Problem Set #9

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Longest Paths

Let $G = (V, E)$ be a directed graph with nodes $v_1, v_2, \ldots, v_n$. We say that G is an ordered graph if it has the following properties.

- Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.
- Each node except v_n has at least one edge leaving it. That is, for every node v_i, $i = 1, 2, \ldots, n-1$, there is at least one edge of the form (v_i, v_j).

The *length* of a path is the number of edges in the path. The goal in this question is to solve the following problem

> Given an ordered graph G, find the length of the longest path that begins at v_1 and ends at v_n.

Give an efficient algorithm using dynamic programming approach that takes an ordered graph G and returns the length of the longest path that begins at v_1 and ends at v_n.

Solution

First, notice that not all nodes v_i necessarily have a path from v_1 to v_i. We set $OPT[i] = -\infty$ in this case. $OPT[1] = 0$ since the longest path from v_1 to v_1 has length 0 (there are no cycles in our graph). What is left, then, is to defined $OPT[i]$ in the general case. $OPT[i] = \max_{(1 \le k < i) \wedge (v_k, v_i) \in E}(OPT[k] + 1)$. The algorithm that implements this dynamic programming algorithm creates an array M of length n. It sets $M[1] = 0$. Then for $i = 2$ to n, the algorithm looks at all edges going *into* i, and computes the max of the longest path for those edges, plus 1. It sets $M[i]$ to this value. When the loop completes, $M[n]$ holds the longest path from v_1 to v_n. The running time is $O(n^2)$ because the loop runs $O(n)$ times, and, in the worst case, a node has $O(n)$ incident edges.

Problem 2: Dynamic Programming

It's the end of the semester, and you're taking n courses, each with a final project. Each project will be graded on a scale of 1 to $g > 1$, where a higher number is a better grade. Your goal is to maximize your average grade on the n projects. (Note: this is equivalent to maximizing the *total* of all grades, since the difference between the total and the average is just the factor n.)

You have a finite number of hours $H > n$ to complete all of your course projects; you need to decide how to divide your time. H is a positive integer, and you can spend an integer number of hours on each project. Assume your grades are deterministically based on time spent; you have a

set of functions $\{f_i : 1, 2, \ldots, n\}$ for your n courses; if you spend $h \leq H$ hours on the project for course i, you will get a grade of $f_i(h)$. The functions f_i are nondecreasing; spending more time on a course project will not *lower* your grade in the course.

To help get you started, think about the (i, h) subproblem. Let the (i, h) subproblem be the problem that maximizes your grade on the first i courses in at most h hours. Clearly, the complete solution is the solution to the (n, H) subproblem. Start by defining the values of the $(0, h)$ subproblems for all h and the $(i, 0)$ subproblems for all i (the latter corresponds to the grade you will get on a course project if you spend *no* time on it).

Give the dynamic programming equation to define the value of the optimal solution (i.e., the value of any (i, h) subproblem, where $0 \leq i \leq n$ and $0 \leq h \leq H$), describe an algorithm for iteratively computing the value of the optimal solution, and describe an algorithm for recreating the optimal solution (i.e., mapping your H hours to your n projects).

Solution

Let $Opt(i, h)$ be the maximum total grade that can be achieved for this subproblem. Then $Opt(0, h) = 0$ for all h and $Opt(i, 0) = \sum_{j=1}^{i} f_j(0)$ for all i. Now in the optimal solution to the (i, h) subproblem, one spends k hours on course i for some value $k \in [0, h]$. Thus:

$$Opt(i, h) = \max_{0 \leq k \leq h} f_i(k) + Opt(i - 1, h - k)$$

To compute the table, we first fill in the values of $Opt(0, h)$ for all h and the values of $Opt(i, 0)$ for all i. Then, starting with $Opt(1, 1)$, we fill in the values in row major order. The value in the table at location (n, H) (the bottom right corner) is the maximum possible grade.

To be able to recreate the optimal solution from the table (i.e., how many hours to spend on each course), we also record the value k that produces the maximum for each decision in the table. Then starting in entry (n, H) in the table, we can trace back through the table of optimal values to find the solution. Specifically, we look at entry (n, H) and the value of k recorded for it. This is how many hours to spend on course n. We then go to entry $(i - 1, h - k)$ for that particular k and repeat.

Problem 3: Network Flow Theory

Decide whether you think each of the following is true or false. If it is true, give a brief explanation. If it is false, give a counter example.

1. Let G be an arbitrary flow network with a source s, a sink t, and a positive integer capacity c_e on every edge e. If f is a maximum s-t flow in G, then f saturates every edge out of s with flow (i.e., for all edges e out of s, we have $f(e) = c_e$.

   Solution

   This is false. Consider a graph with nodes s, v, w, t, edges $(s, v), (v, w), (w, t)$, capacities of 2 on (s, v) and (w, t), and capacity of 1 on (v, w). Then the maximum flow has value 1, and this does not saturate the edge out of s.

2. Let G be an arbitrary flow network with a source s, a sink t, and a positive integer capacity c_e on every edge e, and let (A, B) be a minimum s-t cut with respect to these capacities

$\{c_e : e \in E\}$. Now suppose we add 1 to every capacity. Then (A, B) is still a minimum s-t cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

Solution

This is also false. Conside a graph with nodes s, v_1, v_2, v_3, w, t, edges (s, v_i) and (v_i, w) for each i and an edge (w, t). There is a capacity of 4 on edge (w, t) and a capacity of 1 on all other edges. Then setting $A = \{s\}$ and $B = V - A$ gives a minimum cut, with capacity 3. But if we add one to every edge, then this cut has capacity 6, more than the capacity 4 on the cut with $B = \{t\}$ and $A = V - B$.

Problem 4: Network Flow

Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.

Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location(so different people will have different options for hospitals, depending on where they are right now).

At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively workout whether they can choose a hospital for each of the injured people in such a way that the load on the hospital is balanced: Each hospital receives at most $\lceil n/k \rceil$ people.

Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible

Solution

We build the following flow network. There is a node v_i for each patient i, a node w_j for each hospital j, and an edge (v_i, w_j) of capacity 1 if patient i is within a half hour drive of hospital j. We then connect a super-source s to each of the patient nodes by an edge capacity 1, and we connect each of the hospital nodes to a super-sink t by an edge of capacity $\lceil n/k \rceil$.

We claim that there is a feasible way to send all patients to hospitals if and only if there is an $s - t$ flow of value n. If there is a feasible way to send patients, then we send one unit of flow from s to t along each of the paths s, v_i, w_j, t, where patient i is sent to hospital j. This does not violate the capacity conditions, in particular on the edges (w_j, t), due to the load constraints. Conversely, if there is a flow of value n, then there is one with integer values. We send patient i to hospital j if the edge (v_i, w_j) carries one unit of flow, and we observe that the capacity condition ensures that no hospital is overloaded.

The running time is the time required to solve a max-flow problem on a graph with $O(n+k)$ nodes and $O(nk)$ edges.