Programming

You are given 2 questions, and need to solve both of them.

Instructions to submit your solutions:

1. The files `QE_prob1.py` and `QE_prob2.py` contain your solution to problems 1 and 2, respectively.
2. Remove any debugging or logging code before you submit. It may disturb the automatic grading process, and as a result, you will likely get a lower score.
3. Compress the three files `QE_prob1.py` and `QE_prob2.py` to a single submission `file 20XX_XXXXX.zip` (`20XX_XXXXX` is your SNU student id, *e.g.*, 2022_12345.zip). The submission file should contain at most three files: `QE_prob1.py` and `QE_prob2.py`.
4. Send the submission file to `gsds_qe@aces.snu.ac.kr` from your SNU email account (if it is not an SNU email account, we will not accept your solution). The title of the submission email should be `[QE] 20XX-XXXXX` (*e.g.*, `[QE] 2022-12345`).
5. Make sure that the attached file is easily downloadable from the email message. We will not accept any submission that requires third-party tools or storages (*e.g.*, Google Drive).

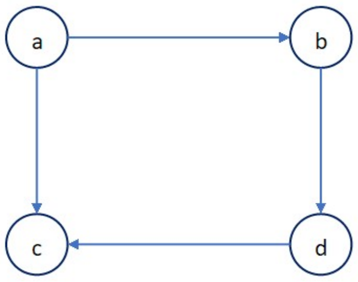
Note: You may use the Internet for API search, but communication with other people in any matter is strictly prohibited. Violation to this will be considered as academic misconduct.

1. [40 pts] A permutation, also called an "arrangement number" or "order," is a rearrangement of the elements of an ordered list `S` into a one-to-one correspondence with `S` itself. Treating a string as an ordered list, we may have a permutation of the string. For example, when the string is "abc", a permutation of the string is "bca". Thus, a string of length n has $n!$ permutations. In this problem, given a string `s`, you will implement a function `str_perm(s)` that returns a list of all permutations of `s`. However, the list contains **no identical permutations**. The list should contain the permutations in lexicographical order (*a.k.a.* alphabetical order). The characters used in a string are only lower-case alphabets.
For example,

- when `s` is "abc", `str_perm(s)` returns:
  `["abc", "acb", "bac", "bca", "cab", "cba"]`
- When s is "abb", `str_perm(s)` returns:
  `["abb", "bab", "bba"]`

The submission file `QE_prob1.py` should only contain the implementations of the function `str_perm(s)`. **You will likely get a lower score if there is any print or debugging code in your submission.**

2. [60 pts] In this problem, given **an acyclic directed graph** `G`, you will implement function `paths(G,s,t)` that returns the list of all paths between two vertices `s` and `t`. A path between two vertices is also a list of vertices. Paths in the list can be in any order, and each element in the path (element in the inner list) should be the `id` (string) of the vertex. For example, for the following graph `G`,



`paths(G,a,c)` returns the following list of paths:

```
[["a","c"],["a","b","d","c"]]
```

A node in a directed graph is defined as follows:

```python
# Node definition.
class GNode:
    def __init__(self, id):
        self.id = id        # id is a string
    def __str__(self):
        return self.id
```

You can freely add members in the `GNode` definition for your conveniences. A directed graph `G` is implemented as an adjacency list using a dictionary as follows:

```
>> a, b, c, d = GNode('a'), GNode('b'), GNode('c'), GNode('d')
>> G = dict()
>> G[a], G[b], G[c], G[d] = [b, c], [d], [], [c]
>> paths(G,a,c)
[['a', 'c'], ['a', 'b', 'd', 'c']]
```

The submission file `QE_prob2.py` should contain only the definition of the `GNode` and implementations of the two functions `paths(G,s,t)`. **You will likely get a lower score if there is any print or debugging code in your submission.**

Hint: Take a path from `s` and start walking on it and check if it leads to `t` then include the path in the path list and backtrack to take another path.