Лабораторная работа № 3***Генерация перестановок***

Дано конечное множество A. Требуется сгенерировать все возможные перестановки его элементов в лексикографическом порядке (*по материалам главы 1, п. 1.3.6, и главы 2, п. 2.2.1*). Требования к заданию множества – в нем не должно быть повторяющихся элементов, кроме того, удобнее использовать или только буквы, или только цифры.

Программа должна сначала упорядочить все элементы заданного множества по возрастанию (это первый – минимальный – набор), затем – посредством МИНИМАЛЬНО ВОЗМОЖНЫХ ПЕРЕСТАНОВОК! – сгенерировать последовательно возрастающие (лексикографически) наборы, вплоть до последнего, в котором все элементы упорядочены по убыванию.

Следует оценивать количество возможных перестановок и в случае, если они не поместятся на экран, выполнять их вывод в файл с выдачей на экран соответствующей информации для пользователя и выполнять поэкранный вывод с ожиданием нажатия клавиши.

Дополнительно: Предоставить пользователю возможность выбора другого варианта работы программы, в котором за исходную точку упорядочивания наборов выбирается не минимальный набор, а набор в таком порядке, как он задан пользователем.

Возможный алгоритм решения (Пример: множество А={1, 2, 3, 4, 5, 6}, |A| = n):

Предположим, что уже построено m наборов. Тогда для получения m+1-го набора:

1. Выполняется проверка последнего (m-го) набора на наличие в его конце некоторого количества символов, упорядоченных по убыванию – пусть это символы ak+1…an.
 3 5 2 **6 4 1**– k=3, символы с 4-го по 6-й упорядочены по убыванию.
2. Если такое k найдено, то поменять местами k-й элемент и наименьший элемент из ak+1…an, больший этого ak.
В нашем примере это 2 и 4:  3 5 4 6 2 1 (это промежуточный набор).
3. После шага 2 упорядочить элементы с k+1-го до последнего по возрастанию. Получен очередной набор  выдать его на печать.
 3 5 4 1 2 6.
4. Если на шаге 1 ответ отрицательный, то поменять местами 2 последних элемента и выдать на печать полученный набор. В частности, после шага 3 это неизбежное действие, т.к. все последние элементы были размещены по возрастанию  целесообразно после выполнения ш.3 задавать признак его выполнения, который будет анализироваться (и сбрасываться) на шаге 1. После шага 3 было  3 5 4 1 **2 6** выдать  3 5 4 1 6 2 .
Если был набор  3 5 2 6 **1 4** выдать  3 5 2 6 4 1 .
5. Если полученный набор не последний (упорядоченный по убыванию), то возврат на шаг 1. В противном случае конец работы.

Лабораторная работа № 4***Генерация подмножеств***

Задано целое положительное число n, которое представляет собой мощность некоторого множества. Требуется с минимальными трудозатратами генерировать все подмножества этого множества, для чего каждое последующее подмножество должно получаться из предыдущего путем добавления или удаления только одного элемента. Множество и все его подмножества представляются битовой шкалой. Для генерации использовать алгоритм построения бинарного кода Грея.

В качестве результата выводить построчно каждое из подмножеств (в виде битовой шкалы), сопровождая их порядковыми номерами. В случае большого количества результирующих строк (превышающего размер экрана) выполнять поэкранную выдачу, а также осуществлять их вывод в файл с выдачей на экран сообщения для пользователя – имя файла, его местонахождение.

**Алгоритм построения бинарного кода Грея**

Вход: n  0 – мощность множества.

Выход: последовательность кодов подмножеств B (битовая шкала).

1. Инициализация массива В и его выдача на печать.

2. В цикле по i (от 1 до 2n–1):

а) Определение элемента для добавления или удаления: p:=Q(i);

б) Добавление или удаление элемента B[p]:=1–B[p];

в) Вывод очередного подмножества – массива B.

Функция Q(i) определяется как число, на единицу превышающее количество “2” в разложении числа i на множители. Очевидно, что для нечетных i значение этой функции равно 1, т.е. для нечетного i значение будет менять крайний правый бит шкалы (нумерация справа налево от 1), а для i, равных степени 2, будет “включаться” бит, соответствующий этой степени 2 (например, для 4 – 3-й бит, для 8 – 4-й бит, …).

Пример: Выполнение алгоритма для n=3. Дополнительно: множество {a,b,c}.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | p |   | B |   |   | Дополнительно | множества |
|   |   | 0 | 0 | 0 |   |  |
| 1 | 1 | 0 | 0 | 1 |   | {с} |
| 2 | 2 | 0 | 1 | 1 |   | {b,c} |
| 3 | 1 | 0 | 1 | 0 |   | {b} |
| 4 | 3 | 1 | 1 | 0 |   | {a,b} |
| 5 | 1 | 1 | 1 | 1 |   | {a,b,c} |
| 6 | 2 | 1 | 0 | 1 |   | {a,c} |
| 7 | 1 | 1 | 0 | 0 |   | {a} |

Лабораторная работа № 5***Поиск компонент связности графа***

Граф задан его матрицей смежности. Требуется определить количество компонент связности этого графа (*по материалам главы 3, п. 3.2.3 и 3.4*). При этом должны быть конкретно перечислены вершины, входящие в каждую компоненту связности.

Выбор алгоритма поиска компонент связности – произвольный. Например, приветствуется использование одного из видов обхода (поиск в глубину или поиск в ширину *по материалам п. 3.4.3*).

Пользователю должна быть предоставлена возможность редактировать исходную матрицу, т.е. изменять исходный граф без выхода из программы. Предусмотреть также возможность изменения количества вершин.

При выполнении работы разрешается (даже рекомендуется!) использовать матрицу бинарных отношений из лабораторной работы №2.

Вход программы: число вершин графа и матрица смежности.

Выход: разбиение множества вершин на подмножества, соответствующие компонентам связности.