CS 4349 Lecture–October 16th, 2017

Main topics for #lecture include #graph_traversal, #depth-first_search, and #topological_sort.

Prelude

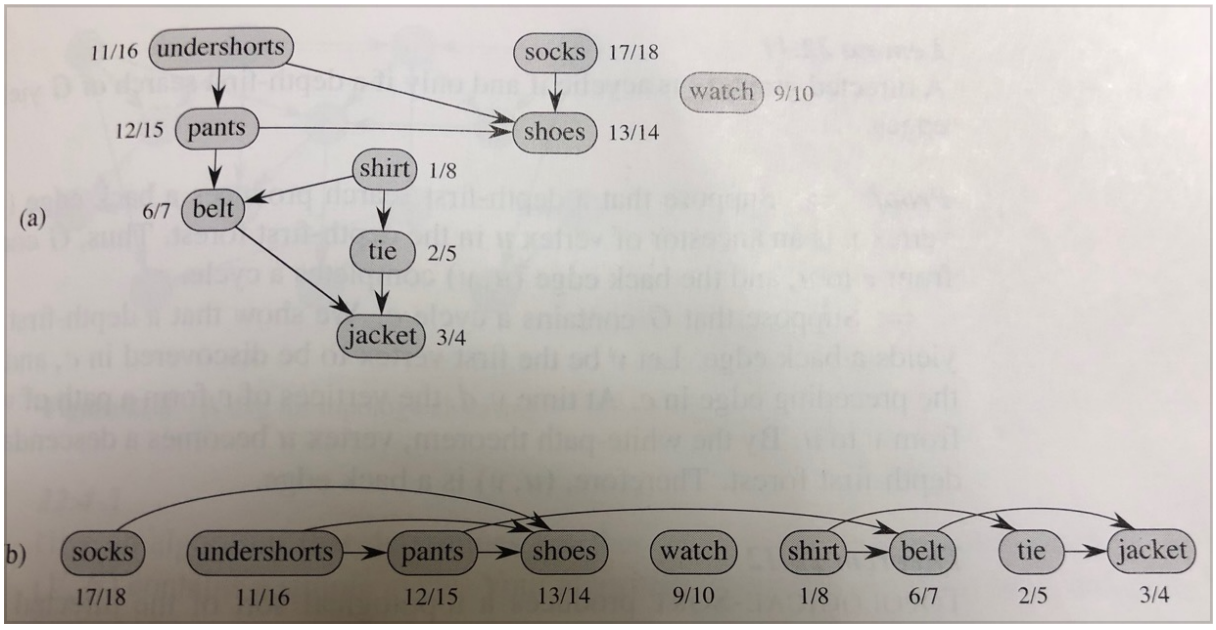
- Homework 6 due Wednesday, October 18th.

```
IsAcyclic(G):
    add vertex s
    for all vertices v ≠ s
        add edge s→v
        status(v) ← New
    return IsAcyclicDFS(s)
```

```
IsAcyclicDFS(v):
    status(v) ← Active
    for each edge v→w
        if status(w) = Active
            return False
        else if status(w) = New
            if IsAcyclicDFS(w) = False
                return False
    status(v) ← Done
    return True
```

DFS Application: Topological Sort

- A *topological ordering* of a directed graph G is a total order on the vertices so that u < v for every edge u ➔ v.
- For example, suppose we have this graph of clothing items. There is an edge u ➔ v if you must put on item u before putting on item v. A topological ordering would tell you what order to put them on to get dressed in the morning.



- One way to think about the ordering is that you can arrange the vertices horizontally so edges only go from left to right. The ordering tells you what order you need to visit vertices

- Topological orderings can only exist in DAGs. If you tried arranging the vertices along a cycle, the rightmost vertex on the cycle would have an edge pointing back to the left.
- On the other hand, every DAG has a topological ordering. We'll prove it by writing an algorithm to find one!
- One algorithm for finding a topological ordering in a DAG is to repeatedly pick a source vertex, add it to the ordering, and delete it from the graph. But this algorithm is tricky to implement efficiently. Instead, we'll use depth-first search to help us.
- The algorithm is based on this claim:
- Lemma: For any directed acyclic graph G, the first vertex marked DONE by IsAcyclic(G) must be a sink.
- Proof: Let v be the first vertex marked done. Suppose to the contrary that there is an edge v ➔ w. Consider when IsAcyclicDFS first considers the edge v ➔ w.
  - If status(w) = DONE, then w was marked DONE first, and we picked the wrong vertex v.
  - If status(w) = NEW, then the algorithm calls IsAcyclicDFS(w) which at some point marks w DONE, meaning we again did not pick the correct v.
  - If status(w) = ACTIVE, then we just found a directed cycle and G isn't a dag after all.
  - All three cases give a contradiction, so v ➔ w must not exist.
- So the first vertex marked DONE is a safe choice for the *end* of the topological ordering. Our algorithm for finding the whole ordering is to add vertices in the reverse order of which they are marked DONE.

```
TopologicalSort(G):
  add vertex s
  for all vertices v ≠ s
      add edge s→v
      status(v) ← NEW

  TopoSortDFS(s)

  for i ← 1 to V
      S[i] ← Pop
  return S[1..V]
```

```
TopoSortDFS(v):
  status(v) ← ACTIVE
  for each edge v→w
      if status(w) = NEW
          ProcessBackwardDFS(w)
      else if status(w) = ACTIVE
          fail gracefully
  status(v) ← DONE
  Push(v)
  return TRUE
```

- A couple notes: In most applications of topological sort, the goal isn't to make a list of vertices but instead to perform some computation on each vertex in topological order or reverse topological order. So instead of pushing into the stack, you may want to just do your computation.
- *Note the computation will happen in reverse topological order.*
- Remember the *dependency graph* where we have a vertex for recursive subproblems and edges between recursive subproblems? Processing vertices in reverse topological order is almost the same thing as dynamic programming.
- The main difference is that in dynamic programming we usually don't compute an explicit

dependency graph but instead it is implied by our recursive structure.

- The other big difference is that dependency graphs often have very nice structure, so we don't need a big formal search to figure out the right order of evaluation.
- But this connection hints at a nice fact. We can solve a lot of difficult problems on directed acyclic graphs in polynomial time by essentially doing dynamic programming. See Erickson 19.6 for an example of how to compute *longest* s, t-paths.
- Final notes: if you know the graph is acyclic, you don't have to do the NEW, ACTIVE, DONE thing. Instead, you can just mark vertices as before and do your pushing or processing immediately after looping over a vertex's edges.
- And if you want to process in topological order, you can compute the reversal of the DAG by reversing each edge, and then do processing in reverse topological order of the reversal.

Grades and the Exam

- You've probably seen your midterm grades. I computed them by weighing your homework and exam equally and then setting a curve. The homework is 30% of your total grade, so you're really hurting yourself if you don't take it seriously. Extra credit did boost a couple students up 1/3 of a letter grade.
- This class is supposed to be hard. It's all about designing things from scratch, and I think the best way to learn that skill is to try and fail, and keep trying until you succeed. I do the lectures so you can see examples and some techniques. The homework is where you're actually learn the techniques by fighting with them until it clicks.
- And you really do want to learn these techniques. If you use your degrees, you will have to design an algorithm at some point, and it may not be something you can look up on the internet. Many companies love asking you to design algorithms during interviews. I successfully interviewed for internships and a full time job at Google for example. It didn't matter that I was doing algorithms in grad school; they still wanted me to prove I could design them on the fly.
- So please, at least read through the homework early so you have time to think about it. I don't expect you to first see these problems on Wednesday and somehow solve them before class starts that evening.
- But if you are trying and it never clicks for certain problems, I don't want to punish you. That's why I have the citation policy as your opportunity to move on from the homework. If you abuse the policy, then you may get fine homework grades, but you'll likely never learn the material which is bad for after the semester ends and for the exam.

- So let's talk about the exam. I meant for it to be hard, but not that hard. I'm sorry.
- I will do things differently on the final. I'll also add more multiple part questions like the

recurrence question so you can demonstrate your understanding better.

- But, you **will** still be asked to design at least one algorithm from scratch like on the midterm so I can check that skill and give you motivation to take the homework seriously.
- There is the curve, but I don't like how much your exact grade is dependent on how well you did on the homework. Especially with the citation policy I use.
- I don't want to promise anything specific yet, but I do want to do *something* so you can prove you know what's going on even if I blindsided you with the specific questions. I'll probably offer some extra credit on the next homework that's similar to the exam questions. The extra credit will not affect the curve, so if you're satisfied with your grade, just keep doing what you're doing.
- I want to explain my take on partial credit in case you're surprised by how much you got or didn't get. If I ask you to design an algorithm and it was clear you had a correct algorithm in mind but with some flaws in the description or analysis, I gave lots of partial credit. Otherwise, I gave very little and likely none. That's because it is not possible for me to distinguish between earnest but flawed approaches and somebody just writing stuff down or making small observations to fill up paper. Correct answers to the wrong problem aren't worth anything either. And correct answers that are incomprehensible while being read aren't worth anything.
- I tried to be careful while grading, but I'm not perfect. If you wrote a correct algorithm and I didn't recognize that, feel free to ask for a regrade on that problem. You may explain to me why what you wrote is correct via email or in my office, but I'm still going to grade what you wrote.

- I'm going to hand back exams now, but I'd like them back before you leave. I'm also handing out some midterm feedback sheets for me, so you can help me teach you better for the rest of the semester. Please fill them out before we leave, and feel free to stack the midterm sheets near the door as you leave so you can remain anonymous.
- We'll spend the rest of the class period going over the last four midterm questions.