Computer Science 35L: Week 8

Graphs and Topological Sorting

Makefile Pasta

Suppose you're preparing a ~*gourmet*~ dinner of roasted vegetable pasta, with the process represented by the Makefile below:

```
toss_pasta_with_veggies: roast_veggies cook_pasta
   echo 'tossed pasta and veggies :3'

preheat_oven:
   echo 'preheated oven'
wash_veggies:
   echo 'washed veggies'
season_veggies: wash_veggies
   echo 'seasoned veggies'
roast_veggies: preheat_oven season_veggies
   echo 'roasted veggies'

boil_water:
   echo 'boiled water'
cook_pasta: boil_water
   echo 'cooked pasta'
```

1. Draw a dependency graph for this Makefile.
    a. The nodes of the graph should be the Makefile targets.
    b. An edge from node A to node B means that A is a prerequisite for B.

```
   toss_pasta_with_veggies
^                          ^
|                          |
roast_veggies              cook_pasta
^            ^                   ^
|            |                   |
preheat_oven season_veggies      boil_water
                 ^
                 |
                wash_veggies
```

2. What are the characteristics of the graph that you drew?
    a. Cyclic or acyclic? acyclic
        i. acyclic
    b. Directed or undirected? directed

3. Derive a valid order for carrying out these steps such that we end up with an edible meal, assuming that we can only do one task at a time. Is your ordering the only possible such order?

Preheat, Wash, Boil, Season, roast, cook, toss
not only possible

CS Course Planning

Now that you're comfortable with graphs, let's learn how to represent them in Python. Here's some code that models CS course requisites as a graph:

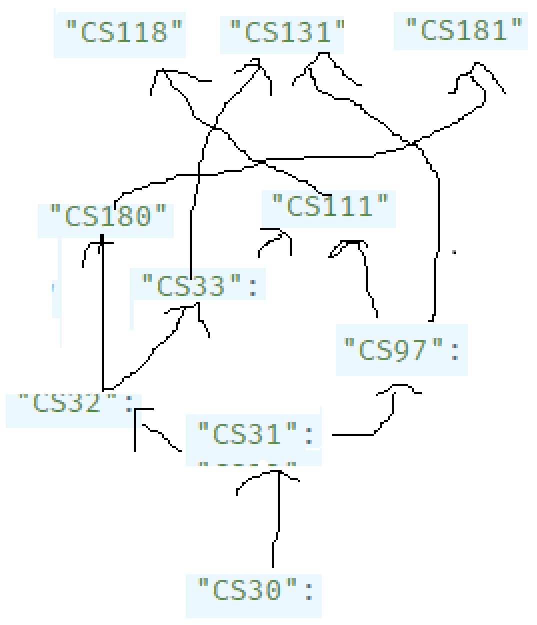
```python
class Course:
    def __init__(self, number, prereqs, req_for):
        self.number = number
        self.prereqs = prereqs
        self.req_for = req_for

    def remove_prereq(self, prereq):
        self.prereqs.remove(prereq)

    def get_num_of_prereqs(self):
        return len(self.prereqs)

courses = {
    "CS111": Course("CS111", ["CS33", "CS97"], ["CS118"]),
    "CS118": Course("CS118", ["CS111"], []),
    "CS131": Course("CS131", ["CS33", "CS97"], []),
    "CS181": Course("CS181", ["CS180"], []),
    "CS31": Course("CS31", ["CS30"], ["CS32", "CS97"]),
    "CS32": Course("CS32", ["CS31"], ["CS180", "CS33"]),
    "CS30": Course("CS30", [], ["CS31"]),
    "CS180": Course("CS180", ["CS32"], ["CS181"]),
    "CS33": Course("CS33", ["CS32"], ["CS131", "CS111"]),
    "CS97": Course("CS97", ["CS31"], ["CS111", "CS131"]),
}
```

4. Can you draw the graph represented by the above code?
    a. The nodes of the graph should be the course numbers (e.g., "CS 111").
    b. What about the edges



```
30
|
31
|      \
32   97
|     |     |
33   |     |
  |  /    /
131    111
           |
         118
```

Not complete

Topological Sorting

In this worksheet, we will use Kahn's algorithm for topological sorting. Here's how it works:

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge
```

```
while S is not empty do
    remove any node n from S
    add n to L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S

if length of L < number of nodes in graph then
    return error   (graph has at least one cycle)
else
    return L       (a topologically sorted order)
```

Note: this algorithm pseudocode was taken from Wikipedia, where you can read more about topological sorting!

5. What would a topological ordering of nodes in this graph represent?

Ordering of the classes such that all dependencies are listed earlier/first

6. Fill in the below function, which will topologically sort the `courses` graph. Here is documentation for the deque class, which is just a queue in Python.

```
from collections import deque

def topological_courses(courses):
    result = [] # list that will contain sorted elements (L)
    no_prereqs = deque() # all nodes w/ no incoming edge (S)

    # Part A. Fill no_prereqs with courses that have no
    # prerequisites to start.
     no_prereqs.append(“CS30”)
    #  YOUR CODE GOES HERE

#Loop through items, check, add to the queue if they fit
# then repeat

    for i in courses:
        if !i.prereqs:
            no_prereqs.append(i.number) #something like this?

    while len(no_prereqs) > 0:
        # Part B. Remove any node N from S and add it to L.
        N = no_prereqs.popleft()
        result.append(N)

        # Part C. For each node M that is a child of N, remove
        # the edge E connecting them.
        for i in courses:
            if N in i.prereqs:
                i.prereqs.remove(N)
                if(i.get_num_of_prereqs() == 0):
                      no_prereqs.append(i)

    # Part D. Check the length of L, in case there are cycles.
    if( len(results) !== len(courses)
          raise exception

    return result
```

Kahn's algorithm for Topological Sorting - GeeksforGeeks
(solution, in python)