Individual report for the COMP1730/6730 project assignment S2 2022

Question 1: Write your name and ANU ID

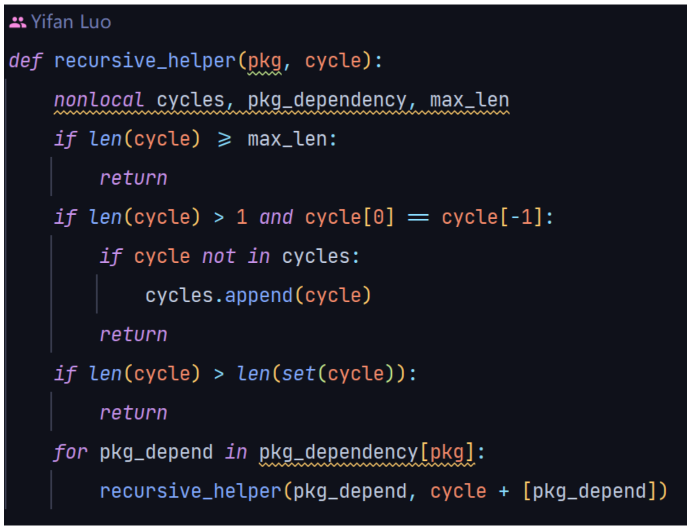
Name: Yifan Luo
ANU ID: u7351505

Question 2: If you are part of a group, write the ANU IDs of ALL members of this group. If you are doing the assignment on your own (not part of a group), just write "not part of a group".

Name: Tanya Babbar
ANU ID: u7015074

Name: Sam Eckton
ANU ID: u7309356

Question 3: Select a piece of code in your assignment solution that you have written, and explain:



Yifan Luo

```python
def recursive_helper(pkg, cycle):
    nonlocal cycles, pkg_dependency, max_len
    if len(cycle) >= max_len:
        return
    if len(cycle) > 1 and cycle[0] == cycle[-1]:
        if cycle not in cycles:
            cycles.append(cycle)
        return
    if len(cycle) > len(set(cycle)):
        return
    for pkg_depend in pkg_dependency[pkg]:
        recursive_helper(pkg_depend, cycle + [pkg_depend])
```

a) What does this piece of code do?

Describe the purpose of the code, including assumptions and restrictions.

The function above is used in task 5 to find cyclic module dependencies.
It has two parameters:

- “pkg”, a string that represents a package name from a set of importable external packages of StdLib defined by tasks 1 – 3.
- “cycle”, a list used to store each step during the recursive progress, i.e., it stores any potential cycle that starts from “pkg”. It is initialized with a void list “[]” at the beginning of each cycle search.

Assumptions: The function assumes “pkg” is a valid importable external package from StdLib. The function is inside a function “find_cycles()” that is also used in task 5. In “find_cycles()”, a set of packages from tasks 1 – 3 is defined as a dictionary “pkg_dependency”. An integer variable “max_len” is defined as the termination condition of the recursion, which restricts the max length (recursion depth) of cycles, and a void list “cycles” is defined to keep all the potential cycles, which are also lists.

Restrictions: The function uses a similar strategy as DFS (Depth-first search) algorithms. Whenever the recursion progress finds a cycle, it is the shortest cycle that the head and the end share the same package name. From wherever it ends, there could be larger cycles that are ignored by this function.

b) How does it work?

Provide a high-level description of the algorithmic idea (and alternative ideas).

In the recursion progress, the cycle that starts from “pkg” have the following outcomes:

- A valid cycle. The cycle that starts from “pkg” also ends with “pkg”, and includes at least one different package name, i.e., the cycle cannot be a length-one node. Any valid cycle will be added to a nonlocal variable “cycles” from the function “find_cycles()”. Before adding, check if it is duplicated.
- An infinite sub-cycle inside. The cycle fails to find a valid cycle, but inside it, an infinite sub-cycle that starts from a successor of “pkg” is been found. Once the function finds any sub-cycle, it will be terminated at the first loop.
- No cycle. When the cycle starts from “pkg” and cannot find an end (without any sub-cycle inside), the length of this cycle keeps growing until it reached the length limit defined by a nonlocal variable “max_len” from function “find_cycles()”.

This function is also known as top-down recursion, which means the recursion starts from the beginning and then keeps and updates its record during the process of deeper-level recursion. When it finds a valid solution during the process, it will directly use the current record as the result of this recursion.

c) What other possible ways did you consider to implement this functionality, and why did you choose the one you did?

There are several methods for this recursion:

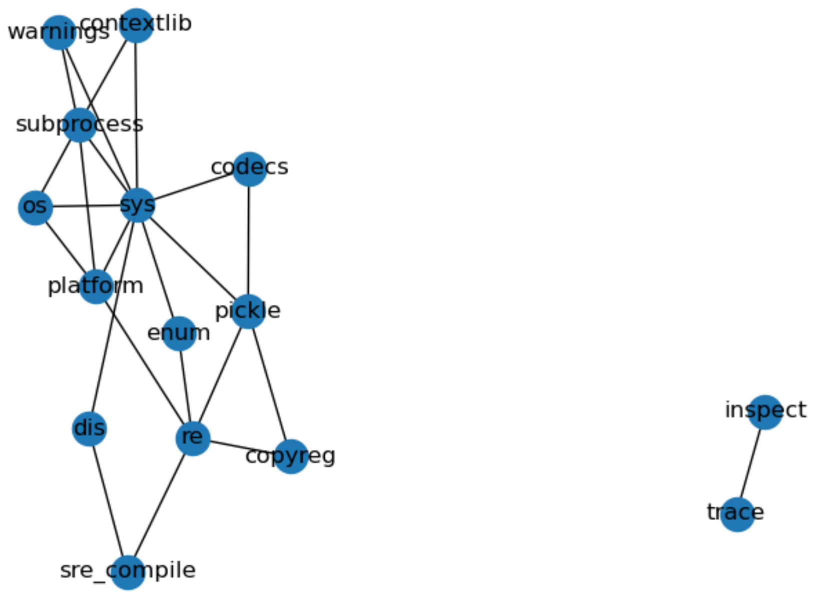
- Same top-down recursion, but replace nonlocal variables with local variables inside the recursion function.
- Bottom-up recursion, where new cycles are reversely linked back to its head at the end of each level of the recursion.
- DFS + stack. Instead of using a recursion function, a stack can be used to implement DFS as well.
- BFS + recursion/queue. BFS can traverse through the same cycles as DFS. It can be implemented by a top-down/bottom-up recursion function or a queue.
- Tree structure with node structure. A tree can be traversed by DFS and BFS to find cycles.
- Other graph algorithms, such as topological sorting and union-find set.

Why this function:

- No need to create tree or graph structures. It benefits the time and space complexity of the project, also easier to implement.
- Nonlocal variables instead of local variables. The recursion progress is implemented as a helper function inside the function “find_cycles()”, using nonlocal variables (“max_len”, “cycles” and “pkg_dependency”) makes the algorithm easy to read and has fewer parameters. No more additions and subtractions of local variables around the entry of recursion are needed.

Overall, depending on different requirements, task scales, datasets, time and space complexity and other conditions, there is no best method for this question.

Task 6: Build the StdLib module connectivity graph



The connectivity of StdLib modules from tasks 1 – 5 is represented as above.