Graph Problem Solving Techniques

Winter 2023

1 Building and Modifying Graphs

One of the most useful tricks in designing graph algorithms is figuring out what graph to apply your algorithm to. You might not want to work directly with the graph that you are originally given. You might even be applying graph algorithms to problems where there isn't an obvious graph to start from! Exactly how you need to modify the graph with vary from problem to problem, so you'll need to think about what exactly you need to do and which graph is most natural to work with.

Some common tricks include things like:

- Adding a new starting vertex or ending vertex and connecting them up to the rest of the graph appropriately.
- Considering a subgraph consisting of only certain edges or certain vertices.
- Doing something involving multiple copies of the original graph where a vertex in the new graph encodes not just a vertex in the old graph but also some additional information.

One fairly general type of graph worth considering is what one might call the *configuration graph* of a problem. Here the vertices correspond to possible states that one might be in. This might encode not just a location (if indeed locations are a thing in this problem), but also information like which tasks still need to be accomplished, or how much time you have left, or whether you have flipped the switch yet or not- whatever information is needed to describe the full relevant state of the problem at that point in time. The edges of the configuration graph should tell you what kind of transitions from one state to another are possible. Often this will just involve moving from one location to another, but you might also need to encode things like flipping a switch or spending money or whatever else the problem in question allows you to do.

[[Work some examples. My Homework 1s tend to all have at least one good example problem of this kind.]]

2 Topological Sort

Topological sorting of DAGs is often a quite powerful tool. For many problems on graphs, there is some function $f(v)$ that you want to compute on each vertex. Often it is the case that if you know $f(v)$ for either all vertices leading *to* v or for all vertices coming *from* v, it is easy to compute $f(v)$. This suggests that we perhaps can compute the values of $f(v)$ for all vertices in the graph one at a time in some order. However, the order here is critical.

If $f(v)$ can be computed by knowing f at all of v's parents, then we need to make sure that f is computed at those parents first. We can do this by ensuring that we compute $f(v)$ over vertices in some topological order. If we need to know f at v's children, we want to use a reverse topological order.

[[Work some examples.]]

3 Algorithms to Know

Some of the algorithms that we have gone over in class that you should make sure that you understand and perhaps can implement by hand:

- Explore/DFS, keeping track of pre-/post-orders.
- Computing (Strongly) Connected Components.
- Topological Sort.

[[Examples.]]