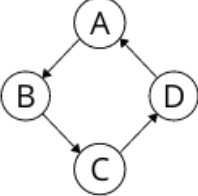
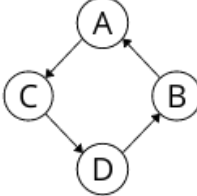
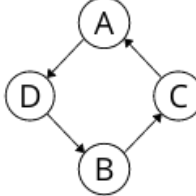
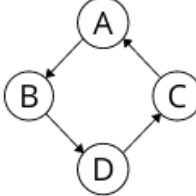
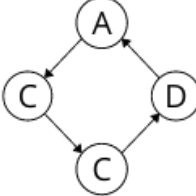
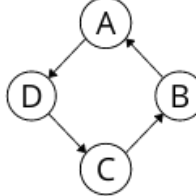


Circular Permutation

In this problem the algorithm and the structure I have used is the following:

I used a weighted graph with adjacency matrix and stored the string elements as the identifiers and all edges equal to 0. In each iteration, after getting the solution of permutation, I set an edge of elements to 1 according to the order of elements and I will have a new matrix with 0's and 1's. Then I check with the previous solutions if the matrix is already in the list of solutions, if not I insert the new matrix inside the list.

Here is the example with str = "ABCD", corresponding graph and the matrix of the solutions:

					
A B C D	A C D B	A D B C	A B D C	A C B D	A D C B
0 1 0 0	0 0 1 0	0 0 0 1	0 1 0 0	0 0 1 0	0 0 0 1
0 0 1 0	1 0 0 0	0 0 1 0	0 0 0 1	0 0 0 1	1 0 0 0
0 0 0 1	0 0 0 1	1 0 0 0	1 0 0 0	0 1 0 0	0 1 0 1
1 0 0 0	0 1 0 0	0 1 0 0	0 0 1 0	1 0 0 0	0 0 1 0

The matrix structure of all other options, such as BACD BADC BCAD BCDA BDAC BDCA CABD CADB CBAD CBDA CDAB CDBA DABC DACB DBAC DBCA DCAB DCBA, have the same structure as the ones above, so they will not be considered as the solution of the problem.

(set **DEBUG** to 1 to see all the permutation results with the concerning matrices).

I used the data structure **graph_t** as a wrapper of **array_t** and **fullSol_t** list to store all the valid solutions of the problem.