FIT2004 S1/2017: Assignment 1 (6 Marks)

THIS PRAC IS ASSESSED. Interview during Week 4 Lab.

Submission deadline: Midnight 11:59pm Sun 19 March 2017

CLASS: This programming exercise must be prepared and finalised in advance, and should be submitted via the unit's Moodle page by 11:59pm Sunday 19 March 2017 (end-of-week-3). Implement this exercise strictly using **Python programming language**. During your **assigned** Week 4 lab, your demonstrator will interview you based on your **submitted** version by downloading it from Moodle. This practical work is marked on the performance of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. You must write **all the code yourself**, and should not use any external library routines (or code), except those that are considered standard. The usual input/output and other unavoidable routines are exempted (i.e., can be used).
DEMONSTRATORS are not obliged to mark programs that do not run or that crash. Time allowing, they will try to help you track down errors, but they are NOT required to mark programs in such a state, particularly those that do not work.
OBJECTIVE: This practical exercise is to help you develop insights into new ways of generating permutations of elements, your understanding of iteration and recursion, taking an algorithm and writing a program from it, and analysing and estimating the time-complexity of algorithms.

Background on Permutations

The number of permutations of N distinct elements is N *factorial*, denoted as $N!$. The factorial of N is the product of all positive integers less than or equal to N: $N! = 1 \times 2 \times 3 \times \cdots \times N$.

Denote the set of all permutations of N (distinct) elements in a set by S_N. For example, the permutation set S_4 corresponding to a set made up of English letters {a,b,c,d} has $4! = 24$ permutations in it, and these are:

```
 0 abcd      6 bacd     12 cabd     18 dabc
 1 abdc      7 badc     13 cadb     19 dacb
 2 acbd      8 bcad     14 cbad     20 dbac
 3 acdb      9 bcda     15 cbda     21 dbca
 4 adbc     10 bdac     16 cdab     22 dcab
 5 adcb     11 bdca     17 cdba     23 dcba
```

Observe that these permutations are printed in the lexicographic order of the letters. Notice also that a decimal number/index is listed in order associated with each permutation. That is, the permutation number 0 represents a fully-ordered permutation, abcd, in the above example.

For any N, there are several ways to print all the permutations in S_N in the lexicographic order. One way relies on converting the permutation number from a decimal system (or base-10

system) into a number in a **new number system**, which we will call here the $\mathcal{F}$-number system. Numbers in the $\mathcal{F}$-number system are denoted here as base-! numbers. In any given permutation set S_N, each permutation number/index $\mathbf{D}$ in base-10 – $(\mathbf{D})_{10}$, in short – where $0 \leq \mathbf{D} < N!$, can be represented **uniquely** as:

$$(\mathbf{D})_{10} = \mathbf{C_{N-1}} \times (N-1)! + \mathbf{C_{N-2}} \times (N-2)! + \cdots + \mathbf{C_1} \times 1! + \mathbf{C_0} \times 0! \tag{1}$$

where $\mathbf{C_{N-1}}, \mathbf{C_{N-2}}, \cdots \mathbf{C_0}$ form the digits in the base-! system.

If the property in Equation 1 holds, then the base-! number $(\mathbf{C_{N-1}C_{N-2}} \cdots \mathbf{C_0})_!$ in $\mathcal{F}$-number system is equivalent to $(\mathbf{D})_{10}$ in the base-10 number system. As a concrete example, below is a table that shows all the base-10 permutation numbers and their corresponding base-! numbers for permutations in S_4 over the letters $\{a, b, c, d\}$.

BASE-10	BASE-!	PERMUTATION	BASE-10	BASE-!	PERMUTATION
(0)_10	(0000)_!	abcd	(12)_10 =	(2000)_!	cabd
(1)_10	(0010)_!	abdc	(13)_10 =	(2010)_!	cadb
(2)_10	(0100)_!	acbd	(14)_10 =	(2100)_!	cbad
(3)_10	(0110)_!	acdb	(15)_10 =	(2110)_!	cbda
(4)_10	(0200)_!	adbc	(16)_10 =	(2200)_!	cdab
(5)_10	(0210)_!	adcb	(17)_10 =	(2210)_!	cdba
(6)_10	(1000)_!	bacd	(18)_10 =	(3000)_!	dabc
(7)_10	(1010)_!	badc	(19)_10 =	(3010)_!	dacb
(8)_10	(1100)_!	bcad	(20)_10 =	(3100)_!	dbac
(9)_10	(1110)_!	bcda	(21)_10 =	(3110)_!	dbca
(10)_10	(1200)_!	bdac	(22)_10 =	(3200)_!	dcab
(11)_10	(1210)_!	bdca	(23)_10 =	(3210)_!	dcba

For instance, we can work out one of the base-10 to base-! conversion (shown above) as:

$$(\mathbf{15})_{10} = \underbrace{\mathbf{2}}_{\mathbf{C_3}} \times 3! + \underbrace{\mathbf{1}}_{\mathbf{C_2}} \times 2! + \underbrace{\mathbf{1}}_{\mathbf{C_1}} \times 1! + \underbrace{\mathbf{0}}_{\mathbf{C_0}} \times 0! = (\mathbf{2110})_!$$

Further, the digits $\mathbf{C_{N-1}}, \mathbf{C_{N-2}}, \cdots \mathbf{C_0}$ have a very special meaning. Notice, from the example above, there are as many base-! digits as there are letters in the permutation string. Each digit in this system gives the number of *inversions* of each letter in the permutation compared to the fully-ordered permutation. What does this mean? Let us see using the same example as above:

```
    (15)_10 = (2110)_!      c b d a
For a better (visual) understanding, the base-! digits are arranged right
under this permutation:
                            c b d a
                            2 1 1 0
Then, these base-! digits have the following meaning:
For the 1st letter 'c', there are 2 letters to the right of it (i.e., 'b' and 'a')
                        that are lexicographically LESS than 'c'.

For the 2nd letter 'b', there is 1 letter to the right of it ('a') that
                        is lexicographically LESS than 'b'.
```

```
For the 3rd letter ‘d’, there is  1 letter  to the right of it (‘a’) that
                          is lexicographically LESS than ‘d’.

For the 4th letter ‘a’, there are 0 letters to the right of it that
                          are lexicographically LESS than ‘a’.
```

In fact, the sum of these base-! digits gives the total number of **transpositions** (or swaps between adjacent letters) needed to convert the permutation into a fully sorted order. In the example we have been considering so far, the sum of digits in (2110)_! $= 2 + 1 + 1 + 0 = 4$, implying that this permutation needs 4 transpositions to convert from cbda to abcd. Let us see how! Working from **right to left** and applying the number of successive swaps as indicated by the base-! digit under each symbol:

```
 c b d a
 2 1 1 0

 SWAPS:          #1       #2       #3       #4
            cbda --> cbad --> cabd --> acbd --> abcd
```

It is also straightforward to construct a permutation string from just the base-! digits (and the total ordering of the N letters in the permutation). Let's follow this with the same example:

```
N = 4
Below is the total ordering of the letter in S_4
Index:               0 1 2 3
Total ordering:      a b c d

Our goal now is to generate the permutation given the following
base_! digits :      2 1 1 0

First digit is 2; find the symbol at index=2 in the total ordering set
and place it as the first symbol of the permutation:
                     c
Remove c from the total ordering:
Index:               0 1 2
Total ordering:      a b d

Next digit is 1; find the symbol at index=1 in this new total ordering set
and place it as the second symbol of the permutation:
                     c b
Remove b from the total ordering:
Index                0 1
Total ordering:      a d

Next digit is 1 again; find and place the symbol at index=1
as the third symbol of the permutation:
                     c b d
Remove d from the total ordering:
Index                0
Total ordering:      a
```

```
Next digit is 0; find and place the symbol at index = 0
as the fourth symbol of the permutation:
                    c b d a
```

Computing matrix determinant using permutations

In linear algebra, the determinant of a square matrix is a value that is commonly computed for various purposes. Coincidentally, since this prac deals with permutations and inversions, one of the ways of computing the determinant of an $N \times N$ real square matrix is using the information of all permutations of the set $\{1, 2, \ldots, N\}$. Consider the matrix for the form:

$$A = \begin{pmatrix} A(1,1) & A(1,2) & \cdots & A(1,N) \\ A(2,1) & A(2,2) & \cdots & A(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ A(N,1) & A(N,2) & \cdots & A(N,N) \end{pmatrix}$$

Then its determinant, denoted by $\det(A)$, is given by the formula:

$$\det(A) = \underbrace{\sum_{\forall perm \in S_N}}_{\substack{\text{Sum over each} \\ perm \text{ in } S_N}} \underbrace{\text{sgn}(perm)}_{\substack{\text{Sign of } perm \\ \text{(see below)}}} \underbrace{\prod_{i=1}^{N}}_{\substack{\text{Product over} \\ N \text{ terms from} \\ \text{the matrix}}} A(perm(i), i)$$

In the above determinant formula:

- S_N contains all permutations of the set $\{1, 2, \ldots, N\}$.
- $perm$ refers to any permutation in S_N. $perm(i)$ gives the ith symbol in that permutation.
- $\text{sgn}(perm)$ denotes the sign function which takes $perm$ as an argument, and returns:
  - $\star$ $+1$, when the **total number** of *inversions** of $perm$ is *even*.
  - $\star$ -1, otherwise.

For example, for a 3×3 matrix, it is easy to see:

$$\begin{aligned} \det(A) = &+ A(1,1)A(2,2)A(3,3) - A(1,1)A(3,2)A(2,3) \\ &- A(2,1)A(1,2)A(3,3) + A(2,1)A(3,2)A(1,3) \\ &+ A(3,1)A(1,2)A(2,3) - A(3,1)A(2,2)A(1,3). \end{aligned}$$

where the 6 permutations of S_3 appear within the 6 terms on the right hand side, shown in red, and the evaluated sign of each term (based on the parity of the total number of inversions) via sgn(perm) is shown in blue.

Based on the above background, address the following questions:

*We saw that this is equivalent the sum of `base-!` digits

Assignment Questions

1. Write a Python program called `permute` which accepts an integer N as its argument. For the given N, your program should iterate over $N!$ numbers, and at each step:

   - Convert each (permutation) number from the `base-10` (decimal) representation to its corresponding `base-!` representation.
   - Convert the `base-!` number into a permutation string, assuming we are dealing with the first N letters of the English alphabet. (Assume that $N \leq 10$ for this exercise.)
   - Compute the sum of these `base-!` digits for the given permutation.

   Having written this program:

   (a) Print, to an output file, all permutations in the lexicographic order containing the following information:
   - Column 1: The `base-10` index/number of the permutation
   - Column 2: The `base-!` digits of that permutation
   - Column 3: The sum of the `base-!` digits
   - Column 4: The corresponding permutation string using the first N letters from the English alphabet as the total ordering.

   (**See sample output given on the last page for formatting your output.**)

   (b) Print a frequency table (**refer output format on the last page**) showing the number of permutations (or frequency) as a function of sum of the `base-!` digits.

   (c) Print the *weighted average* of the sum of `base-!` digits over all permutations.

   (d) How is this *weighted average* growing asymptotically as a function of N? Justify your answer with a clear reasoning.

   (e) This growth informs the complexity of a certain sorting algorithm we discussed specifically in weeks 2 and 3. Which one and what case – Best, Average or Worst – does your answer to (d) illuminate?

   (f) For any arbitrary N, what range of values does the sum of the `base-!` digits take? Why?

2. Using the background information, write another script that takes any two permutations of a finite set of distinct letters and returns as an answer the **smallest number of (adjacent) transpositions** required to convert one permutation to another. (E.g.: for converting the permutation `cadb` to `bacd` or vice versa, the answer would be 4.)

3. Finally, using the background on computing matrix determinant, write a Python program to compute the determinant of a given square matrix. Your program should accept an $N \times N$ matrix as a comma separated file (see last page for input format), check if the matrix is square, and if yes: compute the matrix determinant using the permutation method, and output the determinant as the answer. (You can validate the output of your script using the resource at WolframAlpha: click here.) (If no: Return 'Error'.)

   (a) What is the time-complexity of determinant computation using the method prescribed here? (Justify your answer.)

   (b) Does this complexity grow slower/faster than $O(2^n)$. Why?

In your submission for this assignment, along with the Python scripts, include the output/answers to all the above questions, 1(a)-(f), 2, 3(a)-(b), collated into a single plain text file.

OUTPUT FORMAT FOR Q1:

```
INPUT TO THE SCRIPT: N = 4
TOTAL NUMBER OF PERMUTATIONS = 24
Base-10       Base-!     Sum   Permutation
( 0)_10     (0000)_!      0       abcd
( 1)_10     (0010)_!      1       abdc
( 2)_10     (0100)_!      1       acbd
( 3)_10     (0110)_!      2       acdb
( 4)_10     (0200)_!      2       adbc
( 5)_10     (0210)_!      3       adcb
( 6)_10     (1000)_!      1       bacd
( 7)_10     (1010)_!      2       badc
( 8)_10     (1100)_!      2       bcad
( 9)_10     (1110)_!      3       bcda
(10)_10     (1200)_!      3       bdac
(11)_10     (1210)_!      4       bdca
(12)_10     (2000)_!      2       cabd
(13)_10     (2010)_!      3       cadb
(14)_10     (2100)_!      3       cbad
(15)_10     (2110)_!      4       cbda
(16)_10     (2200)_!      4       cdab
(17)_10     (2210)_!      5       cdba
(18)_10     (3000)_!      3       dabc
(19)_10     (3010)_!      4       dacb
(20)_10     (3100)_!      4       dbac
(21)_10     (3110)_!      5       dbca
(22)_10     (3200)_!      5       dcab
(23)_10     (3210)_!      6       dcba

 FREQUENCY   TABLE
-------------------
   SUM      FREQ.
    0         1
    1         3
    2         5
    3         6
    4         5
    5         3
    6         1
 Weighted average of sum = 3
```

OUTPUT FORMAT FOR Q2:

```
Input Permutation 1 =   cadb
Input Permutation 2 =   bacd
Output (smallest number of inversion) = 4
```

INPUT FILE FORMAT FOR Q3:

```
1,2,1
1,1,0
0,1,1
```

OUTPUT FORMAT FOR Q3:

```
N = 3
Input Matrix:
  1 2 1
  1 1 0
  0 1 1
Determinant = 0
```

-=o0o=-

END

-=o0o=-