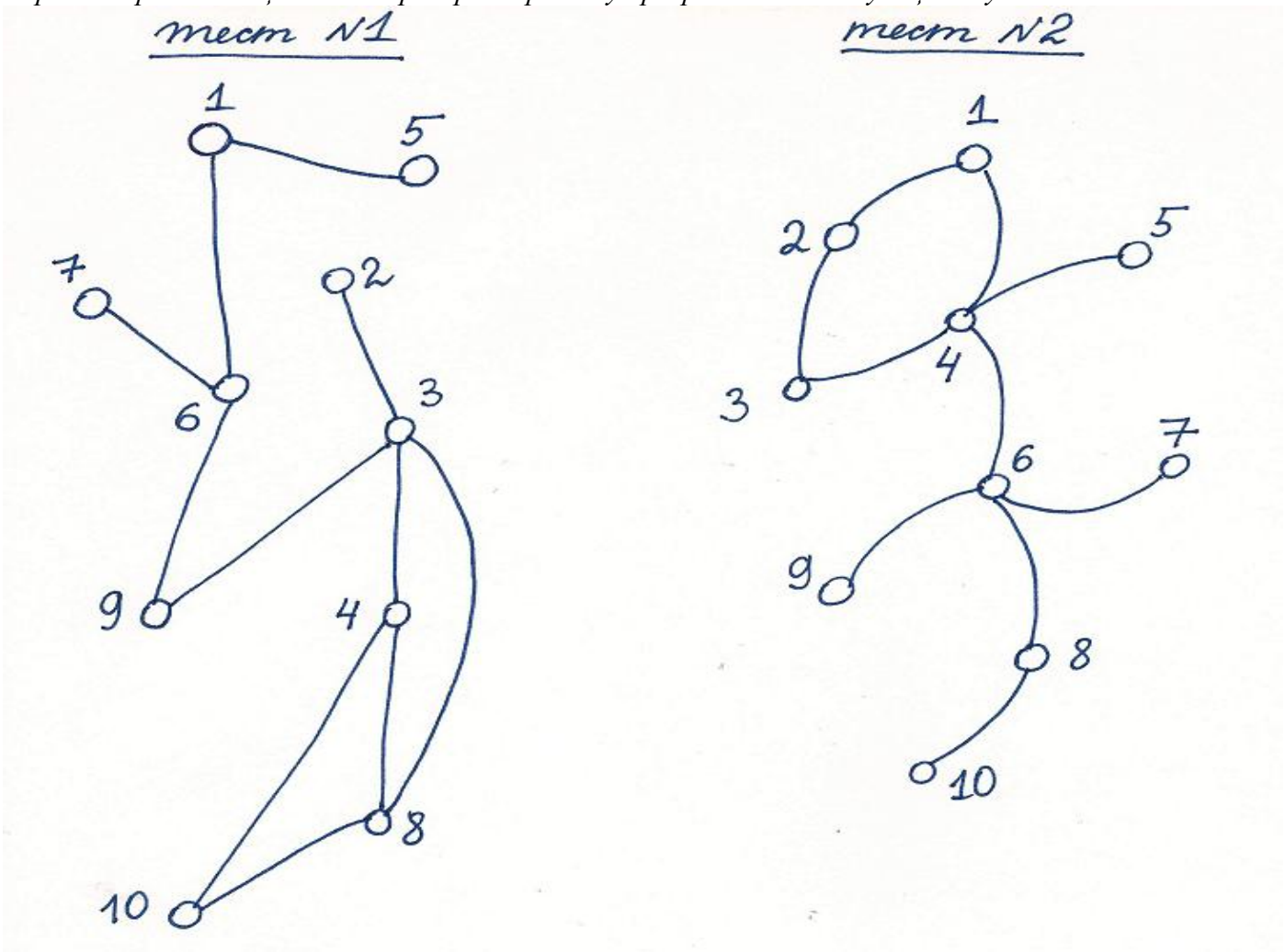
Блок-6 Машины (обязательные) по теме «Рекурсия, часть 3»

(всего 2 задачи, срок их сдачи до 3 декабря включительно):

Задача 1 **12.35 (из задачника)** (решать без **goto** !) – по аналогии с разобранным на семинаре решением задачи **12.34**. В основной программе в первую очередь необходимо сформировать булевскую матрицу смежности: сначала инициализировать все её элементы значением `false`, а затем, в цикле запрашивать пары пунктов, соединённых дорогой, и корректировать соответствующие элементы матрицы. Далее программа запрашивает у пользователя начальный и конечный пункты, после чего выдает ответ о наличии пути. Не забудьте сформировать и распечатать найденный путь. Программа должна быть оттранслирована для значения **n=10**.

Перед отправкой тщательно проверить работу программы на следующих двух тестах:



Подробности решения (для нуждающихся): сформировать так называемую матрицу "смежности": `var P: array[1..n,1..n] of boolean;` {где n – число городов}
Матрица должна отражать информацию о наличии прямых дорог из города **i** в город **j**; если город **i** соединен дорогой с городом **j**, то `P[i,j]= P[j,i]= true` (т.к. движение двухстороннее); считать также, что все `P[i,i]=false` (в основной программе изначально заполнить все элементы этой матрицы значениями `false`, а далее, по мере ввода данных пользователем заполнять нужные элементы значениями `true`). Итак, матрица симметрична, значения на главной диагонали = `false`.
Сформировать также массив `Visited`, отражающий, в каких городах уже побывали в процессе поиска пути: `Visited: array[1..n] of Boolean;` {до начала поиска заполнить все его элементы значениями `false`, кроме одного: `Visited[first]:=true`, так как сейчас уже находимся в городе с номером `first`}

Завести массив `Way: array [1..n] of 1..n` {фиксирует найденный путь; выполнить до начала поиска `Way[1]:=first`, так как начальный пункт маршрута – это город с номером `first`}.
Завести также глобальную целочисленную переменную `Length` для вычисления длины искомого пути (т.е. общего количества пройденных городов, включая начальный и конечный пункты; до начала поиска `Length:=1`, так как в одном городе находимся в начальный момент; очевидно, что длина искомого пути не будет превышать общего количества городов).
Описать рекурсивную **логическую функцию**: `Path(i,j)`, которая проверяет (на основе анализа элементов матрицы `P`), есть ли путь из города **`i`** в город **`j`** . Нерекурсивная ветвь: обращение к ф-ции при **`i=j`** (значение функции вычислено и равно `true`). Рекурсивная ветвь: перебор в цикле `while` или `repeat` всех пар **`i`** и **`k`** (где **`k`**`=1..n`). Если какая-то пара городов соединена прямой дорогой (т.е. `P[i,k]=true`) и при этом в городе **`k`** еще не побывали в процессе поиска, то выполняется переход на следующий уровень рекурсии путем обращения `Path(k,j)` (важно только перед выполнением этого обращения скорректировать текущие данные: `Visit[k]:=true` {чтобы повторно не попасть в город **`k`** при дальнейшем поиске}; `Length:= Length+1` {появился новый город в маршруте}; `Way[Length]:=k` {заносим номер нового города в состав маршрута}). Если обращение `Path(k,j)` увенчалось успехом, то завершаем работу ф-ции `Path(i,j)` с положительным ответом (в массиве `Way` – найденный путь), иначе - переходим к следующему шагу цикла для очередного **`k`** (предварительно исключив только что проверенный пункт **`k`** из формируемого маршрута: `Length := Length -1`). Если ни для одной пары **`i`** и **`k`** путь не найден, то дальнейший поиск не имеет смысла (завершаем работу ф-ции `Path(i,j)` с отрицательным ответом).

Задача 2 **12.38 (из задачника)** - 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 равным единице (это значит, что полностью сгенерирована очередная перестановка и пора распечатать текущий вид массива `A`).

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*.

См. следующую страницу

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**

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.)

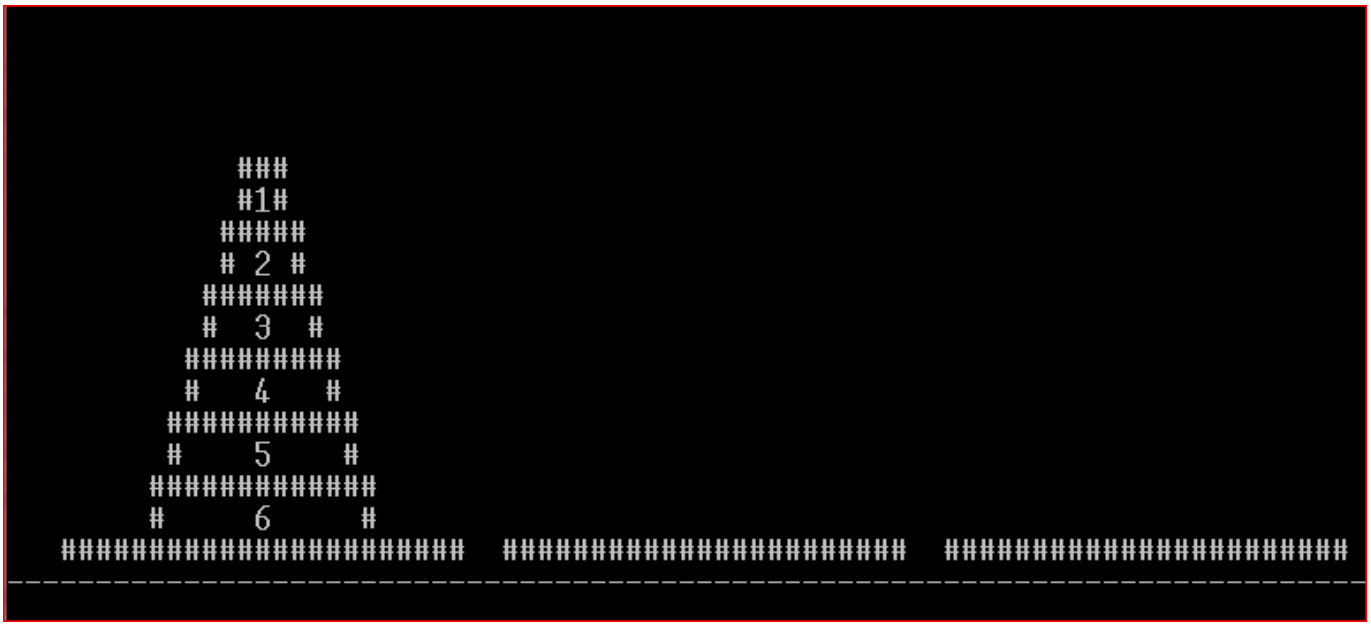
Требуемая перестановка элементов выполняется так. Выбранный элемент копируется в некоторую переменную *q*. Последовательность просматривается *слева направо*, пока не встретится элемент, больший или равный *q*, а затем просматривается *справа налево* до элемента, меньшего или равного *q*. Оба этих элемента 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 (10 очков)

Промоделировать на экране (визуализировать) работу Ханойских башен (алгоритм для задачи 12.33).

Можно предполагать, что число дисков задается как константа, либо вводится с клавиатуры (при этом следует установить ограничения на число дисков с учетом текстового экрана из 80 столбцов и 25 строк). В решении разрешается задействовать необходимые возможности модуля CRT (установка цветного текстового режима, позиционирование курсора, установка цвета символа и цвета фона и прочие возможности - при желании).

Желающие выполнить задачу могут попросить меня выслать пример работы программы (в виде exe-файла, запускаемого из-под windows). Не самый красочный пример, но зато идея проясняется. Более красивые программы я, к сожалению, не сохранила.



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