Блок-6 Машины (обязательные) по теме «Рекурсия, часть 3»

Изменения в ханойских (визуал)

(всего 3 задачи, срок их сдачи до 22 ноября включительно):

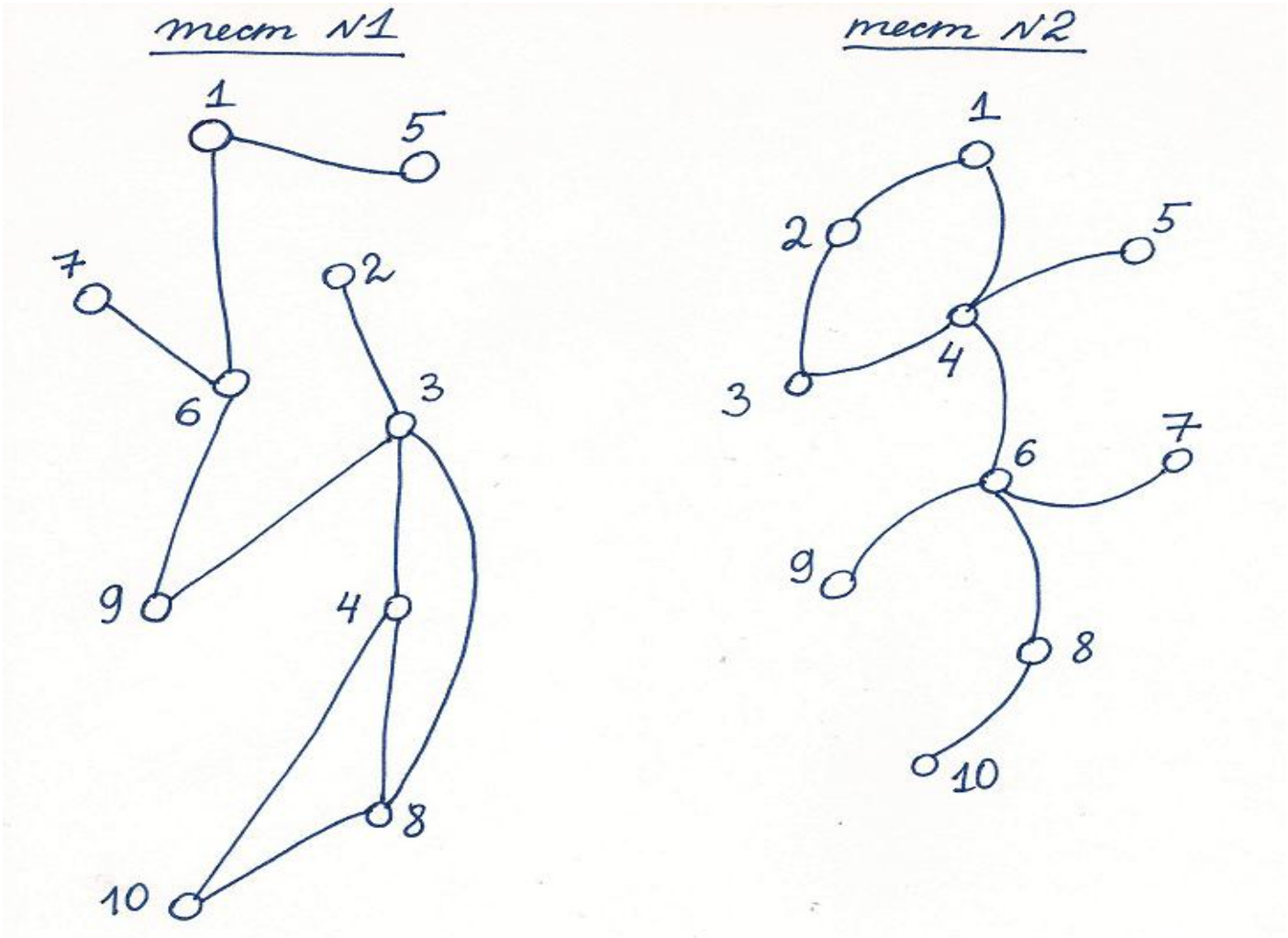
Задача 1 **12.35 (из задачника)** (без **goto** !) – сначала разобрать задачу **12.34**, решённую на семинаре в субботу 31.10.2020

Программа должна быть оттранслирована для значения **n=10**.

В основной программе сформировать булевскую **матрицу смежности** (подробности см. ниже). Сначала инициализировать (заполнить) все её элементы значением `false`, а затем (в цикле) запросить у пользователя все пары городов, соединённых дорогой, и на основании этой информации скорректировать значения соответствующих элементов матрицы (см. **второй абзац** условия задачи **12.35**).

Далее программа должна запросить у пользователя номера начального (`first`) и конечного (`last`) городов, выполнить поиск пути (любого из возможных) между этими городами и выдать ответ о результатах поиска. Если путь существует, программа должна распечатать найденный путь (любой из возможных) в виде номеров городов (включая начальный и конечный города).

Перед отправкой тщательно проверить работу программы на следующих двух тестах:



см. далее

Подробности решения (для нуждающихся):

В основной программе сформировать **матрицу смежности** Roads:

`var Roads: array[1..n,1..n] of boolean;` {где n – число городов}

Матрица Roads хранит информацию о наличии прямой дороги из города **i** в город **j**. Если город **i** соединен прямой дорогой с городом **j**, то `Roads[i,j]=Roads[j,i]=true` (т.к. движение двустороннее). Удобно считать, что все `Roads[i,i]=false` .

В основной программе изначально заполнить все элементы этой матрицы значениями `false`, а далее, по мере ввода данных (полученных от пользователя) заменять нужные элементы значениями `true`. Итак, полученная матрица симметрична, значения на главной диагонали = `false`.

Завести **массив посещений** Visited, отражающий, в каких городах уже побывали в процессе поиска пути (`Visited[i]=true`, если на данный момент поиска мы уже побывали в городе `i`):

`Visited: array[1..n] of boolean;` {до начала поиска пути заполнить все его элементы значениями `false`, кроме одного: `Visited[first]:=true`, так как сейчас уже находимся в городе с номером `first`, из которого стартуем}

Завести **массив** Way **для хранения пути** (фиксирует пункты/города пройденного пути):

`Way: array [1..n] of 1..n;` {до начала поиска пути выполнить присваивание `Way[1]:=first`, так как начальный пункт маршрута – это город с номером `first`}.

Завести также целочисленную переменную Length для вычисления **длины искомого пути**, т.е. общего количества пройденных городов, включая начальный и конечный пункты. До начала поиска выполнить `Length:=1`, так как в исходном городе (с номером `first`) уже находимся к моменту поиска. Очевидно, что длина искомого пути не будет превышать общего количества городов.

Описать рекурсивную **логическую функцию Path(t1,t2)**, которая проверяет (на основе анализа матрицы **Roads** и с учетом информации в массиве **Visited**), есть ли путь из города **t1** в **t2**.

Замечание: матрицу **Roads**, массивы **Visited** и **Way**, переменную **Length** рекомендуется описать в основной программе (т.е. функция **Path** в процессе работы будет взаимодействовать с этими глобальными объектами напрямую, а не через параметры).

Нерекурсивная ветвь: обращение к функции при **t1=t2** (значение функции вычислено и равно `true`). Рекурсивная ветвь: перебор в цикле `while` или `repeat` всех пар **t1** и **t** (где **t**=`1..n`). Если какая-то пара городов соединена прямой дорогой (т.е. **Roads**`[t1,t]=true`) и при этом в городе **t** еще не побывали в процессе поиска (т.е. `Visited[t]=false`), то выполняется переход на следующий уровень рекурсии путем обращения `Path(t,t2)` (т.е. рассматривается аналогичная более простая задача). Перед выполнением этого обращения важно скорректировать текущие данные: 1) `Visited[t]:=true` {чтобы не попасть повторно в город **t** в процессе дальнейшего поиска}; 2) `Length:=Length+1` {появился новый город в формируемом маршруте}; 3) `Way[Length]:= t` {заноси номер нового города в состав маршрута}. Если обращение `Path(t,t2)` увенчалось успехом, то завершаем работу функции `Path(t1,t2)` с положительным ответом (в массиве `Way` при этом зафиксирован найденный путь длины `Length`), иначе - переходим к следующему шагу цикла для очередного **t** (предварительно исключив только что проверенный город **t** из формируемого маршрута выполнением оператора `Length := Length -1`). Если ни для одной рассмотренной пары **t1** и **t** (где `t=1..n`) путь не найден, то завершаем работу функции **Path(t1,t2)** с ответом **false**. **см. далее**

Задача 2 **12.33 (из задачника)**

Читайте внимательно условие и смотрите ***рисунок 6*** (в задачнике):

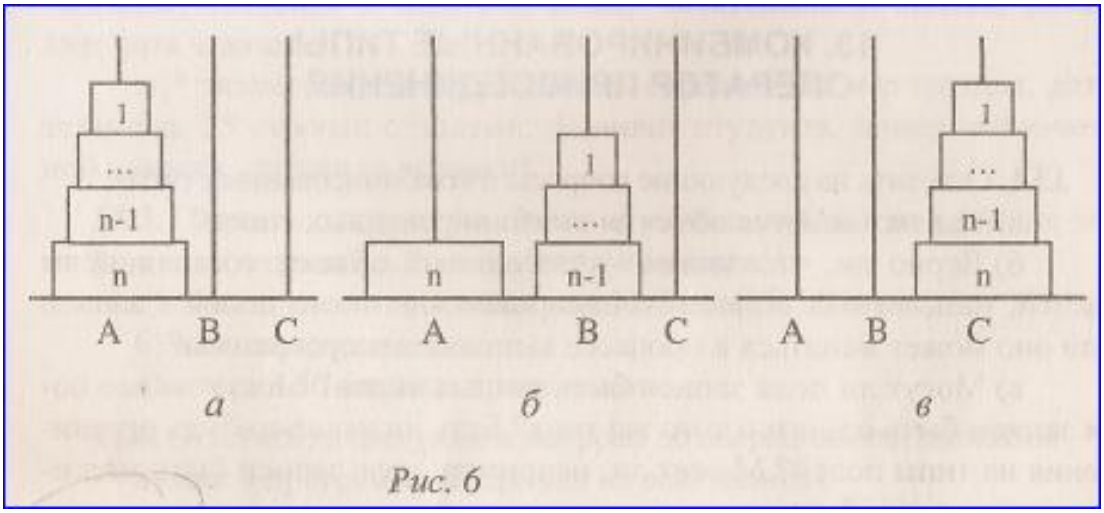


Рис. 6

Требовние: реализовать рекурсивную процедуру **`Move(n:integer;A,B,C:char)`**, где **`A`** – исходный стержень, **`B`** – промежуточный (вспомогательный) стержень, **`C`** – целевой (конечный) стержень.

Идея работы процедуры.
ЭТАП_1: пусть мы сумели переложить (**`n-1`**) колец с **`A`** на **`B`**, используя **`C`** как промежуточный (это возможно с учетом подсказки к задаче).
ЭТАП_2: после этого перекладываем самое большое кольцо с **`A`** на **`C`**.
ЭТАП_3: остается переложить (**`n-1`**) колец с **`B`** на **`C`**, используя **`A`** как промежуточный стержень.

Внимание: на **1** и **3 ЭТАПАХ** (им соответствуют рекурсивные вызовы) по-разному задаются фактические параметры к процедуре **Move** (т.к. меняется назначение (смысл) стержней). Печать сообщения о переносе дисков производится на **ЭТАПЕ_2**.

Задача 3 **12.38 (из задачника)**

Программа должна быть оттранслирована для значения **`n=4`**.

Для представления `n` заданных чисел используется массив `A`.

Для генерации нужных перестановок из основной программы вызывается процедура `generate(n)`. Процедура `generate(k)` генерирует все перестановки элементов `A[1], A[2], …, A[k]` (первый раз вызывается из программы с параметром `n`).

Идея: оставим `k`-ый элемент `A[k]` на своём месте и сгенерируем все перестановки элементов `A[1], …, A[k-1]` (вызвав для этого рекурсивно процедуру `generate(k-1)`). Далее повторяем процесс, поменяв `A[k]` местами с `A[i]` (последовательно для всех значений `i=k-1,…,1`) и сгенерировав соответствующие этому перестановки с помощью `generate(k-1)`. Цепочка рекурсивных вызовов завершается, когда число элементов, которые должны быть переставлены, станет равно единице (это значит, что полностью сгенерирована очередная перестановка и пора распечатать текущий вид массива `A`).

см. далее

Блок-6 Машины (дополнительные) по теме «Рекурсия, часть 3»

Задача 1 12.32 (10 очков)

Требование: решение дать обязательно с использованием *косвенной рекурсии*; до начала решения полезно разобрать задачу **12.31** и присланный пример на “опережающее описание”.

Предусмотреть взаимно-рекурсивные булевские функции **`text`** и **`elem`** (они могут быть с параметрами, а могут быть и нет – дело хозяйское – как сумеете), разрешается при необходимости вводить глобальные переменные (которые функции будут менять в процессе работы). Других функций не вводить, не усложнять решение, только эти две функции! Действуйте четко в соответствии с определениями понятий *текст* и *элемент*.

Задача 2 12.36 (5 очков) – решать для **n=5** (подсказок к этой задаче не будет)

Задача 3 Быстрая сортировка (5 очков) – решать для **n=15** и целочисленных элементов

Выбирается некоторый элемент (например, средний) и все элементы последовательности переставляются так, чтобы выбранный элемент оказался *на своем окончательном месте*, т.е. чтобы *слева* от него были только меньшие или равные ему элементы, а *справа* - только большие или равные. Затем этот же метод **рекурсивно** применяется к левой и правой частям последовательности, на которые ее разделил выбранный элемент. (*Замечание*: если в части оказалось два-три элемента, то упорядочивать ее следует более простым способом.)

Требуемая перестановка элементов выполняется так. Выбранный элемент копируется в некоторую переменную q. Последовательность просматривается *слева направо*, пока не встретится элемент, больший или равный q, а затем просматривается *справа налево* до элемента, меньшего или равного q. Оба этих элемента меняются местами, после чего просмотры с обоих концов последовательности продолжаются со следующих элементов, и т.д. В итоге выбранный элемент окажется в той позиции, где просмотры сошлись, это и есть его окончательное место.

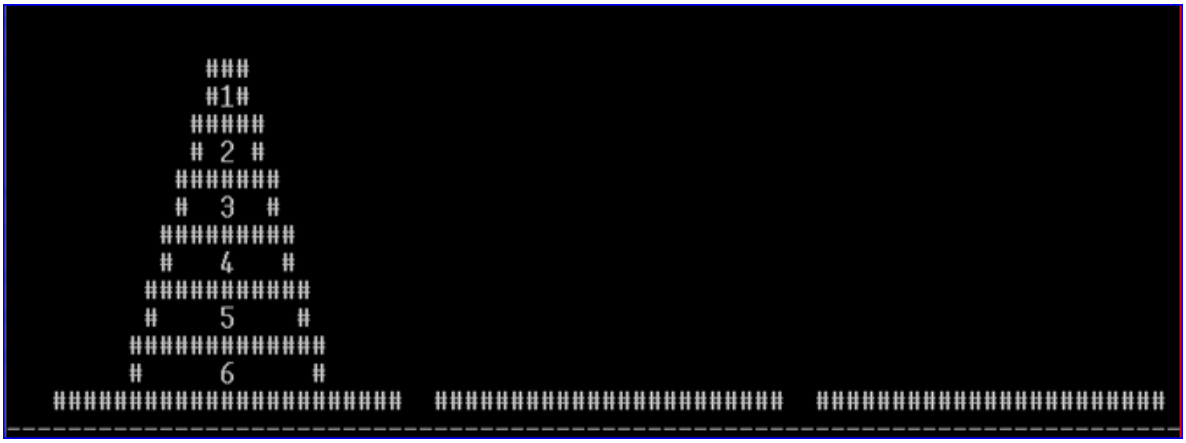
Задача 4 Визуализация Ханойских башен (максимум 30 очков)

Промоделировать на экране (визуализировать) работу “Ханойских башен” (т.е. алгоритм для задачи **12.33**).

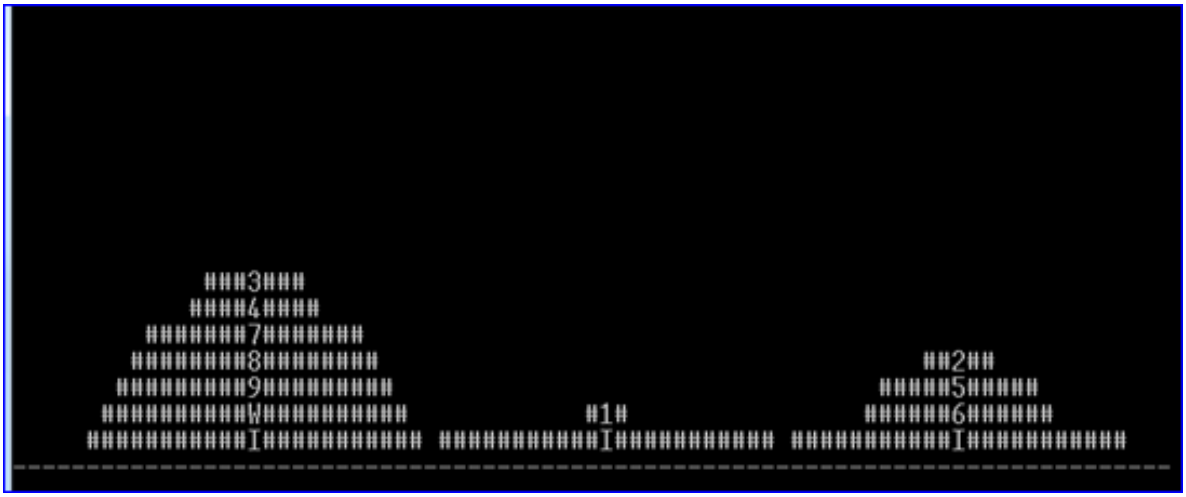
Следует установить ограничения на количество возможных дисков (с учетом текстового экрана из 80 столбцов и 25 строк). В рамках этих ограничений (зависит от конкретной реализации) запросить у пользователя число дисков, над которыми должен выполняться алгоритм. В решении разрешается задействовать необходимые возможности модуля CRT (установка цветного текстового режима, позиционирование курсора, установка цвета символа и цвета фона, а также прочие изобразительные возможности, при желании). Графический режим в решении не использовать! Необходимую дополнительную литературу можно посмотреть по ссылке http://arch32.cs.msu.su/semestr1/

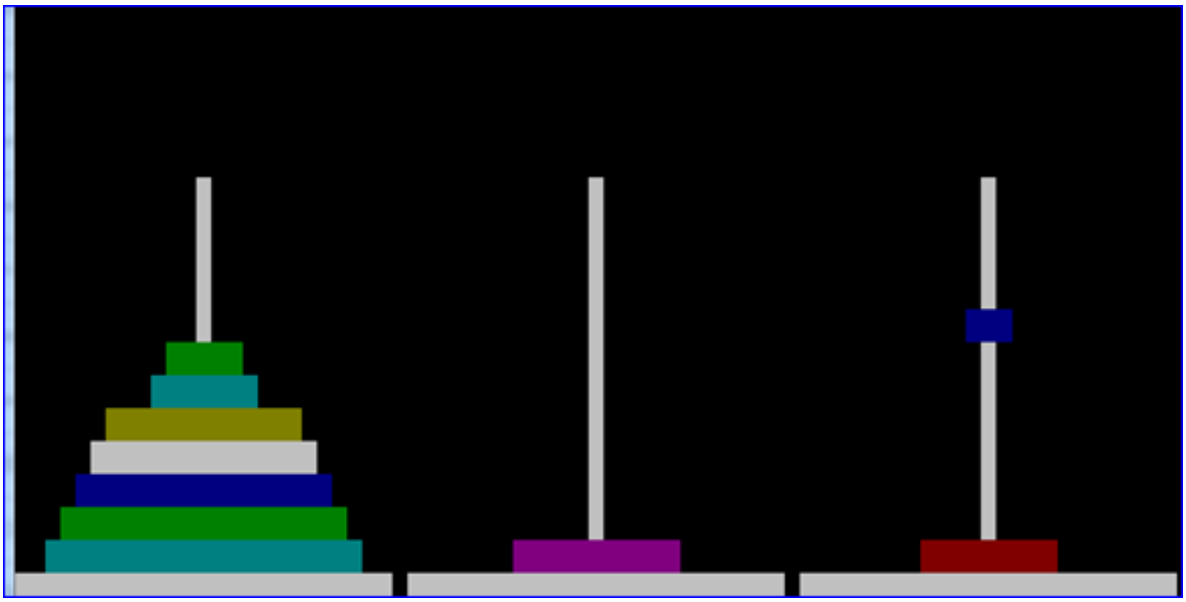
Желающие выполнить задачу могут попросить меня выслать примеры работы программы (в виде exe-файлов, запускаемых из-под windows).

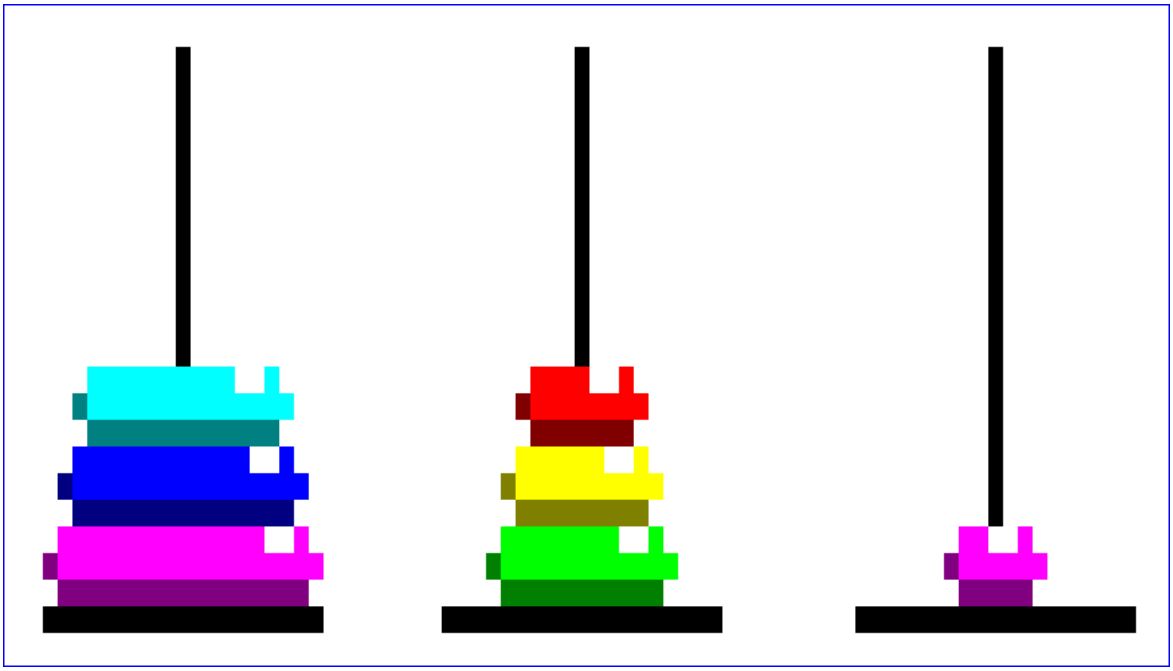
Примеры того, как может выглядеть экран в процессе работы программы (**см. далее**).


1
2
3
4
5
6


8
9
10
3
4
7
1
5
6


3
4
7
8
9
W
I
1
I
2
5
6
I





Текущая картинка на экране, далее, при нажатии на клавишу – перемещение верхнего диска на нужный стержень и т.п. (все перемещения отображаются в динамике на экране).

Базовый вариант: перемещение очередного диска соответствует новому статичному кадру (с новым положением диска), без анимации (т.е. последовательная смена картинок в соответствии с новым местоположением дисков) - 15 очков

+ анимация (демонстрация движения диска с одного колышка на другой) +5 очков

+ креатив (улучшенный дизайн, с активным использованием цветовых возможностей) +5 или +10 очков (по моему усмотрению)

Задача 5 12.35 (изменённая) Печать всех путей (5 очков)

Решить задачу 15.35 в следующей постановке: если из заданного начального (`first`) города можно добраться в заданный конечный (`last`) город, то распечатать на экран **все возможные пути** (каждый путь печатать с новой строки).

Программа должна быть оттранслирована для значения **`n=10`**.

Подсказка. Выполнить проверку всех возможных направлений поиска (полный перебор). Печать очередного найденного пути должна быть осуществлена в момент, когда дойдём до конечного пункта (т.е. попадем в ветку для **`t1=t2`**). Кроме того, необходимо устранить все «следы», оставленные при прохождении по найденному пути (подумайте, где это могло произойти). Рекомендуется также оформить **`path(…)`** в виде процедуры, а не функции.