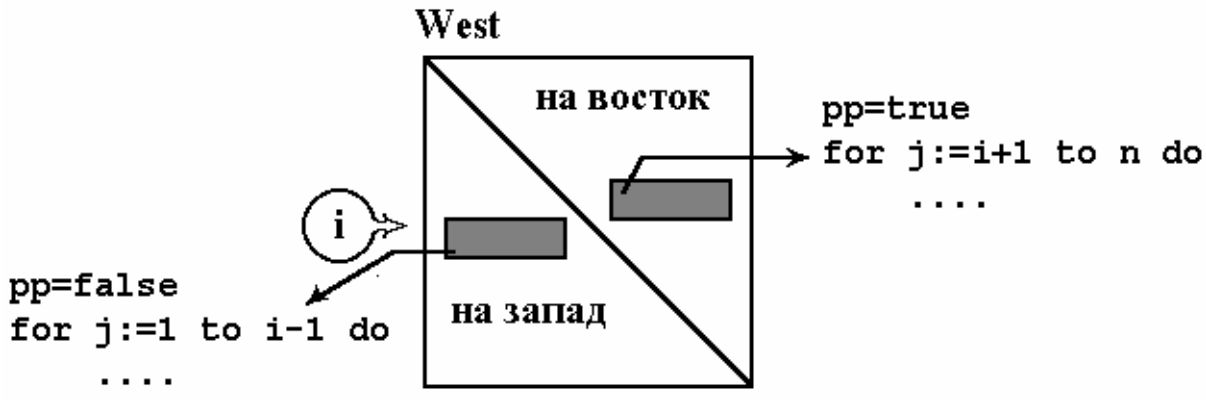
Осталось привести процедуру нахождения A (ее фрагмент).

```
...
FillChar(A,SizeOf(A),0);
A[1,1].d:=1;
for i:=2 to n do
 for j:=1 to i do
  if (i<>j) or (i=n) then
   for k∈Q_i do
          if ((k<>j) or (j=1)) and (A[k,j].d<>0) and (A[k,j].d+1>A[i,j].d) then
            begin
                  A[i,j].d+A[k,j].d+1;A[j,i].d:=A[i,j].d;
                  A[i,j].l:=k;A[j,i].l:=A[i,j].l;
            end;
```

Рассмотрим решения задачи методом полного перебора вариантов. Для этого требуется ряд дополнительных структур данных, а именно:

```
New:array[1..max] of boolean;
way_r:array[1..2*max] of byte;
```

Первый массив необходим для хранения признака - посещался или нет город (New[i]=true - города с номером i нет в маршруте). Во втором и третьем массивах запоминаются по принципу стека текущий и наилучший маршруты соответственно. Для работы со стеками необходимы указатели - переменные yk и yk_max. Логика перебора поясняется на нижеприведенном рисунке. Ищем путь из города с номером i.




Фрагмент основной программы.

```
begin
...
          FillChar(New,SizeOf(New),true);
          yk:=1;Way[yk]:=1;pp:=true;yk_max:=0;
          rec(1);
```

```
...
(* Вывод результата *)
...
end.
```

И процедура поиска решения.

```
procedure rec(i:byte);
 var j,pn,pv:byte;
 begin
   if i=n then pp:=false; (* меняем направление *)
   if (i=1) and not pp then begin
     (* вернулись в первый город *)
     if yk>yk_max then begin (* запоминаем решение *) yk_max:=yk;
                                                                  way_r:=way;
                                  end;
    pp:=true; (* продолжаем перебор *)
  end;
(* по  направлению определяем номера  просматриваемых городов*)
if pp then begin pn:=i+1; pv:=n end
 else begin pn:=1; pv:=i-1 end;
for j:=pn to pv do begin if West[i,j] and New[j] then begin
 (* в город с номером j летают самолеты  из города i и город j еще не посещался *)
 (* включаем город j в маршрут *)
 Inc(yk);Way[yk]:=j;New[j]:=false;
 rec(j);
 (* исключаем город j из маршрута *)
 New[j]:=true;Way[yk]:=0;Dec(yk);
 (* если исключается самый восточный город (n),  то меняем направление          *)
  if j=n then pp:=true;
 end;
end;
```

Следует отметить, что при этой реализации упрощается вывод результата.

```
writeln(n);
writeln(yk_max-1);
for i:=1 to yk_max do writeln(name(Way_r[i]]);
```

Целесообразно сравнение  решений при больших значениях n по времени выполнения.

## 2.3. Метод «решета»

Идея метода. Решето представляет собой метод комбинаторного программирования, который рассматривает конечное множество и исключает все элементы этого множества, не представляющие интереса. Он является логическим дополнением к процессу поиска с возвратом (backtrack), который перечисляет все элементы множества решений,  представляющие интерес.

### 2.3.1. Решето Эратосфена

Эратосфен - греческий математик, 275-194 гг. до нашей эры. Классическая задача поиска простых чисел в интервале [2..N] - инструмент для обсуждения метода.  Объяснение строится на основе следующего рисунка и фрагмента логики.

```
Const N= ...;{верхняя граница интервала чисел}
Type Number=Set of 2..N;{решето, N<256}
```

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
2 3 **4** 5 **6** 7 **8** 9 **10** 11 **12** 13 **14** 15 **16** 17 **18** 19 **20** 21 **22** 23 **24** 25 **26** 27 **28** 29 **30** 31 **32**
Числа, выделенные «жирным» шрифтом, отбрасываются.
2 3 **4** 5 **6** 7 **8** *9* **10** 11 **12** 13 **14** *15* **16** 17 **18** 19 **20** *21* **22** 23 **24** 25 **26** *27* **28** 29 **30** 31 **32**
Числа, выделенные курсивом, отбрасываются.

```
Var S:Number;
procedure Poisk(var S:Number);
 var i,j:2..N;
 begin
          S:=[2..N];
          for i:=2 to N div 2 do
           if i in S then begin
            j:=i+i;
```

```
        while j<=N do begin S:=S-[j];j:=j+i;end;
       end;
end;
```

Логическим развитием задачи является ее решение при N больших 256. В этом случае необходимо ввести массив с множественным типом данных элементов.

## 2.3.2. Быки и коровы

(модификация задачи А.А. Суханова)

Компьютер и ребенок играют в следующую игру. Ребенок загадывает последовательность из четырех (не обязательно различных) цветов, выбранных из шести заданных. Для удобства будем обозначать цвета цифрами от 1 до 6. Компьютер должен отгадать последовательность, используя информацию, которую он получает из ответов ребенка.

Компьютер отображает на экране последовательность, а ребенок должен ответить (используя для ввода ответа клавиатуру) на два вопроса:

- сколько правильных цветов на неправильных местах;
- сколько правильных цветов на правильных местах.

Пример. Предположим, что ребенок загадал последовательность 4655. Один из возможных способов отгадать последовательность такой:

| Компьютер | Ответ ребенка | |
|---|---|---|
| 1234 | 1 | 0 |
| 5156 | 2 | 1 |
| 6165 | 2 | 1 |
| 5625 | 1 | 2 |
| 5653 | 1 | 2 |
| 4655 | 0 | 4 |

Написать программу, которая всегда отгадывает последовательность не более чем за шесть вопросов, в худшем случае - за десять.

Правильный выбор структур данных - это уже половина решения. Итак, структуры данных и процедура их инициализации.

```
Const Pmax=6*6*6*6;
Type Post=String[4];
Var     A:array[1..Pmax] of Post;
        B:array[1..Pmax] of boolean;
        cnt:integer;{счетчик числа ходов}
        ok:boolean;{найдено решение}
procedure Init;
 var i,j,k,l:integer;
 begin
 for i:=1 to 6 do
  for j:=1 to 6 do
   for k:=1 to 6 do
    for l:=1 to 6 do A[(i-1)*216+(j-1)*36+(k-1)*6+l]:=
        Chr(i+Ord('0'))+Chr(j+Ord('0'))+Chr(k+Ord('0'))+Chr(k+Ord('0'));
 for i:=1 to Pmax do B[i]:=true;
 cnt:=0;ok:=false;
end;
```

Поясним на примере идею решета. Пусть длина последовательности равна двум, а количество цветов - четырем. Ребенок загадал 32, а компьютер спросил 24. Ответ ребенка 1 0, фиксируем его в переменных kr (1) и bk(0).

| A | B | B (после анализа bk) | B (после анализа kr) | Результат |
|---|---|---|---|---|
| 11 | true | | false | False |
| 12 | true | | | **True** |
| 13 | true | | false | False |
| 14 | true | false | | False |
| 21 | true | false | | false |
| 22 | true | false | | False |
| 23 | true | false | | False |
| 24 | true | false | | False |
| 31 | true | | false | False |
| 32 | true | | | **True** |
| 33 | true | | false | False |
| 34 | true | false | | False |

| | | | | |
|---|---|---|---|---|
| 41 | true | | | **True** |
| 42 | true | | false | False |
| 43 | true | | | **True** |
| 44 | true | false | | False |

Итак, было 16 возможных вопросов, после первого осталось четыре - работа решета. Функция, реализующая анализ элемента массива A, по значениям переменных kr и bk, имеет вид:

```
function Pr(a,b:Post;kr,bk:integer):boolean;
 var i,x:integer;
 begin
{проверка по “быкам”}
        x:=0;
        for i:=1 to 4 do if a[i]=b[i] then inc(x);
        if x<>bk then begin Pr:=false;exit;end;
{проверка по “коровам”}
        x:=0;
        for i:=1 to 4 do if (a[i]<>b[i]) and (Pos(b[i],a)<>0) then inc(x);
        if x<>kr then begin Pr:=false;exit;end;
        Pr:=true;
 end;
```

Логика - “сделать отсечение” по значению хода h.

```
procedure Hod(h:Post);
 var i,kr,bk:integer;
 begin
        inc(cnt);write(cnt:2,’.’,h,’-’);
        readln(kr,bk);
        if bk=4 then begin ok:=true;<вывод результата>;end
         else for i:=1 to Pmax do if B[i] and Not Pr(A[i],h,kr,bk) then B[i]:=false;
 end;
```

Общая логика.

```
.....
begin
        clrscr;
        init;
        hod(‘1223’);
        while not ok do begin
```

| выбор очередного хода из A |
|---|

```
                hod(h);
         end;
end.
```

Дальнейшая работа с задачей требует ответа на следующие вопросы:

- Зависит ли значение cnt от выбора первого хода? Установить экспериментальным путем.
- Как логика выбора очередного хода влияет на результат игры? Исследовать. Например, выбрать тот элемент A (соответствующий элемент B должен быть равен true), в котором количество несовпадающих цифр максимально.

## 2.4. Задачи

1. Какое наименьшее число ферзей можно расставить на доске так, чтобы они держали под боем все ее свободные поля? Модификация задачи. Найти расстановку ферзей, которая одновременно решает задачу для досок 9*9, 10*10 и 11*11.

2. Расставить на доске N*N(N≤12) N ферзей так, чтобы наибольшее число ее полей оказалось вне боя ферзей.

3. Расставить на доске как можно больше ферзей так, чтобы при снятии любого из них появлялось ровно одно неатакованное поле.

4. За какое наименьшее число ходов ферзь может обойти все поля доски 8*8?

5. Расставить на доске 8*8 максимальное число ферзей так, чтобы каждый из них нападал ровно на p(p≤2) ферзей. (Ответ. При p=1 десять, а при p=2 четырнадцать.)

6. Задача о коне Аттилы (“Трава не растет там, где ступил мой конь!”). На шахматной доске стоят белый конь и черный король. Некоторые поля доски считаются “горящими”. Конь должен дойти до неприятельского короля, повергнуть его и вернуться на исходное место. При этом ему запрещено становиться как на горящие поля, так и на поля, которые уже пройдены.

7. Магараджа - это фигура, которая объединяет в себе ходы коня и ферзя. Для доски 10*10 найти способ расстановки 10 мирных магараджей.

8. Пронумеровать позиции в матрице(таблице) размером 5*5 следующим образом. Если номер i ($1 \le i \le 25$) соответствует позиции с координатами (x,y), вычисляемыми по одному из следующих правил:

- $(z,w)=(x\pm3,y)$;
- $(z,w)=(x,y\pm3)$;
- $(z,w)=(x\pm2,y\pm2)$.

Требуется:

- написать программу, которая последовательно нумерует позиции матрицы 5*5 при заданных координатах позиции, в которой поставлен номер 1 (результаты должны быть представлены в виде заполненной матрицы);
- вычислить число всех возможных расстановок номеров для всех начальных позиций, расположенных в правом верхнем треугольнике матрицы, включая ее главную диагональ.

9. Прямоугольники. N прямоугольников различных цветов располагаются на белом прямоугольном листе бумаги, имеющем размеры A см в ширину и B см в длину. Стороны прямоугольников параллельны краям листа, а сами прямоугольники не выходят за пределы листа. В результате образуются различные одноцветные фигуры. Если два прямоугольника одного цвета имеют хотя бы одну общую точку, то они являются частями одной фигуры. Задача состоит в вычислении площади каждой из видимых фигур для каждого цвета. A и B - четные положительные целые числа, не превосходящие 30.

Начало системы координат находится в центре листа, а оси параллельны краям листа.

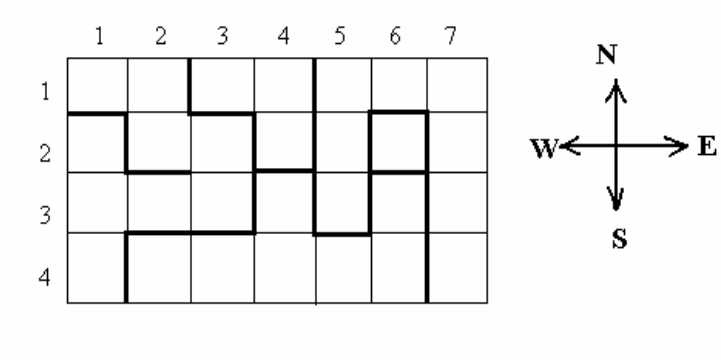
Наборы данных для нескольких тестов записаны во входном файле следующим образом. A, B и N находятся в первой строке каждого набора данных и разделены пробелом. В каждой из следующих N строк находятся:

- целочисленные координаты точки, в которую помещена левая нижняя вершина прямоугольника;
- за ними следуют целочисленные координаты точки, в которую помещена правая верхняя вершина прямоугольника;
- затем следует цвет прямоугольника, заданный целым числом от 1 до 64, белый цвет представлен числом 1.

Порядок строк соответствует порядку, в котором прямоугольники размещались на листе от первого до последнего. Наборы данных для различных тестов разделены пустой строкой.

Необходимо написать программу, которая вычисляет площадь каждой одноцветной фигуры.

10. Замок. На рисунке изображен план замка.




Написать программу, которая определяет:

- количество комнат в замке;
- площадь наибольшей комнаты;
- какую стену в замке следует удалить, чтобы получить комнату наибольшей площади.

Замок условно разделен на m*n клеток ($m \le 50$, $n \le 50$). Каждая такая клетка может иметь от 0 до 4 стен.

План замка содержится во входном файле в виде последовательности чисел, по одному числу, характеризующему каждую клетку.

- В начале файла расположено число клеток в направлении с севера на юг и число клеток в направлении с запада на восток.
- В последующих строках каждая клетка описывается числом p ($0 \le p \le 15$). Это число является суммой следующих чисел: 1 (если клетка имеет западную стену), 2 (северную), 4 (восточную), 8 (южную). Внутренняя стена считается принадлежащей обеим клеткам. Например, южная стена в клетке (1,1) также является северной стеной в клетке (2,1).
- Замок содержит по крайней мере две комнаты.

Пример.

```
4
7
11 6 11 6 3 10 6
7 9 6 13 5 15 5
1 10 12 7 13 7 5
13 11 10 8 10 12 13
```

В выходном файле должны быть три строки. В первой строке содержится число комнат, во второй - площадь наибольшей комнаты (измеряется количеством клеток). Третья строка содержит три числа, определяющих удаляемую стену: номер строки, номер столбца клетки, содержащей удаляемую стену и положение этой стены в клетке (N - север, W - запад, S - юг, E - восток).

11. Структура некоторых биологических объектов представляется последовательностью их составляющих. Эти составляющие обозначаются заглавными буквами. Биологи

интересуются разложением длинной последовательности в более короткие последовательности. Эти короткие последовательности называются примитивами. Говорят, что последовательность **S** может быть образована из данного множества примитивов **P**(их количество $1 \le N \le 100$), если существует n примитивов $p_1,...,p_n$ в **P**, таких, что их конкатенация (сцепление) $p_1...p_n$ равняется **S**. При конкатенации примитивы $p_1,...,p_n$ (их длины не превышают 20 символов) записываются последовательно без разделительных пробелов. Некоторые примитивы могут встречаться в конкатенации более одного раза, и не обязательно все примитивы должны быть использованы.

Например, последовательность ABABACABAAB может быть образована из множества примитивов {A, AB, BA, CA, BBC}.

Первые **K** символов строки **S** называют префиксом строки **S** длины **K**.

Написать программу, которая для заданных множества примитивов **P** и последовательности **T (**ее длина не меньше 1 и не больше 500000) определяет длину максимального префикса последовательности **T**, который может быть образован из множества примитивов **P**.

# Глава 3. АЛГОРИТМЫ НА ГРАФАХ

## 3.1. Представление графа в памяти компьютера

Определим граф как конечное множество вершин V и набор E неупорядоченных и упорядоченных пар вершин и обозначим G=(V,E). Мощности множеств V и E будем обозначать буквами N и M. Неупорядоченная пара вершин называется ребром, а упорядоченная пара - дугой. Граф, содержащий только ребра, называется неориентированным; граф, содержащий только дуги, - ориентированным, или орграфом. Вершины, соединенные ребром, называются смежными. Ребра, имеющие общую вершину, также называются смежными. Ребро и любая из его двух вершин называются инцидентными. Говорят, что ребро (u, v) соединяет вершины u и v. Каждый граф можно представить на плоскости множеством точек, соответствующих вершинам, которые соединены линиями, соответствующими ребрам. В трехмерном пространстве любой граф можно представить таким образом, что линии (ребра) не будут пересекаться.

*Способы описания.* Выбор соответствующей структуры данных для представления графа имеет принципиальное значение при разработке эффективных алгоритмов. При решении задач используются следующие четыре основных способа описания графа: матрица инциденций; матрица смежности; списки связи и перечни ребер. Мы будем использовать только два: матрицу смежности и перечень ребер.

Матрица смежности - это двумерный массив размерности N*N.

$$A[i,j]=\begin{cases} 1, & \textit{вершина с номером } i \textit{ смежна с вершиной с номером } j \\ 0, & \textit{вершина с номером } i \textit{ не смежна с вершиной с номером } j \end{cases}$$

Для хранения перечня ребер необходим двумерный массив R размерности M*2. Строка массива описывает ребро.

## 3.2. Поиск в графе

Множество алгоритмов на графах требует просмотра вершин графа. Рассмотрим их.

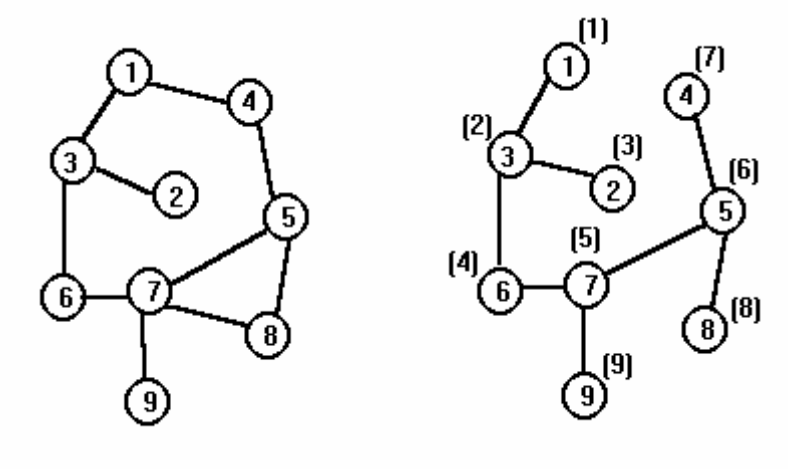
### 3.2.1. Поиск в глубину

*Идея метода.* Поиск начинается с некоторой фиксированной вершины v. Рассматривается вершина u, смежная с v. Она выбирается. Процесс повторяется с вершиной u. Если на очередном шаге мы работаем с вершиной q и нет вершин, смежных с q и не рассмотренных ранее (новых), то возвращаемся из вершины q к вершине, которая была до нее. В том случае, когда это вершина v, процесс просмотра закончен. Очевидно, что для фиксации признака, просмотрена вершина графа или нет, требуется структура данных типа:

```
Nnew : array[1..N] of boolean.
```

*Пример.*

Пусть граф описан матрицей смежности A. Поиск начинается с первой вершины. На левом рисунке приведен исходный граф, а на правом рисунке у вершин в скобках указана та очередность, в которой вершины графа просматривались в процессе поиска в глубину.




*Логика.*

```
procedure
Pg(v:integer);{Массивы
Nnew и A глобальные}

 var j:integer;
 begin
        Nnew[v]:=false; write(v:3);
        for j:=1 to N do if (A[v,j]<>0) and Nnew[j] then Pg(j);
 end;
```

Фрагмент основной логики.

```
...
FillChar(Nnew,SizeOf(Nnew),true);
for i:=1 to N do if Nnew[i] then Pg(i);
...
```

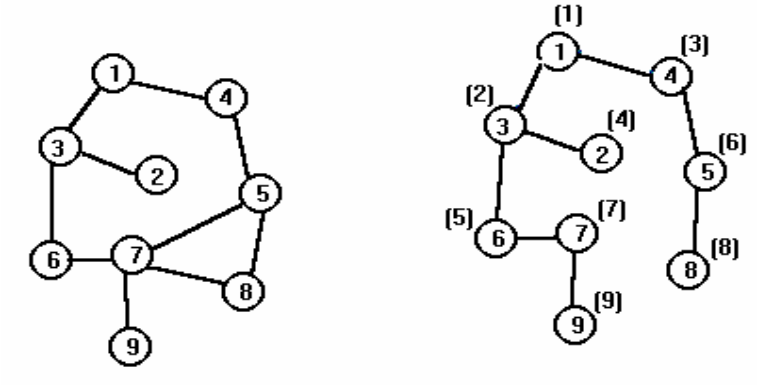
В силу важности данного алгоритма рассмотрим его нерекурсивную реализацию. Глобальные структуры данных прежние: A - матрица смежностей; Nnew - массив признаков. Номера просмотренных вершин графа запоминаются в стеке St, указатель стека - переменная yk.

```
procedure PG1(v:integer);
 var    St:array[1..N] of integer;yk:integer;t,j:integer;pp:boolean;
begin
 FillChar(St,SizeOf(St),0); yk:=0;
Inc(yk);St[yk]:=v;Nnew[v]:=false;
while yk<>0 do begin {пока стек не пуст}
        t:=St[yk];          {выбор “самой верхней“ вершины из стека}
        j:=0;pp:=false;
        repeat
                if (A[t,j+1] <>0) and Nnew[j+1] then pp:=true
                 else Inc(j);
        until pp or (j>=N); {найдена новая вершина или все вершины,
        связанные с данной вершиной, просмотрены}
        if pp then begin
                        Inc(yk);St[yk]:=j+1;Nnew[j+1]:=false;{добавляем в стек}
                end
         else Dec(yk); {“убираем” вершину из стека}
 end;
end;
```

## 3.2.2. Поиск в ширину

*Идея метода.* Суть (в сжатой формулировке) заключается в том, чтобы рассмотреть все вершины, связанные с текущей. Принцип выбора следующей вершины - выбирается та, которая была раньше рассмотрена. Для реализации данного принципа необходима структура данных “очередь”.

*Пример.* Исходный граф на левом рисунке. На правом рисунке рядом с вершинами в скобках указана очередность просмотра вершин графа.




Приведем процедуру реализации данного метода обхода вершин графа.

Логика просмотра вершин.

```
procedure PW(v:integer);
 var    Og:array[1..N] of 0..N;
        {очередь}
        yk1,yk2:integer;
        {указатели очереди, yk1 - запись; yk2 - чтение}
        j:integer;
  begin
        FillChar(Og,SizeOf(Og),0);yk1:=0;yk2:=0;{начальная инициализация}
        Inc(yk1);Og[yk1]:=v;Nnew[v]:=false;{в очередь - вершину v}
        while yk2<yk1 do begin    {пока очередь не пуста}
         Inc(yk2);v:=Og[yk2];write(v:3);{“берем” элемент из очереди}
         for j:=1 to N do {просмотр всех вершин, связанных с вершиной v}
          if (A[v,j]<>0) and Nnew[j] then begin{если вершина ранее не просмотрена}
                Inc(yk1);Og[yk1]:=j;Nnew[j]:=false;{заносим ее в очередь}
               end;
        end;
  end;
```

## 3.3. Деревья

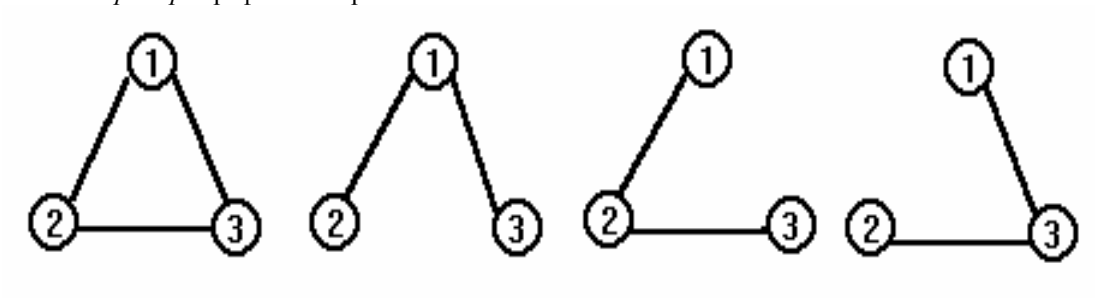
### 3.3.1. Основные понятия. Стягивающие деревья

Деревом называют произвольный связный неориентированный граф без циклов. Его можно определить и по-другому: связный граф, содержащий N вершин и N-1 ребер, либо граф, в котором каждая пара вершин соединена одной и только одной простой цепью. Для произвольного связного неориентированного графа G=<V,E> каждое дерево <V,T>, где T⊆E называют стягивающим деревом (каркасом, остовом). Ребра такого дерева называют ветвями, а остальные ребра графа - хордами.

*Примечание.* Понятие цикла будет дано позже. В данной теме ограничимся его интуитивным пониманием.

Число различных каркасов полного связного неориентированного помеченного графа с N вершинами было найдено впервые Кэли и равно $N^{(N-2)}$.

*Пример.* Граф и его каркасы.




*Поиск стягивающего дерева (каркаса).* Рассмотрим два алгоритма нахождения каркасов (одного), основанных на методах просмотра графа поиском в глубину и в ширину. Граф описывается матрицей смежности A, массив Nnew (*array[1..N] of boolean*) служит для хранения признаков. Значение Nnew[i], равное true, говорит о том, что вершина с номером i еще не просмотрена. Для хранения ребер, образующих каркас, будем использовать структуру данных Tree (*array[1..2,1..N] of integer*).

Известно, что как при поиске в глубину, так и при поиске в ширину просматриваются все вершины связного графа. Использование массива Nnew обеспечивает “подключение” очередного ребра к каркасу без образования циклов. Действительно, цикл образуется, если мы соединяем две просмотренные вершины. Но в нашем случае “подключается” ребро, соединяющее просмотренную вершину с непросмотренной. И наконец, по самой логике методов поиска в глубину и в ширину мы строим связный граф.

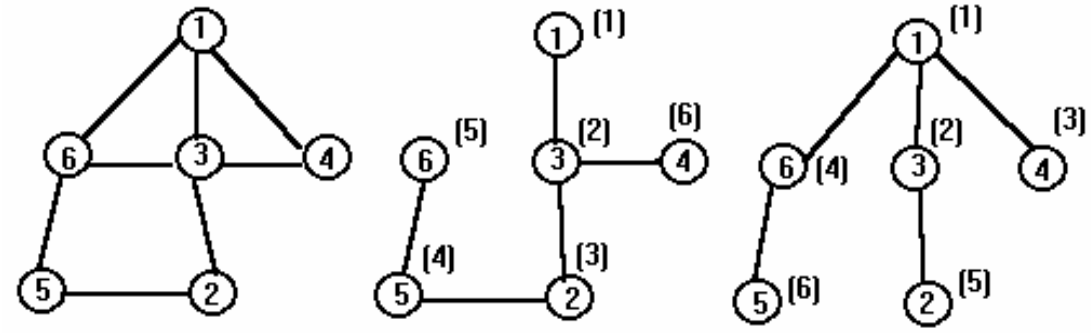
Построение каркаса для связного графа методом поиска в глубину.

```
procedure Tree_Depth(v:integer); {рекурсивная процедура}
        {A, Nnew, Tree, yk - глобальные структуры данных}
 var i:integer;
        begin
          Nnew[v]:=false;
          for i:=1 to N do if (A[v,i]<>0) and Nnew[i] then begin
                                        {добавляем ветвь в каркас}
                                         Inc(yk);Tree[1,yk]:=v;Tree[2,yk]:=i;
                                         Tree_Depth(i);
                                        end;
        end;
{фрагмент основной логики}
begin
        FillChar(Nnew,SizeOf(Nnew),true);yk:=0;
        Tree_Depth(1); {строим от первой вершины}
        <вывод каркаса>;
end.
```

Построение каркаса для связного графа методом поиска в ширину.

```
procedure Tree_Width(v:integer);
{A, Tree, yk - глобальные структуры данных}
 var  Nnew:array[1..N] of boolean;
        Turn:array[1..N] of integer;yr,yw,i:integer;
 begin
  FillChar(Nnew,SizeOf(Nnew),true);
  FillChar(Turn,SizeOf(Turn),0);yr:=0;yw:=0;
  Inc(yw); Turn[yw]:=v;Nnew[v]:=false;
  while yw<>yr do begin
        Inc(yr);v:=Turn[yr];
        for i:=1 to N do  if (A[v,i]<>0) and Nnew[i] then begin
                                        Inc(yw);Turn[yw]:=i;Nnew[i]:=false;
                                        Inc(yk);Tree[1,yk]:=v;Tree[2,yk]:=i;
                                       end;
   end;
 end;
```

*Пример*. Граф и его каркасы, построенные методами поиска в глубину и в ширину. В круглых скобках указана очередность просмотра вершин графа при соответствующем поиске.




### 3.3.2. Порождение всех каркасов графа

*Дано.* Связный неориентированный граф G=<V,E>. *Найти.* Все каркасы графа.

Каркасы не запоминаются. Их необходимо перечислить. Для порождения очередного каркаса ранее построенные не привлекаются, используется только последний. Множество всех каркасов графа G делится на два класса: содержащие выделенное ребро <v,u> и не содержащие. Каркасы последовательно строятся в графах $G_{<v,u>}$ и G-<v,u>. Каждый из графов $G_{<v,u>}$ и G-<v,u> меньше, чем G. Последовательное применение этого шага уменьшает графы до тех пор, пока не будет построен очередной каркас, либо графы станут несвязными и не имеющими каркасов.

Для реализации идеи построения каркасов графа G используются следующие структуры данных:

- очередь - Turn (*array[1..N] of integer*) с нижним (down) и верхним (up) указателями;
- массив признаков Nnew (см. предыдущее занятие);

- список ребер, образующих каркас - Tree (см. предыдущее занятие);
- число ребер в строящемся каркасе - numb.

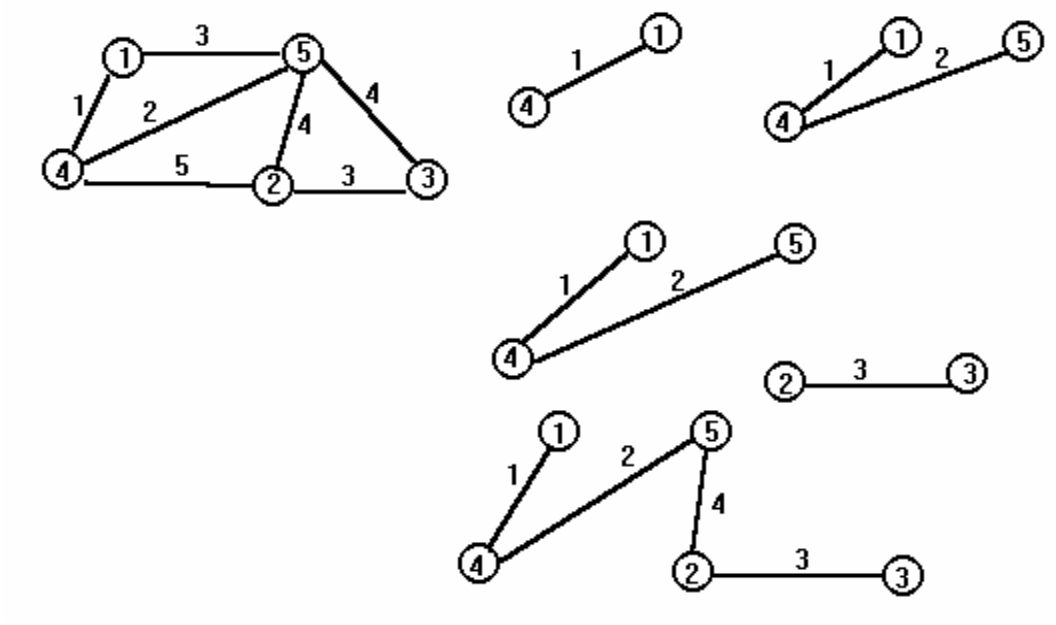
Начальное значение переменных:

```
                .....
FillChar(Nnew,SizeOf(Nnew),true);
FillChar(Tree,SizeOf(Tree),0);
Nnew[1]:=false;
{*в очередь заносим первую вершину*}
Turn[1]:=1; down:=1;up:=2; numb:=0;
                .....
Procedure Solve(v,q:integer);
{*v - номер вершины, из которой выходит ребро *}
{*q - номер вершины, начиная с которой следует искать очередное
   ребро каркаса *}
 var j:integer;
 begin
        if down>=up then exit;
        j:=q;
        while (j<=N) and (numb<N-1) do begin {*просмотр ребер,
        выходящих из вершины с номером v *}
          if (A[v,j]<>0) and Nnew[j] then begin {*есть ребро, и вершины с
                номером  j еще нет в каркасе *}
           {*включаем ребро в каркас*}
            Nnew[j]:=false;
             inc(numb);Tree[1,numb]:=v;Tree[2,numb]:=j;
             {*включаем вершину с номером j в очередь*}
             Turn[up]:=j;inc(up);
              Solve(v,j+1); {*продолжаем построение каркаса*}
             {*исключаем ребро из каркаса*}
              dec(up);Nnew[j]:=true; dec(numb);
            end;
           inc(j);
         end;
 if numb=N-1 then begin <вывод каркаса>; exit end;
{*все ребра, выходящие из вершины с номером v, просмотрены*}
{* переходим к следующей вершине из очереди и так до тех пор,
   пока не будет построен каркас *}
 if j=N+1 then begin     inc(down);
                                  Solve(Turn[down],1);
                                  dec(down);
                  end;
end; {*Solve*}
```

### 3.3.3. Каркас минимального веса. Метод Краскала

*Дано.* Связный неориентированный граф G=<V,E>. Ребра имеют вес. Граф описывается перечнем ребер с указанием их веса. Массив P (*array[1..3,1..N*(N-1) div 2] of integer*). *Результат.* Каркас с минимальным суммарным весом Q=<V,T>, где T⊂E.

*Пример.* Граф и процесс построения каркаса по методу Краскала.




Шаг 1. Начать с вполне несвязного графа G, содержащего N вершин.
Шаг 2. Упорядочить ребра графа G в порядке неубывания их весов.
Шаг 3. Начав с первого ребра в этом перечне, добавлять ребра в графе Q, соблюдая условие: добавление не должно приводить к появлению цикла в Q.
Шаг 4. Повторять шаг 3 до тех пор, пока число ребер в Q не станет равным N-1. Получившееся дерево является каркасом минимального веса.

Какие структуры данных требуются для реализации шага 3? Введем массив меток вершин графа (Mark:*array[1..N] of integer*). Начальные значения элементов массива равны номерам соответствующих вершин (Mark[i]=i для i от 1 до N). Ребро выбирается в каркас в том случае, если вершины, соединяемые им, имеют разные значения меток. В этом случае циклы не образуются. Для примера, приведенного выше, процесс изменения Mark показан в таблице.

| Номер итерации | Ребро | Значения элементов Mark |
|---|---|---|
| начальное значение | - | [1,2,3,4,5] |
| 1 | <1,4> | [1,2,3,1,5] |
| 2 | <4,5> | [1,2,3,1,1] |
| 3 | <2,3> | [1,2,2,1,1] |
| 4 | <2,5> | [1,1,1,1,1] |

И логика этого фрагмента.

```
procedure Chang_Mark(l,m:integer);
 {массив Mark глобальный}
 var i,t:integer;
 begin
        if m<l then begin t:=l;l:=m;m:=t end;
        for i:=1 to N do if Mark[i]=m then Mark[i]:=l;
 end;
```

Фрагмент основной части логики.

```
program Tree1;
 const N=..;
 var   P:array[1..3,1..N*(N-1) div 2] of integer;
        Mark:array[1..N] of integer;k,i,t:integer;
        M:integer;{количество ребер графа}
 begin
 <ввод описания графа - массив P>;
 <сортировка массива P по значениям весов ребер>;
 for i:=1 to N do Mark[i]:=i;
 k:=0;t:=M;
 while k<N-1 do begin i:=1;
  while (i<=t) and (Mark[P[1,i]]=Mark[P[2,i]])  and (P[1,i]<>0) do Inc(i);
  Inc(k);
  <вывод или запоминание ребра каркаса>;
```

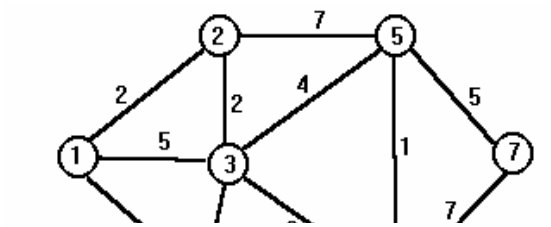
```
 Change_Mark(Mark[P[1,i]],Mark[P[2,i]]);
 end;
end;
```

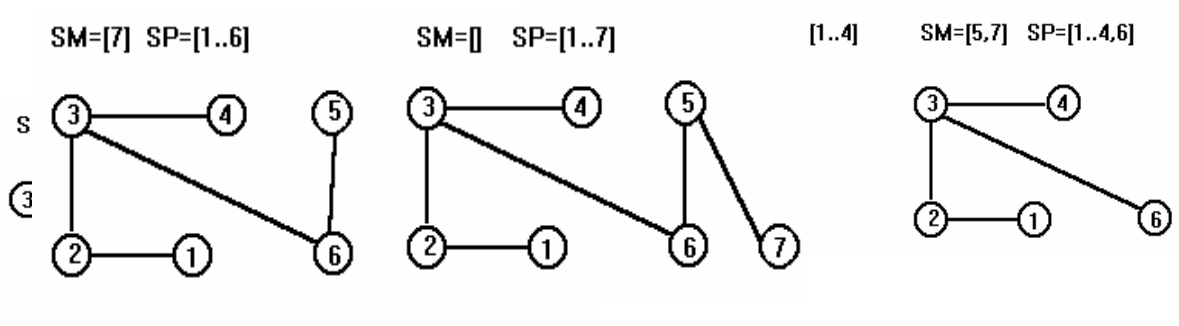
### 3.3.4. Каркас минимального веса. Метод Прима

*Дано.* Связный неориентированный граф G=<V,E>. Ребра имеют вес. Граф описывается матрицей смежности A (*array[1..N,1..N] of integer*). Элемент матрицы, не равный нулю, определяет вес ребра. *Результат.* Каркас с минимальным суммарным весом Q=<V,T>, где T⊂E.

Отличие от метода Краскала заключается в том, что на каждом шаге строится дерево, а не ациклический граф. Для реализации метода необходимы две величины множественного типа SM и SP (*set of 1..N*). Первоначально значением SM являются все вершины графа, а SP пусто. Если ребро <i,j> включается в T, то номера вершин i и j исключаются из SM и добавляются в SP.

*Пример.* Граф и его каркас.




Логика построения каркаса.

```
procedure Tree2;
        {A - глобальная структура данных}
 var  SM,SP:set of 1..N; min,i,j,l,k,t:integer;
 begin
        min:=maxint; SM:=[1..N];SP:=[];{включаем первое ребро в каркас}
        for i:=1 to N-1 do  for j:=i+1 to N do
          if (A[i,j]<min) and (A[i,j]<>0) then begin min:=A[i,j];l:=i;t:=j; end;
         SP:=[l,t];SM:=SM-[l,t];   <выводим или запоминаем ребро <l,t>>;
        {основной цикл}
        while SM<>[] do begin
         min:=maxint;l:=0;t:=0;
         for i:=1 to N do if Not(i in SP) then for j:=1 to N do
                                    if (j in SP) and (A[i,j]<min) and (A[i,j]<>0) then
                                       begin min:=A[j,k]; l:=i;t:=j;end;
         SP:=SP+[l];SM:=SM-[l];  <выводим или запоминаем ребро <l,t>>;
        end;
  end;
```

## 3.4. Связность

### 3.4.1. Достижимость

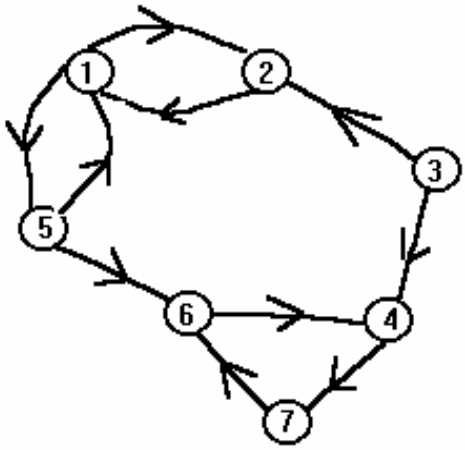
Путем (или ориентированным маршрутом) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Простой путь - это путь, в котором каждая дуга используется не более одного раза.

Элементарный путь - это путь, в котором каждая вершина используется не более одного раза.

Если существует путь из вершины графа v в вершину u, то говорят, что u достижима из v. Матрицу достижимостей R определим следующим образом:

$$R[v,u]=\begin{cases}1, & \text{если вершина с номером u достижима из v}\\ 0, & \text{при недостижимости}\end{cases}$$




Множество R(v) - это множество таких вершин графа G, каждая из которых может быть достигнута из вершины v. Обозначим через Γ(v) множество таких вершин графа G, которые достижимы из v с использованием путей длины 1. $\Gamma^2(v)$ - это Γ(Γ(v)), то есть с использованием путей длины 2 и так далее. В этом случае:

$R(v)=\{v\}\cup\Gamma(v)\cup\Gamma^2(v)\cup...\cup\Gamma^p(v)$.

При этом p - некоторое конечное значение, возможно, достаточно большое.

*Пример*(для рисунка).

$R(1)=\{1\}\cup\{2,5\}\cup\{1,6\}\cup\{2,5,4\}\cup\{1,6,7\}=\{1,2,4,5,6,7\}$

Выполняя эти действия для каждой вершины графа, мы получаем матрицу достижимостей R.

```
        procedure Reach; {формирование матрицы R, глобальной переменной}
{исходные данные - матрица смежности A, глобальная переменная}
 var    S,T:set of 1..N;i,j,l:integer;
begin
        FillChar(R,SizeOf(R),0);
         for i:=1 to N do begin {достижимость из вершины с номером i}
                T:=[i];
                repeat
                        S:=T;
                        for l:=1 to N do if l in S then{по строкам матрицы A,
                        принадлежащим множеству S}
                         for j:=1 to N do if A[l,j]=1 then T:=T+[j];
                until S=T;{если T не изменилось, то найдены все вершины
        графа, достижимые из вершины с номером i}
                for j:=1 to N do if j in T then R[i,j]:=1;
        end;
end;
```

Матрицу контрдостижимостей Q определим следующим образом:

$$Q[v,u]=\begin{cases}1, & \text{если из u можно достигнуть v}\\ 0, & \text{при недостижимости}\end{cases}$$

Множеством Q(v) графа G является множество таких вершин, что из любой его вершины можно достигнуть вершину v. Из определения следует, что столбец v матрицы Q совпадает со строкой v матрицы R, то есть $Q=R^t$, где $R^t$ - матрица, транспонированная к матрице достижимостей R.

Для примера на рисунке матрицы A, R и Q имеют вид:

| | | | | | |
|---|---|---|---|---|---|
| | 0 1 0 0 1 0 0 | | 1 1 0 1 1 1 1 | | 1 1 1 0 1 0 0 |
| | 1 0 0 0 0 0 0 | | 1 1 0 1 1 1 1 | | 1 1 1 0 1 0 0 |
| | 0 1 0 1 0 0 0 | | 1 1 1 1 1 1 1 | | 0 0 1 0 0 0 0 |
| A= | 0 0 0 0 0 0 1 | R= | 0 0 0 1 0 1 1 | Q= | 1 1 1 1 1 1 1 |
| | 1 0 0 0 0 1 0 | | 1 1 0 1 1 1 1 | | 1 1 1 0 1 0 0 |
| | 0 0 0 1 0 0 0 | | 0 0 0 1 0 1 1 | | 1 1 1 1 1 1 1 |
| | 0 0 0 0 0 1 0 | | 0 0 0 1 0 1 1 | | 1 1 1 1 1 1 1 |

Дополнения.

1. Граф называется транзитивным, если из существования дуг (v,u) и (u,t) следует существование дуги (v,t). Транзитивным замыканием графа G=(V,E) является граф Gz=(V,E∪E'), где E' - минимально возможное множество дуг, необходимых для того, чтобы граф Gz был транзитивным. Разработать программу для нахождения транзитивного замыкания произвольного графа G.

2. R(v) - множество вершин, достижимых из v, а Q(u) - множество вершин, из которых можно достигнуть u. Определить, что представляет из себя множество R(v)∩Q(u). Разработать программу нахождения этого типа множеств.

### 3.4.2. Определение связности

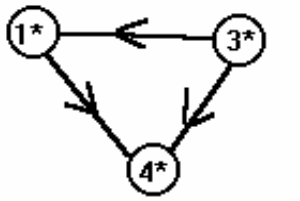
*Определения*. Неориентированный граф G связен, если существует хотя бы один путь в G между каждой парой вершин i и j. Ориентированный граф G связен, если неориентированный граф, получающийся из G путем удаления ориентации ребер, является связным. Ориентированный граф сильно связен, если для каждой пары вершин i и j существует по крайней мере один ориентированный путь из i в j и по крайней мере один из j в i. Максимальный связный подграф графа G называется связной компонентой графа G. Максимальный сильно связный подграф называется сильно связной компонентой.

Рассмотрим алгоритм нахождения сильных компонент графа. Идея достаточна проста. Для вышеприведенного примера значение R(1)={1}∪{2,5}∪{6}∪{4}∪{7}={1,2,4,5,6,7}, а Q(1)={1} ∪ {2,5} ∪{3}. Пересечение этих множеств дает множество C(1)={1,2,5} вершин графа G, образующих сильную компоненту, которой принадлежит вершина графа с номером 1. Продолжим рассмотрение: R(3)={1,2,3,4,5,6,7}, Q(3)={3} и C(3)={3}; R(4)={4}∪{7}∪{6}={4,6,7} и Q(4)={4}∪{6}∪{7}={4,6,7} и C(4)={4,6,7}. Итак, мы нашли сильные компоненты графа G. Граф G*=(V*,E*) определим так: каждая его вершина представляет множество вершин некоторой сильной компоненты графа G, дуга (i*, j*) существует в G* тогда и только тогда, когда в G существует дуга (i,j), такая, что i принадлежит компоненте, соответствующей вершине i*, а j - компоненте, соответствующей вершине j*. Граф G* называется конденсацией графа G.

Дополнения.

1. Доказать, что в конденсации графа не содержится циклов.

2. Доказать, что в ориентированном графе каждая вершина i может принадлежать только одной сильной компоненте.




3. В графе есть множество вершин P, из которых достижима любая вершина графа и которое является минимальным в том смысле, что не существует подмножества в P, обладающего таким свойством достижимости. Это множество вершин называется базой графа. Показать, что в P нет двух вершин, которые принадлежат одной и той же сильной компоненте графа G.

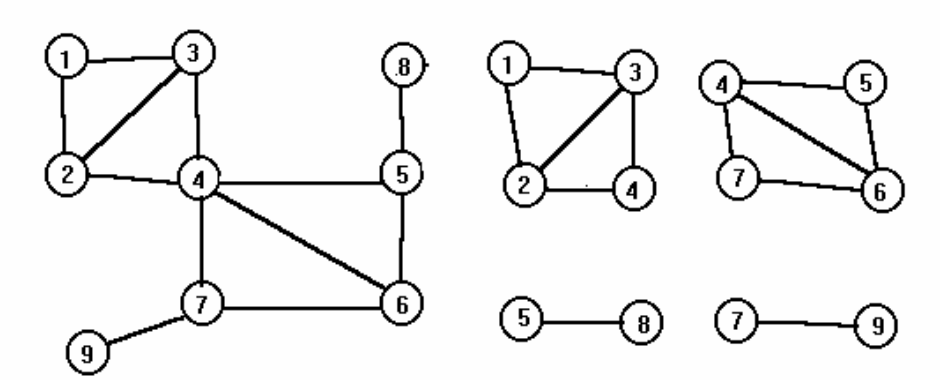
Показать, что любые две базы графа G имеют одно и то же число вершин.

Разработать программу нахождения базы графа. Схема решения. База P* конденсации G* графа G состоит из таких вершин графа G*, в которые не заходят ребра. Следовательно, базы графа G можно строить так: из каждой сильной компоненты графа G, соответствующей вершине базы P* конденсации G*, надо взять по одной вершине - это и будет базой графа G.

### 3.4.3. Двусвязность

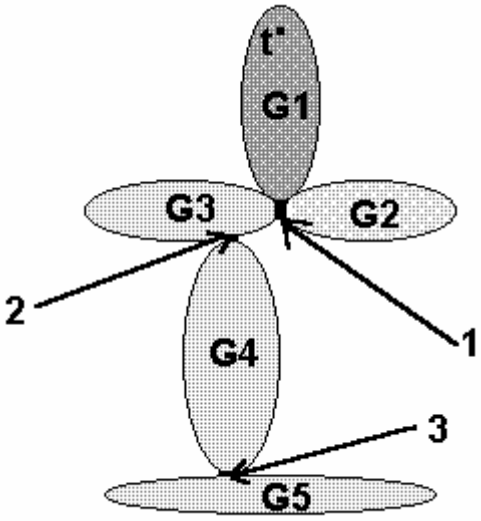
Иногда недостаточно знать, что граф связен. Может возникнуть вопрос, насколько "сильно связен" связный граф. Например, в графе может существовать вершина, удаление которой вместе с инцидентными ей ребрами разъединяет оставшиеся вершины. Такая вершина называется точкой сочленения, или разделяющей вершиной. Граф, содержащий точку сочленения, называется разделимым. Граф без точек сочленения называется двусвязным или неразделимым. Максимальный двусвязный подграф графа называется двусвязной компонентой или блоком.

*Пример.* Разделимый граф и его двусвязные компоненты. Точки сочленения вершины с номерами 4, 5 и 7.

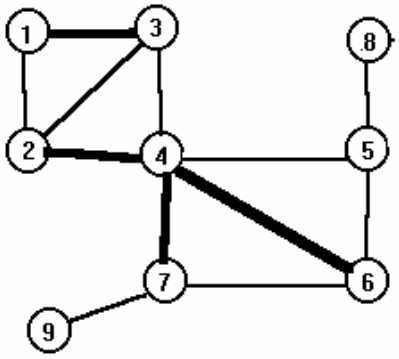



Точку сочленения можно определить иначе. Вершина t является точкой сочленения, если существуют вершины u и v, отличные от t, такие, что каждый путь из u в v (предполагаем, что существует по крайней мере один) проходит через вершину с номером t.

Наша задача - найти точки сочленения и двусвязные компоненты графа. Основная идея. Есть двусвязные компоненты G1, G2, G3, G4 и G5 и точки сочленения 1, 2, 3. Осуществляем поиск в глубину из вершины t, принадлежащей G1. Мы можем перейти из G1 в G2, проходя через вершину 1. Но по свойству поиска в глубину все ребра G2 должны быть пройдены до того, как мы вернемся в 1. Поэтому G2 состоит в точности из ребер, которые проходятся между заходами в вершину 1. Для других чуть сложнее. Из G1 попадаем в G3, затем в G4 и G5. Предысторию процесса прохождения ребер будем хранить в стеке. Тогда при возвращении в G4 из G5 через вершину 3 все ребра G5 будут на верху стека. После их удаления, то есть вывода двусвязной компоненты из стека, на верху стека будут ребра G4, и в момент прохождения вершины 2 мы можем их опять вывести. Таким образом, если распознать точки сочленения, то, применяя поиск в глубину и храня ребра в стеке в той очередности, в какой они проходятся, можно определить двусвязные компоненты. Ребра, находящиеся на верху стека в момент обратного прохода через точку сочленения, образуют двусвязную компоненту.




Итак, проблема с точками сочленения. Рассмотрим граф. В процессе просмотра в глубину все ребра разбиваются на те, которые составляют дерево (каркас), и множество обратных ребер. Обратные ребра выделены на рисунке более жирными линиями.




Что нам это дает? Пусть очередность просмотра вершин в процессе поиска в глубину фиксируется метками в массиве Num. Для нашего примера Num - (1,2,3,4,5,6,7,9,8). Если мы рассматриваем обратное ребро (v,u), и v не предок u, то информацию о том, что Num[v] больше Num[u], можно использовать для пометки вершин v и u как вершин, принадлежащих одной компоненте двусвязности. Массив Num использовать для этих целей нельзя, поэтому введем другой массив Lowpg и постараемся пометить вершины графа, принадлежащие одной компоненте двусвязности одним значением метки в этом массиве. Первоначальное значение метки совпадает со значением соответствующего элемента массива Num. При нахождении обратного ребра (v,u) естественной выглядит операция: Lowpg[v]:=Min(Lowpg[v],Num[u]) - изменения значения метки вершины v, так как вершины v и u из одной компоненты двусвязности. К этой логике необходимо добавить смену значения метки у вершины v ребра (v,u) на выходе из просмотра в глубину в том случае, если значение метки вершины u меньше, чем метка вершины v (Lowpg[v]:= Min(Lowpg[v],Lowpg[u])). Для нашего примера массив меток Lowpg имеет вид: (1,1,1,2,4,4,4,9,8). Осталось определить момент вывода компоненты двусвязности. Мы рассматриваем ребро (v,u), и оказывается, что значение Lowpg[u] больше или равно значению Num[v]. Это говорит о том, что при просмотре в глубину между вершинами v и u не было обратных ребер. Вершина v - точка сочленения, и необходимо вывести очередную компоненту двусвязности, начинающуюся с вершины v.

Итак, логика.

```
procedure Dvy(v,p:integer);{вершина p - предок вершины v}
{массивы A, Num, Lowpg и переменная nm - глобальные}
 var u:integer;
 begin
        Inc(nm);Num[v]:=nm;Lowpg[v]:=Num[v];
        for u:=1 to N do
         if A[v,u]<>0 then if Num[u]=0 then begin
                                <сохранить ребро (v,u) в стеке>;
                                 Dvy(u,v);
                                 Lowpg[v]:=Min(Lowpg[v],Lowpg[u]);
                                {функция, определяющая минимальное из двух
        чисел }
                                 if Lowpg[u]>=Num[v] then <вывод
                                        компоненты>;
                                end
                                else if (u<>p) and (Num[v]>Num[u]) then begin
                                {u не совпадает с предком вершины v}
                                        <сохранить ребро (v,u) в стеке>;
                                        Lowpg[v]:=Min(Lowpg[v],Num[u]);
                                        end;
end;
```

Фрагмент основной логики:

```
....
FillChar(Num,SizeOf(Num),0);
FillChar(Lowpg,SizeOf(Lowpg),0);
nm:=0;
for v:=1 to N do if Num[v]=0 then Dvy(v,0);
....
```

Дополнение. Мостом графа G называется каждое ребро, удаление которого приводит к увеличению числа связных компонент графа. Разработать программу нахождения всех мостов графа. Покажите, что мосты графа должны быть в каждом каркасе графа G. Каким образом знание мостов графа может изменить (ускорить) логику нахождения всех его каркасов?

## 3.5. Циклы

### 3.5.1. Эйлеровы циклы

*Определение.* Эйлеров цикл — это такой цикл, который проходит ровно один раз по каждому ребру.

*Теорема.* Связный неориентированный граф G содержит эйлеров цикл тогда и только тогда, когда число вершин нечетной степени равно нулю.

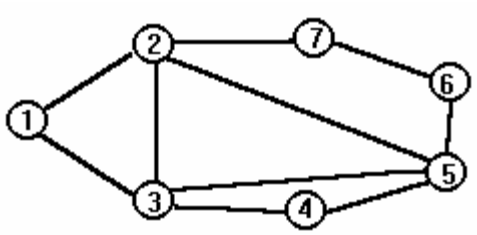
Не все графы имеют эйлеровы циклы, но если эйлеров цикл существует, то это означает, что , следуя вдоль этого цикла, можно нарисовать граф на бумаге, не отрывая от нее карандаша. Дан граф G, удовлетворяющий условию теоремы. Требуется найти эйлеров цикл. Используется просмотр графа методом поиска в глубину, при этом ребра удаляются. Порядок просмотра (номера вершин) запоминается. При обнаружении вершины, из которой не выходят ребра, мы их удалили, ее номер записывается в стек, и просмотр продолжается от предыдущей вершины. Обнаружение вершин с нулевым числом ребер говорит о том, что найден цикл. Его можно удалить, четность вершин (количество выходящих ребер ) при этом не изменится. Процесс продолжается до тех пор, пока есть ребра. В стеке после этого будут записаны номера вершин графа в порядке, соответствующем эйлерову циклу.

*Логика.*

```
procedure Search(v:integer);
{*глобальные переменные*}
{*A - матрица смежности, Cv - стек*}
{* yk - указатель стека*}
 var j:integer;
 begin
                for j:=1 to N do if A[v,j]<>0 then begin
                                A[v,j]:=0;A[j,v]:=0;Search(j)
                                end;
        Inc(yk);
        Cv[yk]:=v;
```

end;

Пример графа и содержимое стека Cv после работы процедуры Search.

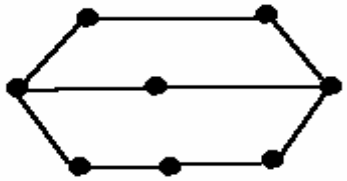



Cv: 1 3 5 6 7 2 5 4 3 2 1

### 3.5.2. Гамильтоновы циклы

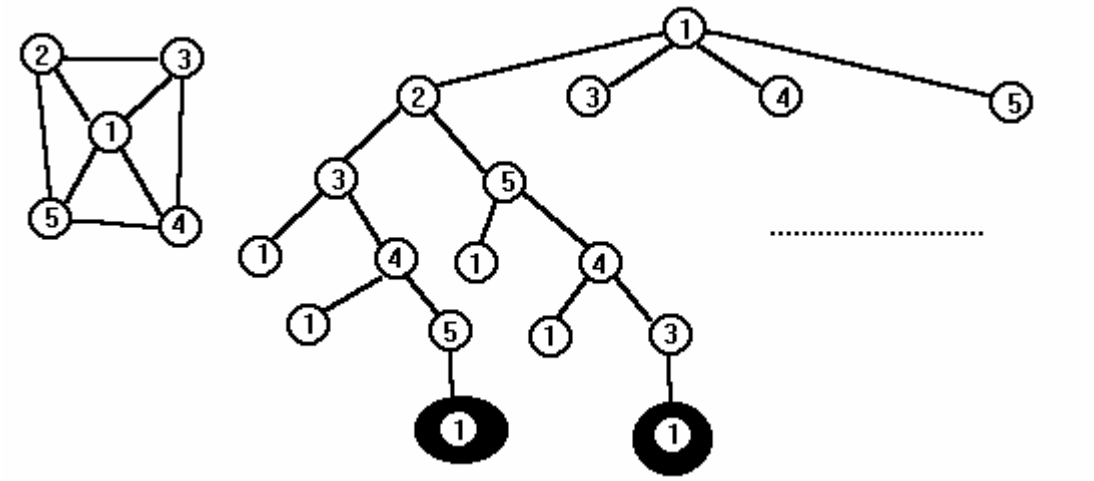
*Определение.* Граф называется гамильтоновым, если в нем имеется цикл, содержащий каждую вершину этого графа. Сам цикл также называется гамильтоновым.

Не все связные графы гамильтоновы.



Дан связный неориентированный граф G. Требуется найти все гамильтоновы циклы графа, если они есть.

Метод решения - перебор с возвратом (backtracking). Начинаем поиск решения, например, с первой вершины графа. Предположим, что уже найдены первые k компонент решения. Рассматриваем ребра, выходящие из последней вершины. Если есть такие, что идут в ранее не просмотренные вершины, то включаем эту вершину в решение и помечаем ее как просмотренную. Получена (k+1) компонента решения. Если такой вершины нет, то возвращаемся к предыдущей вершине и пытаемся найти ребро из нее, выходящее в другую вершину. Решение получено при просмотре всех вершин графа и возможности достичь из последней первой вершины. Решение (цикл) выводится, и продолжается процесс нахождения следующих циклов. Пример графа и часть дерева, показывающего механизм работы данного метода.




*Логика решения*.

```
procedure Gm(k:integer); {* k - номер итерации*}
{*глобальные переменные*}
{* A - матрица смежности, St - массив для хранения порядка просмотра вершин графа, Nnew - массив признаков: вершина просмотрена или нет*}
 var j,v:integer;
 begin
        v:=St[k-1]; {*номер последней вершины*}
        for j:=1 to N do if (A[v,j]<>0)  then {*есть ребро между
                                                вершинами с номерами v и j*}
                                  if (k=N+1) and (j=1) then <вывод цикла>
                                   else if Nnew[j] then {*вершина не
                                          просмотрена*}
                                          begin St[k]:=j;Nnew[j]:=false;
                                                Gm(k+1);Nnew[j]:=true;end;
 end;
```

Фрагмент основной логики.

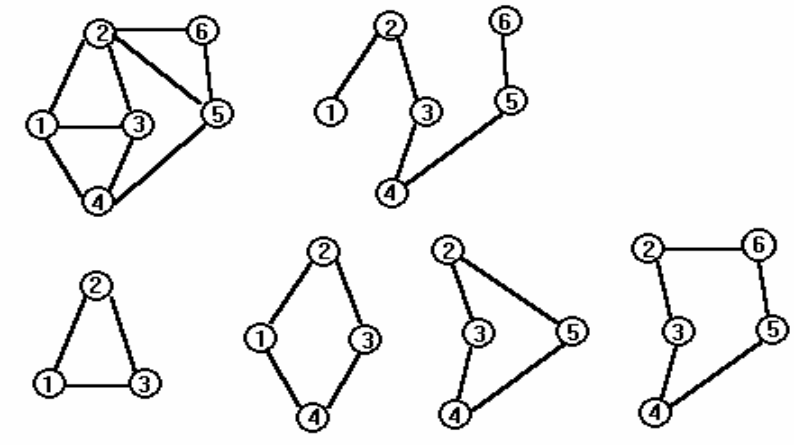
```
.....
St[1]:=1;Nnew[1]:=false; Gm(2]);
```

.....

### 3.5.3. Фундаментальное множество циклов

Каркас (V,T) связного неориентированного графа G=<V,E> содержит N-1 ребро, где N - количество вершин G. Каждое ребро, не принадлежащее T, то есть любое ребро из E-T, порождает в точности один цикл при добавлении его к T. Такой цикл является элементом фундаментального множества циклов графа G относительно каркаса T. Поскольку каркас состоит из N-1 ребра, в фундаментальном множестве циклов графа G относительно любого каркаса имеется M-N+1 циклов, где M - количество ребер в G.

Пример графа, его каркаса и множества фундаменталь-ных циклов.




Поиск в глубину является естественным подходом, используемым для нахождения фундаментальных циклов. Строится каркас, а каждое обратное ребро порождает цикл относительно этого каркаса. Для вывода циклов необходимо хранить порядок обхода графа при поиске в глубину (номера вершин) - массив St, а для определения обратных ребер вершины следует “метить” (массив Gnum) в той очередности, в которой они просматриваются. Если для ребра <v,j> оказывается, что значение метки вершины с номером j меньше, чем значение метки вершины с номером v, то ребро обратное и найден цикл.

Начальная инициализация переменных.

```
        ......
num:=0;yk:=0;
for j:=1 to N do Gnum[j]:=0;
        .......
```

*Логика*.

```
procedure Circl(v:integer);
{глобальные переменные: A - матрица смежности графа; St - массив для хранения номеров вершин графа в том порядке, в котором они используются при построении каркаса; yk - указатель записи в массив St; Gnum - для каждой вершины в соответствующем элементе массива фиксируется номер шага (num), на котором она просматривается при поиске в глубину}
 var j:integer;
 begin
        Inc(yk);St[yk]:=v;Inc(num);
        Gnum[v]:=num;
        for j:=1 to N do if A[v,j]<>0 then
                                if Gnum[j]=0 then Circl[j] {вершина j не
                                        просмотрена}
                                {j не предыдущая вершина при просмотре, и
        она была просмотрена ранее}
                                 else if (j<>St[yk-1]) and (Gnum[j]<Gnum[v]) then
                                        <вывод цикла из St>;
        Dec(yk);
 end;
```

Дополнения.

Название “фундаментальный” связано с тем, что каждый цикл графа может быть получен из циклов этого множества. Для произвольных множеств A и B определим операцию симметрической разности $A \oplus B=(A \cup B) \setminus (A \cap B)$. Известно [9], что произвольный цикл графа G можно однозначно представить как симметрическую разность некоторого числа фундаментальных циклов. Однако не при всех операциях симметрической разности получаются циклы (вырожденный случай). Исследовательская работа - разработать программу нахождения всех циклов графа.

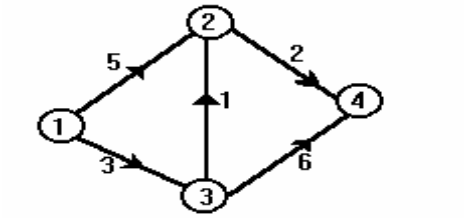
## 3.6. Кратчайшие пути

### 3.6.1. Постановка задачи. Вывод пути

*Постановка задачи*. Дан ориентированный граф G=<V,E>, веса дуг - A[i,j] (i,j=1..N, где N - количество вершин графа), начальная и конечная вершины - s, t∈V. Веса дуг записаны в матрице смежности A, если вершины i и j не связаны дугой, то A[i,j]=∞. Путь между s и t оценивается $\sum_{i,j\in\text{пути}} A[i,j]$. Найти путь с минимальной оценкой.

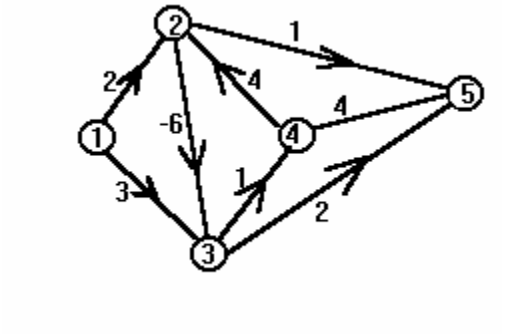
*Пример*.

Кратчайший путь из 1 в 4 проходит через 3-ю и 2-ю вершины и имеет оценку 6.




Особый случай - контуры с отрицательной оценкой.

*Пример.*




При s=1 и t=5, обходя контур 3→4→ 2→ 3 достаточное число раз, можно сделать так, что оценка пути между вершинами 1 и 5 будет меньше любого целого числа. Оценку пути назовем его весом или длиной. Будем рассматривать только графы без контуров отрицательного веса.

Нам необходимо найти кратчайший путь, то есть путь с минимальным весом, между двумя вершинами графа. Эта задача разбивается на две подзадачи: сам путь и значение минимального веса . Обозначим ее через D[s,t]. Неизвестны алгоритмы, определяющие только D[s,t], все они определяют оценки от вершины s до всех остальных вершин графа. Определим D, как *array[1..n] of integer*. Предположим, что мы определили значения элементов массива D - решили вторую подзадачу. Определим сам кратчайший путь. Для s и t существует такая вершина v, что D[t]=D[v]+A[v,t]. Запомним v (например, в стеке) . Повторим процесс поиска вершины u, такой, что D[v]=D[u]+A[u,v], и так до тех пор, пока не дойдем до вершины с номером s. Последовательность t, v, u, ..., s дает кратчайший путь.

```
procedure Way(s,t:integer);
{D, A - глобальные структуры данных}
 var     v,u:integer;
         Stack - локальная структура данных для хранения номеров вершин;
         procedure Print;
         {выводит содержимое Stack}
          begin
                  ...
          end;
begin
         <почистить Stack>;
         <занести вершину с номером t в Stack>;v:=t;
         while v<>s do begin
                  u:=<номер вершины, для которой D[v]=D[u]+A[u,v]>;
                  <занести вершину с номером v в Stack>;
                  v:=u;
          end;
end;
```

Итак, путь при известном D находить мы умеем. Осталось научиться определять значения кратчайших путей, то есть элементы массива D. Идея всех известных алгоритмов заключается в следующем. По данной матрице весов A вычисляются первоначальные верхние оценки. А затем пытаются их улучшить до тех пор, пока это возможно. Поиск улучшения, например для D[v], заключается в нахождении вершин u, таких, что D[u]+A[u,v]<D[v]. Если такая вершина u есть, то значение D[v] можно заменить на D[u]+A[u,v].

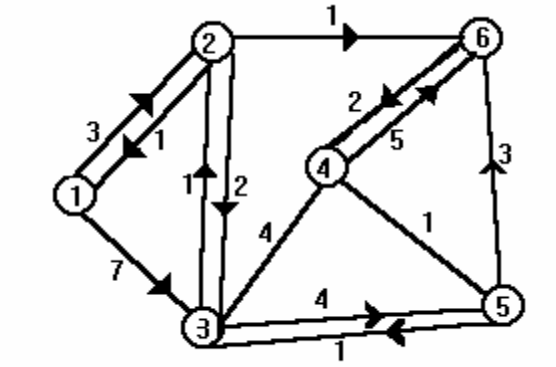
### 3.6.2. Алгоритм Дейкстры

*Дано.* Ориентированный граф G=<V,E>, s - вершина источник;
матрица смежности A (A:*array[1..n,1..n] of integer*); для любых u, v∈V вес дуги неотрицательный (A[u,v]≥0).*Результат.* Массив кратчайших расстояний - D.

В данном алгоритме формируется множество вершин T, для которых еще не вычислена оценка расстояние и, это главное, минимальное значение в D по множеству вершин, принадлежащих T, считается окончательной оценкой для вершины, на которой достигается этот минимум. С точки зрения здравого

смысла этот факт достаточно очевиден. Другой “заход” в эту вершину возможен по пути, содержащему большее количество дуг, а так как веса неотрицательны, то и оценка пути будет больше. В книгах [11], [12] можно найти доказательство этого факта.

*Пример*




Его матрица смежности:

$$A=\begin{pmatrix} \infty & 3 & 7 & \infty & \infty & \infty \\ 1 & \infty & 2 & \infty & \infty & 1 \\ \infty & 1 & \infty & 4 & 4 & \infty \\ \infty & \infty & \infty & \infty & 1 & 5 \\ \infty & \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & \infty & 2 & \infty & \infty \end{pmatrix}$$

В таблице приведена последовательность шагов (итераций) работы алгоритма. На первом шаге минимальное значение D достигается на второй вершине. Она исключается из множества T, и улучшение оценки до оставшихся вершин (3,4,5,6) ищется не по всем вершинам, а только от второй.

| № итерации | D[1] | D[2] | D[3] | D[4] | D[5] | D[6] | T |
|---|---|---|---|---|---|---|---|
| 1 | 0 | **3** | 7 | ∞ | ∞ | ∞ | [2,3,4,5,6] |
| 2 | 0 | 3 | 5 | ∞ | 11 | **4** | [3,4,5,6] |
| 3 | 0 | 3 | **5** | 6 | ∞ | 4 | [3,4,5] |
| 4 | 0 | 3 | 5 | **6** | 9 | 4 | [4,5] |
| 5 | 0 | 3 | 5 | 6 | **7** | 4 | [5] |

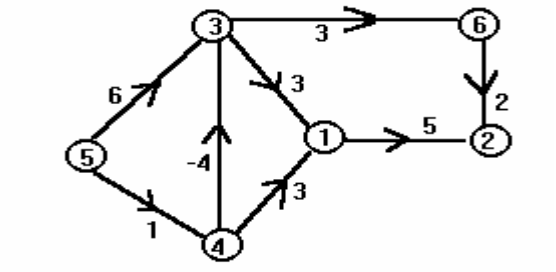
```
procedure Distance;
        {A, D, s - глобальные}
 var    v,u: integer;T:set of 1..N;
  begin{инициализация D}
        for v∈V do D[v]:=A[s,v];
        D[s]:=0; T:=[1..N]-[s];
        {основной цикл}
        while T<>[ ] do begin
          u:=<то значение l, при котором достигается  min(D[l])>;
          T:=T-[u];                                                    l∈T
         for v∈T  do D[v]:=min(D[v],D[u]+A[u,v]);
       end;
  end;
```

Время работы алгоритма пропорционально $N^2$.

### 3.6.3. Пути в бесконтурном графе

*Дано.* Ориентированный граф G=<V,E> без контуров, веса дуг произвольны. *Результат.* Массив кратчайших расстояний(длин) D от фиксированной вершины до всех остальных. *Утверждение.* В произвольном бесконтурном графе вершины можно перенумеровать так, что для каждой дуги <i,j> номер вершины i будет меньше номера вершины j.

*Пример.*




Введем следующие структуры данных:

- массив NumIn, NumIn[i] определяет число дуг, входящих в вершину с номером i;
- массив Num, Num[i] определяет новый номер вершины i;
- массив Stack, для хранения номеров вершин, в которые заходит нулевое количество дуг. Работа с массивом осуществляется по принципу стека;
- переменная nm, текущий номер вершины.

Суть алгоритма. Вершина i, имеющая нулевое значение NumIn, заносится в Stack, ей присваивается текущее значение nm(запоминается в Num), и изменяются значения элементов NumIn для всех вершин, связанных с i. Процесс продолжается до тех пор, пока Stack не пуст. Трассировка работы алгоритма для нашего примера приведена в таблице.

| № итерации | NumIn | Num | Stack | Nm |
|---|---|---|---|---|
| начальная | [2,2,2,1,0,1] | [0,0,0,0,0,0] | [5] | 0 |
| 1 | [2,2,1,0,0,1] | [0,0,0,0,1,0] | [4] | 1 |
| 2 | [1,2,0,0,0,1] | [0,0,0,2,1,0] | [3] | 2 |
| 3 | [0,2,0,0,0,0] | [0,0,3,2,1,0] | [6,1] | 3 |
| 4 | [0,1,0,0,0,0] | [0,0,3,2,1,4] | [1] | 4 |
| 5 | [0,0,0,0,0,0] | [5,0,3,2,1,4] | [2] | 5 |
| 6 | [0,0,0,0,0,0] | [5,6,3,2,1,4] | [ ] | 6 |

```
procedure Change_Num;
        {A, Num - глобальные структуры данных}
 var    NumIn,Stack:array[1..N] of integer;
        i,j,u,nm,yk:integer;
 begin
        {инициализация данных}
        FillChar(NumIn,SizeOf(NumIn),0);
        for i:=1 to N do
         for j:=1 to N do if A[i,j]<>0 then Inc(NumIn[j]);
        nm:=0;yk:=0;
        for i∈V do if NumIn[i]=0 then begin Inc(yk);Stack[yk]:=i; end;
        {основной цикл}
        while yk<>0 do begin
          u:=Stack[yk];Dec[yk];Inc(nm);Num[u]:=nm;
          for i:=1 to n do
           if A[u,i]<>0 then  begin   Dec(NumIn[i]);
                                          if NumIn[i]=0 then begin
                                           Inc(yk); Stack[yk]:=i; end;
                                end;
        end;
 end;
```

Итак, пусть для графа G выполнено условие утверждения и нам необходимо найти кратчайшие пути (их длины) от первой вершины до всех остальных . Пусть мы находим оценку для вершины с номером i. Достаточно просмотреть вершины, из которых идут дуги в вершину с номером i. Они имеют меньшие номера, и оценки для них уже известны. Остается выбрать меньшую из них.

```
procedure Distance;
        {D, A - глобальные]
        var i,j:integer;
        begin
          D[1]:=0;
          for i:=2 to N do D[i]:=∞;
        {основной цикл}
         for i:=2 to N do
          for j:=1 to i-1 do if A[j,i]<>∞ then D[i]:=min(D[i],D[j]+A[j,i]);
        end;
```

Время работы алгоритма пропорционально $N^2$.

### 3.6.4. Кратчайшие пути между всеми парами вершин.

### Алгоритм Флойда

*Дано.* Ориентированный граф G=<V,E> с матрицей весов A(*array[1..N,1..N] of integer*). *Результат.* Матрица D кратчайших расстояний между всеми парами вершин графа и кратчайшие пути.

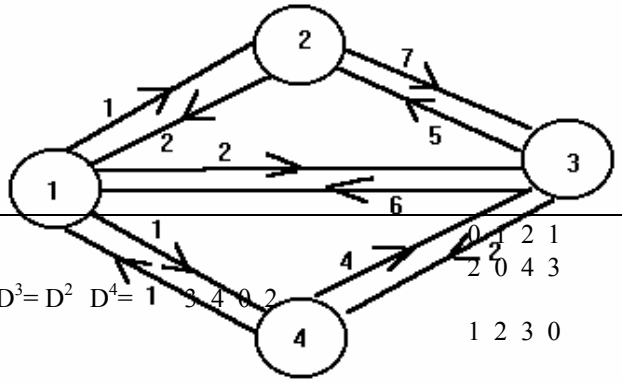
*Идея алгоритма*. Обозначим через $D^m[i,j]$ оценку кратчайшего пути из i в j с промежуточными вершинами из множества [1..m]. Тогда имеем: $D^0[i,j]:=A[i,j]$ и $D^{(m+1)}[i,j]=\min\{D^m[i,j],D^m[i,m+1]+D^m[m+1,j]\}$. Второе равенство требует пояснения. Пусть мы находим кратчайший путь из i в j с промежуточными вершинами из множества [1..(m+1)]. Если этот путь не содержит вершину (m+1), то $D^{(m+1)}[i,j]=D^m[i,j]$. Если же он содержит эту вершину, то его можно разделить на две части от i до (m+1) и от (m+1) до j.

Время работы алгоритма пропорционально $N^3$.

```
procedure Distance; {A, D - глобальные структуры данных}
```

```
var m,i,j:integer;
begin
        for i:=1 to N do  for j:=1 to N do D[i,j]:=A[i,j];
        for i:=1 to N do D[i,i]:=0;
        for m:=1 to N do  for i:=1 to N do for j:=1 to N do {основной
                                                        цикл}
        D[i,j]:=min{D[i,j],D[i,m]+D[m,j]};
end;
```

*Пример*




*Примечание*. Верхний индекс у D указывает номер итерации (значение m в процедуре Distance).

$$D^0=\begin{matrix}0&1&2&1\\2&0&7&\infty\\6&5&0&2\\1&\infty&4&0\end{matrix}\quad D^1=\begin{matrix}0&1&2&1\\2&0&4&3\\6&5&0&2\\1&2&3&0\end{matrix}\quad D^2=D^1\quad D^3=D^2\quad D^4=\begin{matrix}0&1&2&1\\2&0&4&3\\1&3&4&0\\1&2&3&0\end{matrix}$$

Расстояния между парами вершин дает D. Для вывода самих кратчайших путей введем матрицу M того же типа, что и D. Элемент M[i,j] определяет предпоследнюю вершину кратчайшего пути из i в j.

Процедура Distance претерпит небольшие изменения. В том случае, когда D[i,j] больше D[i,m]+D[m,j], изменяется не только D[i,j], но и M[i,j]. M[i,j] присваивается значение M[m,j]. Для нашего примера изменения M выглядят следующим образом.

Например, необходимо вывести кратчайший путь из 3-й вершины во 2-ю. Элемент M[3,2] равен 1,

$$M^0=\begin{matrix}1&1&1&1\\2&2&2&2\\3&3&3&3\\4&4&4&4\end{matrix}\quad M^1=\begin{matrix}1&1&1&1\\2&2&1&1\\3&3&3&3\\4&1&1&4\end{matrix}\quad M^2=M^1\quad M^3=M^2\quad M^4=\begin{matrix}1&1&1&1\\2&2&1&1\\4&1&3&3\\4&1&1&4\end{matrix}$$

поэтому смотрим на элемент M[3,1]. Он равен четырем. Сравниваем M[3,4] с 3-й. Есть совпадение, мы получили кратчайший путь: $3\rightarrow4\rightarrow1\rightarrow2$.

```
procedure All_Way(i,j:integer);{вывод пути между вершинами i и j}
        begin
                if M[i,j]=i then  if i=j write(i) else write(i,'-',j)
                 else begin All_Way(i,M[i,j]);All_Way(M[i,j],j);end;
        end;
```

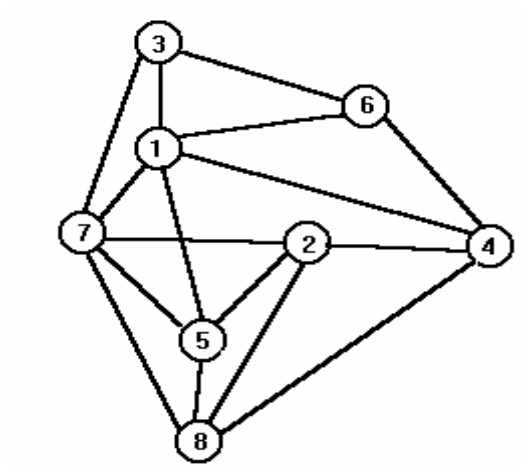
## 3.7. Независимые и доминирующие множества

Задача поиска подмножеств множества вершин V графа G, удовлетворяющих определенным условиям, свойствам, возникает достаточно часто.

### 3.7.1. Независимые множества

Дан неориентированный граф G=(V,E). Независимое множество вершин есть множество вершин графа G, такое, что любые две вершины в нем не смежны, то есть никакая пара вершин не соединена ребром. Следовательно, любое подмножество S, содержащееся в V, и такое, что пересечение S с множеством вершин смежных с S пусто, является независимым множеством вершин.

*Пример*.



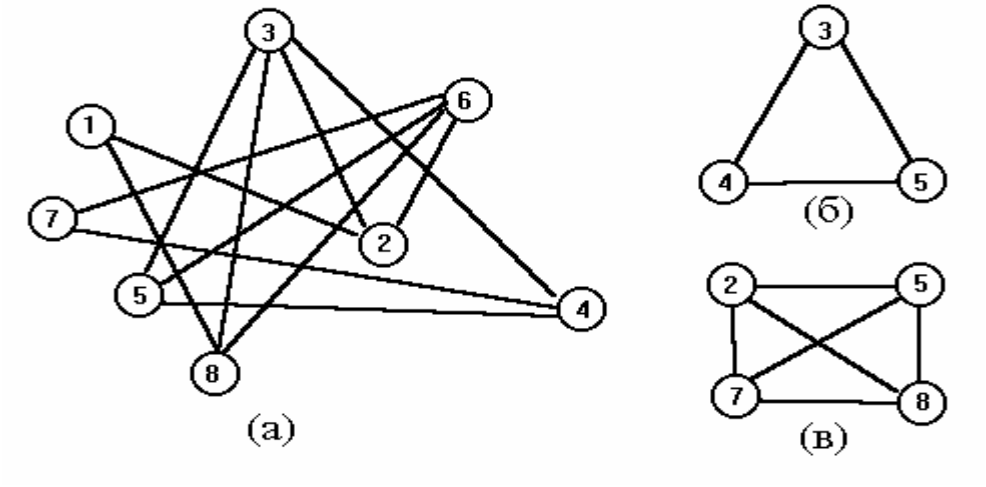

Множества вершин (1, 2), (3, 4, 5), (4, 7), (5, 6) - независимые. Независимое множество называется максимальным, когда нет другого независимого множества, в которое оно бы входило. Если Q является семейством всех независимых множеств графа G, то число

$$\alpha[G]=\max_{S\in Q}|S|$$

называется числом независимости графа G, а множество S*, на котором этот максимум достигается, называется наибольшим независимым множеством. Для нашего примера $\alpha[G]=3$, а S* есть (3, 4, 5).

Понятие, противоположное максимальному независимому множеству, есть максимальный полный подграф (клика). В максимальном независимом множестве нет смежных вершин, в клике все вершины попарно смежны. Максимальное независимое множество графа G соответствует клике графа G', где G' - дополнение графа G.

Для нашего примера дополнение G' приведено на следующем рисунке, клика графа G' соответствует максимальному независимому множеству графа G. Число независимости графа G' равно 4, максимальное независимое множество (2, 5, 7, 8), ему соответствует клика графа G.




### 3.7.2. Метод генерации всех максимальных независимых множеств графа

Задача решается перебором вариантов. "Изюминкой" является отсутствие необходимости запоминать генерируемые множества с целью проверки их на максимальность путем сравнения с ранее сформированными множествами. Идея заключается в последовательном расширении текущего независимого множества (k - номер шага или номер итерации в процессе построения). Очевидно, что если мы не можем расширить текущее решение, то найдено максимальное независимое множество. Выведем его и продолжим процесс поиска. Будем хранить текущее решение в массиве Ss (Ss:*array[1..N] of integer*), его первые k элементов определяют текущее решение. Логика вывода.

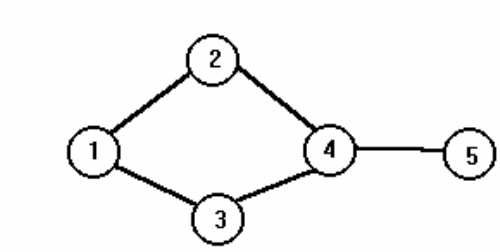
```
procedure Print(k:integer);
 var i:byte;
 begin
        writeln;for i:=1 to k do write(Ss[i],' ');
  end;
```

В процессе решения нам придется многократно рассматривать вершины графа, смежные с данной. При описании графа матрицей смежности - это просмотр соответствующей строки, при описании списками связи - просмотр списка. Упростим нахождение смежных вершин за счет использования нового способа описания графа. Используем множественный тип данных. Введем тип данных:

```
type Sset=set of 1..N; и переменную
var     A:array[1..N] of Sset;     .
```

Итак, чтобы определить вершины графа, смежные с вершиной i, необходимо просто вызвать соответствующий элемент массива A.

*Пример*.




A[1]=[2,3]
A[2]=[1,4]
A[3]=[1,4]
A[4]=[2,3,5]
A[5]=[4]

Основная сложность алгоритма в выборе очередной вершины графа. Введем переменную Gg для хранения номеров вершин - кандидатов на расширение текущего решения. Значение переменной формируется на каждом шаге k. Что является исходной информацией для формирования Gg? Очевидно, что некоторое множество вершин, свое для каждого шага (итерации) алгоритма. Логически правомерно разбить это множество вершин на не использованные ранее (Qp) и использованные ранее (Qm). Изменение значений Qp и Qm происходит при возврате на выбор k-го элемента максимального независимого множества. Мы выбирали на k шаге, например, вершину с номером i, и естественно исключение ее из Qp и Qm при поиске следующего максимального независимого множества. Кроме того, при переходе к шагу с номером (k+1) из текущих множеств Qp и Qm для следующего шага необходимо исключить вершины, смежные с вершиной i, выбранной на данном шаге (из определения независимого множества) и, разумеется, саму вершину i. Итак, общая логика.

```
procedure Find(k:integer; Qp,Qm:Sset);
 var    Gg:Sset;
        i:byte;
 begin
        if (Qp=[ ]) and (Qm=[ ]) then begin Print(k);exit end;
        {черный ящик А}
        <формирование множества кандидатов Gg для расширения текущего решения (k элементов в
массиве Ss) по значениям Qp и Qm>;
        i:=1;
        while i<=N do begin
                if i in Gg then begin
                 Ss[k]:=i;
                 Find(k+1,Qp-A[i]-[i],Qm-A[i]-[i]);
                {черный ящик Б}
                 <изменение Qp, Qm для этого уровня (значения k) и, соответственно,
изменение множества кандидатов Gg>;
                end;
          Inc(i);
        end;{while}
 end;{Find}
```

Продолжим уточнение логики. Нам потребуется простая функция подсчета количества элементов в переменных типа Sset.

```
        function Number(A:Sset):byte;
         var i, cnt:byte;
         begin
                cnt:=0;for i:=1 to N do if i in A then Inc(cnt);Number:=cnt;
         end;
```

Используем метод трассировки для того, чтобы найти дальнейшую логику уточнения решения. Будем делать разрывы таблицы для внесения пояснений в работу черных ящиков.

| k | Qp | Qm | Gg | Ss | Примечания |
|---|---|---|---|---|---|
| 1 | [1..5] | [ ] | [1..5] | (1) | Выбираем первую вершину |
| 2 | [4,5] | [ ] | [4,5] | (1,4) | Итак, первое уточнение черного ящика А. |

```
        if Qm<>[ ] then <черный ящик АА >
                        else Gg:=Qp;
```

Его суть в том, что если выбирать из ранее использованных вершин нечего, то множество кандидатов совпадает со значением Qp, и далее по логике мы “тупо” выбираем первую вершину из Qp. Переходим к следующему вызову процедуры Find.

| | | | | | |
|---|---|---|---|---|---|
| 3 | [ ] | [ ] | [ ] | (1,4) | Вывод первого максимального независимого множества и выход в предыдущую копию Find. |
| 2 | [5] | [4] | [5] | (1,5) | Исключаем вершину 4 из Qp и включаем ее в Qm. |

Продолжает работу цикл while процедуры Find. Выбираем следующую вершину - это вершина 5. И вызываем процедуры Find с другими значениями параметров.

| | | | | | |
|---|---|---|---|---|---|
| 3 | [ ] | [ ] | [ ] | (1,5) | Вывод второго максимального независимого множества. |
| 2 | [ ] | [4,5] | [ ] | | Цикл while закончен, выход в предыдущую копию |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | процедуры Find. |

Уточнение черного ящика Б. Первое: необходимо исключить вершину i из Qp и включить ее в Qm. Второе: следует откорректировать множество Gg. Выбор на этом шаге вершин, не смежных с i, приведет к генерации повторяющихся максимальных независимых множеств, поэтому следует выбирать вершины из пересечения множеств Qp и A[i]. Итак, черный ящик Б.

Qp:=Qp-[i];Qm:=Qm+[i];

if Number(Qp*A[i])<Number(Gg) then Gg:=Qp*A[i]⋙ Следующий шаг - выход в предыдущую версию Find, при этом значение k равно 1.

| | | | | | |
|---|---|---|---|---|---|
| 1 | [2..5] | [1] | [1..5] | | Однако после работы черного ящика Б имеем следующие значения переменных |
| 1 | [2..5] | [1] | [2..3] | (2) | |
| 2 | [3,5] | [ ] | [3,5] | (2,3) | |
| 3 | [5] | [ ] | [5] | (2,3,5) | |
| 4 | [ ] | [ ] | [ ] | | Вывод третьего максимального независимого множества. |
| 3 | [ ] | [5] | [ ] | | |
| 2 | [5] | [3] | [ ] | | Согласно логике черного ящика Б множество кандидатов Gg становится пустым. |
| 1 | [3..5] | [1,2 ] | [2,3] | (3) | |
| 2 | [5] | [2] | | | Итак, мы первый раз попадаем в процедуру Find, и множество Gm при этом не пусто. |

Должна работать логика черного ящика АА. *Замечание 1.* Если существует вершина j, принадлежащая Qm, такая, что пересечение A[j] и Qp пусто, то дальнейшее построение максимального независимого множества бессмысленно - вершины из A[j] не попадут в него. *Замечание 2.* Если нет вершин из Qm, удовлетворяющих первому замечанию, то какую вершину из Qp следует выбирать? Ответ: вершину $i \in (Qp \cap A[j])$ для некоторого значения $j \in Qm$, причем мощность пересечения множеств A[j] и Qp минимальна. Данный выбор позволяет сократить перебор. Итак, логика черного ящика АА.

```
begin
 delt:=N+1;
 for j:=1 to N do if j in Qm then if Number(A[j]*Qp)<delt then
                                  begin i:=j; delt:=Number(A[j]*Qp);end;
 Gg:=Qp*A[i];
end
```

Закончим трассировку примера.

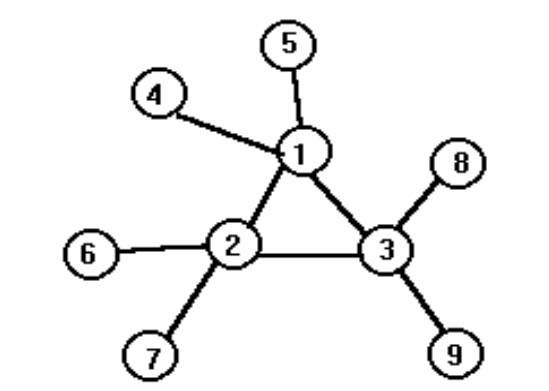
| | | | | | |
|---|---|---|---|---|---|
| 2 | [5] | [2] | [ ] | | |
| 1 | [5] | [1,2,3] | [ ] | | Выход в основную программу. |

Мы нашли все максимальные независимые множества.

### 3.7.3. Доминирующие множества

Для графа G=(V,E) доминирующее множество вершин есть множество вершин $S \subset V$, такое, что для каждой вершины j, не входящей в S, существует ребро, идущее из некоторой вершины множества S в вершину j. Доминирующее множество называется минимальным, если нет другого доминирующего множества, содержащегося в нем.

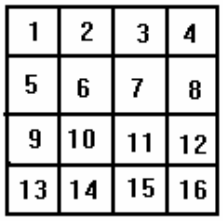
*Пример*.

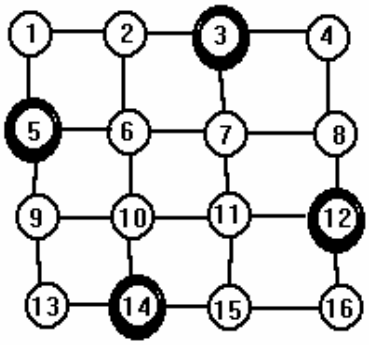



Доминирующие множества (1, 2, 3), (4, 5, 6, 7, 8, 9), (1, 2, 3, 8, 9), (1,2, 3, 7) и т. д. Множества (1, 2, 3), (4, 5, 6, 7, 8, 9) являются минимальными. Если Q - семейство всех минимальных доминирующих множеств графа, то число $\beta[G]=\min_{S \in Q} |S|$ называется числом доминирования графа, а множество S*, на котором этот минимум достигается, называется наименьшим доминирующим множеством. Для нашего примера $\beta[G]=3$.

*Задача.* Пусть город можно изобразить как квадрат, разделенный на 16 районов. Считается, что опорный пункт милиции, расположенный в каком-либо районе, может контролировать не только этот район, но и граничащие с ним районы. Требуется найти наименьшее количество пунктов милиции и места их размещения, такие, чтобы контролировался весь город.

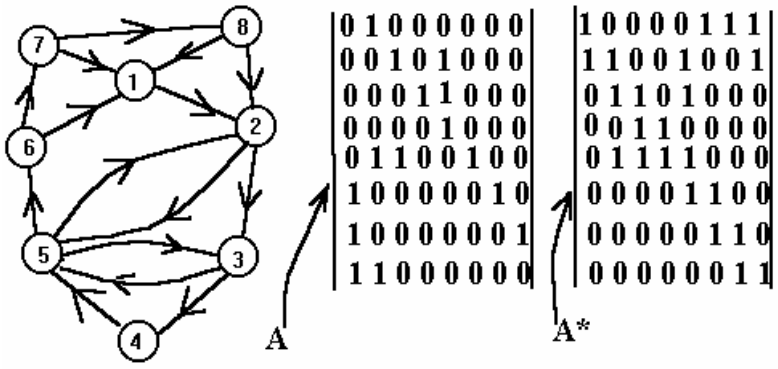
Представим каждый район вершиной графа и ребрами соединим только те вершины, которые соответствуют соседним районам. Нам необходимо найти число доминирования графа и хотя бы одно наименьшее доминирующее множество. Для данной задачи β[G]=4, и одно из наименьших множеств есть (3, 5, 12, 14). Эти вершины выделены на рисунке.




### 3.7.4. Задача о наименьшем покрытии

Рассмотрим граф. На рисунке показана его матрица смежности A и транспонированная матрица смежности с единичными диагональными элементами A*. Задача определения доминирующего множества графа G эквивалентна задаче нахождения такого наименьшего множества столбцов в матрице A*, что каждая строка матрицы содержит единицу хотя бы в одном из выбранных столбцов.

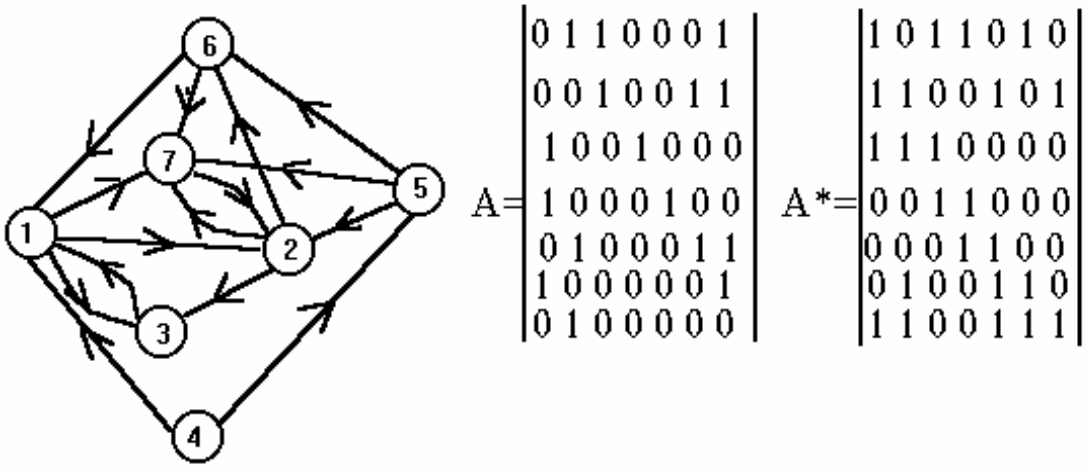



Задачу о поиске наименьшего множества столбцов, "покрывающего" все строки матрицы, называют *задачей о наименьшем покрытии.* В общем случае матрица не обязательно является квадратной, кроме того, вершинам графа (столбцам) может быть приписан вес, в этом случае необходимо найти покрытие с наименьшей общей стоимостью. Если введено дополнительное ограничение, суть которого в том, чтобы любая пара столбцов не имела общих единиц в одних и тех же строках, то задачу называют *задачей о наименьшем разбиении.*

*Замечание*. 1. Если некоторая строка матрицы A* имеет единицу в единственном столбце, то есть больше нет столбцов, содержащих единицу в этой строке, то данный столбец следует включать в любое решение. 2. Рассмотрим множество столбцов матрицы A*, имеющих единицы в конкретной строке. Для нашего примера: $U^1$=(1, 6, 7, 8), $U^2$=(1, 2, 5, 8), $U^3$=(2, 3, 5), $U^4$=(3, 4), $U^5$=(2, 3, 4, 5), $U^6$=(5, 6), $U^7$=(6, 7), $U^8$=(7,8). Видим, что $U^4 \subset U^5$. Из этого следует, что 5-ю строку можно не рассматривать, поскольку любое множество столбцов, покрывающее 4-ю строку, должно покрывать и 5-ю. Четвертая строка доминирует над пятой.

### 3.7.5. Метод решения задачи о наименьшем разбиении

Попытаемся осознать метод решения задачи, рассматривая, как обычно, пример. У нас есть ориентированный граф, его матрица смежности и транспонированная матрица смежности с единичными диагональными элементами. Исследуем структуру матрицы A*. Нас интересует, какие столбцы содержат единицу в первой строке, какие столбцы содержат единицу во второй строке и не содержат в первой и так далее. С этой целью можно было бы переставлять столбцы в матрице A*, но оставим ее "в покое". Будем



использовать дополнительную матрицу Bl, ее тип:

type Pr=array[1..MaxN,1..MaxN+1] of integer;

var Bl:Pr; , где MaxN - максимальная размерность задачи. Почему плюс единица (технический прием - "барьер"), будет ясно из последующего изложения (процедура Press).

При инициализации матрица Bl должна иметь вид:

- в первой строке - [1 2 3 .. N 0];
- все остальные элементы равны нулю.

То есть наше исходное предположение заключается в том, что все столбцы матрицы A* имеют единицы в первой строке. Проверим его. Будем просматривать элементы очередной строки (i) матрицы Bl. Если Bl[i,j]<>0, то со значением Bl[i,j], как номером столбца матрицы A*, проверим соответствующий элемент A*. При его неравенстве нулю элемент Bl остается на своем месте, иначе он переписывается в следующую строку матрицы Bl, а элементы текущей строки Bl сдвигаются вправо, сжимаются (Press). Итак, для N-1 строки матрицы Bl. Для нашего примера матрица Bl после этого преобразования будет иметь вид:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 3 | 4 | 6 | 0 | ... | 0 |
| | 2 | 5 | 7 | 0 | 0 | .... | 0 |
| Bl= | 0 | 0 | 0 | 0 | 0 | ... | 0 |
| | | | | | | ..... | |
| | 0 | 0 | 0 | 0 | 0 | | 0 |

В нашей задаче определены стоимости вершин графа или стоимости столбцов матрицы A*, и необходимо найти разбиение наименьшей стоимости. Пусть стоимости описываются в массиве Price (Price:*array[1..MaxN] of integer*) и для примера на рисунке имеют значения [15 13 4 3 8 9 10]. Осталась чисто техническая деталь - отсортировать элементы каждой строки матрицы Bl по возрастанию стоимости соответствующих столбцов матрицы A*. Результирующая матрица Bl имеет вид:
а логика формирования приведена ниже по тексту (Blocs).

| | | | |
|---|---|---|---|
| | 4 3 6 1 0 | ... | 0 |
| | 5 7 2 0 | ... | 0 |
| Bl= | 0 | | 0 |
| | | .... | |
| | 0 | ... | 0 |

```
procedure Blocs;{выделения блоков}
        {Bl - глобальная переменная}
procedure Sort;
        {Price и Bl - глобальные переменные}
begin
        ...
end;
procedure Press(i,j:integer);{Сдвигаем элементы строки с номером i, начиная с позиции (столбца) j, на одну позицию вправо}
        {Bl - глобальная переменная}
 var k:integer;
  begin
        k:=j;
        while Bl[i,k]<>0 do begin{Поэтому размерность матрицы с плюс единицей. В последнем столбце строки всегда записан 0.}
                Bl[i,k]:=Bl[i,k+1];
                Inc(k);
        end;{while}
  end;{Press}

var i,j,cnt:integer;
begin
        FillChar(Bl,SizeOf(Bl),0);
        for i:=1 to N do Bl[1,i]:=i;{предполагается, что в первом блоке все
        столбцы}
        for i:=1 to N-1 do begin
         j:=1;cnt:=0;
         while Bl[i,j]<>0 do begin
                if A*[i,Bl[i,j]]=0 then begin{столбец не в этом блоке}
                        Inc(cnt);
                        Bl[i+1,cnt]:=Bl[i,j];{переписать в следующую строку}
                        Press(i,j);
                        Dec(j);
                end;{if}
          Inc(j);
         end;{while}
        end; {for}
```

```
        Sort;
end;{Blocs}
```

После этой предварительной работы мы имеем вполне “приличную” организацию данных для решения задачи путем перебора вариантов. Матрица Bl разбита на блоки, и необходимо выбрать по одному элементу (если соответствующие строки ещё не покрыты) из каждого блока. Процесс выбора следует продолжать до тех пор, пока не будут включены в “покрытие” все строки или окажется, что некоторую строку нельзя включить.

Продолжим рассмотрение метода. Если при поиске независимых множеств мы шли “сверху вниз”, последовательно уточняя логику, то сейчас попробуем идти “снизу вверх”, складывая окончательное решение из сделанных “кирпичиков”. Как обычно, следует начать со структур данных. Во-первых, мы ищем лучшее решение, то есть то множество столбцов, которое удовлетворяет условиям задачи (непересечение и “покрытие” всего множества строк), и суммарная стоимость этого множества минимальна. Значит, необходима структура данных для хранения этого множества и значения наилучшей стоимости и, соответственно, структуры данных для хранения текущего (очередного) решения и его стоимости. Во-вторых, в решении строка может быть или не быть. Следовательно, нам требуется как-то фиксировать эту информацию. Итак, данные.

```
        type Model=array[1..MaxN] of boolean;
        var     Sbetter:Model;Pbetter:integer;{лучшее решение}
                S:Model;P:integer;{текущее решение}
                R:Model;{R[i]=true - признак того, что строка i “покрыта”
        текущим решением}
```

Логика включения (исключения) столбца с номером k в решение (из решения) имеет вид:

```
procedure Include(k:integer); {включить столбец в решение}
 {A*, R, Price, S, P - глобальные переменные}
 var j:integer;
 begin
        P:=P+Price[k];{текущая цена решения}
        S[k]:=true;{столбец с номером k в решение}
        for j:=1 to N do
         if A*[j,k]=1 then R[j]:=true; {строки, “покрытые” столбцом k}
 end;{Include}

procedure Exclude(k:integer);{исключить столбец из решения}
 var j:integer;
 begin
        p:=p-Price[k];
        S[k]:=false;
        for j:=1 to N do if (A*[j,k]=1) and R[j] then R[j]:=false;
 end;{Exclude}
```

Проверка, сформировано ли решение, заключается в том, чтобы просмотреть массив R и определить, все ли его элементы равны истине.

```
function Result:boolean;
 var j:integer;
 begin
        j:=1;
        while (j<=N) and R[j] do Inc(j);
        if j=N+1 then Result:=true else Result:=false;
 end; {Result}
```

Кроме перечисленных “кирпичиков”, нам необходимо уметь определять, можно ли столбец с номером k включать в решение. Для этого следует просмотреть столбец с номером k матрицы A* и проверить, нет ли совпадений единичных элементов со значением true соответствующих элементов массива R.

```
function Cross(k:integer):boolean; {пересечение столбца с частичным
        решением, сформированным ранее}
 var j:integer;
 begin
        j:=1;
        while (j<=N) and Not(R[j] and (A*[j,k]=1)) do Inc(j);
        if j=N+1 then Cross:=true else Cross:=false;
 end;{Cross}
```

Заключительная логика поиска (Find) имеет в качестве параметров номер блока (строки матрицы Bl) - переменная bloc и номер позиции в строке. Первый вызов - Find(1,1).

```
procedure Find(bloc,jnd:integer);
 {переменные глобальные}
 begin
        if Result then begin if P<Pbetter then begin Pbetter:=P;
                                                              Sbetter:=S;
                                                    end;
                        end
         else if Bl[bloc,jnd]=0 then exit
                 else if Cross(Bl[bloc,jnd]) then begin
                                                    Include(Bl[bloc,jnd]);
                                                    Find(bloc+1,1);
                                                    Exclude(Bl[bloc,jnd]);
                                                    end
                        else if R[bloc] then Find(bloc+1,1);
         Find(bloc,jnd+1);
 end;{Find}
```

Нам осталось дать общую логику, но после выполненной работы она не вызывает затруднений.

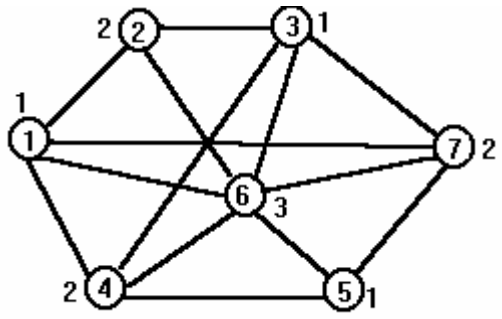
```
program R_min;
 const MaxN=...;
 type ...  var ...
 procedure Init; {ввод и инициализация данных}
        begin
        ...
        end;
procedure Print;{вывод результата}
        begin
        ...
        end;
{процедуры и функции, рассмотренные ранее}
  {основная логика}
 begin
        Init;
        Blocs;
        Find(1,1);
        Print;
 end.
```

## 3.8 Раскраски

### 3.8.1 Правильные раскраски

Пусть G=(V,E) - неориентированный граф. Произвольная функция f:V→{1,2,...,k}, где k принадлежит множеству натуральных чисел, называется вершинной k-раскраской графа G. Раскраска называется правильной, если f(u)≠f(v), для любых смежных вершин u и v. Граф, для которого существует правильная k-раскраска, называется k-раскрашиваемым. Минимальное число k, при котором граф G является k-раскрашиваемым, называется хроматическим числом графа и обозначается χ(G).

*Пример.* χ(G)=3. Меньшим количеством цветов граф правильно раскрасить нельзя из-за наличия треугольников. Рядом с "кружками" - вершинами графа - указаны номера цветов.




Метод получения правильной раскраски. Идея достаточно проста. Закрашиваем очередную вершину в минимально возможный цвет. Введем следующие структуры данных.

```
Const Nmax=100; {максимальное количество вершин графа}
Type V=0..Nmax;
        Ss=Set of V;
        My_array=array[1..Nmax] of V;
```

Var Gr:My_array;{Gr - каждой вершине графа определяется номер цвета}

Для примера, приведенного выше, массив Gr имеет вид: Gr[1]=Gr[3]=Gr[5]=1; Gr[2]=Gr[4]=Gr[7]=2; Gr[6]=3.

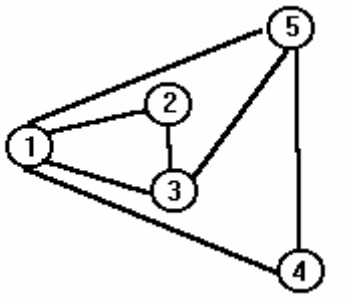
Фрагмент основной логики.

```
....
<формирование описания графа>;
for i:=1 to N do Gr[i]:=Color(i,0);
<вывод решения>;
```

Поиск цвета раскраски для одной вершины можно реализовать с помощью следующей функции:

```
function Color(i,t:V):integer;{i - номер окрашиваемой вершины, t - номер цвета, с которого следует искать раскраску данной вершины}
{A - матрица смежности, Gr - результирующий массив}
 var    Ws:Ss;
        j:byte;
 begin
        Ws:=[ ];
        for j:=1 to i-1 do {формируем множество цветов, в которые окрашены смежные вершины с меньшими номерами}
         if A[j,i]=1 then Ws:=Ws+[Gr[j]];
        j:=t;
        repeat {поиск минимального номера цвета, в который можно окрасить данную вершину}
         Inc(j);
        until Not(j in Ws);
        Color:=j;
 end;
```

*Пример*. Получаем правильную раскраску: Gr[1]=1, Gr[2]=Gr[4]=2, Gr[3]=3, Gr[5]=4. Однако минимальной раскраской является: Gr[1]=1, Gr[2]=Gr[5]=2, и Gr[3]=Gr[4]=3, и хроматическое число графа равно трем.




### 3.8.2. Поиск минимальной раскраски вершин графа

Метод основан на простой идее[7], и в некоторых случаях он дает точный результат. Пусть получена правильная раскраска графа, q - количество цветов в этой раскраске. Если существует раскраска, использующая только q-1 цветов, то все вершины, окрашенные в цвет q, должны быть окрашены в цвет g, меньший q. Согласно логике формирования правильной раскраски вершина была окрашена в цвет q, потому что не могла быть окрашена в цвет с меньшим номером. Следовательно, необходимо попробовать изменить цвет у вершин, смежных с рассматриваемой. Но как это сделать? Найдем вершину с минимальным номером, окрашенную в цвет q, и просмотрим вершины, смежные с найденной. Попытаемся окрашивать смежные вершины не в минимально возможный цвет. Для этого находим очередную смежную вершину и стараемся окрасить ее в другой цвет. Если это получается, то перекрашиваем вершины с большими номерами по методу правильной раскраски. При неудаче, когда изменить цвет не удалось или правильная раскраска не уменьшила количества цветов, переходим к следующей вершине. И так, пока не будут просмотрены все смежные вершины.

Опишем логику, используя функцию Color предыдущего параграфа.

```
var MaxC,MinNum,i:V;
procedure MaxMin(var c,t:V);
 begin
        ...
 end;
 procedure Change(t:V);
  begin
        ...
  end;
begin
        <ввод данных, инициализация переменных>;
        for i:=1 to N do Gr[i]:=Color(i,0);{первая правильная
                        раскраска}
        MaxC:=0;MinNum:=0;
```

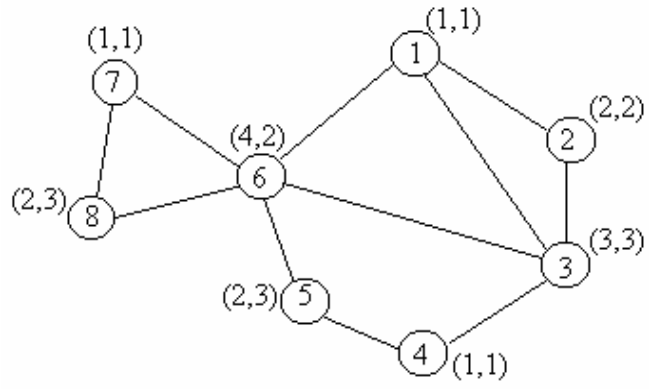
```
        repeat
        <найти вершину с минимальным номером(MinNum), имеющую максимальный цвет
раскраски(MaxC) - процедура MaxMin(MaxC,MinNum) >;
        <изменить цвет раскраски в соответствии с описанной идеей - процедура
Change(MinNum)>;
        until MaxC=Gr[MinNum];
    <вывод минимальной раскраски>;
    end.
```

Если работа процедуры MaxMin достаточно очевидна, то Change требует дальнейшего уточнения.

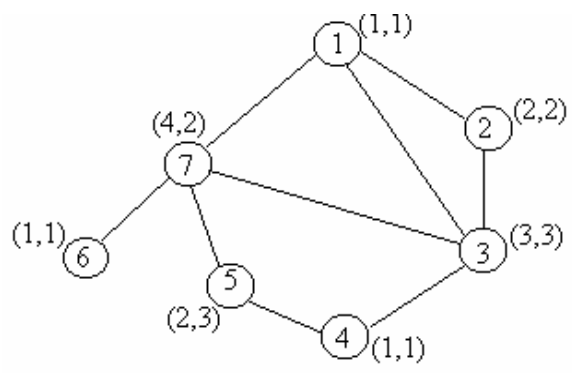
```
    procedure Change(t:V);
     var    r,q:V;
            Ws:Ss;
    function MaxCon(l:V;Rs:Ss):V;{находим смежную с l вершину с наибольшим номером и не
принадлежащую множеству Rs}
            var i:V;
            begin
                    for i:=l-1 downto 1 do
                            if (A[l,i]=1) and( not(i in Rs) then begin
                                    MaxCon:=i;
                                                                    exit
                                                            end
                    MaxCon:=0;
            end;
     begin
            Ws:=[];
            q:=MaxCon(t,Ws);
            while q<>0 do begin
                    r:=Color(q,Gr[q]);
                    if r<MaxC then <изменить цвет у вершин с номерами
            большими t по методу правильной
    раскраски>
                                    else Ws:=Ws+[q];
                    q:=MaxCon(t,Ws);
            end;
    end;
```

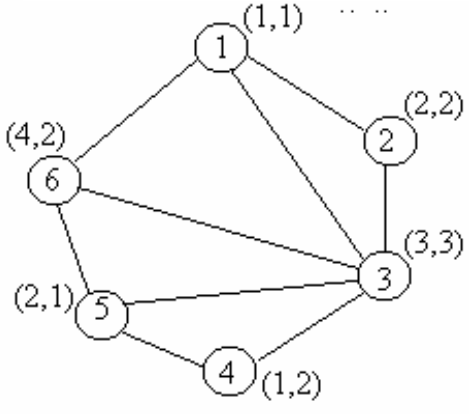
Осталось заметить, что при изменении цвета у вершин с номерами, большими значения t, по методу правильной раскраски следует запоминать в рабочих переменных значения Gr и MaxC, так как при неудаче ( раскраску не улучшим) их необходимо восстанавливать, и на этом закончить уточнение логики.

При любом упорядочении вершин допустимые цвета j для вершины с номером i удовлетворяют условию j≤i . Это очевидно, так как вершине i предшествует i-1 вершина, и, следовательно, никакой цвет j>i не использовался. Итак, для вершины 1 допустимый цвет 1, для 2 - цвет 1 и 2 и так далее. С точки зрения скорости вычислений вершины следует помечать так, чтобы первые q вершин образовывали наибольшую клику графа G. Это приведет к тому, что каждая из этих вершин имеет один допустимый цвет и процесс возврата в алгоритме можно будет заканчивать при достижении вершины из этого множества.



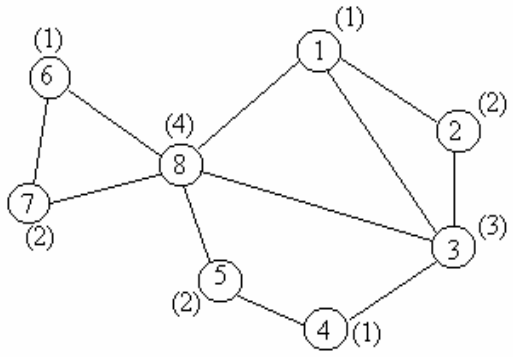

В алгоритме рассматриваются только вершины смежные с вершиной, окрашенной в максимальный цвет. Однако следует работать и с вершинами являющимися смежными для смежных. На рисунке приведены примеры графов и их раскраски. Первая цифра в круглых скобках обозначает значение цвета при правильной



раскраске, вторая - при минимальной.

Для графов на первом и третьем рисунках алгоритм дает правильный результат. Для графа на втором рисунке необходимо просматривать вершины, не смежные для вершины с номером 6, а для этого необходимо модифицировать алгоритм. Самый интересный случай на последнем рисунке. Мы не можем получить минимальную раскраску с помощью рассмотренного алгоритма, хотя она, конечно, существует и совпадает с предыдущей.

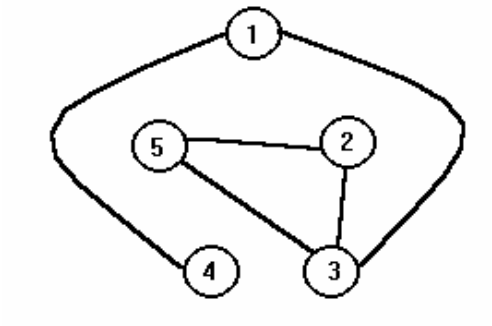



### 3.8.3. Использование задачи о наименьшем покрытии при раскраске вершин графа

При любой правильной раскраске графа G множество вершин, окрашиваемых в один и тот же цвет, является независимым множеством. Поэтому раскраску можно понимать как разбиение вершин графа на независимые множества. Если рассматривать только максимальные независимые множества, то раскраска - это не что иное, как покрытие вершин графа G множествами этого типа. В том случае, когда вершина принадлежит не одному максимальному независимому множеству, допустимы различные раскраски с одним и тем же количеством цветов. Эту вершину можно окрашивать в цвета тех множеств, которым она принадлежит.

Исходным положением метода является получение всех максимальных независимых множеств и хранение их в некоторой матрице M (N*W), где W - количество максимальных независимых множеств. Элемент матрицы M[i,j] равен единице, если вершина с номером i принадлежит множеству с номером j, и нулю в противном случае. Если затем каждому столбцу поставить в соответствие единичную стоимость, то задача раскраски не что иное, как поиск минимального количества столбцов в матрице M, покрывающих все ее строки. Каждый столбец соответствует определенному цвету.

*Пример*. Имеем пять максимальных независимых множеств и два варианта решения задачи о покрытии (строки A и B). В обоих случаях вершина 4 может быть окрашена как во второй, так и в третий цвета.




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | **1** | **1** | **0** | **0** | **0** |
| 2 | **1** | **0** | **0** | **1** | **0** |
| 3 | **0** | **0** | **1** | **0** | **0** |
| 4 | **0** | **0** | **1** | **1** | **1** |
| 5 | **0** | **1** | **0** | **0** | **1** |
| A | * | | * | | * |
| B | | * | * | * | |

## 3.9. Потоки в сетях, паросочетания

### 3.9.1. Постановка задачи

Одной из задач теории графов является задача определения максимального потока, протекающего от некоторой вершины s графа (источника) к некоторой вершине t (стоку). При этом каждой дуге (граф ориентированный) (i,j) приписана некоторая пропускная способность C(i,j), определяющая максимальное значение потока, который может протекать по данной дуге. Содержательных интерпретаций задачи достаточно много, и, безусловно, они усилят и сделают более понятными сложные занятия по этой проблематике.

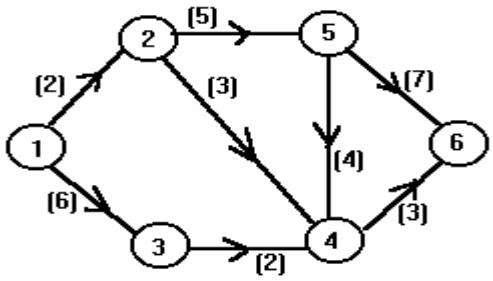
Метод решения задачи о максимальном потоке от s к t был предложен Фордом и Фалкерсоном, и их “техника меток” составляет основу других алгоритмов решения многочисленных задач, являющихся обобщениями или расширениями указанной задачи.

Одним из фундаментальных фактов теории потоков в сетях является классическая теорема о максимальном потоке и минимальном разрезе. Разрезом называют множество дуг, удаление которых из сети приводит к “разрыву” всех путей, ведущих из s в t. Пропускная способность разреза - это суммарная пропускная способность дуг, его составляющих. Разрез с минимальной пропускной способностью называют минимальным разрезом.

*Теорема* (Форд и Фалкерсон). Величина каждого потока из s в t не превосходит пропускной способности минимального разреза, разделяющего s и t, причем существует поток, достигающий этого значения.

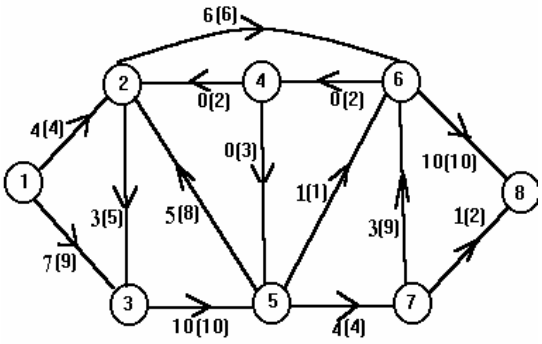
Теорема устанавливает эквивалентность задач нахождения максимального потока и минимального разреза, однако не определяет метода их поиска.

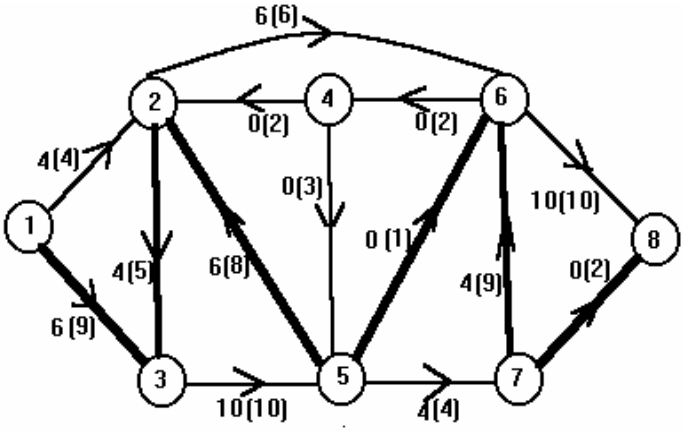
*Пример*. Показана сеть, источник - вершина 1, сток - вершина 6, в круглых скобках у дуг указаны их пропускные способности. Минимальный разрез - дуги (1, 2) и (3, 4), следовательно, согласно теореме максимальный поток равен 4. Разрез определен путем простого перебора. Логика его “лобового” поиска очевидна. Осуществляем перебор по дугам путем генерации всех возможных подмножеств дуг. Для каждого подмножества дуг проверяем, является ли оно разрезом. Если является, то вычисляем его пропускную способность и сравниваем ее с минимальным значением. При положительном результате сравнения запоминаем разрез и изменяем значение минимума. Удачный выбор данных позволяет сделать программный код компактным, но очевидно, что даже при наличии различных отсечений в переборе метод применим только для небольших сетей. Однако, как найти максимальный поток, то есть его распределение по дугам, по - прежнему открытый вопрос.




“Техника меток” Форда и Фалкерсона заключается в последовательном (итерационном) построении максимального потока путем поиска на каждом шаге увеличивающейся цепи, то есть пути (последовательности дуг), поток по которой можно увеличить. При этом узлы (вершины графа) специальным образом помечаются. Отсюда и возник термин “метка”.

Пример из работы [9]. Рядом с пропускными способностями дуг указаны потоки, построенные на этих дугах. На рисунке поток через сеть



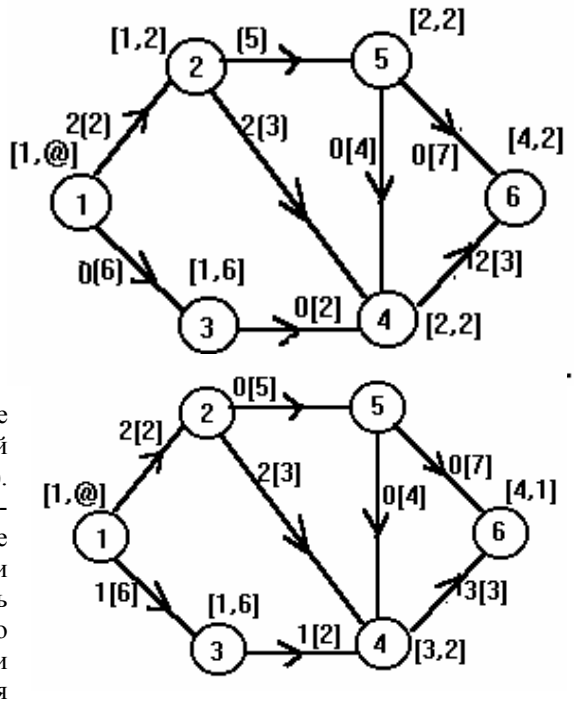




равен 10 и найдена увеличивающаяся цепочка, выделенная “жирными” линиями. Обратите внимание на ориентацию дуг, входящих в цепочку. По данной цепочке можно пропустить поток, равный 1, пропускная способность дуги (5, 6). Изменяем суммарный поток, его значение становится равным 11. Поток увеличен, необходимо продолжить поиск увеличивающихся цепочек; если окажется, что построить их нельзя, то результирующий поток максимален. Заметим, что для данного примера это значение потока окончательное. Обратите внимание на то, как изменен поток на дугах сети в зависимости от их ориентации.

### 3.9.2. Метод построения максимального потока в сети

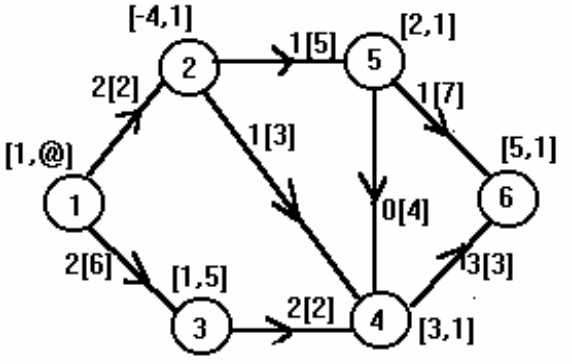
Рассмотрим метод на примере. Пусть дана сеть G=(V,E), узлом-источником является вершина 1, узлом-стоком - вершина 6. Построим максимальный поток (F) между этими вершинами. Начальное значение F нулевое. Очевидно, что структурой данных для описания F является матрица того же типа, что и матрица C, в которой определены пропускные способности дуг.

*Первая итерация*. Присвоим вершине 1 метку [1,@]. Первый шаг. Рассмотрим дуги, началом которых является вершина 1 - дуги (1,2) и (1,3). Вершины 2 и 3 не помечены, поэтому присваиваем им метки, для 2-й - [1,2] и 3-й - [1,6]. Что представляют из себя метки? Первая цифра - номер вершины, из которой идет поток, вторая цифра - численное значение потока, который можно передать по этой дуге. Второй шаг. Выберем помеченную, но не просмотренную вершину. Первой в соответствующей структуре данных записана вершина 2. Рассмотрим дуги, для которых она является началом - дуги (2,4) и (2,5). Вершины 4 и 5 не помечены. Присвоим им метки - [2,2] и [2,2]. Итак, на втором шаге вершина 2 просмотрена, вершины 3, 4, 5 помечены, но не просмотрены, остальные вершины не помечены. Третий шаг. Выбираем вершину 3. Рассмотрим дугу (3,4). Вершина 4 помечена. Переходим к следующей вершине - четвертой, соответствующая дуга - (4,6). Вершина 6 не помечена. Присваиваем ей метку [4,2]. Мы достигли вершины-стока, тем самым найдя путь (последовательность дуг), поток по которым можно увеличить. Информация об этом пути содержат метки вершин. В данном случае путь или увеличивающаяся цепочка 1→2→4→6. Максимально возможный поток, который можно передать по дугам этого пути, определяется второй цифрой метки вершины стока, то есть 2. Поток в сети стал равным 2.

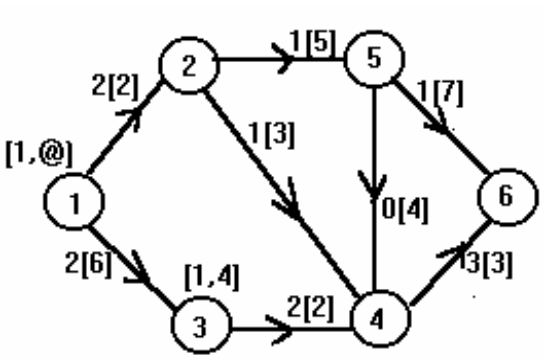



*Вторая итерация*. Первый шаг. Присвоим вершине 1 метку [1,@]. Рассмотрим дуги, началом которых является помеченная вершина 1. Это дуги (1,2) и (1,3). Вершина 2 не может быть помечена, так как пропускная способность дуги (1,2) исчерпана. Вершине 3 присваиваем метку [1,6]. Второй шаг. Выберем помеченную, но не просмотренную вершину. Это вершина 3. Повторяем действия. В результате вершина 4 получает метку [3,2]. Третий шаг. Выбираем вершину 4, только она помечена и не просмотрена. Вершине 6 присваиваем метку [4,1]. Почему только одна единица потока? На предыдущей итерации израсходованы две единицы пропускной способности данной дуги, осталась только одна. Вершина-сток достигнута. Найдена увеличивающая поток цепочка, это 1→3→4→6, по которой можно ”протащить” единичный поток. Результирующий поток в сети равен 3.

*Третья итерация*. Вершине 1 присваиваем метку [1,@]. Первый шаг. Результат - метка [1,5] у вершины 3. Второй шаг - метка [3,1] у вершины 4. Третий шаг. Пропускная способность дуги(4,6) израсходована полностью. Однако есть *обратная* дуга (2,4), по которой передается *поток, не равный нулю* (обратите внимание на текст, выделенный курсивом - “изюминка” метода). Попробуем перераспределить поток. Нам необходимо передать из вершины 4 поток, равный единице (зафиксирован в метке вершины). Задержим единицу потока в вершине 2, то есть вернем единицу потока из вершины 4 в вершину 2. Эту особенность зафиксируем в метке вершины 2 - [-4,1]. Тогда единицу потока из вершины 4 мы передадим по сети вместо той, которая задержана в вершине 2, а единицу потока из вершины 2 попытаемся “протолкнуть” по сети, используя другие дуги. Итак, вершина 4 просмотрена, вершина 2 помечена, вершины 5 и 6 не помечены. Четвертый и пятый шаги очевидны. Передаем единицу потока из вершины 2 в вершину 6 через вершину 5. Вершина-сток достигнута, найдена цепочка 1→3→4→2→5→6, по которой можно передать поток, равный единице. При этом по прямым дугам поток увеличивается на единицу, по обратным - уменьшается. Суммарный поток в сети - 4 единицы.



*Четвертая итерация*. Вершине 1 присваиваем метку [1,@]. Первый шаг. Помечаем вершину 3 - [1,4]. Второй шаг. Рассматриваем помеченную, но не просмотренную вершину 3. Одна дуга - (3,4). Вершину 4 пометить не можем - пропускная способность дуги исчерпана. Помеченных вершин больше нет, и вершина-сток не достигнута. Увеличивающую поток цепочку построить не можем. Найден максимальный поток в сети. Можно заканчивать работу.




Итак, в чем суть “техники меток” Форда и Фалкерсона? Первое. На каждой итерации вершины сети могут находиться в одном из трех состояний: вершине присвоена метка, и она просмотрена; вершине присвоена метка, и она не просмотрена, то есть не все смежные с ней вершины обработаны; вершина не имеет метки. Второе. На каждой итерации мы выбираем помеченную, но не просмотренную вершину v и пытаемся найти вершину u, смежную с v, которую можно пометить. Помеченные вершины образуют множество вершин сети G, достижимые из вершины-источника. Если среди этих вершин окажется вершина-сток, то это означает успешный результат поиска цепочки, увеличивающей поток, при неизменности этого множества работа заканчивается - поток изменить нельзя.

*Алгоритм. Входные данные*. Описание сети G=(V,E) матрицей пропускных способностей C[1..N,1..N], где N - количество вершин. Вершина-источник s и вершина-сток t. *Выходные данные*. Поток, описываемый матрицей F[1..N,1..N]. *Рабочие переменные*. Структура данных для хранения меток - P[1..N,1..2]. Элемент P[i,1] - номер вершины, из которой можно передать поток, равный P[i,2], в вершину с номером i. Логическая переменная Lg, значение true - есть цепочка, увеличивающая поток, false - нет.

*Основная логика*.

```
begin
        <ввод данных и инициализация переменных(Lg:=true)>;
        while Lg do begin
                FillChar(P,SizeOf(P),0);
                <процедура расстановки меток(Mark), если вершину t не смогли  пометить, то Lg:=false; результат работы - значение P (метки вершин) >;
                if Lg then <процедура Stream(t) - изменение потока по дугам найденной цепочки от вершины-стока t до вершины-источника s; входные данные - массив P, результат - измененный массив F>;
         end;
        <вывод потока F>;
end.{конец обработки}
```

Уточним логику расстановки меток (нелучший вариант).

```
procedure Mark;
 var  M:set of 1..N;i,l:integer;
begin
 M:=[1..N]; {непросмотренные вершины}
 P[s,1]:=s;P[s,2]:=maxint;{присвоим метку вершине-источнику}
 l:=s;
 while (P[t,1]=0) and Lg do begin
        for i:=1 to N do {поиск непомеченной вершины}
         if (P[i,1]=0) and ((C[l,i]<>0) or (C[i,l]<>0)) then
                if F[l,i]<C[l,i] then begin{дуга прямая?}
                        P[i,1]:=l; if P[l,2]<C[l,i]-F[l,i] then P[i,2]:=P[l,2]
                                        else P[i,2]:=C[l,i]-F[l,i];
                        end
                else if F[i,l]>0 then begin{дуга обратная?}
                                P[i,1]:=-l;
                                if P[l,2]<F[i,l] then P[i,2]:=P[l,2] else P[i,2]:=F[i,l];
                                end;
        M:=M-[l];{вершина с номером l просмотрена}
        l:=1;{находим помеченную и непросмотренную вершину}
        repeat Inc(l) until (l>N) or ((P[l,1]<>0) and (l in M));
        if l>N then Lg:=false;
 end;
end;
```

Логика изменения потока F имеет вид:

```
procedure Stream(q:integer);
```

```
begin
        {определяем тип дуги - прямая или обратная, знак минус у номера
вершины - признак обратной дуги}
        if P[q,1]>0 then F[P[q,1],q]:=F[P[q,1],q]+P[t,2]
         else F[q,abs(P[q,1])]:=F[q,abs(P[q,1])]-P[t,2];
        {если не вершина-источник, то переход к предыдущей вершине
цепочки}
        if abs(P[q,1])<>s then begin
                q:=abs(P[q,1]);Stream(q);end;
end;
```

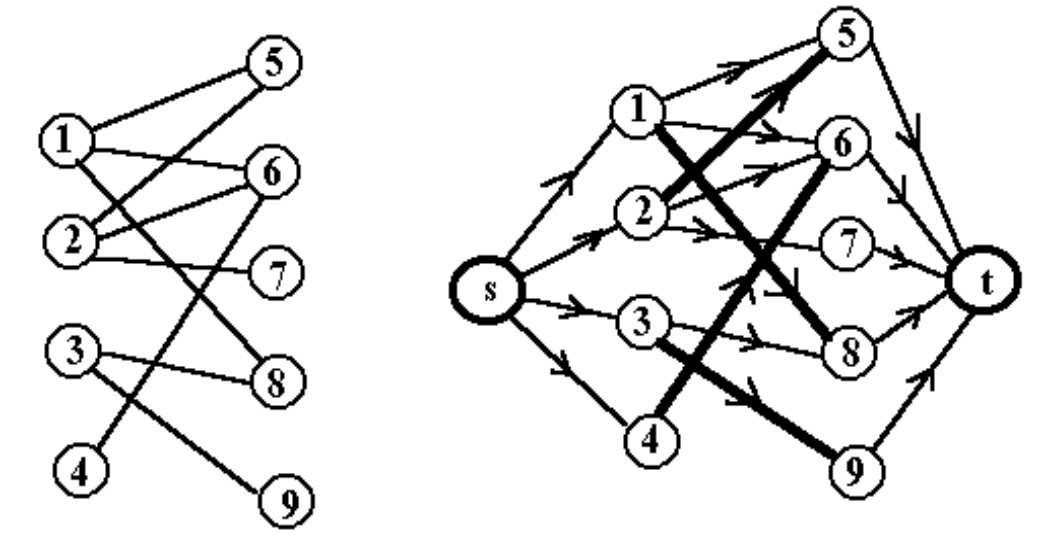
Итак, рассмотрение метода построения максимального потока в сети завершено.

### 3.9.3. Наибольшее паросочетание в двудольном графе

Паросочетанием в неориентированном графе G=(V,E) называется произвольное множество ребер $M \subseteq E$, такое, что никакие два ребра из M не инцидентны одной вершине. Вершины в G, не принадлежащие ни одному ребру паросочетания, называют свободными. Граф G называют двудольным, если его множество вершин можно разбить на непересекающиеся множества - $V=X \cup Y$, $X \cap Y=\varnothing$, причем каждое ребро $e \in E$ имеет вид e=(x,y), где $x \in X$, $y \in Y$. Двудольный граф будем обозначать G=(X,Y,E).

Задача нахождения наибольшего паросочетания в двудольном графе сводится к построению максимального потока в некоторой сети. Рассмотрим схему сведения. На рисунке показан исходный двудольный граф G. Построим сеть S(G) с источником s и стоком t следующим образом:

- источник s соединим дугами с вершинами из множества X;
- вершины из множества Y соединим дугами со стоком t;
- направление на ребрах исходного графа будет от вершин из X к вершинам из Y;
- пропускная способность всех дуг C[i,j]=1.




Найдем максимальный поток из s в t для построенной сети. Он существует[11]. Насыщенные дуги потока (они выделены “жирными” линиями на рисунке соответствуют ребрам наибольшего паросочетания исходного графа.

*Примечание*. Найденное наибольшее паросочетание не единственное. Проверьте, это ли паросочетание получается при помощи алгоритма нахождения максимального потока. Найдите другие паросочетания.

Рассмотрим другой метод построения наибольшего паросочетания в двудольном графе. Введем понятие *чередующейся цепи* из X в Y относительно данного паросочетания M. Произвольное множество дуг $P \subseteq E$ вида:

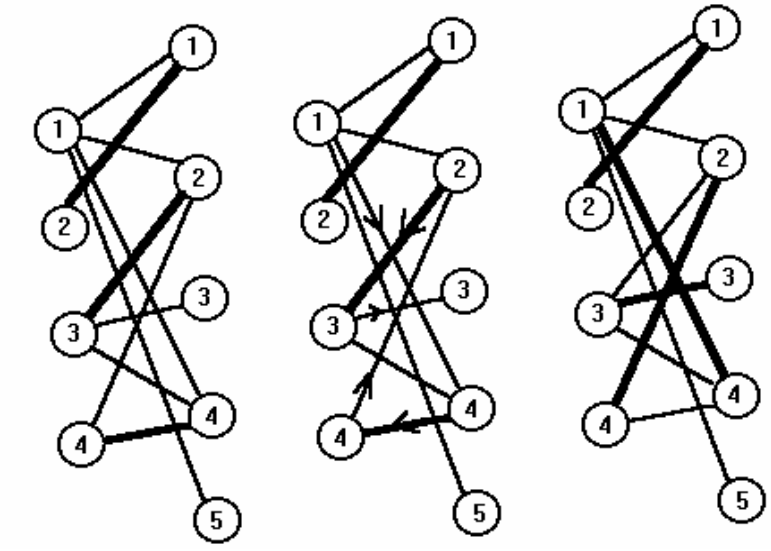
$$P=\{(x_0,y_1), (y_1,x_1), (x_1,y_2), ..., (y_k,x_k), (x_k,y_{k+1})\},$$

где k>0, все вершины различны, $x_0$ - свободная вершина в X, $y_{k+1}$ - свободная вершина в Y, и каждая вторая дуга принадлежит M, то есть

$$P \cap M=\{(y_1,x_1), (y_2,x_2), ..., (y_k,x_k)\}.$$

*Теорема*[3]. Паросочетание M в двудольном графе G наибольшее тогда и только тогда, когда в G не существует чередующейся цепи относительно M.

Теорема подсказывает реальный путь нахождения наибольшего паросочетания. Пусть найдено некоторое паросочетание в графе G, на рисунке оно выделено “жирными” линиями. Ищем чередующуюся цепь - ее дуги выделены линиями со стрелками.




Она состоит из “тонких” дуг и “жирных”, причем начинается и заканчивается в свободных вершинах. Длина цепи нечетна. Делаем “тонкие” дуги “жирными” и наоборот. Паросочетание увеличено на единицу.

*Общая логика.*

```
begin
        <ввод и инициализация данных>;
        <найти первое паросочетание>;
        while <есть чередующаяся цепочка?> do <изменить паросочетание>;
        <вывод результата>;
end.
```

Очередь за определением данных и последовательным уточнением логики. Граф описывается матрицей A[N,M], где N - количество вершин в множестве X, а M в Y. Паросочетание будем хранить в двух одномерных массивах XN и YM. Для рассмотренного примера XN=[0,1,2,4] и YM=[2,3,0,4,0]. Уточнение фрагмента “найти первое паросочетание” не требует разъяснений.

```
for i:=1 to N do begin {назначение переменных i, j, pp следует
очевидно}
        j:=1; pp:=true;
        while (j<=M) and pp do begin
                if (A[i,j]=1) and (YM[j]=0) then begin
                                                        XN[i]:=j;YM[j]:=i;
                                                        pp:=false;
                                                end;
                inc(j);
         end;
 end;
```

Как лучше хранить чередующуюся цепочку, учитывая требование изменения паросочетания и продумывая логику её построения? Естественно, массив, пусть это будет *Chain:array[1..NMmax] of -NMmax.. NMmax*, где NMmax - максимальное число вершин в графе, и его значение для рассмотренного примера равно [1, -4, 4, -2, 3, -3], то есть вершины из множества YM хранятся со знаком минус (вспомните метод построения максимального потока). Итак, поиск чередующейся цепочки.

```
function FindChain:boolean;
 var p,yk,r:word;
        begin
        <инициализация данных, в частности yk:=1;>;
        <нахождение свободной вершины в  XN>;
        if <вершина не найдена> then begin FindChain:=false;exit;end;
         p:=<номер первой свободной вершины>;
         Chain[yk]:=p;
         repeat
                r:=<очередная вершина из Chain>;
                for <для всех вершин, связанных с r> do
                 if <вершина из XN>
                        then begin if <ребро “тонкое”> then
                                        <включить ребро в цепочку>
                             end
                        else begin if <ребро “толстое”> then
                                        <включить ребро в цепочку>
                                end;
```

until <просмотрены все вершины из Chain> or <текущая вершина принадлежит YM, и она свободна>;

FindChar:=<текущая вершина принадлежит YM, и она свободна>;

end;

Процесс дальнейшего уточнения логики вынесем в самостоятельную часть работы.

## 3.10. Методы приближенного решения задачи коммивояжера

### 3.10.1. Метод локальной оптимизации

Попытаемся отказаться от перебора вариантов, рассмотренного в главе 2. Найдем первое решение (первую оценку), а затем попытаемся улучшить оценку, просматривая только соседние города найденного пути.

Шаг 1. Получить приближенное решение (первую оценку). При этом не следует забывать, что после получения первой оценки предыдущим методом его работа заканчивается.

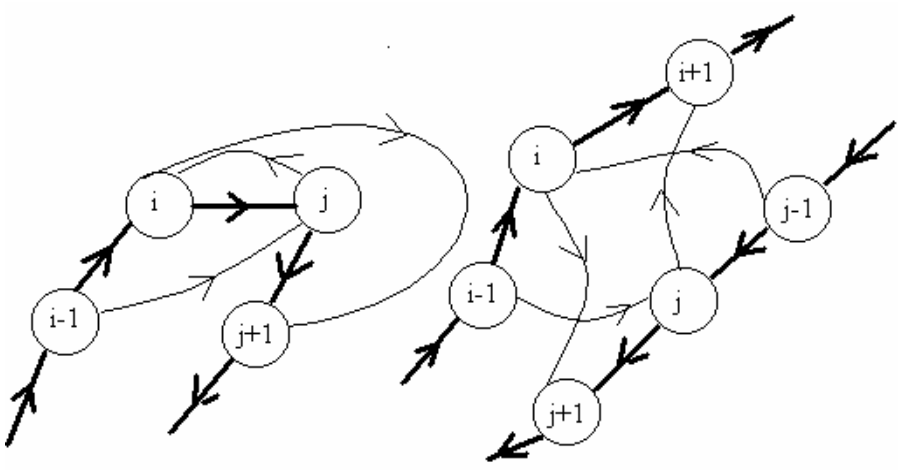
Шаг 2. Пока происходит улучшение решения, выполнять следующий шаг, иначе перейти на шаг 4.

Шаг 3. Для всех пар номеров городов i,j, удовлетворяющих неравенству ($1 \le i < j \le n$), проверить:

$d_{i-1,i} + d_{i,j} + d_{j,j+1} > d_{i-1,j} + d_{j,i} + d_{i,j+1}$ для смежных городов, то есть $j=i+1$

$d_{i-1,i} + d_{i,i+1} + d_{j-1,j} + d_{j,j+1} > d_{i-1,j} + d_{j,i+1} + d_{j-1,i} + d_{i,j+1}$ для несмежных городов.

Примечание. На рисунке даны графические иллюстрации первого и второго неравенств. “Жирными” линиями обозначены участки старых маршрутов, “тонкими” - новых.




Если одно из неравенств выполняется, то найдено лучшее решение. Следует откорректировать ранее найденное решение и вернуться на шаг 2.

Шаг 4. Закончить работу алгоритма.

Для реализации логики (из рассмотренных в предыдущем алгоритме структур данных) достаточно матрицы расстояний (A) и массива для хранения пути коммивояжера (Way). Вид общей логики:

```
begin
    init;{Ввод из файла матрицы расстояний, инициализация глобальных переменных}
    one_way;{Поиск первого варианта пути коммивояжера}
    local;{Локальная оптимизация}
    out;{Вывод результата}
end.
```

Примечание. Владение технологиями “сверху вниз” и “снизу вверх” - необходимые составляющие структурной парадигмы мышления.

Продолжим уточнение логики. Работа процедур init, one_way, out достаточно очевидна. Естественным приемом является вынесение ее в самостоятельную часть работы школьников на занятии. Нам необходимо уточнить процедуру local. Работаем не с частностями, а на содержательном уровне. Предположим, что мы имеем функции best1 и best2, их параметры - индексы элементов массива way, определяющих номера городов в пути коммивояжера, а выход естественный - истина или ложь, в зависимости от того, выполняются неравенства или нет. Рассуждаем дальше. Если неравенство выполняется, то нам необходимо изменить путь (соответствующие элементы массива way). Пока не будем заниматься деталями. Пусть эту работу выполняет процедура swap, ее параметры - индексы элементов массива way). Эта процедура, вероятно, должна сообщать о том, что она «что-то изменила», ибо нам необходимо продолжать работу до тех пор, пока что-то меняется, происходят улучшения. Итак, логика процедуры local.

```
Procedure local;
 var i,j:integer; change:boolean;
 <здесь функции best1 и best2, а также процедура  swap>;
```

```
begin
        repeat
        change:=false;
        for i:=1 to n-1 do
         for j:=i+1 to n do
                if i=j+1 then begin if best1(i,j) then swap(i,j);end
                 else if (i=1) and (j=n) then begin if best1(i,j) then swap(i,j); end{Об этой проверке лучше первоначально умолчать, чтобы было о чем спросить, «если совсем будет плохо» - все понимают и нет вопросов}
                                else if best2(i,j) then swap(i,j);
        until not(change);
end.
```

### 3.10.2. Алгоритм Эйлера

Этот алгоритм и следующий работоспособны в том случае, если выполняется неравенство треугольника. Его суть в том, что для любой тройки городов i, j, k (между которыми есть связь) выполняется неравенство di,j+dj,k>di,k. Рассмотрим идею алгоритма.

Шаг1. Строится каркас минимального веса (алгоритмы Прима или Краскала).

Примечание. Это не есть первое приближение, как в предыдущем алгоритме.

Шаг 2. Путем дублирования каждого ребра каркас преобразуется в эйлеров граф.

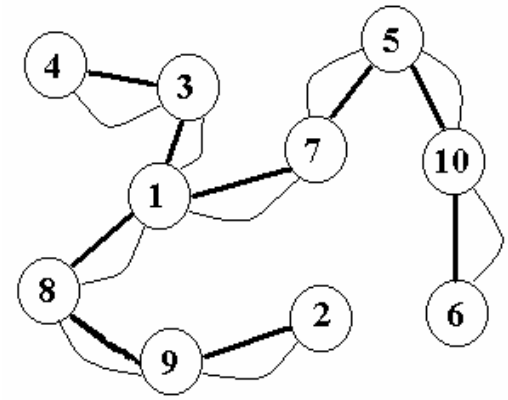
Шаг 3. Находим в построенном графе эйлеров цикл.

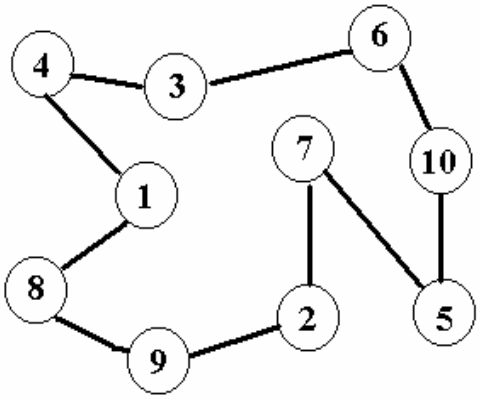
Шаг 4. Эйлеров цикл преобразуем в гамильтонов цикл (или маршрут коммивояжера). Метод преобразования: последовательность вершин эйлерова цикла сокращается так, чтобы каждая вершина графа в получившемся цикле встречалась ровно один раз.

Шаг 5. Закончить работу алгоритма. Получено приближенное решение задачи коммивояжера.

Покажем , что стоимость приближенного решение CostAp не превосходит удвоенной стоимости оптимального решения CostBet. Пусть стоимость каркаса - CostFr. Тогда CostFr<CostBet, так как при удалении из оптимального пути коммивояжера ребра получаем каркас с весом не большим, чем CostAp. Из правила построения эйлерова графа получаем, что вес построенного эйлерова цикла 2*CostFr. Неравенство треугольника обеспечивает результат: следующую оценку шага 4 - CostAp<2*CostFr, а значит, CostAp<2*CostBet.

*Рассмотрим пример*.




Маршрут Эйлера:
1-8-9-2-9-8-1-7-5-10-6-10-5-7-1-3-4-3-1

Для исходных данных, приведенных в п. 2.1.7, на рисунке жирными линиями выделен каркас минимального веса. Тонкие линии добавляются на шаге 2. Получаем эйлеров граф. Затем эйлеров цикл (маршрут) преобразуется в путь коммивояжера (шаг 4). Согласно неравенству треугольника мы получаем: $d_{2,9}+d_{9,8}>d_{2,8}$; $d_{2,8}+d_{8,1}>d_{2,1}$; $d_{2,1}+d_{1,7}>d_{2,7}$. Суммируя левые и правые части неравенств, получаем: $d_{2,9}+d_{9,8}+d_{8,1}+d_{1,7}>d_{2,7}$. Для рассмотренного примера CostAp равна 202. Это ~ 1.28*CostBet.

Структуры данных. Помимо названных ранее массивов A - матрица расстояний и массива Way - хранение искомого пути, требуется «что-то» для хранение описания каркаса. Пусть это будет массив B (B:array[1..Nmax,1..Nmax] of boolean). Элемент B[i,j], равный true, говорит о том, что ребро (i,j) графа принадлежит каркасу.

Общая логика.

```
Begin
        init;{ввод описания - матрица расстояний; инициализация данных}
        solve;{решатель}
        out;{вывод результата}
end.
```

Процедуры init и out очевидны (должны писаться «на автомате»). Уточняем процедуру solve. Первый «набросок».

```
Procedure solve;
 var ?
 <процедуры и функции>;
 begin
        init_solve;{инициализация переменных процедуры solve}
        find_tree;{построение каркаса}
        eiler_way;{поиск эйлерова цикла}
        komm_way;{поиск пути коммивояжера}
end;
```

Прежде чем продолжать уточнение, необходимо определить структуры данных этой части логики и взаимодействие ее составных частей. Во-первых, при построении каркаса необходимо иметь список ребер, отсортированный в порядке возрастания их весов (алгоритмы Краскала и Прима). Следовательно, на входе процедуры find_tree матрица расстояний A, на выходе - B, рабочие структуры - массив для хранения списка ребер. Внутренние процедуры: формирование списка ребер и сортировки. Продолжим рассмотрение. Эйлеров цикл необходимо где-то хранить, пусть это будет массив St (St:array[1..n*(n-1) div 2] of byte. Количество не нулевых элементов в St - значение переменной Count. Эти величины описываются в разделе переменных процедуры solve, их инициализация - в процедуре init_solve.Процедуру поиска эйлерова цикла сделаем рекурсивной, поэтому первый вызов изменится - eiler_way(1). Выбор начальной вершины при поиске эйлерова цикла не имеет значения.

Итак, классическая логика поиска эйлерова цикла. Приводится с целью показа работы процедуры komm_way, ибо последняя не есть поиск гамильтонова цикла в обычной трактовке.

```
procedure eiler_way(v:byte);
 var j:integer;
 begin
        for j:=1 to N do
         if b[v,j] then begin b[v,j]:=false;b[j,v]:=false;eiler_way(j);end;
         inc(count);St[count]:=v;{заносим номер вершины в эйлеров цикл}
end;
procedure komm_way;
 var s:set of 1..Nmax;
        i,j:integer;
 begin
        i:=0;s:=[];
        for j:=1 to Count do{исключаем повторяющиеся номера вершин из эйлерова цикла}
         if Not(St[j] in s) then begin
                                        inc(i);way[i]:=St[j];s:=s+[St[j]];
                                end;
 end;
```

### 2.10.3. Алгоритм Кристофидеса

Отличается от предыдущего логикой построения маршрута из каркаса минимального веса. Неравенство треугольника выполняется.

Шаг 1. Строится каркас минимального веса (алгоритм Прима или Краскала).

Шаг 2. На множестве вершин каркаса(обозначим их через V), имеющих нечетное число ребер каркаса, находится паросочетание минимального веса. Таких вершин в любом каркасе четное число. Метод поиска паросочетания - чередующиеся цепочки.

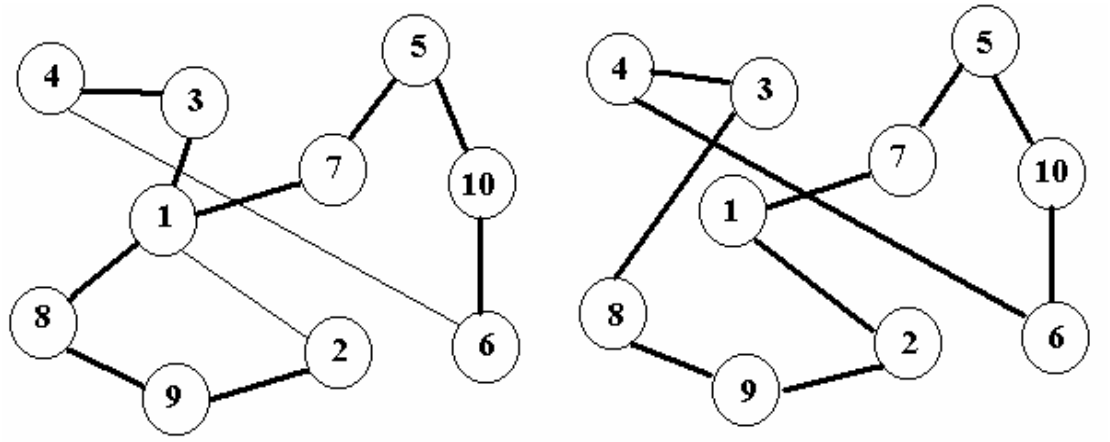
Шаг 3. Каркас преобразуется в эйлеров граф путем присоединения ребер, соответствующих найденному паросочетанию.

Шаг 4. В построенном графе находится эйлеров маршрут.

Шаг 5. Эйлеров маршрут преобразуется в маршрут коммивояжера:

Итак, данный метод отличается от предыдущего только шагами 2 и 3.

Все этапы реализуются за полиномиальное время. Известно[1], что для этого метода оценка приближенного метода имеет вид CostAp<1.5*CostBest и эта оценка является лучшей для приближенных полиномиальных алгоритмов решения задачи о коммивояжере с неравенством треугольника. На данных из п. 2.1.7 на рисунке показан пример каркаса (жирными линиями) и паросочетание минимального веса (тонкими линиями), построенное на вершинах каркаса с нечетными степенями. Путь коммивояжера имеет стоимость CostAp, равную 191, что составляет ~ 1.2*CostBest.




Маршрут Эйлера:

1-7-5-10-6-4-3-1-8-9-2-1

В логику solve (см. предыдущий алгоритм) добавляется процедура построения паросочетания минимального веса (P). Назовем ее pair. Ее входными данными является матрица B (описывает каркас), выходными - новая матрица C (элементы логического типа), соответствующая графу получаемому добавлением к каркасу ребер P.

```
Procedure pair;
 var ?;
 <процедуры pair>;
 begin
        init_pair;{инициализация переменных процедуры, формирование массива с номерами вершин, имеющих нечетную степень}
        first;{поиск первого паросочетания}
        find;{поиск P}
        ad;{добавление ребер, образующих P к каркасу  - матрица C}
end;
```

Примечание. Возможен вариант уточнения логики до работающей программы (это уже теоретическое занятие, которые мы очень не любим). Однако перед этим занятием школьники повторяют ранее изученные алгоритмы построения паросочетаний в графе, так что на этой стадии детализации можно и остановиться.

## 3.11. Задачи

1. Дан ориентированный граф с N вершинами (N<50). Вершины и дуги окрашены в цвета с номерами от 1 до M (M≤6). Указаны две вершины, в которых находятся фишки игрока и конечная вершина. Правило перемещения фишек: игрок может передвигать фишку по дуге, если ее цвет совпадает с цветом вершины, в которой находится другая фишка; ходы можно делать только в направлении дуг графа; поочередность ходов необязательна. Игра заканчивается если одна из фишек достигает конечной вершины.

Написать программу поиска кратчайшего пути до конечной вершины, если он существует.

2. Некоторые школы связаны компьютерной сетью. Между школами заключены соглашения: каждая школа имеет список школ-получателей, которым она рассылает программное обеспечение всякий раз, получив новое бесплатное программное обеспечение (извне сети или из другой школы). При этом, если школа B есть в списке получателей школы A, то школа A может не быть в списке получателей школы B.

Требуется написать программу, определяющую минимальное количество школ, которым надо передать по экземпляру нового программного обеспечения, чтобы распространить его по всем школам сети в соответствии с соглашениями.

Кроме того, надо обеспечить возможность рассылки нового программного обеспечения из любой школы по всем остальным школам. Для этого можно расширять списки получателей некоторых школ, добавляя в них новые школы. Требуется найти минимальное суммарное количество расширений списков, при которых программное обеспечение из любой школы достигло бы всех остальных школ. Одно расширение означает добавление одной новой школы-получателя в список получателей одной из школ.

3. Задан неориентированный граф. При прохождении по некоторым ребрам некоторые (определенные заранее) ребра могут исчезать или появляться.

Найти кратчайший путь из вершины с номером q в вершину с номером w.

4. Заданы два числа N и M (20≤M≤N≤150), где N - количество точек на плоскости. Требуется построить дерево из M точек так, чтобы оно было оптимальным. Дерево называется оптимальным, если

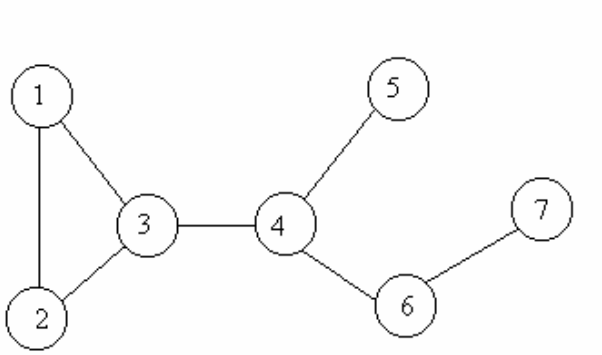
сумма всех его ребер минимальна. Все ребра - это расстояния между вершинами, заданными координатами точек на плоскости.

5. Даны два числа N и M. Построить граф из N вершин и M ребер. Каждой вершине ставится в соответствие число ребер, входящих в нее. Граф должен быть таким, чтобы сумма квадратов этих чисел была минимальна.

6. Задан ориентированный граф с N вершинами, каждому ребру которого приписан неотрицательный вес. Требуется найти простой цикл, для которого среднее геометрическое весов его ребер было бы минимально.

Примечание. Цикл называется простым, если через каждую вершину он проходит не более одного раза, петли в графе отсутствуют.

7. Компьютерная сеть состоит из связанных между собой двусторонними каналами связи N компьютеров (N не превосходит 50, а количество каналов N+5), номера которых от 1 до N. Эта сеть предназначена для распространения сообщения от центрального компьютера всем остальным. Компьютер, получивший сообщение, владеет им и может распространять его дальше по сети. Запрещается передавать сообщения одному и тому же компьютеру дважды. Время передачи сообщения по каналу связи занимает одну секунду, при этом передающий и принимающий компьютеры заняты на все время передачи данного сообщения. На рисунке приведен возможный вариант такой сети.




В начальный момент времени центральный компьютер может передать сообщение одному из непосредственно связанных с ним компьютеров, т.е. соседу. После окончания передачи, этим сообщением будут владеть оба компьютера. Каждый из них может передать сообщение одному из своих соседей и так далее, пока все компьютеры не будут владеть сообщением. Для сети, показанной на рисунке, возможный порядок распространения сообщения от центрального компьютера с номером 6 имеет вид:

Секунда 1: 6→4

Секунда 2: 4→3

6→7

Секунда 3: 3→1

4→5

Секунда 4: 3→2

Написать программу определения и вывода порядка передачи сообщения за минимальное время.

8. Внутрь квадрата с координатами левого нижнего угла (0,0) и координатами верхнего угла (100, 100) поместили N (1≤N≤30) квадратиков. Необходимо найти кратчайший путь из точки (0,0) в точку (100,100), который бы не пересекал ни одного из этих квадратиков. Ограничения:

- длина стороны каждого квадратика равна 5;
- стороны квадратиков параллельны осям координат;
- координаты всех углов квадратиков - целочисленные;
- квадратики не имеют общих точек.

9. Задан неориентированный граф с N вершинами, пронумерованными целыми числами от 1 до N. Написать программу, которая последовательно решает следующие задачи:

- выясняет количество компонент связности графа;
- находит и выдает все такие ребра, что удаление любого из них ведет к увеличению числа компонент связности;
- определяет, можно ли ориентировать все ребра графа таким образом, чтобы получившийся граф оказался сильно связным;
- ориентирует максимальное количество ребер, чтобы получившийся граф оказался сильно связным;
- определяет минимальное количество ребер, которые следует добавить в граф, чтобы ответ на третий пункт был утвердительным.

10. Задан ориентированный граф с N (1≤N≤33) вершинами, пронумерованными целыми числами от 1 до N. Напишите программу, которая подсчитывает количество различных путей между всеми парами вершин графа.

11. Ребенок нарисовал кружки и некоторые из них соединил отрезками. Кружки он пометил целыми числами от 1 до N (1≤N≤30), а на каждом отрезке поставил стрелочку. Затем он приписал каждому кружочку

вес в виде некоторого целого числа и определил начальный и конечный кружочки. Из первого он должен выйти, а во второй попасть.

Ребенок решил для себя следующее:

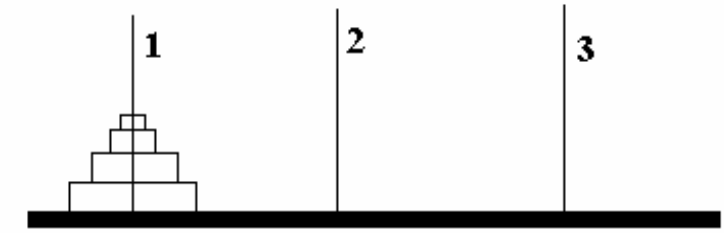
- набрать максимально возможное суммарное количество очков;
- по каждому отрезку пройти ровно один раз;
- если в кружок он попадает при движении по направлению стрелки, то к суммарному количеству очков вес этого кружка прибавляется;
- если в кружок он попадает при движении против направления стрелки, то из суммарного количества очков вес этого кружка вычитается.

Написать программу, которая бы помогла ребенку построить путь, удовлетворяющий всем этим требованиям.

# Глава 4. ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ

## 4.1. Олимпиада - 89

r89_1 В конце XIX века в Европе появилась игра под названием "Ханойские башни". Реквизит игры состоит из 3 игл, на которых размещается башня из колец. Цель игры - перенести башню с левой иглы(1) на правую(3), причем за один раз можно переносить только одно кольцо, кроме того, запрещается помещать большее кольцо над меньшим. Составить алгоритм решения данной задачи (количества колец - N<10).




r89_2 Составить алгоритм поиска заданного слова в тексте. Слово и текст являются массивами символов заданной длины. Если заданное слово присутствует в тексте, то алгоритм должен возвращать номер позиции совпадения, в противном случае - значение -1.

r89_3 Составить алгоритм заполнения прямоугольной таблицы размером N*N целыми числами от 1 до N*N по спирали.

Пример для N=5.

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 16 | 17 | 18 | 19 | 6 |
| 15 | 24 | 25 | 20 | 7 |
| 14 | 23 | 22 | 21 | 8 |
| 13 | 12 | 11 | 10 | 9 |

r89_4 По кругу стоят N спортсменов с номерами от 1 до N. Начиная с какого-то человека, по кругу удаляется каждый k-ый спортсмен. После каждого удаления круг смыкается. Составить алгоритм определения последовательности номеров удаляемых спортсменов.

r89_5 Дан какой-то язык программирования, в котором, кроме стандартных конструкций исполнителя, есть еще функция СИМВОЛ (переменная, N). Она выделит N-символ из переменной. Например, СТРОКА:='АБВГДЕЙКА' и А:=СИМВОЛ(СТРОКА,6). Тогда переменная А будет иметь значение Е. Написать программу нахождения первого появления слова КОТ в произвольной переменной СТРОКА, состоящей из 50 символов.

r89_6 Составить алгоритм подсчета количества способов, которыми можно разменять рубль медными монетами (достоинством 1, 2, 3, 5 копеек).

r89_7 Составить алгоритм поиска всех двузначных чисел, сумма цифр которых не меняется при умножении числа на 2, 3, 4, 5, 6, 7, 8, 9.

r89_8 По правилам машинописи и полиграфии после запятой в тексте всегда должен стоять пробел (код 32). Составить алгоритм для исправления ошибок (отсутствия пробела после запятой). Текст является массивом символов произвольной длины, заканчивающийся нулевым символом (код 0 - признак конца текста).

r89_9 Составить алгоритм заполнения таблицы A(10,10) единицами и минус единицами в шахматном порядке.

о89_1 Составить алгоритм преобразования строки S в эквивалентное ей вещественное число. Алгоритм должен обрабатывать необязательный знак и вещественную точку, а также целую и дробную части, каждая из которых может как присутствовать, так и отсутствовать. Представление числа в строке может содержать ведущие пробелы и заканчиваться любым нецифровым символом. При решении считать строку массивом целых, содержащим коды символов. Для получения кода символа его следует заключить в апострофы ('). Например, строка " -327.45A", представленная в виде массива,

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| ‘ ‘ | ‘ ‘ | ‘-‘ | ‘3’ | ‘2’ | ‘7’ | ‘.’ | ‘4’ | ‘5’ | ‘A’ |

преобразуется в число -327.45 .

При выполнении преобразований считать, что 'N'-'0'=N для N от 0 до 9.

Примечания:

- при реализации алгоритма на Бейсике недопустимо использование функции VAL( );
- максимальный размер массива - 20.

о89_2 Натуральное число называется совершенным, если оно равно сумме всех своих делителей, не считая его самого. Вывести все совершенные числа, меньшие, чем заданное M.

о89_3 Дано натуральное число M и целочисленный массив A[1..M]. Сосчитать и вывести количество различных чисел в массиве.

о89_4 Даны два целочисленных массива A[1..N] и B[1..M]. Причем элементы A образуют монотонно возрастающую (A[1]<=A[2]<=...<=A[N]) последовательность, а B - монотонно убывающую (B[1]>=B[2]>=...>=B[M]). Сформировать массив S[1..K], содержащий все элементы массивов A и B и образующий монотонно возрастающую последовательность.

о89_5 Заданы прямоугольные координаты X1, Y1, X2, Y2, X3, Y3 вершин треугольника и координаты X и Y точки. Определить, находится ли точка в треугольнике.

о89_6 Функция f(n) для целых неотрицательных n определена так: f(0)=0, f(1)=1, f(2*n)=f(n), f(2*n+1)=f(n)+f(n+1). Для данного n найти и вывести f(n). Учесть, что n столь велико, что недопустимо вводить массив из n чисел, равно как и массив, длина которого растет с ростом числа n, поэтому в решении использование массива недопустимо.

о89_7 На плоскости N точек заданы своими координатами $A_1[X_1,Y_1]$, $A_2[X_2,Y_2]$, ..., $A_N[X_N,Y_N]$. Найти диаметр этого множества, т.е. максимальное расстояние между его точками.

о89_8 Упростить логическое выражение: Y=(((A и B) или не(B)) и (не(A) или (A и B))) или (не (B и A) и A).

о89_9 Построить правильный многоугольник со стороной, равной A, и центром в центре экрана.

о89_10 Имеется прямоугольный массив A[21,21], заполненный нулями. Построить в массиве элементами, равными единице, окружность радиуса R, не превышающего 10. Центр окружности зафиксирован в элементе A[11,11].

## 4.2. Олимпиада - 90

г90_1 На сельской улице живут Ивановы и Петровы. Необходимо, используя минимальное число обменов, расселить их так, чтобы Ивановы жили с одного конца улицы, а Петровы - с другого.

г90_2 Так как экран компьютера представляет точечный растр, произвольная прямая линия на экране состоит из некоторого множества горизонтальных отрезков - штрихов. Составьте алгоритм (программу) для построения на экране произвольного отрезка прямой с координатами: X0, Y0 - координаты первой точки отрезка; X1, Y1 - координаты последней точки отрезка. Внимание: ваш компьютер имеет команды только для построения точек или горизонтальных и вертикальных штрихов.

г90_3 У кур и кроликов N ног. Сколько среди них кур, сколько кроликов? Составить алгоритм, который перечисляет все возможные комбинации.

г90_4 Какое значение примет X после выполнения алгоритма:

```
алг ALFA(нат N, вещ таб A[1..N], вещ X)
арг N,A; рез X
        нач
        если N=1
                то X:=A[1]
                иначе ALFA(N-1,A,X)
                        если A[N]>X
                                то X:=A[N]
                        все
        все
кон
```

г90_5 Дана линейная таблица A, состоящая из N элементов. Сформировать таблицу B, содержащую неповторяющиеся элементы таблицы A.

o90_1 Найти все простые числа в интервале от 1000 до 2000. Оценивается не только решение задачи, но и время (в количестве операций сравнения).

o90_2 На шахматной доске стоят ферзь и конь. Каждая фигура задана своими координатами (номера клетки по горизонтали и вертикали). Определить количество полей, которые находятся под боем.

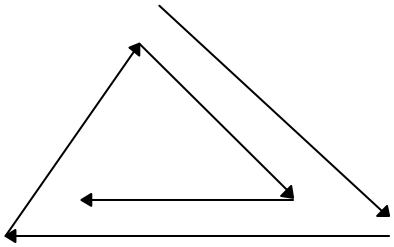
o90_3 Некто родился в XIX веке. В 1901 г. сумма цифр числа лет, прожитых им, равнялась сумме цифр года его рождения. Составить программу определения его возраста.

o90_4 Написать программу расшифровки сообщения, закодированного по описанному ниже принципу. Пусть дан шифр (набор цифр 432513) и текст - " настоящий виновник кражи алмазов". Записываем текст без пробелов и под ним цифры шифра:

настоящийвиновникккражиалмазов
43251343251343251343251343251
сгучпвэллжйртепнлнфгинборгйуг .

Каждая буква алфавита заменяется на букву, номер которой равен номеру исходной буквы в алфавите плюс цифра, стоящая под ней. При этом буква с номером 33+К есть буква с номером К.

o90_5 Составить алгоритм расстановки натуральных чисел в правильный треугольник по следующей схеме:



Количество чисел на стороне вводится(N<14).

Пример: N=6.

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

o90_6 Ввести координаты вершин произвольного многоугольника (без самопересечений и самокасаний сторон), который должен полностью принадлежать некоторому прямоугольнику. Построить многоугольник и оценить его площадь по отношению к площади прямоугольника.

o90_7 Интроспективными называются программы, результатом которых является свой собственный текст, выведенный на дисплей. Составьте ( и выполните) такую программу, не прибегая к системным "штучкам" типа:

10 LIST

Программа должна работать честно: выведенный текст должен в точности соответствовать исходному; недопустимо использовать средства операционной системы (монитора), для вывода использовать стандартные средства выбранного языка программирования.

Критерий оценки: размер исходного текста программы.

## 4.3. Олимпиада - 91

r91_1 Дан алгоритм:

```
алг задание 1(цел М)
 арг М
 нач цел А,К
          К:=М;А:=2
          пока К<>1
           нц
                    если остаток(К,А)=0
                     то вывод А;К:=К/А
                     иначе А:=А+1
                    все
           кц
 кон
```

Какие числа будут напечатаны при исполнении алгоритма "задание 1" для М>1.

r91_2 Составить программу нахождения и вывода всех целых четырехзначных чисел, сумма цифр которых равна 22.

r91_3 Товарный поезд состоит из одного или двух локомотивов впереди состава, одной или более платформ, за платформами в конце состава следует вагон с охраной. Написать алгоритм распознавания товарного поезда.

r91_4 Составить алгоритм заполнения прямоугольной таблицы размером N на M (M>N) вложенными рамками, каждая из которых изображается числом, равным разности между N и номером рамки. Примечание: внешнюю рамку считать первой.

Таблицу вывести на экран. Пример для N=5, M=8:

44444444
43333334
43222234
43333334
44444444

r91_5 Составить программу перевода обыкновенной дроби в периодическую. Примеры: 1/3=0.(3), 4/5=0.8( ), 1/6=0.1(6), 3/14=0.2(142857).

o91_1 Дана строка из шести произвольных различных символов. Разработать программу вывода всех возможных подмножеств (подстрок), составленных из символов данной строки. Каждый символ строки входит в подмножество не более одного раза.

o91_2 Новобранцы выстроились в ряд, и старшина скомандовал: "Направо!", после чего каждый повернулся в произвольном направлении на 90 градусов, но, увидев лицо соседа, тут же повернулся на 180 градусов. Перестанут ли они когда-нибудь вертеться? Результат обосновать. Составить программу, иллюстрирующую эту ситуацию.

o91_3 Написать программу возведения в натуральную степень натурального числа N (1<N<30).

o91_4 Составить программу обхода конем шахматной доски (Z*Q). Результат отобразить на экране в следующем виде: (для доски 3*4)

| | | | |
|---|---|---|---|
| 8 | 11 | 6 | 3 |
| 1 | 4 | 9 | 12 |
| 10 | 7 | 2 | 5 |

На каждом шахматном поле конь может побывать только по одному разу.

o91_5 Имеется N карточек. На каждой стороне каждой карточки записано целое число. Известно, что каждое из чисел 1,2,...,N встречается на карточках дважды. Требуется узнать, можно ли карточки выложить так, чтобы каждое из чисел 1,2,...,N было на верхней стороне одной из карточек. Если можно, то указать для каждой карточки, как ее класть.

o91_6 Числовой прямоугольник имеет N рядов и M столбцов: в каждую его клетку случайным образом заносится натуральное число из интервала от 1 до N*M. Каждая клетка может обменять свое значение с соседней правой или левой, верхней или нижней клетками.

Составить программу, которая:

- позволяет сменить значения двух указанных клеток;
- располагает между двумя указанными клетками числа по возрастанию в строках, считая первую указанную клетку начальной, а вторую конечной;
- располагает все числа в порядке возрастания в строках, начиная с первой клетки.

Программа должна демонстрировать на экране все обмены.

## 4.4. Олимпиада - 92

r92_1 Для каждой буквы заданного текста указать, сколько раз она встречается в тексте.

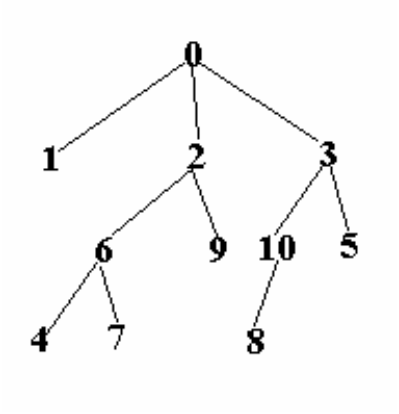
r92_2 Дана квадратная матрица A[10,10] и два непустых множества $S_1$ и $S_2$, состоящие из каких-то элементов от 1 до 10, например $S_1$=[7,2,6] и $S_2$=[5,8,6,5]. Вычислить сумму тех элементов матрицы, номера строк и столбцов которых принадлежат соответственно множествам $S_1$ и $S_2$. Если в $S_1$ или $S_2$ один элемент встречается несколько раз, то соответствующие элементы A должны входить в сумму только один раз.

r92_3 Заданы целые числа $A_1$, $A_2$, ..., $A_n$, $A_{(n+1)}$, (n<=10). Определить, имеет ли уравнение $A_1*X_1+A_2*X_2+...+A_n*X_n=A_{(n+1)}$ хотя бы одно решение, при котором каждая из переменных $X_1,X_2,...,X_n$ равна нулю или единице.

r92_4 Имеется N населенных пунктов, перенумерованных от 1 до N (N=10). Некоторые пары пунктов соединены дорогами. Определить, можно ли попасть по этим дорогам из первого пункта в N-й. Если да, то найти несколько маршрутов.

o92_1 Заданы три натуральных числа A,B,N. Найти все натуральные числа, не превосходящие N и не равные нулю, которые можно представить в виде суммы (произвольного числа) слагаемых, каждое из которых A или B. Каждое найденное число должно быть выведено на экран не более одного раза.

o92_2 Операционные системы типа UNIX, PC-DOS (старшие версии) поддерживают древовидную структуру хранения данных на диске. При этом минимальная именованная единица информации называется файлом. Файлы объединяются в каталоги, каждый из которых тоже имеет имя. Множество каталогов и отдельных файлов могут в свою очередь также образовать каталог более высокого уровня. Каталог самого высокого уровня называется корневым. Количество уровней вложенности не ограничено. Пример.




На данной схеме элемент 0 является корневым каталогом; элементы 2,3,6,10 - каталогами; элементы 1,4,7,9,8,5 - файлами.

Здесь и далее будем рассматривать только случаи без пустых каталогов (т.е. каталогов, не имеющих файлов).

Составить программу для вывода на экран по уровням:

- списка всех элементов данных, "вложенных" в произвольный каталог на всех уровнях;
- списка вех каталогов, "вложенных" в произвольный каталог;
- списка всех файлов, "вложенных" в произвольный каталог.

Пример. Для каталога 2 первое задание:

2
6
4
7
9

Примечание: для всех заданий вместо номеров использовать имена (текстовые значения).

o92_3 В картинной галерее каждый сторож работает в течение некоторого времени. Расписанием стражи называется множество пар $[T_1(i),T_2(i)]$ - моменты начала и конца дежурства i -го сторожа из интервала [0,T].

Для заданного расписания стражи требуется проверить, в любой ли момент в галерее находится не менее двух сторожей и если это условие не выполняется, то:

- перечислить все интервалы времени с недостаточной охраной (менее двух сторожей);
- добавить наименьшее число сторожей с заданной, одинаковой для всех длительностью дежурств так, чтобы получить правильное расписание, удовлетворяющее первому условию.

Примечание. Если момент начала дежурства сторожа A совпадает с окончанием дежурства сторожа B, то считать дежурным только сторожа A.

o92_4 Двумерный массив A[10,10] случайным образом "заполняется" 0 и 1, т.е. A[I,J]=0 или 1.

Разработать программу, определяющую, можно ли из элементов A, равных 1, образовать соединяющее множество.

Пояснения:

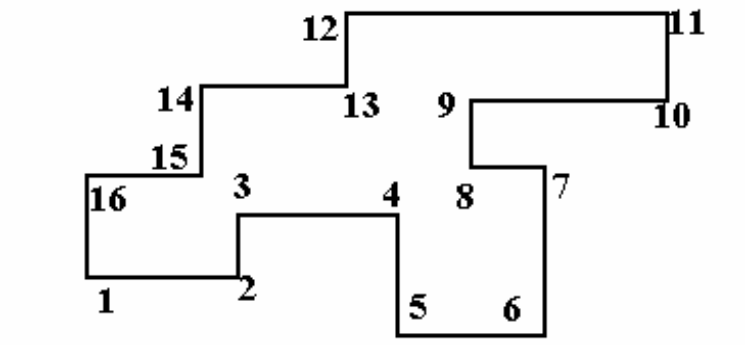
- границами назовем элементы A из первой строки, первого столбца, десятой строки, десятого столбца;
- "соседями" элемента A[i,j] являются A[i-1,j], A[i+1,j], A[i,j-1], A[i,j+1];
- соединяющее множество - это такое множество элементов массива A, равных 1, что, начиная свой "путь" из любого элемента этого множества и переходя к его "соседям", мы можем рано или поздно попасть на каждую из границ.

## 4.5. Олимпиада - 93

r93_1 На координатной плоскости x0y заданы целочисленные координаты n точек $(x_1,y_1),...,(x_n,y_n)$, в которых замкнутая ломаная претерпевает излом.

Выполняются следующие условия:

- номера точек совпадают с номерами "излома";
- ломаная «ломается» только под прямым углом;
- отсутствуют самопересечения или самокасания ломаной.




Определить площадь S области, которую ограничивает ломаная.

Примечание. Можно использовать формулу S=n+m/2-1, где n - количество точек с целочисленными координатами внутри, а m - на границе области.

r93_2 Разработать программу перевода римских чисел в десятичную систему счисления.

r93_3 Обычные алгебраические выражения можно записывать также в обратной польской нотации - записи без скобок (предложил польский математик Ян Лукашевич). Например, A+(B-C)*D-F/(G+H) преобразуется в ABC-D*+FGH+/- или (A+B)*C-D+E/F/(G+H) в AB+C*D-EF/GH+/+.

Разработать программу преобразования выражения в обратную польскую запись и его вычисления в обратной польской записи.

Примечания:

- в выражении используются только натуральные числа;
- выражение заканчивается знаком "=".
- допустимые операторы и их приоритет:

| Оператор | Приоритет |
|---|---|
| * | 3 |
| / | 3 |
| + | 2 |
| - | 2 |
| ( | 1 |
| = | 0 |

o93_1 Пункты с номерами 1,2,...,N (N <=50) связаны сетью дорог, длины которых равны 1. Дороги проходят на разной высоте и пересекаются только в пунктах. В начальный момент времени в некоторых пунктах находятся M роботов. Все роботы начинают двигаться с постоянной скоростью 1. Останавливаться или менять направление они могут только в пунктах.

а)Требуется найти минимальное время $T_1$, через которое все роботы могут встретиться в одном пункте, указать этот пункт или сообщить, что такая встреча невозможна.

б) Если встреча возможна, то найти время $T_2 <= T_1$, через которое встреча может произойти и вне пунктов.

в) Пусть роботам запрещена какая-либо остановка, и скорость равна 1 или 2. При этих условиях найти минимальное время T, через которое произойдет их встреча, или сообщить, что встреча невозможна.

Примечания:

- Для задачи (в) можно указать, что M равно 2 или 3.
- При решении задач (а) и (б) данные о скоростях игнорируются.

o93_2 В таблице N*N, где N<13, клетки заполнены случайным образом цифрами от 0 до 9.

Найти маршрут из клетки A[1,1] в клетку A[N,N], который удовлетворяет следующим условиям:

- состоять из отрезков, соединяющих центры клеток, имеющих общую сторону;
- длина маршрута минимально возможная;
- из всех маршрутов, удовлетворяющих условиям 1 и 2, искомый маршрут тот, сумма цифр в клетках которого максимальна.

Организовать ввод данных для таблицы и вывод маршрута как последовательности пар координат клеток, через которые он проходит (первая координата - номер столбца, вторая - номер строки).

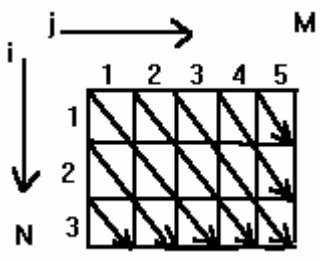
o93_3 Координаты всех домов в городе заданы парой целых чисел, таких, что любой из домов можно однозначно изобразить в виде точки на мониторе компьютера. Мэрия решила выделить средства на постройку кольцевой автодороги вокруг города. Изобразить на экране дорогу минимальной длины в виде прямых отрезков, соединяющих дома, мимо которых должна пройти дорога, при условии, что внутри дороги окажутся все остальные дома города.

o93_4 Бумажная полоска разбита на N<=40 клеток. Двое играющих по очереди выбирают и зачеркивают K<=N пустых смежных клеток. Выигрывает сделавший последний ход. Запросить числа N и K и написать программу:

- позволяющую играть в описанную игру 2 игрокам;
- написать программу, которая бы сама играла либо за первого, либо за второго игрока (программа запрашивает, за кого она будет играть). В обоих случаях каждый ход должен отображаться на экране. В случае 1 программа должна показывать, чей сейчас ход.

## 4.6. Олимпиада - 94

r94_1 Для заданной целочисленной матрицы найти максимум среди сумм элементов диагоналей, параллельных главной диагонали. Программа должна находить решение для любых матриц N*M ($0 \le N, M \le 10$).




r94_2 Найти наименьшее общее кратное всех чисел, содержащихся в заданной последовательности натуральных чисел.

Примечания.

- количество чисел в последовательности N (0<N<=10);

•числа последовательности вводятся с клавиатуры. Наименьшее общее кратное чисел - это наименьшее число, которое без остатка делится на каждое число последовательности.

r94_3 На расстоянии N шагов от магазина винно-водочных изделий стоит Алкоголик. Каждую минуту он выбирает, куда сделать шаг: к магазину или в противоположном направлении. Сколько способов у Алкоголика попасть в магазин, пройдя ровно K шагов?

Ограничения: N<=10, K<=20.

r94_4 Доска 8*8 раскрашена в два цвета: белый и черный. Необходимо пройти из левого нижнего угла в правый верхний так, чтобы цвета клеток перемежались. За один ход разрешается перемещаться на одну клетку по вертикали или горизонтали.

Программа должна (если путь существует):

•находить хотя бы один путь;

•находить путь минимальной длины.

r94_5 Дана матрица N*N(N<=10), состоящая из целых чисел. За одну операцию разрешается изменить знаки всех чисел произвольно выбранной строки или столбца на противоположные. Требуется сделать суммы чисел во всех строках и столбцах неотрицательными.

Программа должна ввести матрицу и вывести в качестве результата последовательность операций. Количество операций должно быть минимальным.

o94_1 Массив A[1..N,1..M] упорядочен по неубыванию по строкам и по столбцам, т.е. A[i,1]<=A[i,2]<=...<=A[i,M] при всех i=1,...,N и A[1,j]<=A[2,j]<=...<=A[N,j] при всех J=1,...,M. Определим C[i,j] как элемент массива C[1..N,1..M], равный количеству операций сравнения при поиске элемента X, равного A[i,j], в массиве A. Эффективность решения задачи поиска будем оценивать (ls) частным от деления суммы элементов массива C на количество элементов (N*M).

Например, если

| | | | |
|---|---|---|---|
| | 1 | 3 | 5 |
| A= | 2 | 4 | 6 |
| | 3 | 7 | 8 |

и мы используем для поиска элементов "лобовой" просмотр массива A, то C будет иметь вид:

| | | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| C= | 4 | 5 | 6 |
| | 7 | 8 | 9, |

а эффективность решения (1+2+3+4+5+6+7+8+9)/9=5.

Разработать программу, обеспечивающую:

•ввод массива A и вывод его на экран;

•эффективный поиск в массиве A (оценивается значением ls);

•вывод массива C и значения ls.

o94_2 Даны две строки символов X и Y. Назовем расстоянием между X и Y (r[X,Y]) количество символов, которыми X и Y различаются между собой.

Например: Если X="abcd", Y="dxxc" то r[X,Y]=4.

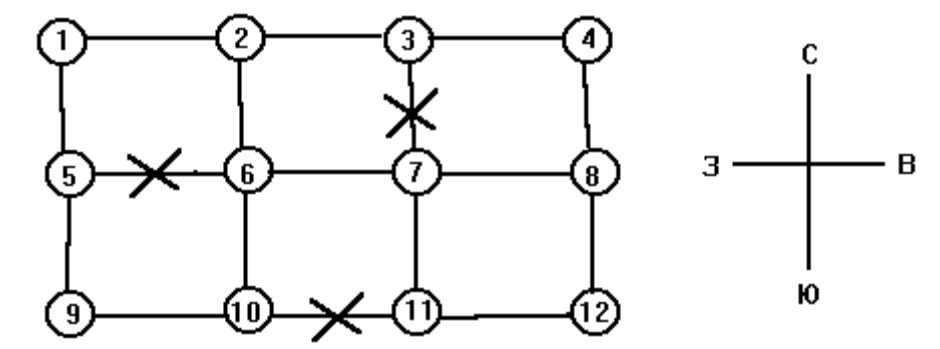
Написать программу, определяющую расстояние между строками.

o94_3 Написать рекурсивную программу вывода на экран следующей картинки:

```
AAAAAAAAA.............AAAAAAAAA   (N раз)
  BBBBBBB.............BBBBBBB      (N-4 раза)
    CCCCC.............CCCCC
      DDD.............DDD
        .............
        III.............III       (N-32 раза)
         J. .............J         (N-36 раз)
        III.............III       (N-32 раза)
        .............
      DDD.............DDD
    CCCCC.............CCCCC
  BBBBBBB.............BBBBBBB     (N-4 раза)
AAAAAAAAA.............AAAAAAAAA   (N раз)
```

Примечание. Значение N определяется исходя из возможностей монитора Вашего компьютера.

o94_4 Черепашка находится в городе, все кварталы которого имеют прямоугольную форму и ей необходимо попасть с крайнего северо-западного перекрестка на крайний юго-восточный. Пример.



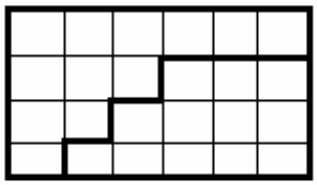

На некоторых улицах проводится ремонт и по ним запрещено движение (например, между 3 и 7 перекрестками, 5 и 6, 10 и 11), длина, а значит и стоимость проезда, по остальным улицам задается. Кроме того, для каждого перекрестка определена стоимость поворота. Так, если Черепашка пришла на 7-й перекресток и поворачивает к 11-му, то она платит штраф, а если идет в направлении 8-го, то платить ей не приходится. Найти для Черепашки маршрут минимальной стоимости.

Исходные данные:

- N - количество перекрестков, определяется через два числа l,m и N=l*m (1<l,m<11);
- Длина улиц (стоимость проезда) и стоимость поворота на перекрестках - целые числа.

o94_5 Написать рекурсивную программу представления натурального числа N (0<N<=8) суммой натуральных чисел. Перестановка слагаемых нового варианта не дает.

o94_6 Задано прямоугольное клеточное поле N*M (2<=N,M<8) и число k.



Построить на экране k различных непрерывных разрезов этого поля на два клеточных поля равной площади. Разрез должен проходить по границам клеток. Пример разреза.

Найти все возможные непрерывные разрезы на поля равной площади.

o94_7 Даны две непустые строки символов X и Y. Строчными (маленькими) буквами обозначаются переменные. Переменные могут быть только в строке X. Значением переменной может быть непустая строка символов без строчных букв.

Задача заключается в определении значений переменных так, чтобы после подстановки их значений в строку X она совпала со строкой Y.

Пример 1. X="tGqtt", Y="RIGPORYRIRI".

Совпадение возможно, если t="RI", q="PORY"&

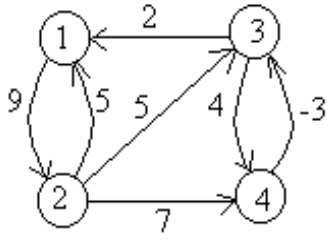
Требования:

- X и Y содержат не более 20 символов.
- Программа должна определять количество решений N и выводить на экран эти решения.
- Если нет решений, то программа должна выдавать сообщение "Решений нет".

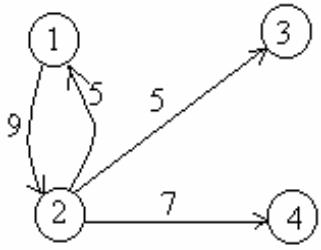
o94_8 В организации N рабочих (2<N<31). Каждый из них оказывает влияние на некоторое количество других рабочих (не обязательно всех). Влияние оценивается некоторым положительным числом. Так, если рабочий с номером i влияет на рабочего с номером j, то степень влияния это число C[i,j]. Итак, совокупность влияний можно описать матрицей N*N. Нулевой элемент матрицы - нет влияния.

Руководитель организации решил упорядочить влияния, считая, что тем самым он повысит производительность труда. Для исследования того, как это сделать, он пригласил социолога.

Первое предложение социолога заключалось в том, чтобы выделить совокупность связей (влияний) максимального веса, при этом на каждого рабочего должен влиять только один человек, все рабочие должны быть охвачены связями, а остальные связи (влияния) запрещаются в директивном порядке на время работы. Пример работы социолога.

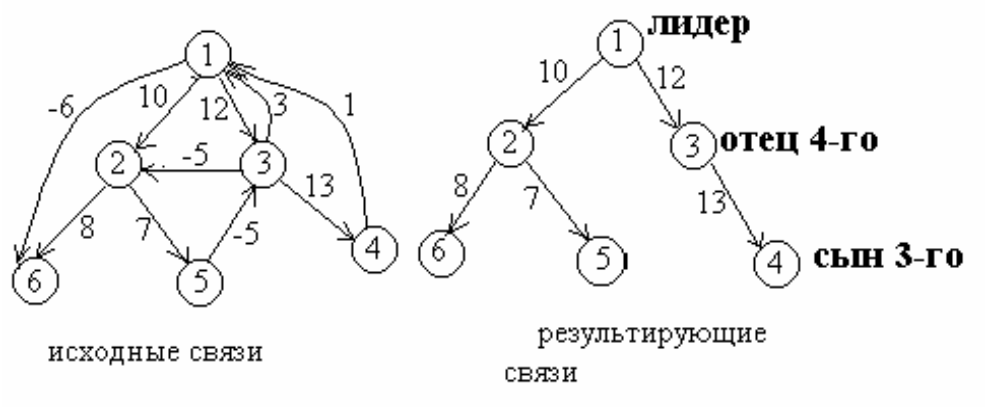



исходные связи




результирующие связи

Второе предложение социолога заключалось в поиске лидера и организации связей так, как это показано на следующем рисунке (если это можно сделать). На лидера не влияет никто. На каждого рабочего может влиять только один человек, в то время как он может влиять на любое количество рабочих. Если рабочий с номером i оказывает влияние на рабочего с номером j, то ни один рабочий с номером j и ни один из его "СЫНОВЕЙ" не может оказывать влияния на рабочего с номером i и на всех его "ПРЕДКОВ". При этом следует выбирать вариант с максимальным суммарным весом связей.




Разработать программу, которая должна обеспечивать помощь социологу в том и другом случаях.

## 4.7. Олимпиада - 95

r95_1 Найти все натуральные числа, не превосходящие заданного N, десятичная запись которых есть строго возрастающая или строго убывающая последовательность цифр.

r95_2 Дана матрица (таблица) A из N строк и M столбцов (2<N,M<12). Элементы матрицы равны 0 или 1. Разрешенная операция - перестановка столбцов. Преобразовать матрицу A так, чтобы первыми в ней были столбцы с единицей в первой строке, затем столбцы с единицей во второй строке (если такие есть) и т. д.

Дополнительную матрицу вводить не разрешается.

| Пример. | |
|---|---|
| 0 1 0 1 0 0 | 1 1 0 0 0 0 |
| 1 0 0 1 0 0 | 1 0 1 0 0 0 |
| 1 1 1 1 0 0 | 1 1 1 1 0 0 |
| 1 0 0 0 1 1 | 0 0 1 0 1 1 |
| 0 0 1 0 0 1 | 0 0 0 1 0 1 |

r95_3 Исходные данные - строка (не более 254 символов) из круглых скобок и знаков вопроса. Данные корректны. Составьте программу, печатающую все правильные скобочные выражения, которые можно восстановить из входной строки заменой знаков вопроса на скобки, либо сообщающую, что решения нет.

Пример 1. Исходная строка: ((??)?

Вывод программы: (((())

(()())

Пример 2. Исходная строка: )?

Вывод программы: восстановить невозможно.

r95_4 Ряд чисел Фибоначчи представляет собой последовательность натуральных чисел, такую, что первое и второе числа равны единице, а каждое следующее равно сумме двух предыдущих: 1 1 2 3 5 8 13 21 34 ... Числа Фибоначчи выписываются одно за другим вплотную. Определите, какой будет 1994-я цифра в такой последовательности. (Обратите внимание, что соответствующее число Фибоначчи может оказаться весьма большим.)

Входных данных нет. На выходе должно быть: фраза "1994 цифра равна" и искомая цифра.

r95_5 На шахматной доске N*N расставить N*N ферзей N цветов (N≤8) так, чтобы ферзи одного цвета не били друг друга. Найти по одному варианту расстановки для значения N.

Входные данные - значение N. На выходе таблица расстановки, если она существует.

o95_1 Перечислить все последовательности длины 2N, составленные из N единиц и N минус единиц, у которых сумма любого количества элементов, начиная с 1-го, неотрицательна (число минус единиц не превосходит числа единиц).

Входные данные**:** значение N (N≤10)

Выходные данные**:** последовательности, удовлетворяющие условию задачи.

| Пример. N=3 |
|---|
| 1-11-11-1 |
| 1-111-1-1 |
| 11-1-11-1 |
| 11-11-1-1 |
| 111-1-1-1 |

o95_2 На шахматной доске стоят белые король и ферзь и

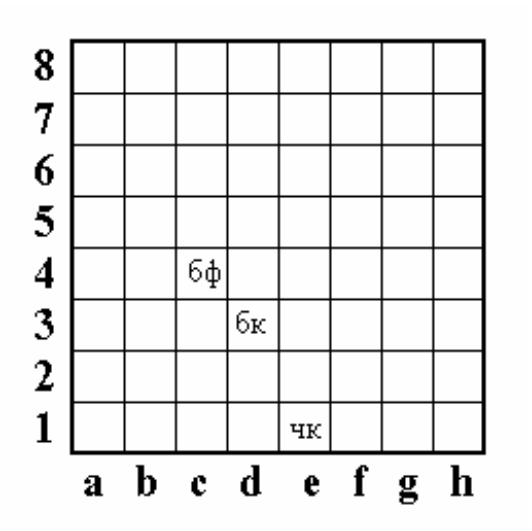
черный король. Белый король не перемещается по шахматной доске. Написать программу, которая, играя белыми, ставит мат черному королю за минимальное число ходов при любом начальном расположении фигур.

Начальные позиции фигур вводятся с клавиатуры в соответствии с правилами шахматной нотации (латинская буква **a**..**h** и цифра **1**..**8**).

Программа не должна допускать ввода некорректных позиций (например, король не может изначально находиться под боем).

Программа начинает игру белыми. Ответные ходы короля вводятся с клавиатуры. На экране должна быть изображена шахматная доска, на которой отображается ход игры (допускается схематичное изображение доски и фигур в текстовом режиме).

Качество игры оценивается по количеству ходов до мата в серии стандартных начальных позиций. За каждый лишний ход сверх минимально возможного количества оценка за тест уменьшается вдвое.




о95_3 Автостоянка имеет форму клеточного поля N*M. Одна клетка на границе является выездом с автостоянки. Любой автомобиль занимает две соседние клетки (по горизонтали или по вертикали, но не по диагонали!).

Написать программу поиска размещения максимального количества автомобилей, такого, что любой из автомобилей имеет возможность выезда.

Автомобиль может выехать, если у его “границ” есть хотя бы одна свободная клетка и, перемещаясь по свободным клеткам, он может “добраться” до выезда.

Входные данные. Натуральные числа N и M(N, M≤100) и координаты клетки выезда.

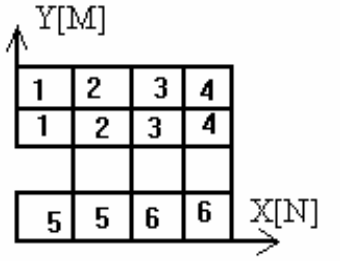
Выходные данные. Число размещенных автомобилей.

Пример.

N, M → 4, 4

клетка выезда → 1, 2

Количество автомобилей → 6




о95_4 Племя из M миссионеров и L людоедов находится по одну сторону реки, через которую необходимо переправиться. В распоряжении имеется одна лодка, которая может выдержать вес только K представителей этого племени (все имеют одинаковый вес). Кроме того, если в какой-то момент времени число людоедов станет больше числа миссионеров, миссионеры будут съедены независимо от того, на каком берегу или в лодке это случится.

Написать программу поиска способа переправы этого племени, если он существует, состоящего из минимального количества перегонов лодки через реку.

Ограничения:

- M, L, K - натуральные числа(M, L≤10000, K≤6);
- лодка сама по себе переправляться через реку не может;
- любой представитель племени способен перегнать лодку с одного берега на другой.

Входные данные - значения M, L, K.

Выходные данные - число перегонов лодки (P).

Пример:

Ввод

(M, L, K) - 3 3 2.

Ответ - 11.

(3, 3, 1) → (2, 2, 0) → (3, 2, 1) → (3, 0, 0) → (3, 1, 1) → (1, 1, 0) → (2, 2, 1) → (0, 2, 0) → (0, 3, 1) → (0, 1, 0) → (1, 1, 1) → (0, 0, 0)

Примечания:

- третье число в примере - признак “где лодка”;
- первые два числа в примере означают число миссионеров и людоедов на исходном берегу;
- вывод способа переправы при значениях M и L меньших 10 оценивается дополнительными баллами.

## 4.8. Олимпиада - 96

r96_1 Переставить две части массива A из n элементов, первая часть - элементы с номерами от 1 до m, вторая - от m+1 до n. При этом порядок элементов в каждой из частей должен быть сохранен и нельзя использовать дополнительные массивы.

Пример. n=9, m=5

Вход: 9 4 7 2 3 5 8 1 6  Выход: 5 8 1 6 9 4 7 2 3

r96_2 Для надежности некоторый текст был передан по линии связи трижды, но каждый раз ровно один символ был принят в искаженном виде. Требуется по трем полученным текстам восстановить исходный текст или установить, что сделать это невозможно.

r96_3 Дан массив N*M, в клетках которого произвольным образом поставлены числа от 1 до N*M. Горизонтальным ходом называется такая перестановка любого числа элементов массива, при которой каждый элемент остается в той строке, в которой он был. Вертикальным ходом называется такая перестановка любого числа элементов массива, при которой каждый элемент остается в том столбце, в котором он был. Разрешаются только горизонтальные и вертикальные ходы.

Разработать программу, которая за возможно меньшее количество ходов расставляет элементы массива по порядку, т.е. в первой строке - от 1 до M, во второй строке - от M+1 до 2M, и т. д.

Примечание. Программа должна выводить массив до каждого хода и после хода.

r96_4 В городе прямоугольной формы, расположенном в холмистой местности, все улицы идут либо с юга на север (N улиц), либо с востока на запад (M улиц), так что все кварталы являются квадратами со стороной 1. Каждый из участков улиц между соседними перекрестками имеет либо только спуск, либо только подъем, либо горизонтален. Матрица A[i,j] (i=1..M, j=1..N) определяет высоту перекрестков над уровнем моря.

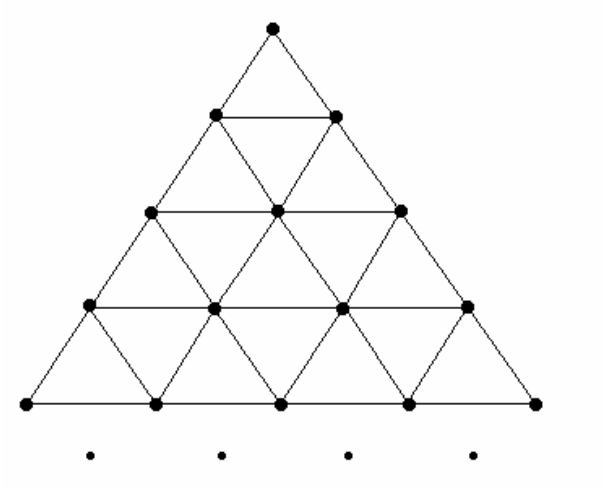
Разработать программу, которая:

- запрашивает размерность матрицы (числа M и N):
- вводит элементы матрицы A и координаты двух перекрестков X и Y:
- определяет, можно ли спуститься из X в Y или из Y в X, непрерывно двигаясь под уклон, и если ответ положительный, то находит маршрут, состоящий из минимального числа кварталов.

o96_1 Напишите программу, которая ищет и выводит на экран все решения числового ребуса ДВА * ДВА = ЧИСЛО.

Здесь, как обычно, одинаковым буквам соответствуют одинаковые цифры,  разным - разные, а первая цифра числа не может быть нулем.

o96_2 Рассмотрим  точки на бесконечной сетке из равносторонних треугольников, показанной ниже:



Пронумеруем отмеченные точки слева направо и сверху вниз. Заметим, что некоторые группы этих точек являются вершинами определенных геометрических фигур. Например, множества точек {1,2,3} и {7,9,18} являются вершинами треугольников, множества {11,13,26,24} и {2,7,9,17} - вершинами параллелограммов, а множества {45,9,13,12,7} и  {8,10,17,21,23,32,34} - вершинами шестиугольников.

Задание. Напишите программу, которая вводит с клавиатуры множество точек этой треугольной сетки, анализирует его и определяет, являются ли эти  точки вершинами одной из следующих фигур: "правильного" треугольника, "правильного" параллелограмма или "правильного" шестиугольника. Многоугольник называется "правильным", если все его стороны идут по ребрам треугольной сетки и имеют равную длину. Предусмотрите  возможность многократного тестирования Вашей программы.

Возможен следующий диалог с Вашей программой:

Число точек: 3

Номера: 1 2 3

Это вершины правильного треугольника.

Число точек: 4

Номера: 11 13 29 31

Эти точки не являются вершинами никакой правильной фигуры.

Число точек: 4

Номера: 26 11 13 24

Эти вершины правильного параллелограмма.

Примечание. Число точек в множестве не будет превышать шести, номера всех точек принадлежат диапазону {1, 32767}

o96_3 2N-значное число называется “*счастливым*”, если сумма его первых N цифр равняется сумме последних N цифр. Напишите программу, которая определяет количество 2N-значных (N≤10) “*счастливых*” чисел в диапазоне [A,B], где A и B - заданные натуральные числа, имеющие в десятичной записи не более двадцати цифр. Учтите, что “счастливое” число может начинаться и с нуля.

Пример работы программы:

Введите N: 3

Введите A: 1

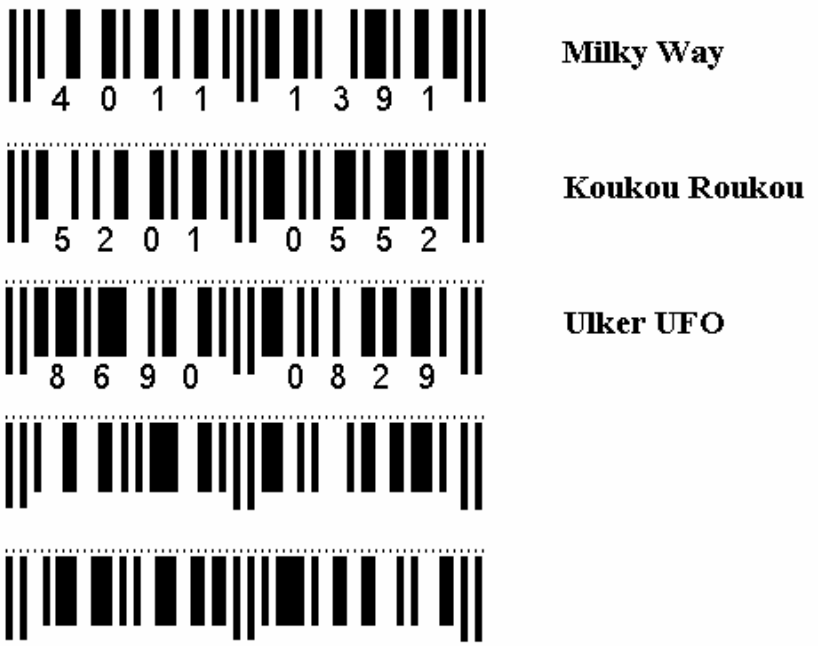
Введите B: 999999

Число счастливых чисел: 55251

o96_4 Существуют натуральные числа, оканчивающиеся на цифру N, такие, что перенесение цифры N в начало числа приводит к увеличению числа в N раз. Например, число 102564, N=4, 410256=102564*4.

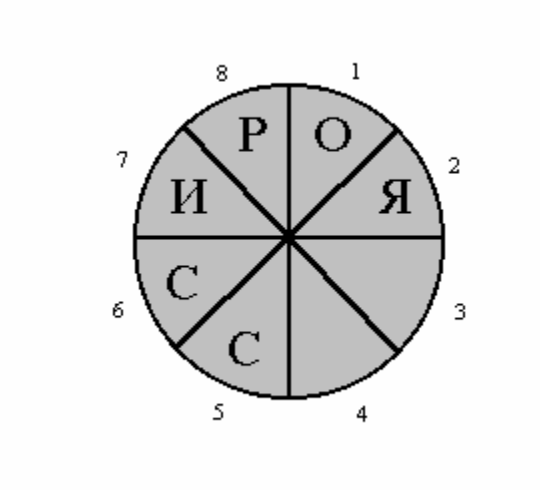
Найдите наименьшие натуральные числа, удовлетворяющие заданному условию при N=2, 3, 5, 6, 7, 8, 9.

o96_5 Штриховой код предназначен для автоматизации учета продажи товаров. При считывании кода светочувствительный элемент движется вдоль штриховой полоски, причем результат считывания не зависит от направления движения. Ниже приведены коды трех всем известных лакомств с расшифровками и два кода без расшифровок. Расшифруйте их.




Напишите программу, расшифровывающую произвольный штриховой код, введенный с клавиатуры. Штриховой код вводится в виде строки, состоящей из цифр от 1 до 4, соответствующих толщине линий. Например, код "Milky Way" можно ввести в виде строки 1111132321122212221111112221141131122221111 (слева направо) или 1111222211311411222111111222122211232311111 (справа налево). В обоих случаях программа должна вывести на экран строку 40111391.

o96_6 Круг разбили на 8 секторов. Какие-то два соседних сектора пусты, а в остальных расположены буквы Р, О, С, С, И, Я в некотором порядке (по одной в секторе).




За один ход разрешается взять любые две подряд идущие буквы и перенести их в пустые сектора, сохранив порядок. При этом сектора. которые занимали перенесенные буквы до хода, становятся пустыми. Номером *хода* назовем номер первого из секторов (в порядке обхода по часовой стрелке (ЧС), содержимое которого переносится.

Состояние круга назовем *правильным,* если начиная с сектора 1 по ЧС можно прочесть слово "РОССИЯ". При этом местоположение пустых секторов может быть произвольным. Состояние круга можно закодировать строкой из 8 символов, получающихся при чтении круга по ЧС, начиная с сектора 1, если пустому сектору поставить в соответствие символ подчеркивания '_'. Эту строку из 8 символов будем называть *кодом* круга.

Пункт 1. Требуется ввести код и проверить его корректность. Если код некорректен, то необходимо выдать сообщение "Не корректно" и закончить работу. Если код корректен, то Ваша программа должна ввести с клавиатуры номер хода и произвести его, выдав код круга после этого хода на экран.

Пункт 2. По заданному коду найти кратчайшую последовательность ходов, переводящую круг в правильное состояние. Если решений несколько, то необходимо выдать только один вариант.

Ваша программа должна запрашивать номер тестируемого пункта (1 или 2). Будет оцениваться, совпадает ли формат ввода - вывода Вашей программы с нашими примерами.

Пример 1. Номер пункта: *1* Введите код: ОЯ_ _ССИР Номер хода: 8 _Я РОССИ_

Пример 2. Номер пункта: 2 Введите код: ОЯ_ _ССИР _Я РОССИ_ Р_ _ОССИЯ

Примечание. Не выдавайте на экран никакой лишней информации и не подключайте модуль CRT. Время тестирования Вашей программы будет ограничено.

## 4.9. Олимпиада - 97

r97_1 Сортировка является одной из наиболее часто встречающихся задач вычислительной техники. В данной задаче сортируемые записи имеют не более трех различных значений ключа. Например, три числа 1, 2 и 3. Требуется отсортировать последовательность в порядке неубывания. Сортировка выполняется с помощью попарных перестановок элементов. При попарной перестановке, определяемой номерами позиций i и j, производится обмен местами двух элементов, находящихся в позициях i и j.

Напишите программу, которая по заданной последовательности значений ключей (их количество 1<=N<=1000) находит минимальное количество операций попарной перестановки, необходимых для того, чтобы отсортировать заданную последовательность, и определяет эту последовательность.

Пример.

9 (количество ключей) 2 2 1 3 3 3 2 3 1

4 (количество перестановок) 1 3 4 7 9 2 5 9

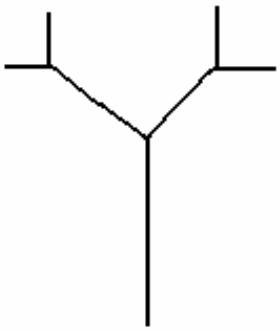
r97_2 Известно, что клетки шахматной доски обозначаются латинскими буквами от A до H по вертикали и цифрами от 1 до 8 по горизонтали(от A1 в левом нижнем углу до H8 в правом верхнем).

| Клетка | Строка |
|---|---|
| A7 | 32 |
| B1 | 16 |
| F4 | B1+C2 |
| C2 | A7+B1 |
| E6 | F4+B1 |

В каждой клетке содержится строка одного из трех типов: пустая строка, десятичная запись положительного числа или сумма названий двух или более клеток. Напишите программу, которая вычисляет заданную таблицу либо определяет, что это невозможно. Пример. Для приведенного примера заполнения доски (остальные клетки пустые) решение имеет вид: A7=32, B1=16, C2=48, F4=64, E6=80

г97_3 Написать программу, которая выводит на графический экран дерево, примерное изображение которого (с пятью ветками) приведено на рисунке. Угол между парой исходящих веток равен 90 градусов. Каждая ветка (исключая ствол) имеет длину, равную 0.56 от длины ветки, из которой она растет. Высота дерева составляет 10 веток (включая ствол). Считайте, что размеры графического экрана вашего компьютера 640 точек по горизонтали и 480 точек по вертикали. В левом верхнем углу экрана расположена точка с координатами (0.0). Считайте, что в вашем языке программирования есть оператор LINE(x1,y1,x2,y2), который изображает на экране отрезок, соединяющий точки с координатами (x1,y1) и (x2,y2).

Начало дерева имеет следующий вид.



о97_1 В длинную деревянную рейку вбили несколько гвоздей (2≤N≤20). Гвозди объединяются в пары веревочками так, чтобы выполнились следующие условия:

- к каждому гвоздю была привязана хотя бы одна веревочка;
- суммарная длина веревочек была бы минимально возможной.

Написать программу, которая определяет пары гвоздей, связанных веревочками, как сказано выше.

Входными данными являются число гвоздей и их координаты (целые числа, по модулю не превосходящие 30000, вводимые в порядке ВОЗРАСТАНИЯ значений), выходные данные - минимальная суммарная длина и пары номеров соединяемых гвоздей.

Пример.

Входные данные:

5

11 12 13 16 17

Выходные данные:

3

1 2 2 3 4 5

о97_2 Заданы две символьные строки A и B, не содержащие пробелов. Требуется вычислить, сколькими способами можно получить строку B из строки A, вычеркивая некоторые символы. Например, если строки A и B имеют соответственно вид СамаринаИрина и Сара, то искомое число равно 7, для строк ааавввссс и авс, это число равно 36.

Написать программу, находящую требуемое число способов.

Примечание. Каждая строка содержит не более 25 символов.

о97_3 В вашем распоряжении имеется по четыре экземпляра каждого из чисел : 2, 3, 4, 6, 7, 8, 9, 10. Требуется написать программу такого размещения их в таблице 6x6, чтобы в клетках, обозначенных одинаковыми значками, находились равные числа и суммы чисел на каждой горизонтали, вертикали и обеих диагоналях равнялись заданному числу N (30<=N<=55).

| | | | | | |
|---|---|---|---|---|---|
| + | | $ | # | | 11 |
| | @ | | | @ | |
| # | @ | 11 | + | | # |
| | | $ | 11 | @ | |
| + | ! | | # | ! | $ |
| 11 | | ^ | + | ^ | $ |

Входные данные - число N.

Выходные данные - заполненная таблица или сообщение «нет решения».

Примечание. Обратите внимание на то, что число 11 уже записано в таблице 4 раза.

о97_4 В выпуклом N-угольнике провели все диагонали, никакие три из которых не пересекаются в одной точке. Требуется написать программу поиска числа частей, на которые оказался разбит N-угольник.

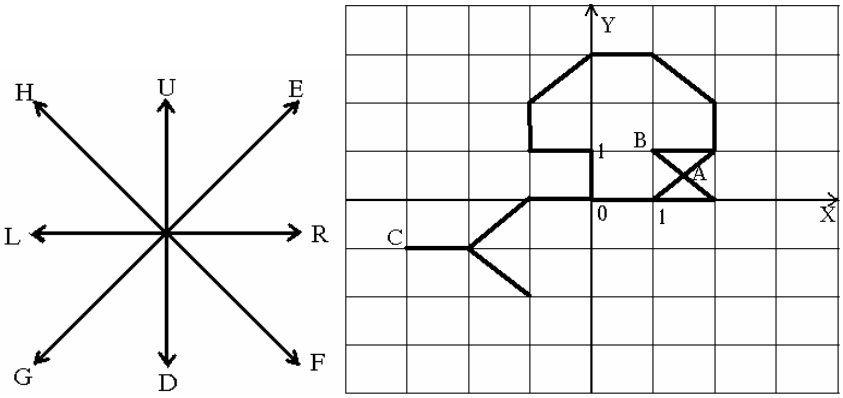
Входные данные: натуральное число N (3≤N≤30). Выходные данные - число частей в разбиении. Например. N=5. Ответ - 11.

о97_5 Возьмем клетчатую доску MxN (1<=M,N<=4). Воткнем в каждую клеточку штырек. В нашем распоряжении есть K (0<K<20) абсолютно одинаковых колечек, каждое из которых можно нанизывать на штырек, причем на один штырек можно надеть несколько колечек. Подсчитать, сколькими способами можно распределить все эти колечки по штырькам. Два распределения считаются разными, если на каком-то из штырьков находится разное количество колечек, и одинаковыми в противном случае.

Входные данные : M N K

Пример : Входные данные : 2 2 2 Результат : 10

о97_6 На клеточном поле вдоль линий сетки и по диагоналям из центра координат (в центре клеточного поля) выкладывается связная фигура, состоящая из спичек длины 1 и √2. Спички длины 1 выкладываются по сторонам клеток, а длины √2 - по диагоналям. Пример.




Ребенок решил сжечь фигуру. Ему разрешено поджечь ее только в одной точке, имеющей целочисленные координаты (например, в точке А нельзя, а в точках В и С можно). Известно, что время сгорания спички не зависимо от длины и равно 2 минутам, если спичка загорается с одного конца, и 1 минуте, если она загорается с двух концов одновременно или со средины.

Написать программу определения координат точки зажигания такой, что:

а) фигура сгорает за минимальное время;

б) при сгорании какая-нибудь спичка загорается одновременно с двух концов.

Входные данные: Строка не более чем из 20 заглавных латинских букв (U,E,R,F,D,G,L,H), определяющих направления спичек при выкладывании фигуры, причем одна и та же спичка может быть описана во входной строке несколько раз.

Выходные данные. Выдать номер решаемого пункта, затем: для пункта а) координаты одной из точек поджигания и время сгорания фигуры; для пункта б) координаты одной из точек поджигания или сообщение «Решения нет».

Примечание: Если возможных точек поджигания несколько - выдать одну из них.

Пример входных данных :

RELFLEUHLGDRDLGLRF

Пример выходных данных для этого теста:

Пункт а): координаты (1,0) время 9

Пункт б): координаты (0,1)

## 4.10. Олимпиада - 98

r98_1 Треугольники, у которых длины сторон и площадь являются целыми числами, называются треугольниками Герона.

Написать программу, которая:

- по введенным с клавиатуры трем числам - длины сторон(a, b, c), определяет, является ли треугольник Героновым, и выводит его площадь;
- находит треугольники Герона, у которых стороны являются соседними числами (например, a=3, b=4, c=5) и значения длин сторон принадлежат интервалу целых чисел от 2 до 20;
- находит треугольники Герона с площадью, равной его периметру (значения длин сторон принадлежат интервалу целых чисел от 2 до 20).

r98_2 Игра между двумя игроками. Дана последовательность из N положительных целых чисел(N - четное число). Значение N и последовательность чисел вводятся с клавиатуры. Игроки ходят по очереди. Ход заключается в том, что игрок выбирает число, расположенное на левом или на правом конце последовательности. Выбранное число стирается. Игра заканчивается, когда чисел не останется. Первый игрок выигрывает, если сумма выбранных им чисел не меньше, чем сумма чисел, выбранных вторым игроком. Первый игрок всегда ходит первым. Известно, что у первого игрока есть выигрышная стратегия.

Требуется написать программу, которая реализует выигрышную стратегию первого игрока (играют компьютер и человек).

r98_3 Арифметический ребус - это зашифрованная запись сложения двух натуральных чисел (например, КОМП+КОМП=СБОРЫ). При этом одинаковым буквам должны соответствовать одинаковые цифры, разным - разные, и ни одно из чисел не может начинаться с нуля.

Требуется написать программу, находящую одно из решений ребуса.

Входные данные: единственная строка с записью ребуса вводится с клавиатуры. Длина строки не превышает 20 символов.

Выходные данные: Одно из решений ребуса (выводится на экран монитора).

Пример:

ЛЕТО+ЛЕТО=ПОЛЕТ

8947+8947=17894

o98_1 Вводится ïîñëåäîâàòåëüíîñòü öåëûõ ÷èñåë (в диапазоне îò -32000 äî 32000). Òðåáóåòñÿ ðàçáèòü èõ íà ïàðû òàê, ÷òîáû ïðîèçâåäåíèÿ ÷èñåë â ïàðàõ áûëè îäèíàêîâû.

Âõîäíûå äàííûå: Ââîäèòñÿ N (количество чисел), 1<=N<=100. Затем вводятся N целых чисел.

Выходные данные: Если разбить на пары можно, то выдать произведение чисел в паре (все варианты), иначе - сообщение "Нет решения".

o98_2 Часы древних марсиан устроены следующим образом: из большой корзины каждую секунду выкатывается шарик и попадает в первую корзину. Как только в ней накапливается пять шариков, корзина переворачивается и четыре шарика возвращаются в большую корзину, а пятый шарик попадает во вторую корзину. Как только во второй корзине накапливается шесть шариков, пять èç íèõ возвращаются â большую корзину, à шестой - â третью корзину è òàê äàëåå (â третьей корзине накапливается 7 шариков, â четвертой 8 ....).

Часы проработали T секунд. Они должны работать еще P секунд. Требуется определить минимальное количество шариков в большой корзине в момент времени T, необходимое для успешной работы часов в течение этих P секунд.

Входные данные: T è P (T≥0, P≥0, T+P<=1 000 000 000).

Выходные äàííûå: êîëè÷åñòâî шаров.

Пример:

Входные данные: 0 32

Ответ: 10

o98_3 Есть N лампочек è M переключателей, каждый из которых какие-то лампочки переключает, а какие-то нет. Сначала часть лампочек включена. Договоримся горящие лампочки обозначать 1, а выключенные - 0. Для переключателей будем писать 1, если переключатель меняет состояние данной лампочки, и 0, если не меняет. Тогда любой переключатель можно представить строкой из N нолей и единиц. Начальное и конечное состояние лампочек также закодируем строкой из 0 и 1. «Применение» переключателя к лампочкам приводит к тому, что включенные лампочки становятся выключенными, а выключенные - включенными (естественно, это справедливо только для лампочек, к которым есть доступ от этого переключателя). Напишите программу, которая определяет, какие переключатели нужно применить, чтобы лампочки перешли в конечное состояние.

Входные данные:

Вводятся числа N и M (1<=N<=50, 1<=M<=50).

Вводится строка, характеризующая начальное состояние лампочек.

Вводится строка, характеризующая конечное состояние лампочек.

Вводится M строк, характеризующих переключатели.

Выходные данные: Номера переключателей, которые нужно применить, причем каждый переключатель можно применить не более 1 раза, либо сообщение "Нет решения".

Пример:

Входные данные:

5 3
10010
11000
11100
10001
10110

Выходные данные:

1 3

o98_4 Последовательность из латинских букв строится следующим образом. Первоначально последовательность пуста. На каждом последующем шаге последовательность удваивается, после чего к ней дописывается очередная буква латинского алфавита (a, b, c, ...). Ниже приведены первые шаги построения последовательности.

Шаг 0. Пустая последовательность

Шаг 1. a

Шаг 2. baa

Шаг 3. cbaabaa

Шаг 4. dcbaabaacbaabaa

.....

Требуется написать программу, которая определяет по заданному числу N ($1 \le N < 2^{26}$) символ, стоящий на N-м месте в последовательности, получившейся после 26-го шага.

Пример. Вход 4. Выход - w.

o98_5 Дано N целых чисел (1≤N≤100). Каждое из них можно изменить не более чем на величину L (целую), как в сторону увеличения, так и в сторону уменьшения или оставить без изменения. Если после такой

операции некоторые из чисел оказываются равными, то они засчитываются за одно. Написать программу поиска такой операции, в результате которой остается наименьшее количество чисел.

Входные данные:

В первой строке значения L (1<=L<=32000) и N.

Во второй строке N чисел (в диапазоне от -32000 до 32000), записанных через пробел.

Выходные данные: количество оставшихся чисел.

Пример.

10 3

11 21 27

Результат - 1.

o98_6 Найти площадь пересечения N прямоугольников со сторонами, параллельными осям координат.

Входные данные:

Вводится число N (количество прямоугольников $2 \le N \le 1000$ ).

Затем вводится N четверок действительных чисел $X_{i1}$ $Y_{i1}$ $X_{i2}$ $Y_{i2}$, задающих координаты левого нижнего и правого верхнего углов прямоугольника.

Выходные данные: вывести площадь пересечения (общей части) всех данных прямоугольников, либо выдать, что они не пересекаются.

Пример:

3

0 0 10 10

5 5 15 15

-1.5 0 7 8

Ответ: 6

## Глава 5. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧ

### 5.1. Олимпиада - 89

r89_1 Задача о ханойских башнях "кочует" по учебникам информатики, не миновала она и кировские олимпиады. Суть решения явно просматривается из текста программы.

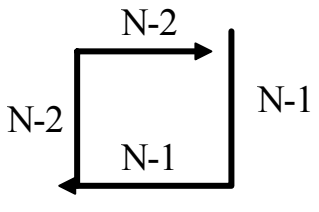
Вычислим число f(n) ходов, необходимых для проведения игры с n кольцами. Число ходов f(n) получается в результате переноса башни из (n-1)-го кольца с иглы 1 на иглу 2, затем один ход на перенос n-го кольца с 1 иглы на 3 и переноса башни из (n-1)-го кольца с иглы 2 на иглу 3. Следовательно, f(n)=2*f(n-1)+1. Введем функцию g(n)=f(n)+1=2*f(n-1)+2=2*(f(n-1)+1)=2*g(n-1). Имеем: f(0)=0, g(0)=f(0)+1, f(1)=1, g(1)=f(1)+1=2. По индукции $g(n)=2^n$, и, наконец, $f(n)=2^n-1$. Для игры с 30 дисками требуется $2^{30}-1$ ходов. Но $2^{10}$ равно 1024, или порядка $10^3$. Следовательно, $2^{30}$ порядка $10^9$. Предположим, что компьютер за одну секунду выполняет $10^6$ ходов. Потребуется порядка 0.27 часа для решения задачи.

r89_2 Ограничение на представление слова и текста в виде массива символов дано "под алгоритмический язык". В 1989 г. это, видимо, еще было необходимо.

Самым простым методом поиска является «прямой поиск». Его суть в последовательном сравнении отдельных символов. Поиск продолжается до тех пор, пока не обнаружится вхождение или пока не будет пройдена вся строка s. При этом можно закончить просмотр, когда i будет равно n-m, так как при следующих значениях i длина любого фрагмента строки s с позиции i меньше m. В Приложении приведен текст программы метода «прямого поиска подстроки». Его временная характеристика - t~O(n*m).

Алгоритмы Р. Бойера, Дж. Мура и Кнута, Мориса, Пратта [5] требуют меньших временных затрат.

r89_3 Первый вариант решения. Заполним элементы первой строки, а затем заполняем матрицу по следующей схеме.




Требуется N-1 раз заполнить такой «прямой угол» элементов, уменьшая на единицу количество записываемых элементов.

Второй вариант решения еще проще и может служить примером при изучении приемов написания рекурсивных программ. Заполнение одного «отрезка прямой», от следующего отличается изменением значений приращений индексов. Внесем номер приращения в параметр рекурсии и получаем короткую и изящную программу. Разумеется, этими вариантами не исчерпывается все множество возможных решений задачи.

r89_4 Идея решения. Номера спортсменов записываются первоначально в массив A, номера удаляемых спортсменов - в массив B. Процесс продолжается до тех пор, пока в A есть элементы. Другое решение основано на использовании множественного типа данных - задачу можно считать учебной при его изучении. Фрагмент реализации.

```
var     A:set of byte;
```

```
        i:integer;
begin
...
 A:=[1..N]; {в кругу все спортсмены}
 i:=<номер спортсмена, с которого начинается счет>;
 repeat {считаем, до тех пор, пока в кругу есть спортсмены}
        for j:=1 to k do{пропускаем k спортсменов}
         repeat
          i:=i mod n+1;
         until i in A;{находим очередного спортсмена}
  <номер i записываем в массив или выводим на экран>;
  A:=A-[i];{исключаем номер i из множества A}
 until A=[];
end.
```

r89_5 Текста программы, приведенного в Приложении, достаточно для понимания задачи.

r89_6 Текста программы, приведенного в Приложении, достаточно для понимания задачи.

r89_7 Идея решения: перебор всех двузначных чисел и проверка для каждого из них условия задачи - «сумма цифр числа не меняется при умножении его на 2, 3, 4, 5, 6, 7, 8, 9». Для проверки условия требуется цикл. С методической точки зрения важно, чтобы он был циклом типа "пока". Если хотя бы для одного из чисел от 2 до 9 условие не выполняется, то осуществляется переход на следующее двузначное число.

r89_8 Текста программы, приведенного в Приложении, достаточно для понимания задачи.

r89_9 Текста программы, приведенного в Приложении, достаточно для понимания задачи.

o89_1 Задача требует аккуратного программирования. Необходимо последовательно убрать пробелы, определить знак числа, преобразовать целую и дробную части числа.

o89_2 Первые четыре совершенных числа 6, 28, 496, 8128 были известны еще древним грекам. В XII в. Церковь утверждала, что для спасения души достаточно найти пятое число. Оно было найдено в XV в., это число 33 550 336. В 1976 г. было известно 24 совершенных числа, причем наибольшее из них было $2^{19936}(2^{19937}-1)$, содержащее около 6000 цифр. К 1989 г. Нашли 52 числа, наибольшее равнялось $2^{56667}(2^{56667}-1)$ [6].

В XVIIIв. Эйлер доказал, что каждое четное совершенное число m может быть представлено в виде $m=2^{n-1}(2^n-1)$, где $2^n-1$ - простое число. Простые числа представимы в виде 6*k+1 или 6*k-1. С учетом этих двух фактов в Приложении приведен текст программы (второй вариант), позволяющей находить пять совершенных чисел.

o89_3 Необходимо выполнить сортировку элементов массива, например, по неубыванию ($t\sim M\log_2 M$). После этого однократный просмотр массива по логике

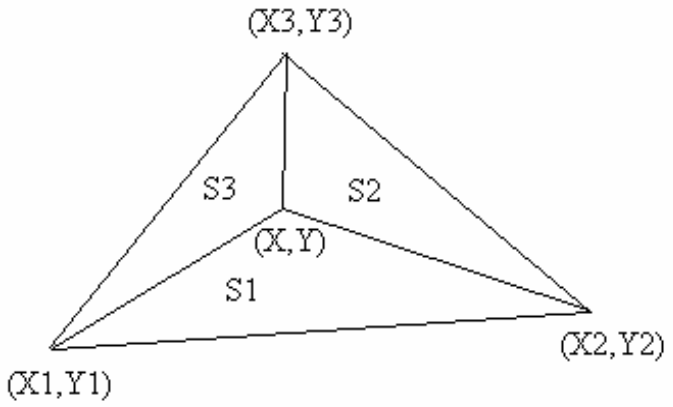
```
s:=1;
for i:=1 to M-1 do if A[i]<>A[i+1] then inc(s);
```

дает ответ задачи.

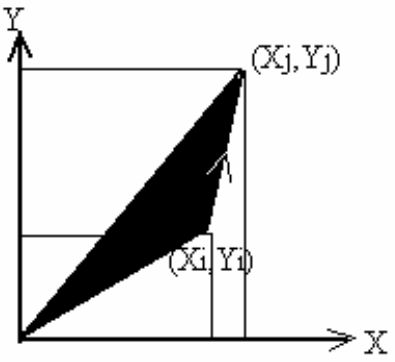
Если сделать предположение о том, что значения элементов массива принадлежат некоторому интервалу чисел, то более приемлемым является вариант решения, приведенный в Приложении.

o89_4 Из всех возможных вариантов решения задачи в Приложении приведен текст программы, реализующей метод однократного просмотра массивов A и B.

o89_5 Если точка находится внутри треугольника и S - площадь треугольника, то S=S1+S2+S3 (рисунок). Для вычисления площадей треугольника можно использовать формулу Герона, однако есть более простые формулы.

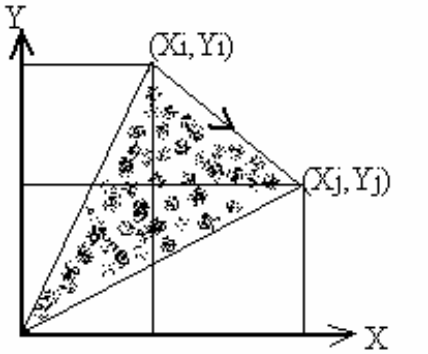


На рисунке площадь темной области (треугольника) вычисляется по следующей формуле: S=Yj*Xj/2-Xi*Yi/2-(Xj-Xi)*(Yi+Yj)/2=(Xi*Yj-Xj*Yi)/2 - из площади большого треугольника вычитаем площадь маленького и трапеции.




На следующем рисунке площадь заштрихованной области вычисляется также по формуле: S=(Xi*Yj-Xj*Yi)/2, но оказывается отрицательной.

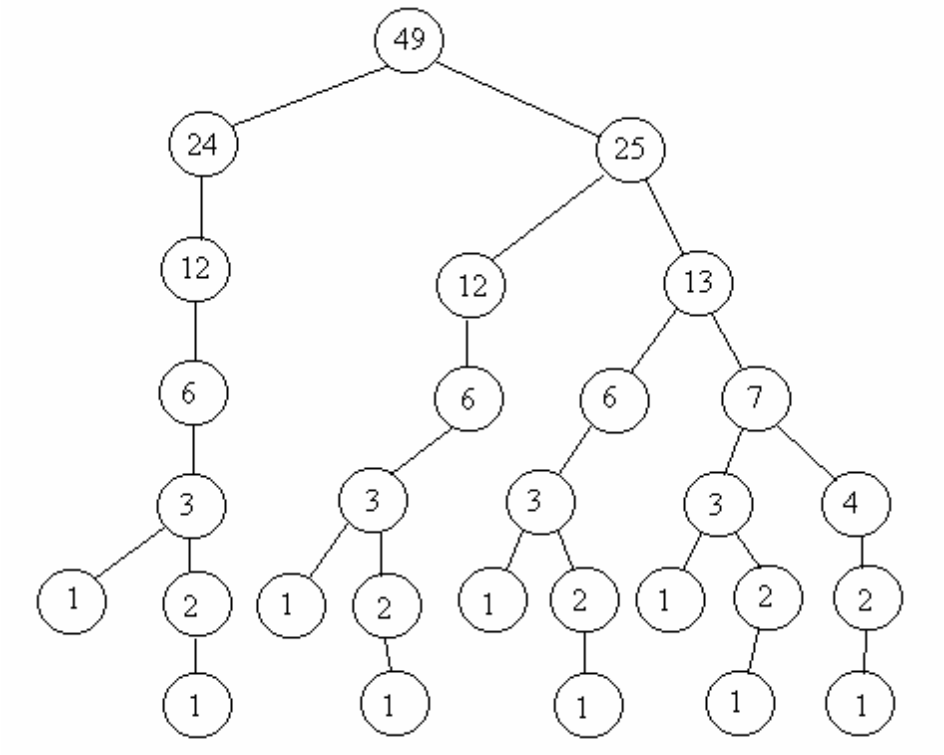
Абсолютное значение S дает площадь треугольника.




o89_6 Первый вариант решения (временно отбрасываем условие о недопустимости использования массива) - подсчитываем подряд все элементы последовательности. Тогда для очередного элемента с номером i уже известны элементы с меньшими номерами, через которые он вычисляется.

```
f[1]:=1;
for i:=2 to n do
 if i mod 2=0 then f[i]:=f[i div 2] else f[i]:=f[(i-1) div 2]+f[(i+1) div 2];
```

Все. Один цикл и есть ответ. Однако мы нарушили условие задачи, да и недостатки решения очевидны. Какую бы размерность массива f мы ни объявляли, всегда можно задать элемент последовательности с номером i на единицу больший значения n. Кроме того, мы вычисляем все элементы последовательности, это недопустимая роскошь - выполнять лишнюю работу удел..., кроме того, не любой компьютер с этим «согласится». Пусть нам необходимо вычислить 49-й элемент. Он вычисляется через сумму 24-го и 25-го элементов, 24-й - равен 12-му, 25-й - через сумму 12-го и 13-го и т. д.(рисунок).




Можно организовать хранение только части элементов, необходимых для вычисления заданного, например в структуре данных типа «стек». В стек «откладываются» невычисленные значения. Текущие вычисления выполняются с элементом из вершины стека, если он не равен единице. В этом случае мы снижаем требования к размерности массива, так для вычисления, например, 1000-го элемента потребуется стек из 17 ячеек, но, по-прежнему, условие задачи не выполнено, массив есть.

Из формулировки задачи явно просматривается рекурсивная схема реализации.

```
function rec(n:integer):integer;
 begin
  if n=1 then rec:=1 else if n div 2 =0 then rec:=rec(n div 2)
                                        else rec:=rec((n-1) div 2)+rec((n+1) div 2);
 end;
```

Массива в явном виде нет. Однако глубина вложенности подпрограмм в конкретных языках программирования ограничена, и это обстоятельство определяет пределы применимости данной логики. Ее можно улучшить, сократив лишние рекурсивные вызовы (рассмотрите случай, когда n=64).

```
function rec(n:integer):integer;
 begin
  while n div 2=0 do n:=n div 2;
  if n=1 then rec:=1 else rec:=rec((n-1) div 2)+rec((n+1) div 2);
 end;
```

И все же это не окончательный вариант решения. Из вышеприведенного рисунка видим, что в процессе эволюции (вычисление по рассмотренной схеме) не возникает более двух различных элементов, меняются только кратности их вхождения.

Введем функцию g(n,i,j)=i*f(n)+j*f(n+1) трех аргументов. Для нее верны рекуррентные соотношения:

g(2n,i,j)=i*f(2n)+j*f(2n+1)=i*f(n)+j*f(n)+j*f(n+1)=g(n,i+j,j),

g(2n+1,i,j)=i*f(2n+1)+j*f(2n+2)=i*f(n)+i*f(n+1)+j*f(n+1)=g(n,i,i+j).

Искомое значение f(n)=g(n,1,0) (напомним, что f(0)=0). Последовательно применяя рекуррентные соотношения, получаем:

f(n)=g(n,1,0)=....=g(0,i,j)=j.

Осталось оформить эту логику в виде программы, что и сделано в Приложении.

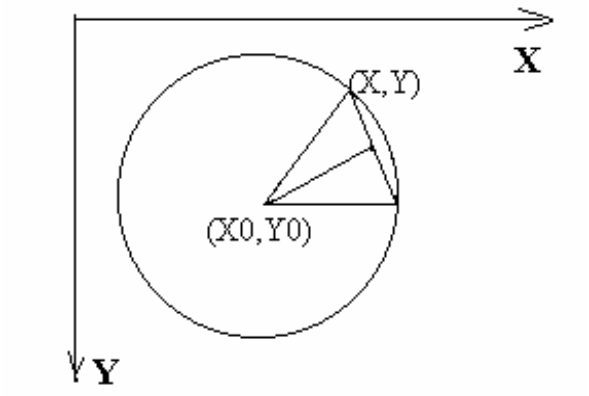
o89_7 Текста программы, приведенного в Приложении, достаточно для понимания задачи.

o89_8 Задача из логики. Маловероятно, чтобы школьники знали правила минимизации сложных высказываний. Однако задачу можно решить "в лоб", основываясь только на таблицах истинности логических операций И, ИЛИ, НЕ (D - промежуточный результат).

| A | B | AиB | (AиB) или не(B) | (AиB) или неA | D | не(AиB) | не(AиB)иA | Y |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

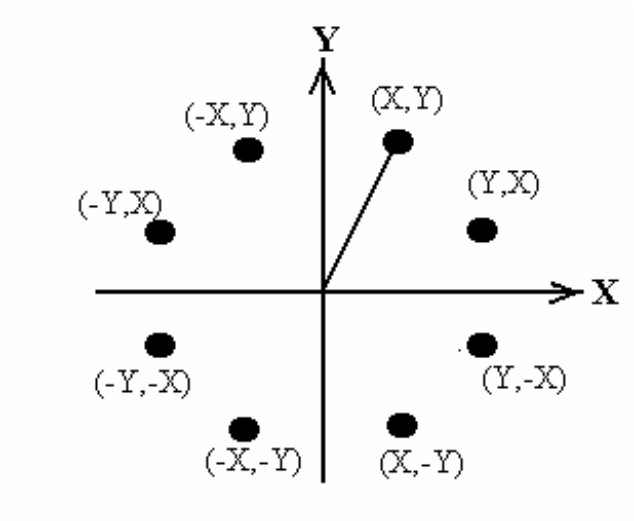
Нетрудно заметить, что Y=A или не B.

o89_9 Основная сложность задачи состоит в том, что вершины многоугольника лежат на окружности через определенные углы. Определить координаты вершин многоугольника можно, только зная N - количество сторон многоугольника. На рисунке показана зависимость между R - радиусом описанной окружности, A - длиной стороны многоугольника и N.



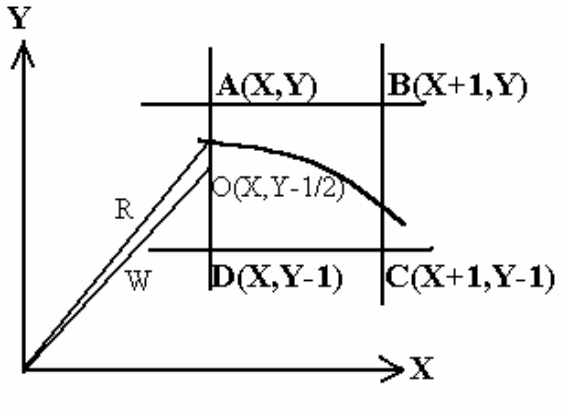

```
l=2*pi/N
R=A/(2*sin(pi/N))
X=X0+R*cos(2*pi/N)
Y=Y0-R*sin(2*pi/N)
```

o89_10 Алгоритм не должен включать тригонометрических функций, квадратного корня, а также операций деления и возведения в степень. Если известно, что точка с координатами (X,Y) принадлежит окружности, то окружности принадлежат и точки, показанные на рисунке.



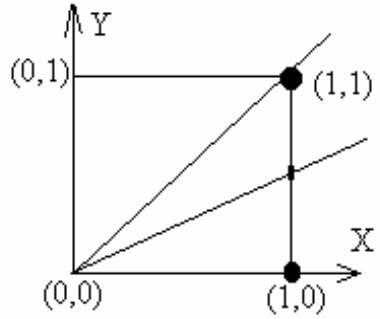

Сложность задачи заключается в выборе точки с целочисленными координатами, которая находится ближе к истинному положению точки на окружности. Какую точку, A или D, выбрать - на этот вопрос нам необходимо ответить. Оценим величину $R^2-W^2$, так как $W^2=X^2+(Y-1/2)^2$, то $R^2-W^2=(R-X)*(R+X)-Y*(Y+1)-1/4$. Итак, если $R^2-W^2>0$, то следует выбирать точку A, при равенстве 0 - безразлично, а если меньше 0, то точку D. Кроме того, при $R^2-W^2<0$ необходимо определить расположение точки C относительно окружности, которое дает знак величины $R^2-((X+1)^2+(Y-1)^2)=(R-X)*(R+X)-Y*(Y+2)-2*(X+1)$. При положительном знаке точка C- "внутри", поэтому при следующей итерации в сравнении должны участвовать точки с координатами (X+1,Y-1) и (X+1,Y-2).

### 5.2. Олимпиада - 90




r90_1 Задача по методам сортировки. Один из способов ее решения заключается в следующем. Пусть Ивановы должны жить в начале улицы, а Петровы - в конце. По индексу i (i<j) ищем первого Петрова, i увеличивается с шагом 1. Если нашли, то ищем Иванова с конца улицы - индекс j, он уменьшается. Если пара составлена, то совершаем обмен, и так до тех пор, пока i будет меньше j.

r90_2 Опишем коротко известный алгоритм Брезенхема [11] вычерчивания отрезков по точкам. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. На единицу, в процессе работы, изменяется одна из координат - либо X, либо Y(в зависимости от углового коэффициента), изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Это расстояние определим как ошибку. Рассмотрим первый октант (отрезки с угловыми коэффициентами, лежащими в диапазоне от 0 до 1).




инициализировать ошибку в -1/2
ошибка=ошибка+dy/dx
1/2<=dy/dx<=1 (ошибка >=0)
0<=dy/dx<1/2 (ошибка <0)

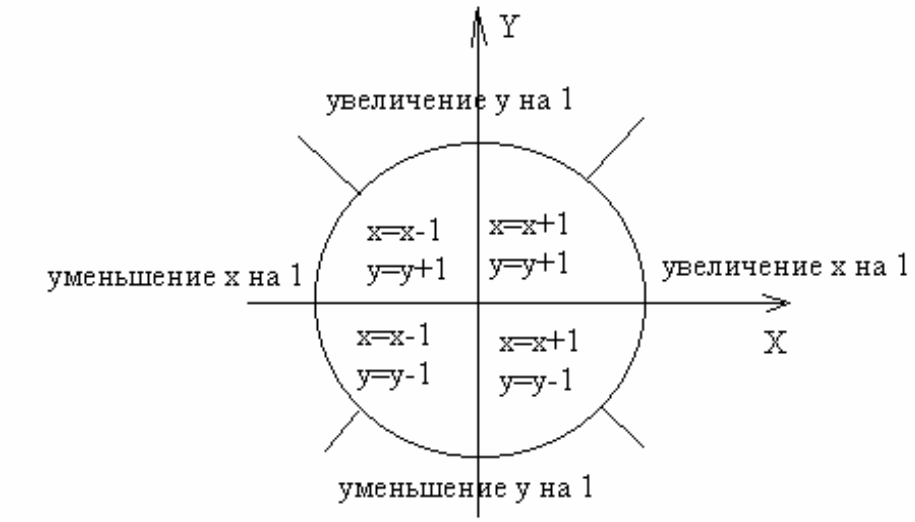
Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0,0) больше, чем 1/2, то его пересечение с прямой x=1 будет расположено ближе к прямой y=1, чем к прямой y=0. Следовательно, точка растра (1,1) лучше аппроксимирует ход отрезка, чем (1,0).

```
{Логика для первого октанта, т.е. для случая 0≤dy≤dx}
var     x,y,dx,dy:integer;
        x1,y1,x2,y2:integer;
        ep:real;
begin
...
 x:=x1;y:=y1;dx:=x2-x1;dy:=y2-y1;
 ep:=dy/dx-0.5;
 for i:=1 to dx do begin
  pset(x,y);
  while ep>=0 do begin y:=y+1;ep:=ep-1;end;
```

```
 x:=x+1;ep:=ep+dy/dx;
end;
...
end.
```

Рассмотрим пример. Отрезок имеет угловой коэффициент 3/8. Если первоначально установить значение ошибки -1/2, то затем можно будет проверять только знак ошибки. Для примера ep=-1/2+3/8=-1/8. Значение ep отрицательно, отрезок пройдет ниже точки (1,1/2), следовательно, точка, (1,0) лучше аппроксимирует положение отрезка. Вычислим ошибку ep=-1/8+3/8=1/4 в следующей точке. Значение ошибки положительно, выбираем точку (2,1). Корректируем ошибку ep=1/4-1=-3/4. Продолжаем вычисления для следующей точки, ep=-3/4+3/8=-3/8, т.е. выбираем точку (3,1). Итак, ошибка - это интервал, отсекаемый по оси Y строящимся отрезком на каждом значении X, относительно -1/2.

Если вместо ep рассматривать величину **ep**=2*ep*dx, то в приведенной логике вычисление ep:=dy/dx-0.5 заменяется на **ep**:=2*dy-dx, ep:=ep-1 - на **ep**:=**ep**-2*dx, а ep:=ep+dy/dx - на **ep**:=**ep**+2*dy. Получаем целочисленный алгоритм Брезенхема. Все случаи для построения общего варианта целочисленного алгоритма Брезенхема приведены на рисунке.




В Приложении приведен текст программы более простого варианта решения задачи.

r90_3 Текста программы решения задачи достаточно для ее понимания.

r90_4 Данный алгоритм относится к разряду рекурсивных. Рекурсивный вызов продолжается до тех пор, пока не дойдем до первого элемента массива. Считаем его максимальным. Затем «на выходе из рекурсии» сравниваем очередной элемент массива со значением максимального.

r90_5 Решение, время которого пропорционально $N^2$, не требует пояснений - первый вариант решения, приведеный в Приложении. Решение со временем $O(N\log_2 N)$ требует использования методов быстрой сортировки элементов массива, например сортировки Хоара, - второй вариант решения. При ограничениях на допустимый диапазон значений элементов исходного массива время решения пропорционально N - третий вариант решения. Можно продолжить модификации задачи, например ее решение без использования дополнительного массива.

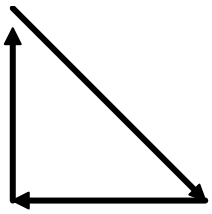
o90_1 Воспользуемся следующим фактом. Простые числа N представимы в виде N=6*I+1 или N=6*I-1. При заданном интервале чисел (от 1000 до 2000) минимальное и максимальное значения I равны соответственно 167 и 333. Кроме того, известно, что делители числа N, если они есть, принадлежат интервалу целых чисел от 2 до SQR(N).

o90_2 Задача требует аккуратного программирования. Клетки, находящиеся под боем коня и ферзя, следует учитывать один раз. В условии задачи не сказано, «непрозрачен» конь для ферзя или нет. Решение для случая «непрозрачности» требует добавления нескольких процедур в текст программы из Приложения.

o90_3 Текста программы, приведенного в приложении, достаточно для понимания задачи.

o90_4 По номеру каждого символа закодированного сообщения определяем число (обычная операция взятия модуля по длине кода), вычитание которого приводит к месту истинного символа в русском алфавите.

o90_5 Формируем элементы треугольника в квадратной матрице (ниже главной диагонали). После этого останется аккуратно выполнить ее вывод. Заполнение осуществляется по принципу.



За одну «прокрутку» заполняется 3*k+q-3 элементов матрицы. После достижения значением t (очередное число) величины 3*k+q-3, q присвоим значение 3*k+q-3 (начальное значение q равно 1), а значение k уменьшим на 3.

o90_6 Нам дан простой плоский многоугольник (многоугольник называется простым, если никакая пара непоследовательных его ребер не имеет общих точек). Соединим последовательно начало координат отрезками с вершинами многоугольника. Вычисляя ориентированные площади треугольников, так, как описано в задаче o89_5, получим площадь многоугольника.

o90_7 Если коротко, то программа должна выводить свой собственный текст. Эта тема была одно время довольно популярна в компьютерных журналах. Если бы мы решали задачу на языке программирования QBasik, в котором допускается не нумеровать строки программы, то решение могло быть следующим.

```
FOR I=1 TO 9
READ A$
PRINT A$
NEXT
RESTORE
FOR I=1 TO 9
READ A$
PRINT "DATA";A$
NEXT
DATA FOR I=1 TO 9
DATA READ A$
DATA PRINT A$
DATA NEXT
DATA RESTORE
DATA FOR I=1 TO 9
DATA READ A$
DATA PRINT "DATA";A$
DATA NEXT
```

Вариант решения на языке программирования MSX-Бейсик имеет вид:

```
1 A$="1 A$=PRINT LEFT$(A$,5)+CHR$(34)+A$+CHR$(34)+CHR$(58);:
PRINT MID$(A$,6,68)":PRINT LEFT$(A$,5)+CHR$(34)+A$+CHR$(34)+
CHR$(58);:PRINT MID$(A$,6,68)
```

Программа состоит из одной строки, в которой формируется строковая константа A$, а затем она выводится с помощью двух операторов PRINT. Функция CHR$ от кода 34 дает символ кавычки, а от кода 58 - двоеточие.

В Приложении приводится один из вариантов решения задачи на языке программирования Паскаль.

## 5.3. Олимпиада - 91

r91_1 Алгоритм определяет и выводит все простые делители числа M.

r91_2 Задача из серии «счастливые билеты», ее простейшая трактовка для четырехзначных чисел. Решается простым перебором.

r91_3 Договоримся, что локомотив обозначаем буквой "L", платформу - "P", охрану -"O". В этом случае нам необходимо уметь распознавать строки вида "L[L]P[P...P]O", в квадратных скобках обозначены символы, которых может не быть.

r91_4 Заполняем по принципу, первый шаг 

второй шаг  , и так (N div 2 + N mod 2)  раз. В зависимости от шага вычисляются верхний и нижний параметры стандартного цикла for.

r91_5 Пусть дробь имеет вид N/M, где N<M, тогда цифры частного L и остатка N получаются по следующим формулам: L=10*N DIV M (деление нацело) и N=10*N-L*M и т. д. Каждый остаток не превосходит M, поэтому если мы определим массив A[1..M], значением элементов которого будет множество частных, полученных для соответствующего остатка, то для очередного остатка N и частного L, присутствие L в A[N] говорит о том, что начался период.

о91_1 Необходимо получить все подмножества множества из шести символов. Будем генерировать шестиразрядные двоичные числа, так, 010101 будет соответствовать подстроке "bdf" для строки "abcdef". Принцип генерации: просматриваем число слева направо до первого нуля, заменяя все 1 на 0, например после 110011 получим 001011. Завершение генерации - число 0000001.

о91_2 Новобранцы перестанут вертеться, ибо крайние, не видя лица соседа остановятся, их соседи, видя затылок, тоже заканчивают процесс верчения и т.д. Значение A[i], равное 0, говорит о том, что новобранец i смотрит налево, A[i], равное 1, - направо. Поворот новобранца с номером i - это изменение значения A[i], они отражены в таблице.

| A[i-1] | A[i] | A[i+1] | Новое значение A[i] |
|---|---|---|---|
| 0 | 0 | - | 0 |
| 1 | 0 | - | 1 |
| - | 1 | 0 | 0 |
| - | 1 | 1 | 1 |

Определение ситуаций, соответствующих 2-й и 3-й строкам таблицы, выполняется с помощью проверки

if A[i+A[i]*2-1]=1-A[i] then B[i]:=1-A[i];{B[i] - новое состояние новобранца с номером i}

о91_3 Задача на умение работать с большими числами. Один из способов хранения большого числа состоит в выделении для каждой его цифры ячейки памяти (элемента массива). В этом случае число, например из 372 цифр, будет представлено элементами массива из 372 ячеек. Можно изменить основание системы счисления, например, на 1 000 000, тогда исходное число разбивается на шестерки цифр и представление нашего числа потребует 62 ячейки.

Рассмотрим начальный этап формирования $3^{512}$.

| ... | B[4] | B[3] | B[2] | B[1] | B[0], количество занятых элементов массива | Номер итерации |
|---|---|---|---|---|---|---|
| | | | | 3 | 1 | 1 |
| | | | | 9 | 1 | 2 |
| | | | 2 | 7 | 2 | 3 |
| | | | 8 | 1 | 2 | 4 |
| | | 2 | 4 | 3 | 3 | 5 |
| | | 7 | 2 | 9 | 3 | 6 |
| | 2 | 1 | 8 | 7 | 4 | 7 |
| ... | ... | ... | ... | ... | ... | ... |

Хранение числа занятых ячеек массива (значение B[0]) позволяет «убрать» из схемы работы лишние умножения.

о91_4 Задача подробно рассмотрена в главе 2.

о91_5 Начнем с примеров. В массиве A представлены состояния карточек, в массиве P - количество соответствующих чисел в верху карточек. Пусть начальное расположение восьми карточек имеет вид:

| Номер примера | Номер шага | A | P |
|---|---|---|---|
| 1 | 0 | 1 1 2 2 3 3 4 4<br>5 6 7 8 5 6 7 8<br>* | 2 2 2 2 0 0 0 0 |
| | 1 | 5 1 2 2 3 3 4 4<br>1 6 7 8 5 6 7 8<br>* | 1 2 2 2 1 0 0 0 |
| | 2 | 5 1 2 2 3 6 4 4<br>1 6 7 8 5 3 7 8<br>* | 1 2 1 2 1 1 0 0 |
| | 3 | 5 1 7 2 3 6 4 4<br>1 6 2 8 5 3 7 8<br>* | 1 1 1 2 1 1 1 0 |
| | 4 | 5 1 7 2 3 6 4 8<br>1 6 2 8 5 3 7 4 | 1 1 1 1 1 1 1 1 |
| 2 | 0 | 1 1 2 2 3 5 4 6<br>5 6 7 8 3 8 7 4<br>* | 2 2 1 1 1 1 0 0 |
| | 1 | 1 1 7 2 3 5 4 6<br>5 6 2 8 3 8 7 4<br>* * | 2 1 1 1 1 1 1 0 |
| | 2 | 5 1 7 2 3 8 4 6<br>1 6 2 8 3 5 7 4 | 1 1 1 1 1 1 1 1 |
| 3 | 0 | 1 1 7 2 3 5 4 6<br>4 3 2 8 6 8 7 5 | 2 1 1 1 1 1 1 0 |

| | | | |
|---|---|---|---|
| | | * * * * | |
| | 1 | 1 3 7 2 6 8 4 5<br>4 1 2 8 3 5 7 6 | 1 1 1 1 1 1 1 1 |

Первый пример. Ищем первый 0 в массиве P, он соответствует числу 5. Находим 5 в нижнем ряду. Нашли. Если вверху число, которое встречается два раза, то переворачиваем эту карточку. Второй пример. Попытаемся 8 получить вверху. На первой карточке вверху двойка. Она встречается один раз, поэтому необходимо искать вторую карточку с восьмеркой внизу. И у этой карточки верхнее число (пять) встречается вверху один раз. Возьмем пятерку в качестве эталона и ищем карточку, у которой на нижней стороне 5. Нашли. У этой карточки в верхнем ряду 1, которая встречается два раза. Переворачиваем две карточки - они отмечены символом * в таблице. Третий пример. Символом "*" выделены те карточки, которые необходимо перевернуть так, чтобы получить правильное их расположение. Для того чтобы перевернуть серию карточек, необходимо заполнить их номера. Процесс поиска серии карточек не может длиться бесконечно - мы так или иначе найдем карточку, у которой открыто число, встречаемое дважды сверху.

o91_6 Логика решения задачи достаточно проста, но требует очень пунктуального программирования. Во втором задании необходимо выполнить сортировку в порядке возрастания значений между двумя клетками, координаты которых задаются. Принцип сортировки - поиск минимального элемента между очередной клеткой и последней введенной и установка минимального элемента в очередную клетку. Этот процесс продолжается до тех пор, пока не дойдем до последней введенной клетки. В третьем задании необходимо все числа расположить в порядке возрастания. Для этого находим клетку, в которой записано число 1 и переставляем 1 в первую клетку. Число 2 - во вторую клетку и т.д.

## 5.4. Олимпиада - 92

r92_1 Идея решения заключается во ведении массива A, индексом которого является код (по ASCII) символа. Другими словами, элемент массива A[P] служит счетчиком числа вхождений в текст символа, код которого равен P.

r92_2 Преобразование S1 и S2 так, чтобы каждый элемент этих множеств встречался только один раз, можно выполнить при вводе элементов. После этого вычисление суммы части элементов матрицы становится простой задачей.

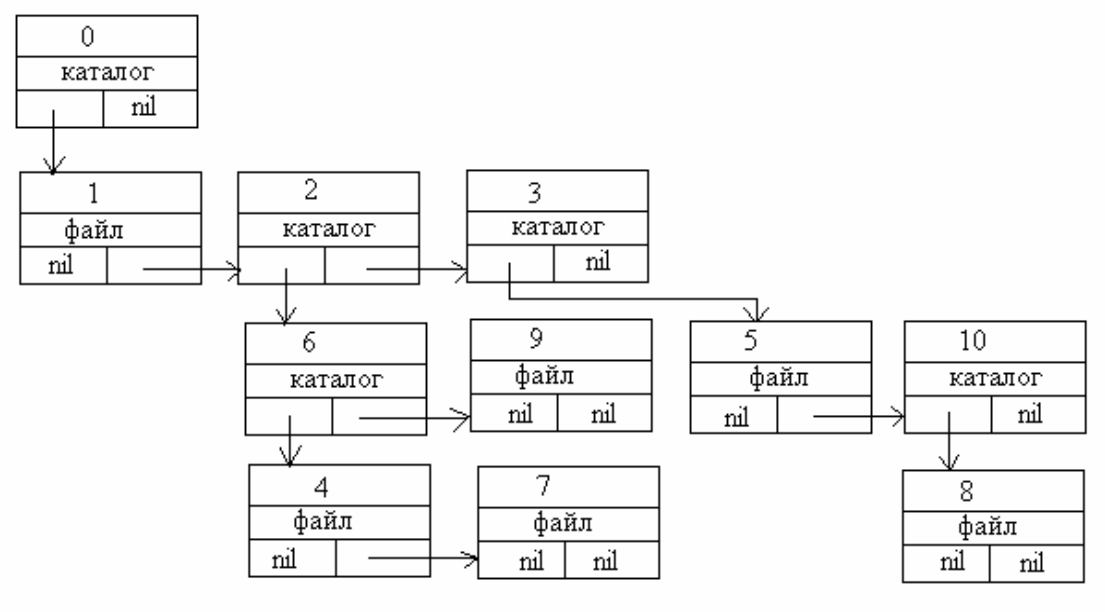
r92_3 Простейший вариант задачи о «рюкзаке», рассмотренной в главе 2. Необходимо получить все подмножества множества из n элементов и из чисел, соответствующих элементам, входящим в очередное множество, образовать сумму. Если она совпадает с A[n+1] элементом, то мы нашли очередное решение уравнения. Решение работоспособно для небольших значений n, количество вариантов $2^n$.

r92_4 Задан граф. Требуется выполнить обход вершин графа. Методы обхода «в глубину» или «в ширину» описаны в главе 3.

o92_1 Решение не требует пояснений.

o92_2 Для решения задачи необходимо создать структуры данных, которые полностью определяют все связи между файлами. Необходимо использовать ссылочный тип данных.

Пусть для примера из формулировки задачи исходные данные вводятся в такой последовательности: 0\, 0\1, 0\2\, 0\2\6\, 0\2\9, 0\2\6\4, 0\2\6\7, 0\3\, 0\3\5, 0\3\10\, 0\3\10\8. Признаком имени каталога является символ \ в конце вводимой строки. Если мы организуем данные, например, в виде двоичного дерева, то проблема обработки различных запросов на вывод информации будет не очень трудной задачей.




Для обработки запросов любого из трех типов необходимо выполнить полный или частичный обход дерева с выбором элементов требуемого типа.

o92_3 Формирование исходных данных задачи не требует пояснений. Запомним время дежурства (начало и конец) сторожей в одном массиве, а в другом, в соответствующих элементах, признаки начала и конца дежурства (1 и -1). После этого выполним сортировку элементов первого массивов в порядке возрастания, переставляя при этом и элементы второго массива. Начнем анализ с момента времени T=0. К счетчику числа сторожей добавляем 1, если время дежурства очередного сторожа совпадает со временем T, и вычитаем 1, если время окончания дежурства у сторожа совпадает со временем T. После этого выбирается новый отсчет времени, в который следует определять количество работающих сторожей, и если количество сторожей меньше 2, то интервал времени, его начало и конец запоминаем. Итак, проверив все моменты времени по этой логике и дойдя до конечного времени дежурства сторожей, мы можем ответить на вопрос, в любой ли момент времени в галерее находится не менее двух сторожей, и перечислить все интервалы времени с недостаточной охраной. Для их перечисления достаточно запомненных значений интервалов времени. Увеличение количества сторожей осуществляется последовательно. Вначале добавляются сторожа в интервалы времени с недостаточной охраной и отстоящие друг от друга на время, большее времени дежурства сторожей. После этого вновь выполняется сортировка и осуществляется поиск интервалов времени с недостаточной охраной. Если они есть, то процесс продолжается, и так до тех пор, пока в любой момент времени в галерее не будут находиться не менее двух сторожей.

o92_4 Рассмотрим пример.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | | 0 | 2 | 0 | 3 | 3 | 0 | 4 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | | 0 | 0 | 3 | 3 | 0 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | | 5 | 5 | 5 | 5 | 0 | 4 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 5 | 0 | 4 | 0 | 0 |

Вторую часть таблицы (справа) получаем с помощью следующей логики (вариации на тему «лабиринт», глава 2).

Фрагмент основной части

```
...
cnt:=1;{счетчик числа областей}
for i:=1 to n do
 for j:=1 to n do
  if A[i,j]=1 then begin inc(cnt);Rec(i,j);end;
...
```

и процедуры

```
procedure Rec(x,y:integer);
 begin
  if A[x,y]<>1 then exit
   else begin A[x,y]:=cnt;Rec(x-1,y);Rec(x+1,y);
```

```
                Rec(x,y-1);Rec(x,y+1);
            end;
        end;
```

Для решения задачи осталось ответить на вопрос, есть ли на границах клетки с одним значением метки.

## 5.5. Олимпиада - 93

r93_1 Идея первого способа решения основана на закраске области, ограниченной заданной ломаной, и подсчете точек, имеющих цвет закраски. Небольшая трудность при этом решении состоит в определении координат точки Q[X,Y] внутри замкнутой области. Однако известно, что средневзвешенное трех произвольных точек ломаной дает координаты внутренней точки.

Идея второго способа решения основана на теореме Пика. Ее формулировка. Площадь многоугольника, вершины которого находятся в целочисленных точках (и не имеющего самопересечений), выражается в виде q+p/2-1, где q - количество целочисленных точек внутри многоугольника, а p - количество целочисленных точек на его границе (оно включает вершины и другие целочисленные точки на его сторонах).

Идея третьего способа решения совпадает с методом решения задачи o90_6.

r93_2 Задача очень известная, она регулярно появляется в изданиях по информатике. Для записи римскими числами используются латинские буквы I, V, X, L, C, D, M, обозначающие соответственно числа 1, 5, 10, 50, 100, 500, 1000. Правила записи. Если большая цифра находится перед меньшей, то они складываются, а если меньшая перед большей, то меньшая вычитается из большей. Цифры M, C, X, I могут повторяться не более трех раз подряд, остальные цифры (D, L, V) - только по одному разу. При решении задачи считаем римскими цифрами не только вышеприведенные цифры, но и их пары IV, IX, XL, XC, CD,CM. В этом случае при переводе числа из римской системы счисления в десятичную получается простое правило: число, записанное римскими цифрами, равно сумме своих цифр. Это правило реализовано в программе, приведенной в Приложении.

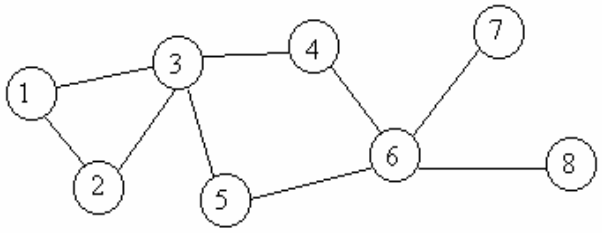
r93_3 В задаче рассматривается простейший вариант трансляции - перевод арифметического выражения (из натуральных чисел) со скобками в бесскобочную форму, называемую обратной польской записью. Например, выражение 9+(6-3)*12-18/(2+7) переводится в 9 6 3 - 12 * + 18 2 7 + / - =. Схема перевода традиционна. Выделяется очередная лексема. Если это число, то оно преобразуется в строковый тип и добавляется к результирующей строке. Если значением лексемы является операция и ее приоритет меньше, чем приоритет ранее записанной в стек операции, то в результирующую строку записывается операция из стека. И наконец, при выделении ‘(‘, последняя просто добавляется в стек, а при ‘)’ из стека в строку выписываются операции до символа ‘(‘. Трассировка логики для приведенного выше примера имеет вид.

| i (номер позиции в исходной строке) | Результирующая строка | Стек |
|---|---|---|
| 0 | ‘’ | = |
| 1 | 9 | = |
| 2 | 9 | = + |
| 3 | 9 | = + ( |
| 4 | 9 6 | = + ( |
| 5 | 9 6 | = + ( - |
| 6 | 9 6 3 | = + ( - |
| 7 | 9 6 3 - | = + |
| 8 | 9 6 3 - | = + * |
| 9,10 | 9 6 3 - 12 | = + * |
| 11 | 9 6 3 - 12 * + | = - |
| 12,13 | 9 6 3 - 12 * + 18 | = - |
| 14 | 9 6 3 - 12 * + 18 | = - / |
| 15 | 9 6 3 - 12 * + 18 | = - / ( |
| 16 | 9 6 3 - 12 * + 18 2 | = - / ( |
| 17 | 9 6 3 - 12 * + 18 2 | = - / ( + |
| 18 | 9 6 3 - 12 * + 18 2 7 | = - / ( + |
| 19 | 9 6 3 - 12 * + 18 2 7 + | = - / |
| | 9 6 3 - 12 * + 18 2 7+ / - | |

Вычисление значение выражения в обратной польской записи опять же реализуется с использованием стека. Если очередная лексема, выделенная из строки, число, то она записывается в стек, а если операция, то последняя выполняется с двумя верхними элементами стека и результат записывается в стек. Продолжим трассировку нашего примера (пробелы из строки исключаются при выделении лексем).

| i (номер лексемы) | Состояние стека |
|---|---|
| 1 | 9 |
| 2 | 9 6 |
| 3 | 9 6 3 |
| 4 | 9 3 |
| 5 | 9 3 12 |
| 6 | 9 36 |
| 7 | 45 |
| 8 | 45 18 |
| 9 | 45 18 2 |
| 10 | 45 18 2 7 |
| 11 | 45 18 9 |
| 12 | 45 2 |
| 13 | 43 |

o93_1 Данные о связях между пунктами будем хранить в массиве Alink[1..N,1..N], элементы которого равны 0 или 1. Значение Alink[i,j]=1 говорит о том, что между пунктами i и j есть дорога. В двумерном массиве Aplace[1..N,1..M] для каждого робота значениями, равными единице, будем указывать те населенные пункты, в которых данный робот может находиться в данный момент времени. Поясним логику решения на примере. Четыре робота находятся в пунктах 1, 2, 7, 8.




Alink

Aplace

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 6 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 | 0 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 8 | 0 | 0 | 0 | 1 |

Что происходит в следующий момент времени? Первый робот может остаться в первом пункте или пойти во второй или третий пункты, в соответствии со связями в матрице Alink. Таким образом, в первом столбце матрицы Aplace во второй и третьей строках вместо 0 должны появится 1. Изменения матрицы Aplace для роботов с номерами 2, 3 и 4 выполняются аналогичным образом.

Aplace через
1 момент времени

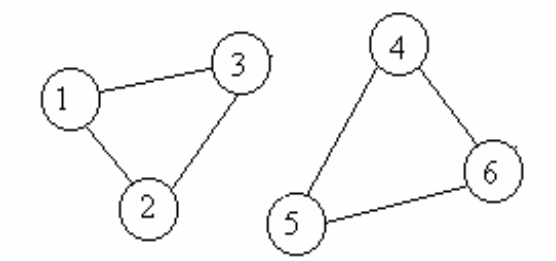
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 |

Aplace в следующий
момент времени

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 1 | 1 |

Итак, появилась строка или строки матрицы Aplace, состоящие из одних единиц. Эта строка будет соответствовать населенному пункту, в котором возможна встреча роботов.

Однако для пунктов




и при начальном расположении двух роботов в пунктах 1 и 6 встреча роботов никогда не произойдет, и строки Aplace, состоящей из одних единиц, не появится.

Это требует введения второй матрицы (Aold), в которой должны фиксироваться достижимые пункты для каждого робота в предыдущий момент времени. Итак, если Aplace и Aold совпадают и нет ни одной строки, состоящей из одних единиц, то встреча роботов невозможна. Это схема решения первого задания. Решение второго задания отличается от первого тем, что требуется найти время T2=T1-1 (T1 - время встречи роботов в одном населенном пункте), в которое все роботы находятся в одном из двух (произвольных) населенных пунктов, соединенных дорогой. В этом случае возможна их встреча и вне населенного пункта. Другими словами, в каждый момент времени необходимо проверять (находить) две строки матрицы Aplace, поэлементная логическая сумма которых дает строку, состоящую из одних единиц. При решении задания три матрицу Aplace следует не дополнять элементами, равными единице, а обновлять в соответствии со связями из матрицы Alink. Причем обновление выполнять не для всех роботов одновременно: в нечетные моменты времени 1, 3, ... для роботов, имеющих скорость 2, а в четные - 2, 4, ... для всех роботов.

Отметим, что выше приведено популярное изложение темы «достижимость» главы 3.

o93_2 Задача на метод динамического программирования. Очевидно, чтобы маршрут удовлетворял условиям 1 и 2, необходимо каждый следующий шаг маршрута делался либо на клетку вправо, либо на клетку вниз, а длина такого маршрута всегда равна K=2*(N-1) (число отрезков). Из таких маршрутов искомый можно найти методом перебора с возвратом из 2 в степени K вариантов.

Однако попробуем решить задачу по-другому. Разберем идею решения на примере. Пусть N=6 и массив(A) заполнен следующим образом.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 4 | 0 | 5 | 3 |
| 3 | 8 | 9 | 7 | 6 | 9 |
| 4 | 7 | 8 | 9 | 4 | 9 |
| 0 | 6 | 0 | 5 | 7 | 9 |
| 2 | 5 | 6 | 1 | 8 | 5 |
| 5 | 9 | 8 | 0 | 6 | 3 |

Для каждого элемента матрицы A вычислим сумму A[I,J] с максимальным значением суммы в "соседних клетках"(с меньшими значениями индексов I и J). Пусть суммы будут храниться в матрице S, тогда логика ее вычисления имеет вид:

```
for i:=2 to n do {1-я строка и 1-й столбец сформированы}
 for j:=2 to n do
if S[i-1,j]>S[i,j-1] then S[i,j]:=A[i,j]+S[i-1,j] else S[i,j]:=A[i,j]+S[i,j-1];
```

Итак, мы получаем матрицу:

| | | | | | |
|---|---|---|---|---|---|
| 1 | 3 | 7 | 7 | 12 | 15 |
| 4 | 12 | 21 | 28 | 34 | 43 |
| 8 | 19 | 29 | 38 | 42 | 52 |
| 8 | 25 | 29 | 43 | 50 | 61 |
| 10 | 30 | 36 | 44 | 58 | 66 |
| 15 | 39 | 47 | 47 | 64 | 69 |

Максимально возможная сумма маршрута(SS) получена - 69. Осталось найти сам маршрут. Начинаем с последней клетки [N,N]. Это конечная клетка маршрута. Уменьшаем значение SS на значение A[N,N] и ищем соседнюю клетку с суммой 66 - это будет предпоследняя клетка маршрута. Продолжаем процесс до тех пор, пока не дойдем до клетки (1,1). Маршрут найден.

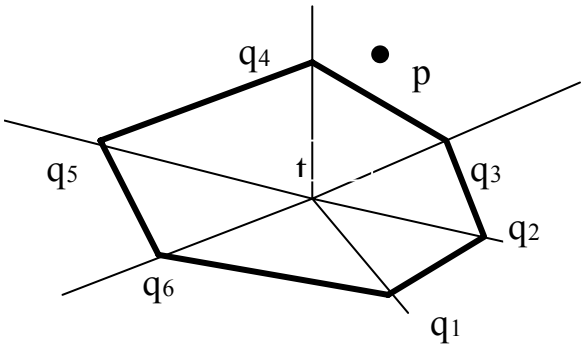
o93_3 Первый способ решения. Соединим каждую точку с каждым отрезком прямой цвета q1. Среди этих отрезков уже есть и искомая дорога, образованная совокупностью каких-то отрезков. Остается выделить ее. Закрасим область вне города цветом q2 до цвета отрезков q1. Если сейчас найти точку внутри области и закрасить внутреннюю область цветом q3 до цвета q2, то граница между двумя цветами q2 и q3 и есть искомое решение.

Второй способ. Задача заключается в построении выпуклой оболочки множества точек на плоскости. Рассмотрим методы построения выпуклых оболочек (в конспективном виде).

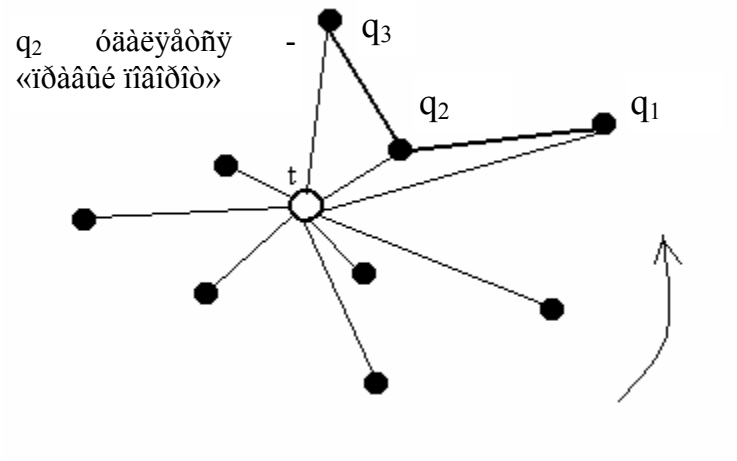
Предварительные замечания. Q-выпуклый многоугольник. Вершины выпуклого многоугольника упорядочены по полярным углам относительно любой внутренней точки. Такая точка t находится путем взятия центра масс вершин треугольника, образованного любой тройкой вершин Q. Задача. Дана точка p и выпуклый N-угольник Q. Определить принадлежность точки p выпуклому N-угольнику Q.

Общая схема.

- Поиск клина. (Методом двоичного поиска. Утверждение. Точка p лежит между лучами, определяемыми $q_i$ и $q_{i+1}$ тогда и только тогда, когда угол ($ptq_{i+1}$) положительный, а угол ($ptq_i$) отрицательный.)
- Сравнить p с единственным ребром. (Утверждение. Точка p внутренняя тогда и только тогда, когда угол ($q_iq_{i+1}p$) отрицательный.)




Первый метод. Алгоритм Грэхема. *Идея.* Тройки последовательных точек многократно проверяются в порядке обхода против часовой стрелки на предмет того, образуют они или нет угол, больший или равный π. Если угол $q_1q_2q_3$ больше или равен π, то считают, что $q_1q_2q_3$ образуют «правый поворот», иначе они образуют «левый поворот».




*Алгоритм.*

- Найти внутреннюю точку t.
- Используя t как начало координат, отсортировать точки множества S по неубыванию в соответствии с полярным углом и расстоянием от t.
- Выполнить обход точек, исключая точки типа $q_2$ (образующие «правый поворот»).

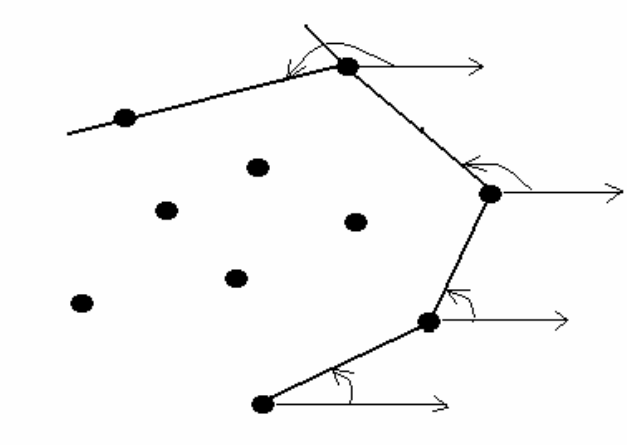
Известно, что сортировка N элементов может быть выполнена за время O(NlogN) (методы Хоара, слияния). Это самая трудоемкая (по времени) часть алгоритма. Обход может быть выполнен за время, пропорциональное N. Значит, выпуклую оболочку можно построить за время, пропорциональное NlogN.

При организации данных (отсортированных точек) в виде двухсвязного списка с указателями СЛЕД и ПРЕД и указателем НАЧАЛО на первую точку логика обхода точек имеет вид:

```
Begin
        v:=НАЧАЛО;w:=ПРЕД[v];f:=false;
        while (СЛЕД[v]<>НАЧАЛО)or Not f do begin
         if СЛЕД[v]=w then f:=true;
         if<три точки v, СЛЕД[v], СЛЕД[СЛЕД[v]] образуют «левый
                поворот»> then v:=СЛЕД[v]
          else begin
                 УДАЛИТЬ СЛЕД[v];
                 v:=ПРЕД[v];
                end;
        end;
end;
```

Второй метод. *Утверждение.* Отрезок L, определяемый двумя точками, является ребром выпуклой оболочки тогда и только тогда, когда все другие точки множества S лежат на L или с одной стороны от него. Из этого утверждения «вытекает» следующая логика. N точек определяют порядка $O(N^2)$ отрезков. Для каждого из этих отрезков за линейное время можно определить положение остальных N-2 точек относительно него. Таким образом, за время, пропорциональное $O(N^3)$, можно найти все пары точек, определяющих ребра выпуклой оболочки. Затем эти точки следует расположить в порядке последовательных вершин оболочки. Джарвис заметил, что данную идею можно улучшить, если учесть следующий факт. Если установлено, что отрезок $q_1q_2$ является ребром оболочки, то должно существовать другое ребро с концом в точке $q_2$. Уточнение этого факта приводит к алгоритму с временем работы,

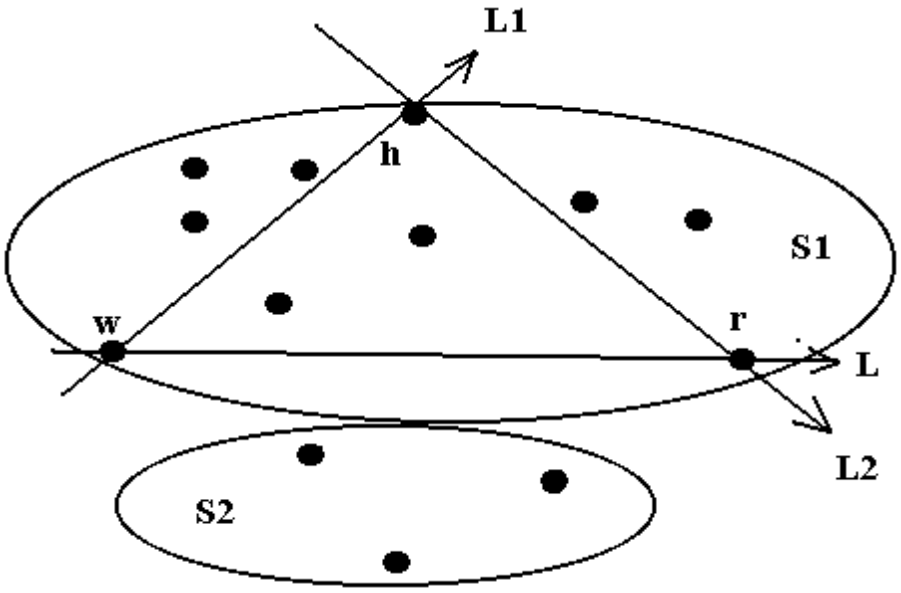
пропорциональным $O(N^2)$. Рисунок, приводимый ниже, может быть, поможет Вам в разработке этого алгоритма. Обратите внимание на то, что в некоторый момент времени необходимо изменить направление оси X, т.е. считать углы относительно отрицательного направления оси X.



Третий метод. *Идея*. Разбиваем множество S из N точек на два подмножества, каждое из которых будет содержать одну из двух ломаных, соединение которых даст многоугольник выпуклой оболочки. Найдем точки w и r, имеющие соответственно наименьшую и наибольшую абсциссы. Проведем через них прямую L, тем самым получив разбиение множества точек S на два подмножества S1 (выше или на прямой L) и S2 (ниже прямой L). Определим точку h, для которой треугольник <whr> имеет наибольшую площадь среди всех треугольников {<wpr>:p∈S1}. Если точек имеется более одной, то выбирается та из них, для которой угол <hwr> наибольший. Точка h гарантированно принадлежит выпуклой оболочке. Действительно, если через точку h провести прямую, параллельную L, то выше этой прямой не окажется ни одной точки множества S. Через точки w и h проведем прямую L1, через точки h и r - прямую L2. Для каждой точки множества S1 определяется ее положение относительно этих прямых. Ни одна из точек не находится одновременно слева как от L1, так и от L2, кроме того, все точки, расположенные справа от обеих прямых, являются внутренними точками треугольника <wrh> и поэтому могут быть удалены из дальнейшей обработки.

Составляющие элементы логики. Функция САМАЯ ДАЛЬНЯЯ ТОЧКА - по значениям S, w, r находить координаты точки h. Функция БЫСТРОЕ ПОСТРОЕНИЕ ОБОЛОЧКИ (Bpo) - по значениям S, w, r возвращает в качестве результата упорядоченный список точек, образующих выпуклую оболочку.

```
Function Bpo(S,w,r):<список точек>;
 var     U1,U2:списки точек;
         S1,S2:множества точек;
         h:точка;
 begin
         if S=[w,r] then возврат (w,r) {выпуклая оболочка состоит из единственного ориентированного ребра}
          else begin
                  h:=САМАЯ ДАЛЬНЯЯ ТОЧКА(S,w,r);
                  S1:=[точки множества S, расположенные слева от wh или на ней];
                  S2:=[точки множества S, расположенные слева от hr или на ней];
                  U1:=Bpo(S1,w,h);
                  U2:=Bpo(S2,h,r);
                  Bpo:=U1+U2;{+ означает сцепление списков U1 и U2}
                  end;
 end;
```




Четвертый метод. Заданы два выпуклых многоугольника Q1 и Q2. Найти выпуклую оболочку их объединения. Метод обработки.

Шаг 1. Находим некоторую внутреннюю точку t многоугольника Q1(центр масс трех любых вершин Q1).

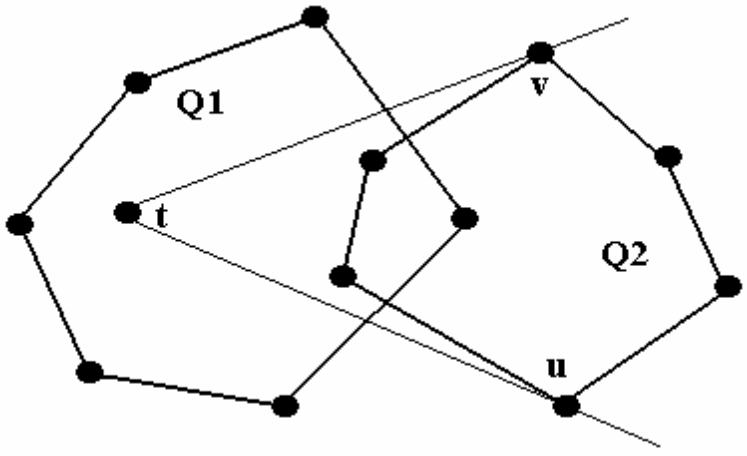
Шаг 2. Определяем, является ли точка t внутренней точкой многоугольника Q2 (За время, пропорциональное O(N)). Если точка t не является внутренней точкой Q2, то переходим к шагу 4.

Шаг 3. Точка t является внутренней точкой Q2. Вершины как Q1, так и Q2 упорядочены в соответствии со значением полярного угла относительно точки t. За время, пропорциональное O(N) получаем упорядоченный список вершин как Q1, так и Q2 путем слияния списков вершин этих многоугольников. Переход к шагу 5.

Шаг 4. Точка t не является внутренней точкой Q2. Если «смотреть» из точки t, то многоугольник Q2 лежит в клине с углом, меньшим или равным π. Этот клин определяем двумя вершинами u и v многоугольника Q2, которые находим за один обход вершин многоугольника Q2. Вершины u и v разбивают границу Q2 на две цепочки вершин, монотонные относительно изменения полярного угла с началом в точке t. При движении по одной цепочке угол увеличивается, при движении по другой - уменьшается. Одну из этих цепочек удаляем (ее точки являются внутренними относительно строящейся оболочки). Другая цепочка и граница Q1 являются упорядоченными списками, содержащими не более N вершин. За время, пропорциональное O(N), их можно соединить в один список вершин, упорядоченный по полярному углу относительно точки t.

Шаг 5. К полученному списку точек применяем метод обхода Грэхема. Затраты времени пропорциональны O(N).

Итак, *утверждение*. Выпуклая оболочка объединения двух выпуклых многоугольников находится за время, пропорциональное суммарному числу их вершин.




Строим выпуклую оболочку множества точек. Метод *«разделяй и властвуй»* основан на принципе сбалансированности, суть которого в том, что вычислительная задача разбивается на подзадачи примерно одинаковой размерности. Они решаются, и затем результаты объединяются. Метод эффективен, если время,

требуемое на объединение результатов, незначительно. Суть данного алгоритма построения выпуклой оболочки. Множество S разделяется на два подмножества S1 и S2, примерно равной мощности. Для каждого из подмножеств строится выпуклая оболочка. Из предыдущего пункта известно, что время объединения оболочек пропорционально количеству вершин в них, т. е. временная зависимость имеет вид: $T(N) \le 2T(N/2)+O(N)$.

```
Procedure Sob(S);
 begin
        if |S|≤k {k - некоторое небольшое целое число}
         then <построить выпуклую оболочку одним из ранее рассмотренных методов>
         else begin
                  <разбить исходное множество S на два подмножества S1 и S2, примерно равной
мощности>;
                   Sob(S1);
                   Sob(S2);
                  <соединить две построенные оболочки>;
                  end;
 end;
```

o93_4 Следующие замечания позволяют написать «уверенно» играющую программу. Выигрышный ход - это такой ход, при котором любой ход человека не изменит результата игры. Выигрышная стратегия. Если первый ход выполняется компьютером и четность чисел N и K совпадает, то первым ходом необходимо зачеркнуть K клеток в центре полоски, а затем всегда следует делать ход, симметричный ходу человека. При отсутствии выигрышной стратегии компьютер может зачеркивать первые найденные K свободных клеток.

## 5.6. Олимпиада - 94

r94_1 Разность индексов (i-j) постоянна для элементов, находящихся на одной диагонали, и изменяется от 1-N до M-1. Этот факт является основным при решении задачи, то есть при формировании массива значений сумм на диагоналях. После этого остается лишь найти максимальный элемент в этом массиве.

r94_2 Один из способов решения задачи очевиден. Наименьшее общее кратное чисел a и b равно (a div НОД(a,b))*b, где НОД - наибольший общий делитель чисел a и b. Осталось выполнить эту операцию для всех чисел последовательности.

Пример. N=3. Числа 24, 32, 44. Ответ - 1056.

r94_3 Задача на метод динамического программирования. Определим массив A(array [0..MaxN+MaxK+1] of longint ), где по условию задачи MaxN равно 10, а MaxK - 20. Элемент A[i] определяет количество способов попадания в магазин при расстоянии i шагов. Очевидно, что A[1] равно 1. Если нам известно количество способов попадания в магазин за t-1 шаг для расстояний i-1 шаг и i+1 шаг, то очень просто найти количество способов при значениях t и i (B[i]:=A[i-1]+A[i+1]), где B - массив того же типа, что и A. A это уже решение. Ответы для некоторых значений N и K: (8,20) - 15504, (4,20) - 25194.

r94_4 Задача на применение метода динамического программирования. Неоднократно задачи этого типа использовались на олимпиадах.

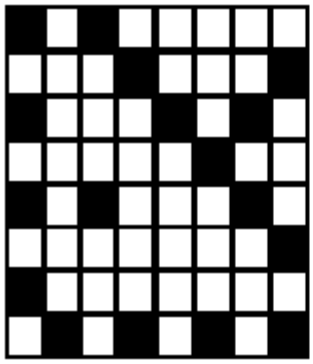
Структуры данных.

```
Const N=8;
Var  A:array[1..N,1..N] of boolean; {*A[i,j]=true - клетка белая,
                                                A[i,j]=false - клетка черная*}
```

B:array[1..N,1..N] of integer;{* начальное значение B[i,j]=-1 для всех значений i и j, B[i,j]=p, где p>0 означает, что в клетку <i,j> можно попасть за p перемещений*}

Формирование значений элементов матрицы A очевидно, с матрицей B чуть сложнее. Определим понятие метки (p). Ее начальное значение равно 1. Пусть мы пометили некоторую клетку значением метки, равным p, тогда при нахождении клетки, в которую мы можем попасть из рассматриваемой клетки за один шаг, метим ее значением на единицу большим, считаем необработанной и помещаем в очередь. Использование очереди для запоминания «необработанных» клеток (по принципу обхода в ширину, глава 3) делает логику более строгой, чем очередной просмотр матрицы для поиска клеток, помеченным текущим значением метки.

Так, для следующего примера матрица B имеет вид:



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 9 | 10 | 11 | -1 | -1 | -1 | 17 |
| 7 | -1 | 11 | 12 | 13 | -1 | 15 | 16 |
| 6 | 7 | -1 | 13 | 14 | 15 | 14 | 15 |
| 5 | -1 | 7 | -1 | 15 | 14 | 13 | -1 |
| 4 | 5 | 6 | 7 | -1 | 13 | 12 | 11 |
| 3 | -1 | 7 | -1 | -1 | -1 | 11 | 10 |
| 2 | 3 | -1 | 5 | -1 | 7 | -1 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

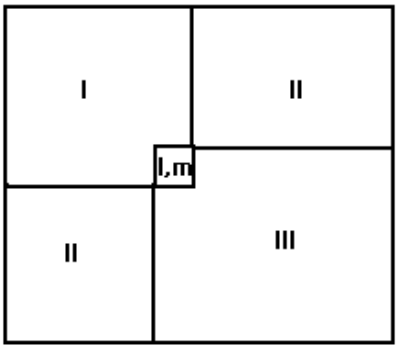
Мы нашли длину минимального пути, но пока еще не сам путь. Для рассматриваемого примера его значение равно 17 и оно записано в B[1,N]. Для поиска пути необходим “обратный просмотр матрицы” B, от элемента B[1,N] к элементу B[N,1]. Его суть в поиске элементов с меньшим на единицу значением p, чем текущий элемент матрицы B.

r94_5 Задача решается перебором вариантов. У нас 2*N элементов, N строк и N столбцов. Необходимо генерировать k элементные подмножества из 2*N элементов, где k изменяется от 1 до 2*N, и проверять матрицу после изменения знаков в строках и столбцах, принадлежащих очередному подмножеству. Если условие задачи выполняется, то процесс генерации заканчивается - задача решена.

o94_1 Решение, в котором время поиска каждого элемента массива A не превосходит значение N+M (количество сравнений), основано на следующем простом факте. Если (на первом шаге) мы сравниваем элемент X с A[N,1], то по результатам сравнения (нет совпадения) можно исключить или элементы первого столбца (X>A[N,1]), или элементы последней строки (X<A[N,1]). Процесс «сужения» области поиска продолжается при каждом следующем сравнении. Этот вариант поиска реализован в программе из приложения.

Задачу можно использовать в качестве учебной при изучении двоичного (бинарного) поиска - «ключевой» идеи в образовательной информатике. Она рассматривается в темах: информация (ее определение, формула Хартли), кодирование (коды Фано, Хаффмена), методы сортировки, вычислительная геометрия (например, метод построения выпуклой оболочки), численные методы решения уравнений.

Итак, решение задачи с использованием идеи двоичного поиска. Если мы сравниваем элемент X с элементом матрицы A[l,m] и нет совпадения, то матрица A разбивается на три части, так, как это показано на рисунке.




В первой части мы должны продолжать поиск, если X меньше A[l,m]; в третьей части, если X больше A[l,m]; и во вторых частях в том и другом случаях.

```
Procedure Poick(i1,j1,i2,j2:Integer);
 Var l,m:Integer;
 Begin
  If pp Then Exit;{pp - глобальная переменная,  начальное
                                значение  -  false,  присваивается
значение true,
                        если      элемент X найден}
If (i1>i2) Or (j1>j2) Then Exit;
l:=(i1+i2) div 2; m:=(j1+j2) div 2;
p:=p+1; {p - счетчик числа сравнений,  начальное значение нуль
        при каждом новом поиске,  глобальная переменная }
If A[l,m]=X Then Begin { элемент X найден}
                                pp:=true End
Else Begin If X>A[l,m] Then Begin{ поиск в 3-й части}
                                        Poick(l+1,m,i2,j2);
                                        Poick(l,m+1,l,j2)
                                        End
                Else Begin { поиск в 1-й части}
                        Poick(l,j1,l,m-1);
                        Poick(i1,j1,l-1,m)
                        End;
                { поиск во 2-й части }
```

```
            Poick(l+1,j1,i2,m-1);
            Poick(i1,m+1,l-1,j2);
        End;
End;
```

Основная программа достаточно очевидна, поэтому приводить ее текст вряд ли целесообразно. Пример исходной матрицы A, результирующей матрицы C и значения ls:

A C ls=5.88

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | | 4 | 3 | 6 | 9 | 12 |
| 6 | 7 | 8 | 9 | 10 | | 5 | 4 | 5 | 10 | 11 |
| 11 | 12 | 13 | 14 | 15 | | 2 | 3 | 1 | 5 | 7 |
| 16 | 17 | 18 | 19 | 20 | | 9 | 11 | 3 | 2 | 4 |
| 21 | 22 | 23 | 24 | 25 | | 9 | 10 | 5 | 3 | 4 |

Для этой же матрицы A значение ls можно уменьшить за счет некоторой модификации процедуры Poick. Попробуйте.

o94_2 Суть решения.

```
If длина(Y)<длина(X) Then <Поменять значения X и Y> ;
 r:=длина(Y);
 For t:=1 To длина(X) Do Begin
  <взять символ с номером t из X>;
  If <этот символ не входит в Y> Then r:=r+1 Else Begin r:=r-1;
                                                    <удалить символ из Y>
                                                    End;
End;
```

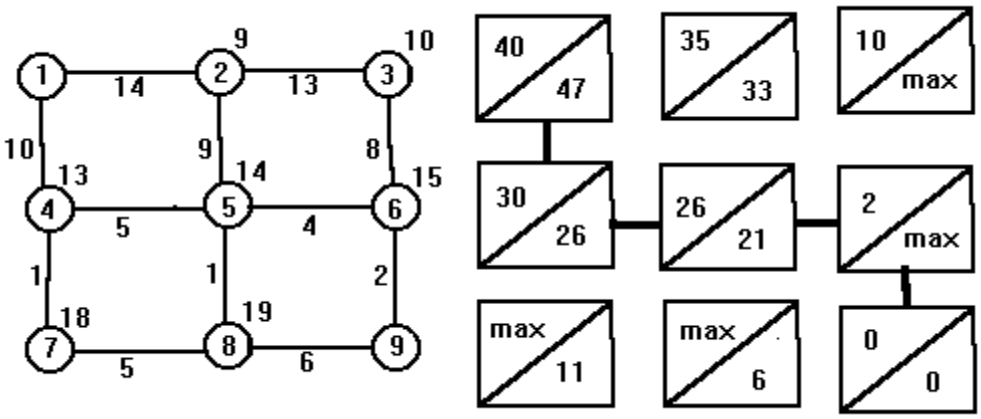
o94_3 Классический вариант рекурсивной задачи. Есть действие на входе в рекурсию - вывод строки. Есть действие на выходе из рекурсии - вывод строки. Есть условие завершения - вывод строки из символов J. Осталось определить параметры рекурсии - символ, количество символов, количество пробелов.

o94_4 Задача на метод динамического программирования. Определим для каждого перекрестка два числа: стоимость перемещения на юг и стоимость перемещения на восток. Начнем их вычисление с крайнего юго-восточного перекрестка. Логика вычисления имеет вид:

A[i,j]/восток+min{A[i,j+1]/восток;A[i,j+1]/юг+A[i,j+1]/поворот}

A[i,j]/юг+min{A[i+1,j]/юг;A[i+1,j]/восток+A[i+1,j]/поворот}.

При этом i изменяется от l-1 до 1, а j от m-1 до 1. Пример (max - обозначено отсутствие движения в этом направлении).




Итак, стоимость минимального маршрута равна 40. Дальнейшее решение зависит во многом от удачности выбора структур данных. Если определить:

```
Const NMax=11;
Type Cross=Record Rigth,Down,Turn:Integer; End;
Var A: Array[1..NMax,1..NMax] Of Cross;       ,
```

то программная реализация логики будет достаточно компактна.

o94_5 Очередной пример рекурсивной логики. Пусть P - функция, которая каждому натуральному числу N ставит в соответствие множество его разбиений. При N=1 - одно разбиение. Будем хранить разбиение числа N в массиве A(A:Array[1..N] Of Integer). В переменной k будем фиксировать число элементов разбиения. Например, если разбиение состоит из самого числа N, то A[1]=N, а все остальные элементы равны 0, при N=1+1+...+1 - все элементы A равны 1. Итак, если на место k массива A мы записываем число i, то с k+1 места нам необходимо последовательно записывать все разбиения числа Sum-i, где Sum - сумма элементов разложения, записанных в A до k-го места.

o94_6 Задача логически разбивается на следующие части:

- построение разреза между двумя точками на границе клеточного поля;
- подсчет площадей;
- вывод поля и разреза на экран.

Наиболее интересной является первая часть. Перенумеруем точки на границе поля по часовой стрелке. После этой операции появляется возможность перебирать по две точки и строить между этими точками разрез. Организация перебора очевидна. Номер первой точки (i ) изменяется от 1 до sp-1, где sp - количество точек на границе, номер второй от значения i+1 до sp. Для построения разреза используем идею классического “волнового алгоритма” - проведения «волны» между двумя точками плоскости или поиска выхода из произвольного лабиринта (глава 2). Дальнейшее решение зависит от удачности выбора структур данных. Целесообразно хранить координаты точек линии разреза. В этом случае площадь - это сумма площадей прямоугольников.

o94_7 Задача на перебор вариантов. Применима рекурсивная схема реализации. Требуется ответить на вопрос о том, что является подстановкой для каждой строчной буквы из строки X. Пусть мы нашли строчную букву в строке X в позиции с номером q и до позиции w в строке Y символы исчерпаны предыдущими заменами. Что нам необходимо сделать? Перебирать все возможные подстановки.

```
...
for i:=w to length(Y)-length(X)+q do begin
 <запомнить, что символу с номером q из строки X сответствует подстрока из Y, начиная с позиции w, длиной i-w+1>;
<продолжить рекурсивный перебор вариантов с позиции q+1 в строке X и позиции i+1 в строке Y>;
<отменить запоминание «вырезанной» подстроки из Y, соответствующей символу на месте q в строке X>;
end;
....
```

o94_8 Опишем связи между сотрудниками с помощью матрицы A. Для первого примера из формулировки задачи она имеет вид.

| | | | |
|---|---|---|---|
| 0 | 9 | 0 | 0 |
| 5 | - | 5 | 7 |
| 2 | 0 | 0 | 4 |
| 0 | 0 | -3 | 0 |

Элементы столбца с номером j описывают все влияния на рабочего с номером j. Итак, если из каждого столбца мы выберем один элемент, причем максимальный, то условие задачи «найти множество связей (влияний) максимального суммарного веса, на каждого рабочего должен влиять только один человек, все рабочие должны быть охвачены связями» будет выполнено. Это не что иное, как пример «жадной» логики.

В главе 3 рассмотрен алгоритм перечисления всех каркасов (остовов) неориентированного связного графа. Используем этот алгоритм. Строим все каркасы, начинающиеся с первой вершины (первый человек - лидер), начинающиеся со второй вершины и т.д. Как обычно, в задачах о перечислениях объектов запоминаем лучшее решение.

Для второго пункта задачи, естественно, возможны и другие решения. Одно из них - поиск пересечения матроидов, но это уже тема отдельного разговора.

## 5.7. Олимпиада - 95

r95_1 Задача на умение работать с различными типами конструкции повторения. Внешний цикл - по числам от 1 до N. Внутренний - по цифрам числа до тех пор, пока они образуют возрастающую или убывающую последовательности.

r95_2 Задача на знание тем массивы и вспомогательные алгоритмы. Требуется одновременный просмотр элементов массива с помощью двух индексов, обеспечивающих в каждой строке поиск очередного нуля слева и очередной единицы справа.

r95_3 Задача на знание строкового типа данных и умение организовать перебор вариантов. Относится к классическому типу задач "на перебор с возвратом". В алгоритме должно учитываться не только совпадение количества открывающих и закрывающих скобок, но и правильность их расстановки - количество закрывающих не должно превышать количества открывающих на любом фрагменте строки, начинающемся с первого символа.

r95_4 Очевидно, что хранить все числа Фибоначчи нет никакой необходимости. Достаточно знать, на какой цифре по номеру начинается последнее число B последовательности, два последних числа и количество цифр в последнем числе (B[0]). Остаток от вычитания из числа 1994-й значения номера цифры, на которую заканчивается предпоследнее число, позволяет определить положение 1994-й цифры последовательности в последнем найденном числе Фибоначчи. Для сложения двух последних чисел Фибоначчи должна использоваться длинная арифметика (см. задачу o91_3). Ответ задачи - цифра 3.

r95_5 Незначительная модификация, вероятно, самой известной задачи в информатике. Любопытна ее математическая трактовка как задачи о латинских квадратах с известной первой строкой. Подсчетом их количества для различных значений N Вы можете на долгое время загрузить свой компьютер.

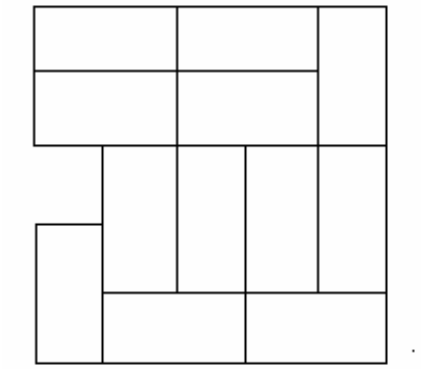
o95_1 Опишем рекурсивную схему реализации задачи (Rec). Параметрами схемы является количество +1 и -1 в сгенерированной части последовательности. Первый вызов - Rec(0,0). Шаг рекурсии. Если можно записать -1 (их количество меньше N и количество +1 больше количества -1), то записываем и переходим к следующему шагу рекурсии, иначе, если количество +1 меньше N, то к последовательности добавляем +1 и переходим на следующий шаг рекурсии. Первой последовательностью будет 1 -1 1 -1 ...1 -1, последней - 11...1 -1 -1 ... -1.

o95_2 Определим понятие ранга позиции. Это число ходов, которые должны сделать белые, чтобы дать мат. Основные структуры данных - массивы WF, BK array[1..8,1..8,1..8,1..8] of word. Элемент $WF[w_i,w_j,b_l,b_k]$ определяется по $w_i$, $w_j$ - номерам вертикали и горизонтали нахождения белого ферзя и $b_l$, $b_k$ - номерами вертикали и горизонтали черного короля. В структуре WF хранятся ранги позиций при ходе белых; BK - при ходе черных.

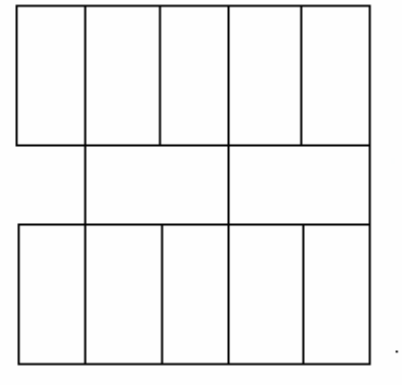
Первый этап решения заключается в заполнении WF и BK. Сначала отмечаются особые позиции: маты, паты и другие особые ситуации (король съел ферзя). Для матовых позиций в BK заносится 0. Принцип заполнения. Для каждого значения t=0, 1, 2, ... :

- Просматриваются все элементы $WF[w_i,w_j,b_l,b_k]$, не имеющие ранга. Если из этой клетки доски можно сделать ход ферзем в клетку $(w_{i1},w_{j1},b_l,b_k)$, в которой значение WF равно t, то $WF[w_i,w_j,b_l,b_k]$ присваивается значение t+1.
- Просматриваются все элементы $BK[w_i,w_j,b_l,b_k]$, не имеющие ранга. Если из этой клетки все ходы королем ведут в позиции, уже имеющие ранги в WF, то в $BK[w_i,w_j,b_l,b_k]$ заносится число t+1.
- Заполнение массивов заканчивается, если при очередном значении t не происходит изменение элементов WF.

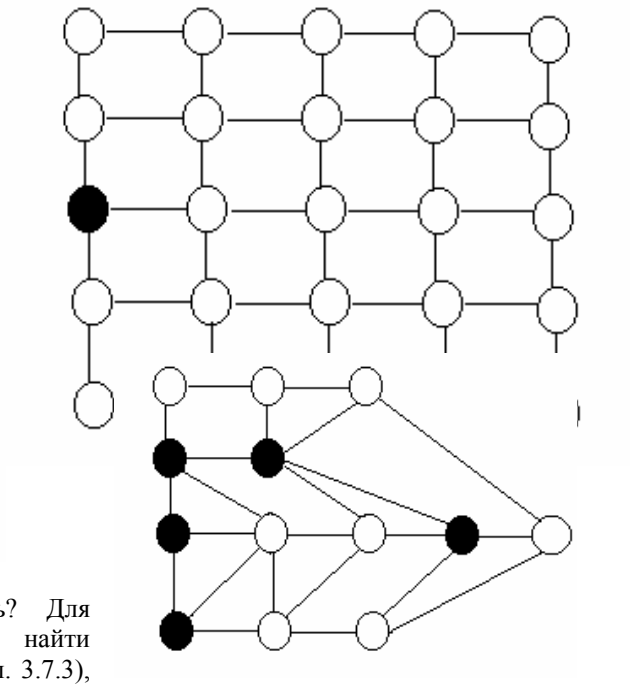
Второй этап решения (после заполнения массивов) - игра. Компьютер, играя белыми, делает очередной ход в позицию с рангом на единицу ниже ранга текущей позиции.



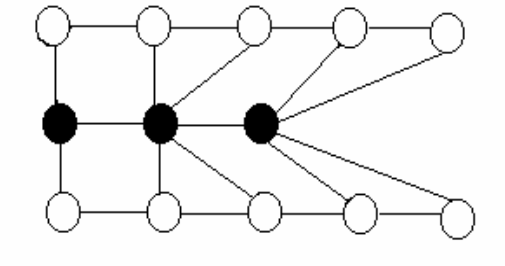
o95_3 Пусть N и M равны пяти и точка выезда имеет координаты (1,3). Есть две произвольные укладки поля (автостоянки) плитками 2*1, показанные на рисунке.



Все клетки поля и взаимосвязи между ними описываются графом.



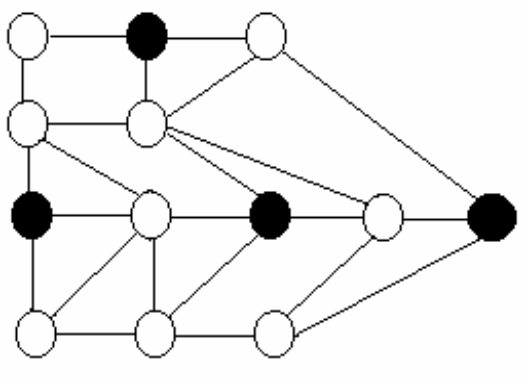
Сжимаем исходный граф. Укладкам соответствуют графы на следующих рисунках.





Итак, что необходимо сделать? Для каждого полученного графа требуется найти минимальное доминирующее множество (п. 3.7.3),

причем вершины этого множества должны образовывать связный подграф (вершины на рисунках выделены темным цветом). Так, для второй укладки доминирующее множество на следующем рисунке нас не устраивает, оно не связно.



Мы нашли идею решения задачи. Последовательно генерировать все укладки. Каждую укладку преобразовывать в граф. Найти для него минимальное связное доминирующее множество.

o95_4 Очевидно, что при L>M или K<2 решений нет. Предположим, что M>L. За две поездки перевозится K-1 представитель этого сообщества. Если (K-1) кратно (M+L-1), то требуется на одну поездку меньше, в противном случае - на одну больше.

Примеры. 1. M=9, L=8, K=5. (9,8,1)→ (6,6,0)→ (7,6,1)→ (4,4,0)→ (5,4,1)→ (2,2,0)→ (3,2,1)→ (0,0,0). Итого, семь поездок.

2. M=7, L=5, K=3. (7,5,1)→ (5,4,0)→ (6,4,1)→ (4,3,0)→ (5,3,1)→ (3,2,0)→ (4,2,1)→ (2,1,0)→ (3,1,1)→ (1,0,0)→ (2,0,1)→ (0,0,0). Итого, одиннадцать поездок.

3. M=6, L=5, K=6. (6,5,1)→ (3,2,0)→ (3,3,1)→ (0,0,0).

Тесты.

| M | L | K | Ответ |
|---|---|---|---|
| 4 | 4 | 2 | нет решения |
| 5 | 5 | 3 | 11 |
| 8 | 7 | 3 | 13 |
| 5 | 5 | 5 | 5 |
| 73 | 73 | 4 | 143 |
| 91 | 44 | 4 | 89 |
| 10000 | 9999 | 3 | 19997 |
| 10000 | 9998 | 4 | 13331 |

Не рассмотрен случай M=L(особый). Его следует рассматривать отдельно для каждого значения K. Пусть K=2. Если M>3, то решений нет. Действительно, при любом способе перевозки возникает ситуация, в которой количество людоедов и миссионеров на том и другом берегу совпадает и кто-то должен перегонять лодку. При M=3 схема перевозки приведена в условии задачи. При M=2 имеем (2,2,1)→ (2,0,0)→ 2,1,1)→ (0,1,0)→ (0,2,1)→ (0,0,0). Аналогично рассматриваются и другие случаи.

## 5.8. Олимпиада - 96

r96_1 Идея решения проста. Рассмотрим на примере.

Исходный массив: 4 1 9 3 5 2 7 8 0 6 n=10, m=5

Шаг 1 5 3 9 1 4 2 7 8 0 6 “перевернули” первую часть.

Шаг 2 5 3 9 1 4 6 0 8 7 2 “перевернули” вторую часть.

Шаг 3 2 7 8 0 6 4 1 9 3 5 “перевернули” весь массив - задача решена .

Итак, требуется “кирпичик” для перевертывания произвольной части массива и обращение к нему с различными участками исходного массива.

r96_2 Задача на знание темы строковый (символьный) тип данных и умение проводить логический анализ задачи Будем рассматривать символы с одним номером из всех трех текстов - колонка из трех символов. Выделим искаженные колонки - хотя бы один символ, отличный от других. Так как в трех текстах точно три искаженных символа, то искаженных колонок не более трех. Рассмотрим случаи.

- Три искаженные колонки. В каждой из искаженных колонок находится точно один искаженный символ, который легко восстанавливается, ибо два других совпадают.
- Ноль искаженных колонок. Случай возможен только при одной и той же ошибке во всех трех текстах - текст восстановить нельзя.
- Одна искаженная колонка. В каждом тексте искажен один и тот же символ - текст восстановить нельзя.
- Две искаженные колонки. Случай самый нетривиальный. В каждой искаженной колонке найдутся либо два одинаковых символа, либо все три попарно различны. Рассмотрим варианты. В обеих колонках символы попарно различны. Случай не возможен - четыре ошибочных символа. В обеих колонках по два совпадающих символа. При (А, А, В) и (Б, Б, Г) - восстановление неоднозначно. Варианта (А, А, В) и (Г, Б, Б) не может быть. В одной колонке два совпадающих, а в другой - все попарно различные символы, например - (А, А, Б) и (В, Г, Д). Текст восстанавливается однозначно.

r96_3 Пусть ($A_1$, ..., $A_n$) - произвольная последовательность множеств. Они могут и пересекаться и совпадать. Системой различных представителей для ($A_1$, ..., $A_n$) называют такую произвольную

последовательность ($a_1$, ..., $a_n$), что $a_i \in A_i$, $1 \le i \le n$, и $ai \ne aj$ для $i \ne j$. Говорят, что в такой системе элемент $a_i$ представляет множество $A_i$.

Теорема Холла. Система различных представителей данного множества существует тогда и только тогда, когда любые k множеств ($1 \le k \le n$) в совокупности содержат не менее k различных элементов.

Теорема Холла утверждает существование системы различных представителей, но не дает способа нахождения этой системы. Очевидно, что теорема Холла не имеет большого значения с алгоритмической точки зрения, так как проверка всех семейств подмножеств, а их $2^n$, требует значительных временных затрат. Рассмотрим метод поиска системы различных представителей для нашей задачи.

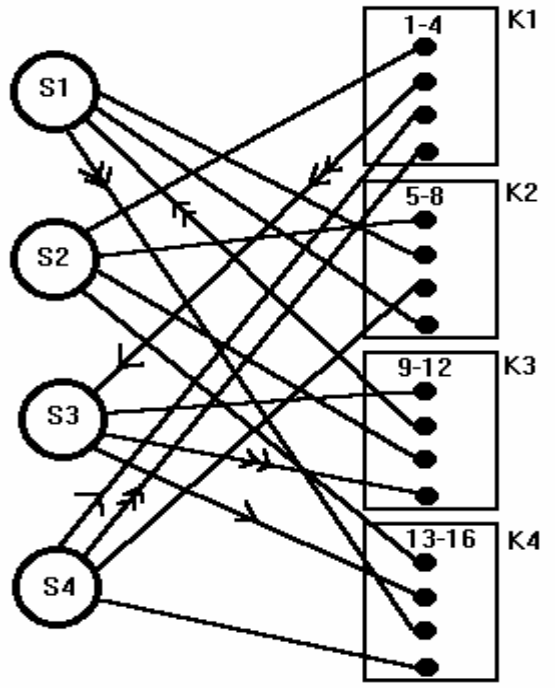
Обозначим элементы каждой строки нашей таблицы как $S_1$, $S_2$, $S_3$, $S_4$. Считаем, что есть множества $S_i$ ($1 \le i \le n$), составленные из элементов, которые необходимо разбить на n классов Ki по принадлежности к строке результирующей таблицы. Эти множества удовлетворяют теореме Холла, то есть любые q множеств содержат элементы не менее чем q классов.

Начинаем работу. *Первая итерация* (первый столбец правой таблицы). Из $S_1$ выбираем $6 \in K_2$, из $S_2$ -

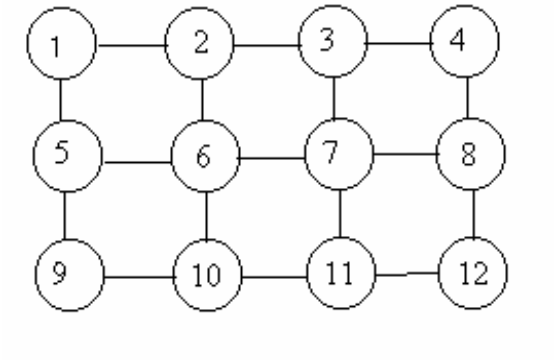
| | | | | |
|---|---|---|---|---|
| $S_1$ | 6 | 15 | 10 | 8 |
| $S_2$ | 5 | 13 | 1 | 11 |
| $S_3$ | 2 | 9 | 12 | 14 |
| $S_4$ | 7 | 3 | 16 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | 8 | 8 | 10 | 15 | 10 |
| 1 | 11 | 11 | 5 | 5 | 13 |
| 9 | 2 | 14 | 2 | 12 | 2 |
| 16 | - | 3 | - | 4 | 7 |

$1 \in K_1$, из $S_3$ - $9 \in K_3$, так как 2 выбрать нельзя, класс $K_1$ занят, из $S_4$ - 16, так как 3, 4 и 7 выбирать не имеем права по той же причине. *Вторая итерация*. По этой же логике из $S_1$ - 8, $S_2$ - 11, $S_3$ - 2. Выбрать из $S_4$ нечего - все классы заняты. Строим чередующуюся цепочку (глава 3, построение паросочетания в двудольном графе), она начинается в $S_4$ и состоит из ребер ($S_4$,3), (3,$S_3$), ($S_3$,14). Эти ребра на рисунке, приведенном ниже, помечены одинарными стрелками. Меняем представителя у $S_3$ на 14 из $K_4$, и для $S_4$ появляется представитель - 3 из $K_1$ (третий столбец правой таблицы). *Третья итерация*. Выбираем для $S_1$ - 10, для $S_2$ - 5, для $S_3$ - 2. Проблемы выбора для $S_4$. Строим чередующуюся цепочку: ($S_4$,4), (2,$S_3$), ($S_3$,12), (10,$S_1$), ($S_1$,15), она выделена на рисунке двойными стрелками и меняем представителей (пятый столбец правой таблицы). *Четвертая итерация*. Выбор представителей однозначен. Второй вертикальный и третий горизонтальный ходы очевидны. При вертикальном - каждый элемент столбца перемещаем в свою строку, а это можно сделать - у нас по одному представителю; при горизонтальном - каждый элемент строки перемещается в свой столбец.



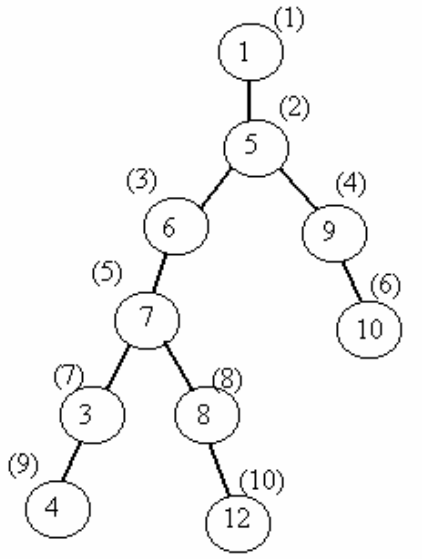

r96_4 В идейном плане задача традиционна - это обход в ширину, рассмотренный в главе 3. Продемонстрируем логику на примере. Дан город (схематическое изображение на рисунке)




и следующие значения высот для перекрестков.

| | | | |
|---|---|---|---|
| 100 | 110 | 40 | 30 |
| 90 | 60 | 50 | 20 |
| 80 | 70 | 90 | 10 |

Обход в ширину дает кратчайший путь. Последовательность просмотра перекрестков (в круглых скобках у номеров перекрестков) приведена на рисунке.




o96_1 Трехзначное число, состоящее из различных цифр, возводится в квадрат. Результат - пятизначное число, также состоящее из различных цифр, причем не совпадающих с цифрами трехзначного числа. Диапазон трехзначных чисел, удовлетворяющий условию задачи, очевиден - от 102 до 316. Итак, в решении задачи одна циклическая конструкция типа for, логика выделения цифр из числа и грамотно организованные проверки на несовпадение.

Ответ: 209 и 259.

o96_2 Рассмотрим «ключевые» моменты задачи. Определим основные структуры данных.

```
P:array[1..6] of integer; {хранится номер точки}
Pos: array[1..6] of Record x,y:integer End;{координаты точки}
```

Введем координатные оси. Начало - точка с номером 1, ось X идет по левой стороне треугольной сетки, ось Y - параллельна основанию. Преобразуем номер точки в ее координаты.

```
Procedure GetXY(k:word; var x,y:integer);;
        var i:word;
        begin i:=1; while i*(i+1) div 2<k do Inc(i);
                        x:=i;y:=k-i*(i-1) div 2
        end;
```

Так, точка с номером 10 имеет координаты (4,4). Естественно, что это преобразование осуществляется при вводе данных.

По условию задачи стороны многоугольника идут по ребрам треугольной сетки и имеют равные длины. Пусть мы имеем две точки с номерами u, v и длину предыдущей стороны многоугольника (значение переменной Len). Ответим на вопрос - могут ли быть соединены стороной многоугольника точки u,v.

```
function Can(u,v:word):boolean;
 var r: word;
 begin
  r:=0;
  if Pos[u].x=Pos[v].x then r:=abs(Pos[u].y-Pos[v].y){параллельна оси Y}
  else if Pos[u].y=Pos[v].y then r:=abs(Pos[u].x-Pos[v].x) {параллельна оси X}
   else if (Pos[u].x-Pos[u].y)=(Pos[v].x-Pos[v].y) then r:=abs(Pos[u].x-Pos[v].x); {параллельна правой стороне треугольной сетки}
 if r=0 then Can:=false {соединить нельзя} else begin Can:=true;
  if Len=0 then Len:=r {первая сторона}else Can:=(Len=r) end;
 end;
```

Естественный путь дальнейшей “раскрутки” решения - перебор всех вариантов построения геометрической фигуры на заданном множестве введенных точек. Если рассматривать это множество как вершины графа, то необходимо найти гамильтонов цикл с ограничениями, приведенными в функции Can. При этом мы должны уметь выяснять для любой точки с номером k возможность построения из нее очередной стороны многоугольника.

o96_3 Достаточно известная задача. Обычно решается для небольших значений N. Пусть N=2, тогда следующая простая логика дает количество счастливых четырехзначных чисел в интервале от 0000 до 9999.

```
...
cnt:=0;
for i:=0 to 9 do
 for j:=0 to 9 do
  for l:=0 to 9 do
   for k:=0 to 9 do
    if i+j-l-k=0 then inc(cnt);
...
```

Как сделать ее независимой от значения N? Может быть, так:

```
Function Rec(k,Sum:integer):comp;
 var i:integer;Ans:longint;
 begin
  if k=2*N+1 then if Sum=0 then Rec:=1{достигли конца числа и сумма цифр равна нулю - число счастливое}
                                  else Rec:=0
                  else begin
                         Ans:=0;
                         for i:=0 to 9 do
                          if k<=N then Ans:=Ans+Rec(k+1,Sum+i)
                           else if i<=Sum then Ans:=Ans+Rec(k+1,Sum-i);
                         Rec:=Ans;
                         end;
 end;
```

Красивая реализация, но ответ можно получить лишь для небольших значений N. Задача не решена, ибо среди тестов были и такие: N=8, A=1122334455667788, B=9999999999999999 или N=10, A=27182818284590452353, B=31415926535897932384. Требуется сократить повторные «просчеты» в этой рекурсивной схеме. Параметрами рекурсии являются k - номер позиции в числе и Sum - сумма цифр в числе до позиции k (подсчет слева направо). Например, при N=2 вызов Rec(4,2) или Rec(3,8) идет многократно. Сократим эти повторные просчеты. Введем структуру данных Matr (Matr:array[1..20,0..90] of comp;). Элемент Matr[k,Sum] предназначен для хранения количества счастливых чисел длины 2N таких, что сумма (k-1)-й цифры числа равна значению Sum. Первоначальное значение элементов Matr равно -1. В таблице указаны значения элементов Matr при N=2. Если элемент не равен -1, то «в клетке» два числа. Первое - это число вариантов, второе - номер в очередности заполнения элементов Matr.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13-18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 658<br>40 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | ... |
| 55<br>21 | 57<br>23 | 69<br>25 | 73<br>27 | 75<br>29 | 75<br>31 | 73<br>33 | 69<br>35 | 57<br>37 | 55<br>39 | -1 | -1 | -1 | ... |
| 1<br>2 | 2<br>4 | 3<br>6 | 4<br>8 | 5<br>10 | 6<br>12 | 7<br>14 | 8<br>16 | 9<br>18 | 10<br>20 | 9<br>22 | 8<br>24 | 7<br>26 | ... |
| 1<br>1 | 1<br>3 | 1<br>5 | 1<br>7 | 1<br>9 | 1<br>11 | 1<br>13 | 1<br>15 | 1<br>17 | 1<br>19 | -1 | -1 | -1 | ... |

Итак, второй вариант функции Rec (в прямоугольнике повторяется часть, выделенная курсивом в первом варианте функции).

```
Function Rec(k,Sum:integer):comp;
 var i:integer;Ans:comp;
 begin
  if k=2N+1 then if Sum=0 then Rec:=1 else Rec:=0
   else if Matr[k,Sum]<>-1 then Rec:=Matr[k,Sum]
          else begin
                 [          ]
               Matr[k,Sum]:=Ans;
               end;
 end;
```

Сокращение времени перебора трудно оценить аналитически. Однако понимания можно добиться, рассматривая, например, вопрос о том, сколько раз мы обращаемся в нашем примере к элементу Matr[3,6] после его первого вычисления.

Остался еще один непоясненный момент. Нам необходимо считать счастливые числа не во всем интервале от 00...0 до 99...9, а в его части - от некоторого числа A до числа B. Он решается путем введения дополнительного параметра в рекурсивную схему - логической переменной, фиксирующей факт совпадения рассматриваемого числа с заданным.

o96_4 Решение задачи в “лоб” - перебор, безнадежно. Так, при n=2 ответом является число 105263157894736842. В каком же диапазоне перебирать? А идея «лежит на поверхности». Умножая известную первую цифру числа на n (т. е. n*n), мы должны получать вторую цифру и может быть что-то в переносе. Умножая n на вторую цифру, а мы ее знаем, - третья цифра плюс перенос. Процесс заканчивается, если перенос равен нулю и очередная цифра равна n.

o96_5 Первое, о чем необходимо догадаться, это то, что в коде указывается толщина белых и черных линий. Второе: первые и последние три единицы, а также пять единиц в середине кода являются разделителями. Отбрасывая разделители и анализируя примеры, мы можем определить кодировку всех цифр

от 0 до 9, кроме кода цифры 7. Ее нет в примерах. Результатом этой работы являются значения элементов массива констант:

```
const Code: array[Boolean,0..9] of String[4] = ((‘1123’,’1222’,’2212’,’1141’,’2311’,’1321’,’4111’,’’,’3121’,’2113’),(‘3211’,’2221’,’2122’,’1411’,’1132’,’1231’,’1114’,’’,’1213’,’3112’));
```

Первый индекс указывает на тип кода - прямой, обратный. После этого осталось выделять по 4 символа из строки и сравнивать их с элементами массива Code. Если результат сравнения отрицательный, то эти 4 символа - код цифры 7, так как в условии задачи не сказано, что могут существовать некорректные данные. Небольшой нюанс - не забыть о совпадении результата при прямом и обратном кодах.

o96_6 Для проверки корректности кода можно отсортировать введенную строку из 8 символов. Ее значение после этой операции должно быть “__ИОРССЯ”. Кроме этого следует добавить проверку следования символов подчеркивания (то, что они записаны подряд). При выполнении хода следует учесть “переход через границу” - сектора 8 и 1.

Второй пункт задания реализуется методом “обхода в ширину”. Его реализация априори дает кратчайшую последовательность ходов. Суть его применения. Из каждого состояния круга генерируются все состояния, в которые можно попасть из него. Запоминаем состояния в структуре данных типа очередь. Так, из первого состояния мы формируем элементы очереди со 2-го по 6-й.

| |
|---|
| ОЯ__ССИР |
| __ОЯССИР |
| ОЯСС__ИР |
| ОЯСИС__Р |
| ОЯИРСС__ |
| _ЯРОССИ_ |
| ... |

После этого по значению указателя считывания из очереди берем элемент “__ОЯССИР” и генерируем все состояния, получаемые из него. Повторяющиеся состояния в очередь не записываются. Процесс генерации заканчивается при получении состояния (после удаления из него символов подчеркивания), совпадающего с состоянием “РОССИЯ”. Осталось сделать вывод последовательности ходов. Если не использовать списковые структуры данных, а на олимпиадах из-за ограниченного времени это нежелательно, то придется ввести дополнительный массив для хранения значений указателей - из какого состояния (его адрес в очереди) получен каждый элемент очереди. Просмотр из конечного состояния очереди по значениям указателей дает последовательность ходов, но только в обратном порядке. Вывод в прямом порядке - чисто техническая деталь решения.

## 5.9. Олимпиада - 97

r97_1 При вводе элементов масива ключей (A) подсчитываем количество ключей первого (N1) и второго (N2) типов. Это позволит определить место окончания записи ключей этих типов в отсортированном массиве. Определим индексы: i - изменяется от 1 до N1 (по ключам первого типа); l - от N1 до N1+N2 (по ключам второго типа); j - от N3 (общее количество ключей) до N1+N2+1. Тогда сортировка выполняется с помощью единственного цикла:

```
i=1;l:=N1+1;j:=N3;
while (i<=N1) or (l<j) do
```

| |
|---|
| Изменение индексов; перестановка ключей |

Будем запоминать перестановки элементов массива A в массиве P. Нам потребуется следующая процедура:

```
procedure Change(v,w:integer;c1,c2:char);
 begin
  inc(cnt);P[cnt,1]:=v;P[cnt,2]:=w;A[v]:=c1;A[w]:=c2;
 end;
```

Анализ блока «изменения индексов и перестановки ключей» приведен в таблице.

| i | l | j | Действие |
|---|---|---|---|
| A[i]=ch1 | - | - | inc(i) |
| - | A[l]=ch2 | - | inc(l) |
| - | - | A[j]=ch3 | dec(j) |
| A[i]=ch2 | A[l]=ch1 | - | Change(i,l,ch1,ch2) |
| A[i]=ch2 | A[l]=ch3 | A[j]=ch1 | Change(i,j,ch1,ch2)<br>Change(l,j,ch2,ch3) |
| A[i]=ch2 | A[l]=ch3 | A[j]=ch2 | Change(l,j,ch2,ch3) |
| A[i]=ch3 | - | A[j]=ch1 | Change(i,j,ch1,ch3) |
| A[i]=ch3 | A[l]=ch1 | A[j]=ch2 | Change(l,j,ch2,ch1)<br>Change(i,j,ch1,ch3) |
| A[i]=ch3 | A[l]=ch3 | A[j]=cch2 | Change(l,j,ch2,ch3) |

r97_2 Последовательно проверяем клетки, содержащие суммы, и пытаемся выполнить вычисления. Если сумму найти можно, то записываем результат в таблицу. При этом фиксируем значение признака того, что «произошло вычисление». Процесс вычислений продолжается до тех пор, пока не будут определены все значения сумм или окажется, что среди невычисленных нет ни одной, которую можно получить.

```
...
repeat
```

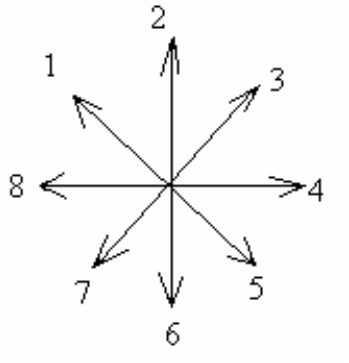
Вычисление сумм

until <вычислены все суммы> or <не было вычислений>;

r97_3 Очевидно, что логика рекурсивная. Если очередной раз мы ее вызвали как Rec(X,Y,Lengt,Angle,Lev:integer), то в ней должно быть два вызова Rec(NewX,NewY,Round(Lengt*0.56),Angle1,Lev+1) и Rec(NewX,NewY,Round(Lengt*0.56),Angle2,Lev+1). Следовательно, требуется по значениям X, Y, Angle вычислить NewX,NewY, Angle1 и Angle2. Определимся с углами. Ветки дерева могут продолжаться в одном из следующих направлений (пронумеруем их).

В таблице показано, какие направления получаются из текущего (первый столбец) угла (Angle).

| Angle | Angle1 | Angle2 |
|---|---|---|
| 1 | 2 | 8 |
| 2 | 3 | 1 |
| 3 | 4 | 2 |
| 4 | 5 | 3 |
| 5 | 6 | 4 |
| 6 | 7 | 5 |
| 7 | 8 | 6 |
| 8 | 1 | 7 |




Итак, Angle1=(Angle+1) Mod 8, а Angle2=(Angle+7) Mod 8. Осталось выяснить логику вычисления NewX, NewY. Заметим, что если направление нечетное, то Lengt следует умножить на 0.70710678 (sqrt(2)/2) и на знак приращения; если направление кратно четырем, то изменяется только X, а при направлениях 2 и 6 - только Y (с учетом знака приращения). Знак приращения также является функцией направления и легко вычисляется. Этих фактов оказывается достаточно для определения NewX, NewY.

o97_1 Решение получаем в результате простой итеративной логики. Предположим, что у нас есть решение для N гвоздей, то есть известна минимальная длина связывания веревочками. Вводим гвоздь с номером N+1. Он соединяется с гвоздем с номером N, а из предыдущих соединений выбираем минимальное из связываний для N и N-1 гвоздей.

o97_2 Рассмотрим идею решения задачи на примере.

| | 0 | a | a | a | b | b | b | b | c | c | c |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| a | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| b | 0 | 0 | 0 | 0 | 3 | 6 | 9 | 12 | 12 | 12 | 12 |
| c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 24 | 36 |

Для клеток, у которых символы на горизонтали и вертикали не совпадают, значение переписывается из соседней по горизонтали клетки. Например, символ ‘a’ по вертикали и первый символ ‘b’ по горизонтали. Способов получение строки ‘a’ из строки ‘aaab’ столько же, сколько и из строки ‘aaa’. Для случая совпадения символов, например ‘b’ по вертикали и второе ‘b’ по горизонтали, количество способов - это сумма способов для строк ‘ab’ и ‘aaab’ (без последнего символа во второй строке), а также - ‘a’ и ‘aaab’ (последние символы строк совпадают).

o97_3 Метод решения задачи традиционный - перебор вариантов. Однако из-за ограничений на время тестирования его следует организовать достаточно грамотно. Заметим, что первоначально необходимо выбрать числа для помеченных клеток таблицы. Если это сделано, то часть клеток заполняется автоматически из знания возможной суммы элементов. Остается найти число, записываемое в клетку [2,1], оставшиеся три клетки заполняются по принципу дополнения до суммы. Единственное значение суммы при котором есть решение, 40 (оно указано в подсказке - количество баллов за задачу). Его вычисление тривиально, сложить все числа и разделить сумму на шесть. В программе достаточно пояснений, она хорошо читается.

o97_4 Для решения задачи требовалось установить, что число частей равно сумме числа диагоналей, числа точек пересечения диагоналей плюс единица. Диагонали (i,j) и (k,l) пересекаются, если выполняется следующее условие (k-i)*(k-j)*(l-i)*(l-j)<0.

o97_5 Заметим, что клетчатая доска дана для “устрашения”. С таким же успехом задачу можно решать на линейке из M*N клеток. Допустим, что найдено решение для L клеток, то есть определено число способов для всех значений колечек от 1 до K. При L+1 клетке решение получается очевидным способом - в клетке L+1 размещаем p колечек, а в клетках от 1 до L - оставшиеся. Традиционная схема, называемая динамическим программированием. Разумеется, это не единственный метод решения задачи.

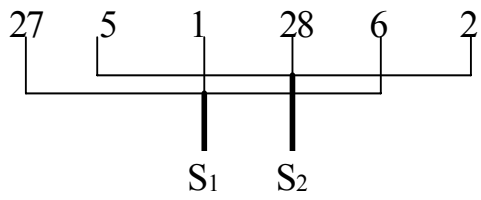
o97_6 Решение задачи достаточно очевидно, если построить соответствующую графовую модель. Преобразуем данные так, что фигура из спичек описывается графом, причем вершинами графа являются не только концы спичек, но и их середины. Время сгорания ребра этого графа равно единице. Найдем расстояния (измеряются количеством ребер графа) между всеми парами его вершин (алгоритм Флойда). Пусть расстояния записываются в матрицу C. Элементы строки матрицы с номером i - это расстояния от вершины i до всех остальных вершин графа. Тогда для ответа на первый пункт задачи достаточно найти в каждой строке максимальный элемент и из этих максимальных выбрать минимальный. Для ответа на второй пункт задачи требуется проверка того, что расстояния до вершин графа, определяющих концы одной спички, совпадают и, кроме того, расстояние до вершины графа, соответствующей середине спички, больше этих расстояний.

## 5.10. Олимпиада - 98

r98_1 Текста программы решения задачи достаточно для ее понимания.

r98_2 При вводе последовательности подсчитываем сумму элементов на нечетных и четных местах ($S_1$ и $S_2$). Если $S_1>S_2$, то начиная с первого всегда берутся элементы последовательности с нечетных мест, в противном - с четных.

Пример.




| Номер хода | Последователь-ность | $S_1$ | $S_2$ |
|---|---|---|---|
| 1 | 27 5 1 28 6 | 2 | 0 |
| 2 | 5 1 28 6 | 2 | 27 |
| 3 | 1 28 6 | 7 | 27 |
| 4 | 1 28 | 7 | 33 |
| 5 | 1 | 35 | 33 |
| 6 | | 35 | 34 |

r98_3 Классический вариант переборной задачи. Определим основные структуры данных. Элемент массива Nnew (array[char] of boolean) показывает, присвоен ли соответствующему символу его числовой эквивалент. В Num (array[char] of char) хранятся цифры (представленные в типе данных char), присвоенные символам исходной строки. И наконец, элемент P (array[0..9] of boolean), равный false, говорит о том, что соответствующая цифра уже задействована при разгадывании ребуса.

Итак, основная логика.

```
Procedure rec(k:word;var s:string);
{k - номер позиции в исходной строке, s - преобразованная строка ребуса, вместо исходных символов записаны символы соответствующих цифр, str - исходная строка}
var i:word;
 begin
  if k>length(str) then begin
                          <преобразование s и проверка ребуса>;
                            end
    else if Not (Nnew[str[k]] then begin
                                     s:=s+Num[str[k]];
                                     rec(k+1,s);
                                     delete(s,k,1);
                                     end
         else if (str[k]='+') or (str[k]='=') then begin
                                                     s:=s+str[k];
                                                     rec(k+1,s);
                                                     delete(s,k,1);
                                                   end
              else for i:=0 to 9 do
                     if P[i] then begin Nnew[str[k]]:=false;
                                  Num[str[k]]:=Char(i);
                                  P[i]:=false;
                                  s:=s+Char(i);
                                  rec(k+1,s);
                                  delete(s,k,1);
                                  P[i]:=true;
                                  Nnew[str[k]]:=true;
                          end;
```

end;

o98_1 В случаях, когда N нечетно или есть нули и их меньше половины, решение отсутствует. Если количество нулей больше половины, то ответ задачи 0. Итак, N четно и нулей нет. Отсортируем числа по неубыванию. Удобнее это сделать отдельно для отрицательных и положительных чисел. Рассмотрим пока случай только положительных чисел. Есть последовательность $a_1 \le a_2 \le ... \le a_{n-1} \le a_n$. Необходимо вычислять произведения типа $a_i * a_{n-i+1}$ (i=1,2,...,n div 2). Только в этом случае задача имеет решение. Действительно, пусть не так, мы рассматриваем произведение, например, пар $a_2 * a_n$ и $a_1 * a_{n-1}$. После несложных преобразований легко усматривается, что выражение ($a_2 * a_n$ - $a_1 * a_{n-1}$) всегда больше нуля. Для последовательности 2 6 12 14 28 84 вычисляем произведения 2*84, 6*28, 12*14. Обобщим результат и посмотрим, нет ли «подводных камней». Есть отрицательные числа. Моментально «напрашивается» ограничение - их количество равно количеству положительных чисел. Логика перемножений в этом случае: очередное наименьшее отрицательное умножаем на очередное наименьшее положительное. Например, для последовательности -14 -7 -1 2 4 28 схема перемножения имеет вид -14*2, -7*4 и -1*28. Оказывается есть еще особый случай, когда количество как отрицательных, так и положительных чисел четно. Задача может иметь как один, так и два ответа, т.е. есть два способа перемножения. Например, для последовательности -84 -28 -14 -12 -6 -2 2 6 12 14 28 84 ответы 168 и -168.

o98_2 Ответим на вопрос, как описывается состояние часов в некоторый момент времени? Количеством шариков в первой, во второй, в третьей и т. д. корзинах. А это число, но не в привычной десятичной системе счисления, а в смешанной. Первая справа цифра по основанию 5, следующая - 6 и т.д. Запишем состояния часов в моменты времени T и T+P, например 740253 и 740432. А что дальше? Необходимо понять, сколько шариков требуется для того, чтобы часы циркулировали от момента времени T до T+P. Они обязаны «перевалить» через момент времени, которому соответствует число из интервала [T,T+P] с максимальной суммой цифр в этой системе счисления (таков должен быть запас большой корзины). Для нашего примера - число 740354 и сумма цифр равна 12 (вычисляем от первой справа цифры, в которой есть несовпадение). Сумма соответствующих цифр числа 740253 равна 10, а ответ 12-10+1=3, то есть в большой корзине должно быть 3 шарика.

o98_3 Пусть есть два переключателя A и B. Данные по ним соответствуют двум строкам матрицы W[1..M,1..N]. Рассмотрим операцию сложения по модулю 2 (для одного разряда переключателей).

| A | B | A+B | (A+B)+A≡B |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |

Какие выводы можно сделать из этого факта? Появляется возможность заменить в W данные по переключателям A и B на данные по A и (A+B) - результат действий не изменится, ибо последний столбик говорит о том, что действие переключателя B равносильно действиям (A+B)+A. Из этого следует, что W преобразуется к виду, когда на главной диагонали матрицы записаны 1, а все элементы ниже главной диагонали равны нулю. После такого приведения W логика обработки сводится к поиску лампочек, у которых начальное (а затем текущее) состояние не совпадает с конечным, и к применению переключателя с единицей на главной диагонали, соответствующей этим лампочкам. При преобразовании строк матрицы W следует помнить, из каких строк она получается. Для примера, приведенного в условии задачи, описанные действия выглядят следующим образом.

| | W | Строки, логическая сумма которых приводит к данной строке (Who:array[1..M] of set of byte) |
|---|---|---|
| 1 | 11100<br>10001<br>10110 | [1]<br>[2]<br>[3] |
| 2 | 11100<br>01101<br>01010 | [1]<br>[1,2]<br>[1,3] |
| 3 | 11100<br>01101<br>00111 | [1]<br>[1,2]<br>[2,3] |

Результат. Применяем второй (новый) переключатель. Состояние лампочек 11111, результирующее множество задействованных переключателей Res равно (Res∪Who[2])\(Res∩Who[2])=[1,2]. Применяем переключатель 3 к новому состоянию лампочек, получаем 11000, множество задействованных переключателей есть (Res∪Who[3])\(Res∩Who[3])=[1,3]. Если полученная строка состояний лампочек совпадает с конечной, то ответ на вопрос задачи положительный, если нет, то преобразование лампочек из начального в конечное состояния невозможно. Ниже приведен текст программы. Его отличие от приведенных рассуждений в том, что в структуре программы блок приведения W не выделен как отдельная процедура.

o98_4 Если N равно 1, то выводим текущий символ и заканчиваем обработку. При N, не равном 1, смотрим, больше ли N половины длины последовательности. При положительном результате сравнения уменьшаем значение N на это значение и продолжаем сравнение.

o98_5 Сортировка исходного массива по неубыванию очевидна. А затем в первую группу объединяем числа, попадающие в интервал $a_1$+2L, во вторую группу - первое число, не попавшее в предыдущий интервал - $a_k$, и числа $a_k$+2L. Этот итеративный процесс продолжаем до тех пор, пока не будут рассмотрены все элементы массива. Утверждение - описанная операция дает в результате наименьшее количество чисел. Пусть по этой логике мы получаем группы $G_1$, $G_2$, ..., $G_t$ и есть оптимальная разбивка - $Q_1$, $Q_2$, ..., $Q_r$, причем $r \le t$. Рассмотрим числа, принадлежащие к группе $Q_1$. Каждое из них можно отнести к одной из групп $G_i$, перебросим эти числа в группы $G_i$. С группами $Q_2$, ..., $Q_r$ поступаем аналогичным образом. В результате получаем, что $t \le r$, следствие $t=r$.

o98_6 Задача нахождения координат прямоугольника, получающегося в результате пересечения исходных, сводится к поиску общей части некоторого заданного множества интервалов на прямой. Последнюю задачу решаем дважды для осей X и Y.

# Заключение

Читатель, вероятно, захочет выполнить некоторые программы из этой книги на компьютере. Программы написаны на языке программирования Турбо Паскаль 7.0. Попытки воспроизведения программ в других диалектах языка потребуют, может быть, их изменения. Чтение текстов программ такое же необходимое умение специалиста по информатике, как и их написание, поэтому не следует пренебрегать работой с текстом чужой программы.

При работе с книгой Вы, наверное, обратили внимание на то, что большинство программ написано по одной и той же схеме.

```
Program имя;
Procedure Init;
 var f:text;a:integer;
 begin
  assign(f,'input.txt');
  reset(f);
  while not seekeof(f) do begin ...
                                        read(f,a);
                                        ...
                                        end;
  close(f);
  {инициализация всех глобальных переменных}
 end;
Procedure Solve; {или Rec}
 begin
          ...
 end;
Procedure Print;
 var f:text;
 begin
  assign(f,'output.txt');
  rewrite(f);
  close(f);
 end;
begin
 Init;
 Solve;
 Print;
end.
```

Программа работает, и она должна оставаться работающей в течение всего времени решения задачи.

Приведем ряд практических рекомендаций для дальнейшей работы с программой.

- Оценить идею решения задачи, временную сложность метода. «Прокрутить» логику решения на конкретном примере. В том случае, если Вы видите, что для данной размерности задачи у Вас нет разумных идей, то сделайте задачу для меньшей размерности. В процессе экспериментов с работающей программой Вы можете найти новые идеи решения, разумные эвристики.

- Если есть идея решения, то выбрать структуры данных и после неоднократного чтения условия задачи согласовать с ними процедуры Init и Print.
- Дальнейшие Ваши действия должны соответствовать технологии нисходящего проектирования (глава 1). Не пишите, по возможности, полный текст программы (это дурной тон). Делайте это поэтапно. На каждом этапе у Вас должна быть компилируемая и работающая программа (не забывайте сохранять ее!!!).
- При написании программы у Вас должен формироваться список контрольных точек. В каждой из них Вы должны полностью представлять значения всех переменных программы. Контроль - пошаговый режим работы программы в контрольных точках. Процесс разработки программы - это подключение к работающей версии вновь написанных процедур и функций. Если состояние данных не соответствует Вашим представлениям, то ни в коем случае не откладывайте выяснение причины «на потом». Ошибки имеют свойство накапливаться. А это приводит к трагическим последствиям - программа написана, но Вы не можете ее отладить. Если бы было время, то это не очень страшно (забросить программу на недельку - стандартный прием), однако обычно его нет.
- У Вас есть полностью работающая (на Ваш взгляд) версия программы. Обычная история, когда на этом процесс работы с программой заканчивают, а он только начинается. Необходимо создать достаточно полную систему тестов и проверить программу. Особо следует обратить внимание на «граничные тесты», ибо программа может работать на «средних» примерах, но давать неверный результат, когда ряд параметров задачи принимают граничные значения.