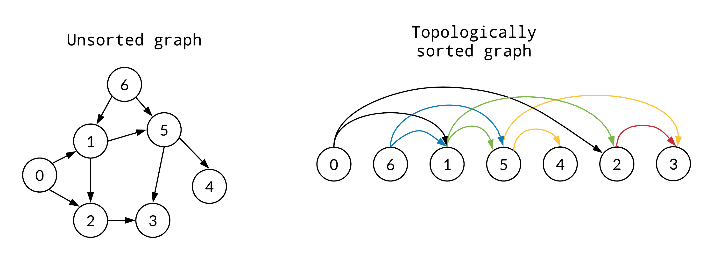
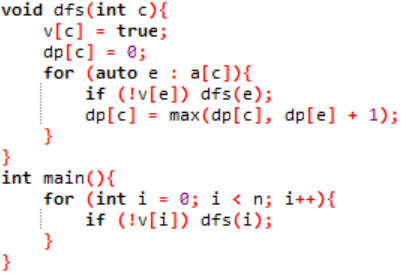
**G - Longest path**

Given a DAG with vertices and directed edges, find the length of the longest path.

Solution 1: Topological sorting

Topological sort, then you can use basic dp because each edge points from some node you already optimized to a future node. (there is no ‘back edges’)

Solution 2: clever dfs

dp[i] is length of longest path starting from node i

Observation: if dfs(i) is called, dp[i] will be optimized (because dfs goes through all possible descendants of i)

2 possibilities: a child of i, let’s say j is visited, which means dp[j] is already optimized; j is not visited, then we need to dfs(j). In both cases update dp[i] because j can be the next step in longest path starting from i regardless of visited or not.

**J - Sushi**

There are dishes, and each dish has 1, 2 or 3 sushi on it. In an operation, you pick a dish out of the dishes, and eat 1 sushi from it if there are more than 0 sushi on it. (If the dish has 0 sushi do nothing) What is the expected number of operations to eat all the sushi?

Intuition: , each dish 3 possibilities (actually 4) hmm maybe can dp[n][n][n]

dp[i][j][k] = expected number of operations to eat i dishes with 1 sushi, j dishes with 2 sushi and k dishes with 3 sushi

Base case: dp[0][0][0] = 0 (expected number of operations is 0)

: probability of choosing a dish with 1 sushi,

: probability of choosing a dish with 2 sushi,

: probability of choosing a dish with 3 sushi,

: probability of choosing a dish with 0 sushi,

( because eating a sushi from a dish with 2 sushi produces a dish with 1 sushi)

yey we have our dp transition!