Sean Blanchard
11/12/20
Math 554

Midterm 2

2-1) Dependencies

- takes as input a list of classes

listOfClasses = []

first we would want to get input from a user, being they enter a list of classes they are interested taking.

- has access to a list of prerequisites for each class

Next we must find the prerequisites of all the classes the user entered.

since we have access to a list of prerequisites for each class, we must add the list of prerequisites to each course.

* A big thing to be aware of is some prerequisites may have prior prerequisites, may or may not be an issue.

- outputs an order in which the classes can be taken such no class is taken until after its prerequisites.

- Do a boolean to check if prerequisites for the course have been taken.
- If no prerequisites were found, add it to the orderedClasses list.
- If prerequisites are found,
  run a loop to see how many prerequisites for each class and store number (int)
- Run a switch comparing 1-5 prerequisites count
  → submit each class in order by how many prerequisites.

```
return orderedClasses;

  bool hasPrerequisites = prerequisites();
       count = count('prerequsites()')
  if hasPrerequisites == 0
      addClass = orderedClasses;
  else
      for i in count
  switch 1:
      addClass = orderedClasses;
        break;
  switch 2:
      addClass = orderedClasses;
        break;
  etc. to 5

return orderedClasses;
```

2-2) Shortest Paths

a) Dijkrsta Algorithm, since we know we have an unexplored vertex, and this is the fastest algorithm.
- some hints why I used Dijkrsta, the graph is weighted and undirectional.

b) Topological Sorting, since we know it is directional and acyclic. (DAG)
This would run O(M+N)
M = # of edges.
- O(M) - Determine the indegree for each node
- O(N) - Find node w/ no incoming edges
- Add nodes until we run out of nodes w/ no incoming edges
O(N)
O(M+N)

c) Bellman Ford Algorithm, since we know that edge weights can be negative. At first it seems almost the same as question a, since it has negative edge weight we can't use Dijkrsta algorithm. It has a slower time complexity then Dijkrsta algorithm.

d) Bellman-Ford algorithm, not enough information is given to assume that the graph will contain only positive weight. I'd want to use Dijkstra, but Bellman-Ford algorithm is the one to be used here to be sure.

2-3 Dynamic Programming

a) essentially we can improve this code with memorization. Running the code will give us the correct output as is, but it reuses alot of calculations that have already been done. This will help us only calculate each node in the sub-tree once. This will give us a huge performance increase

```
C(n,k)
    IN: n, k ∈ 0, 1, 2, ..., n≥k
    if k==0 or n==k
        return 1;
    else if memorizationTable[n][k] != 0
    int memorizationTable[][] = {{}}
        return memorizationTable[n][k]
    else
        return C(n-1, k-1) + C(n-1, k)
```

b) subproblems are problems that can be broken down into "subproblems" which are reused several times or a recursive algorithm for a problem that solves the same subproblem over and over again. The best example to show is the Fibonacci sequence that has overlapping subproblems.

NON-MEMORIZATION

$f(5) = f(4) + f(3) = 5$

$f(3) = f(2) + f(1) = 2$

$f(1) = 1$

$f(2) = 1$

$f(4) = f(3) + f(2) = 3$

$f(2) = 1$

$f(3) = f(2) + f(1) = 2$

$f(1) = 1$

$f(2) = 1$

Ideal

OR W/ MEMORIZATION

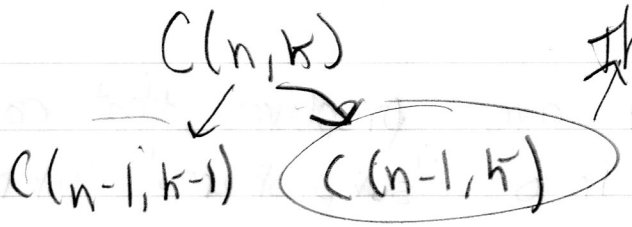
$f(5) = f(4) + f(3) = 5$

$f(4) = f(3) + f(2) = 3$

$f(3) = f(2) + f(1) = 2$

$f(1) = 1$

$f(2) = 1$



Thus creates the subproblems

This puts the topological order in the center 2.