Week 6: Lab 5

Lab 5

This lab covers several dynamic programming questions.

Flavours Galore

Go to the DOMjudge server (http://domjudge.gozz.au/) select lab5 and look at "flavoursgalore".

Hint 1: This problem is very similar to the introductory DP in lecture 9. It involves finding the longest path. The only issue is that the graph isn't guaranteed to be a directed acyclic graph. How do we deal with that?

Hint 2: We can detect if a graph has a cycle. If it does, the answer is -1. If notm then it is a directed acyclic graph and we can apply DP! There are many ways to check if a graph contains a cycle. One method is to use Khan's algorithm from lecture 8. If Khan's algorithm fails to process all the nodes, it means the graph must contain a cycle.

This problem was taken from the South Pacficic ICPC.

GPU Meme

Next, look at "gpumeme". This is a problem similar to 0-1 knapsack, which is covered in lecture 10.

Hint 1: Suppose that we have a DP function f(i,w) that gives us the minimum monetary cost possible for the first i GPUs such that the environmental cost is **exactly** w (not up to w as in 0-1 knapsack). We could use it to find the solution by trying different values of w for f(N,w) and taking the largest w for which a solution exists. You will need to modify the 0-1 knapsack DP to compute f(i,w).

Hint 2: The base case for f(i, w) is when i=0. It is only possible to achieve an environmental cost of zero with zero GPUs, so f(0,0)=0 and f(0,w)=None otherwise.

Hint 3: The recurrence for f(i,w) = min(f(i-1,w), f(i-1, w-C[i])+M[i] if C[i] <= w). Note that C[i] is the environmental cost and M[i] is the monetary cost. This is similar to 0-1 knapsack and the two cases represent either not taking item i or taking item i. We also need to deal with the cases when f(i-1,w) or f(i-1, w-C[i]) are None.

This problem was taken from the South Pacficic ICPC.

Master of Jenga (optional challenge question)

This is a challenge question and not all students are expected to solve it.

See the "masterofjenga" problem. This problem can be solved with DP. However, did you notice that the output for this problem involves giving the actual bricks in your solution? For this problem, you will need to code the traceback too! Tracebacks are covered in lecture 10.

Efficient City (optional challenge question)

This is a challenge question and not all students are expected to solve it.

See the "efficientcity" problem. This problem is not a dynamic programming problem! It can be solved using divide and conquer.

A note about recursion. Python and PyPy are inefficient with recursion and also may stack overflow. While your solution should be to divide and conquer, we suggest not coding it recursively. Instead, you can use a stack data structure and manually simulate the recursion. This will lead to faster code that also will not stack overflow.

This problem was taken from the South Pacficic ICPC.