COSC 4P03 – Advanced Algorithms
Winter 2019
Assignment #2

Due Date: 11th March, noon **Late Date: 14th March, noon**
This assignment accounts for 10% of your final grade and is worth a total of 100 marks.

In this assignment you are to use backtracking to solve a problem related to non-linear error-correcting codes.
A non-linear binary code of length *n* and minimum distance *d* is a set of binary vectors, (called *codewords*) of length *n*, such that the distance between any pair of codewords is at least *d*. The (Hamming) distance between two vectors *v* and *w* is defined as the number of positions in which they differ. For example, the Hamming distance between *v*=111000 and *w*=101010 is 2, because the second and fifth symbols differ while all other symbols are the same.

The following is a non-linear code of length 6 and minimum distance 4:

```
000000
001111
110011
111100
```

The problem we wish to solve is to determine the maximum number of vectors in a non-linear binary code of minimum distance 4, for different lengths *n*. We will call this value A(*n*, 4).

Two binary non-linear codes are equivalent if one can be obtained from the other by any permutation of the coordinates and/or permutations of the values in one or more columns. For example, the following two codes are equivalent as one can be obtained from the other by exchanging the second and third columns and then exchanging 0s and 1s in the last column.

```
000000          000001
001111          010110
110011          101010
111100          111101
```

Question 1 (70 marks): Write a program which uses a backtracking algorithm to compute A(*n*, 4), for all values of *n* from 4 up to whatever value your program can complete in no more than 1 day of CPU time on a single processor. You should run this program once for each different value of *n*.

Input: *n* (integer)
Output: The value of A(*n*, 4), followed by a list of the vectors in *one* of the codes of maximal length.
Important note: Marks will be allocated for efficiency. See the recommendations below. **Ensure that you have adequate documentation to explain the decisions you have made to help efficiency.**

Question 2 (30 marks): Write a program to estimate the number of nodes at each level of the backtracking search tree for your solution to question 1 above. Use this program to estimate the number of nodes at each level of the tree when used to compute A(n, 4) for all values of n between 7 and 11, inclusive. Your program should randomly choose at least 5 *different* paths for each one of these values, and average out their results to produce your estimate.

Input: n (integer)
Output: A table indicating, for each level of the search, the estimated number of nodes at that level. Note: if the estimated number is 0, then it is not necessary to print further levels as these will also be 0. Save your output for each run in a separate text file, and submit all of these files.

Recommendations:
Given a large enough value of n, your program from part 1 will take an extremely long time to run. I strongly recommend you do all of the following:

1. Try small values of n before moving on to the larger ones. You might be very surprised to discover the huge difference in CPU time to go from order n to order n+1.
2. Carefully track the progress of your program so that you will know how much of the search remains, allowing you to "kill" it if necessary.
3. Think about what constitutes a "level" of the search.
4. Think carefully about which candidates you actually need to test at each level of the search.
5. Maintain an ordering to the candidate vectors (see the maximal clique problem for an example of this).
6. Think carefully about when and how to test for equivalence.
7. Think carefully about whether there are any assumptions you can make about the structure of the matrix (or list of words) you are generating.
8. Consider storing each vector as an integer (bit-vector), and perform bit manipulations, rather than storing a vector as an array. You could write several routines to make this easier:
   - Conversion of an array to a bit-vector
   - Conversion of a bit-vector to an array
   - Pre-compute a table that stores the weights (number of 1s) of each bit-vector of length n.
   - Use the above table to determine the distance between vectors. The distance between vectors x and y is the weight of (x+y), using bitwise modular addition.

   For example, we can store vectors of length 8 as integers with values from 0 to 255.
   Suppose x = 11110000 and y = 11001100.
   If stored as an integer, x = 240 and y = 204.
   The distance between x and y is weight(x + y) = weight(00111100) = 4.

Additional Notes:

1. Marks are allocated for producing the correct results while following the given instructions. Correct output with incorrect implementation and/or without appropriate documentation will result in a poor grade.
2. In order to simplify marking, try to adhere to the requested format as much as possible.

Submission Requirements:
All of the following must be placed in a sealed envelope in the 4P03 assignment box:

1. A cover sheet, available from `http://www.cosc.brocku.ca/forms/cover`, completely filled out. Your assignment will not be marked unless one is submitted with the assignment.
2. Commented and properly documented listings for all source code for your program.
3. Adequate documentation to explain (and show the validity of) the decisions you have made to improve efficiency.
4. Printouts of all output as specified above.
5. Any information required to run your programs.

You must also submit your assignment electronically so that it can be checked for plagiarism using `MOSS`. To do this, create a directory on Sandcastle containing all files for this assignment, and run the script `submit4p03` from this directory.