15-451/651 Algorithms, Spring 2019

Homework #4 **Due: Monday–Thursday, March 4–7, 2019**

(25 pts) 1. **(Fall Foliage in the Spring)**

Let $G = (V, E)$ be an undirected graph with the vertex set V and edge set E. Each edge has an integer weight $w_e > 0$. A $(k, n-k)$-*partition* of G is a coloring of the vertices with two colors (red/blue) such that there are k red nodes, and $n-k$ blue nodes.

The goal is to find a $(k, n-k)$-partition of G where the total weight of *split* edges (edges with one endpoint red and the other blue) is minimized. In general there is no known polynomial time algorithm for this problem. However, if the graph is a tree, the problem becomes easier.

Design a polynomial-time algorithm to solve this problem when the input graph G is a *binary tree*. Formally, a binary tree (for our purposes) is a tree rooted at some node r, and each node has at most two children. Prove the correctness of your algorithm and analyze its running time. For full credit, your algorithm should take $O(n^3)$ time.

Solution: We will write a dynamic program to solve this problem. Lets just focus on finding the cost of the minimum K-partition. Define $C(v, K, c) =$ cost of min K-partition of tree rooted at v where v is colored c.

Our base case is as follows: $C(v, 1, R) = C(v, 0, B) = 0$ when v is a leaf, and every other entry $C(v, K, R) = C(v, K, B) = \infty$ for $K \geq 2$. Now, for v an internal node with children l, r:

$$\begin{aligned} C(v, blue, K) = \min_{a,b \in \{red,blue\}^2} \{ & \min_{j \leq K} \{ C(l, a, j) + C(r, b, K-j) \\ & + w_{vl}(1 - \delta(blue, a)) + w_{vr}(1 - \delta(blue, b)) \} \} \\ C(v, red, K) = \min_{a,b \in \{red,blue\}^2} \{ & \min_{j \leq K-1} \{ C(l, a, j) + C(r, b, K-1-j) \\ & + w_{vl}(1 - \delta(red, a)) + w_{vr}(1 - \delta(red, b)) \} \} \end{aligned}$$

Here $\delta(x, y)$ is the Kronecker delta (1 if $x = y$ and 0 otherwise).

To summarize the DP, if a vertex v can have K red children and wants to be colored red, it will have $K-1$ remaining red vertices under it, which will be split between its two children in some way. Similarly, if v wants to be blue, it will have K remaining red vertices for its children. It will give each of its children arbitrary colors and some number of red vertices, and then take the min cost over all of the choices it gave to its children.

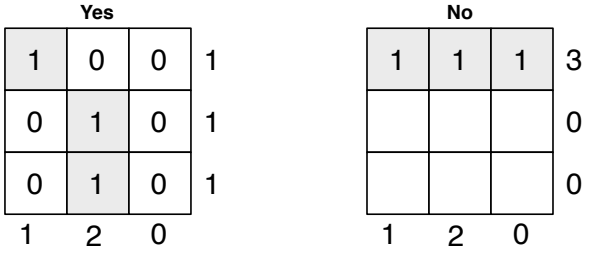
The final answer we return is $\min(C(r, red, k), C(r, blue, k))$. The total number of vertices in the DAG representing this DP is $4nk = O(n^2)$, and at each vertex we do $O(n)$ work taking a min over $4k$ values. Thus the total work for the DP is $O(n^3)$.

To get the actual assignment, we can go back into the DP table and look at which sub-calls were taken in the min and keep building up the assignment.

(25 pts) 2. **(Sort of Magic Square)**

Consider the GENERALIZED MAGIC SQUARE problem: Given k, and two lists of n non-negative integers $\vec{r} = (r_1, \ldots, r_n)$ and $\vec{c} = (c_1, \ldots, c_n)$, we ask whether there is an $n \times n$ grid of integers from the set $\{0, 1, \ldots, k\}$ such that row i sums to r_i and column j sums to c_j. We assume $\sum_i r_i = \sum_j c_j$.

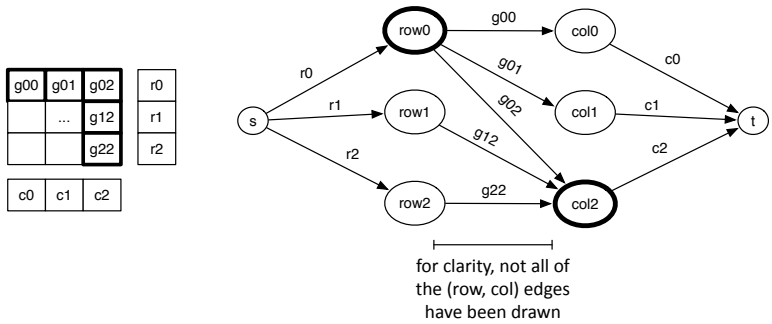
Examples: With $k = 1$ and $n = 3$ with $\vec{c} = (1, 2, 0)$ and $\vec{r} = (1, 1, 1)$ (answer=yes) or $\vec{r} = (3, 0, 0)$ (answer=no):



Use network flow to create a polynomial-time algorithm to decide whether it is possible to design a grid containing integers from 0 to k that obeys the given $\vec{r}$ and $\vec{c}$ sums. Illustrate your construction with the "yes" example above.

Solution: We construct a flow network in the following way: Create a node for each row and a node for each column. Create a source vertex s and a sink vertex t. We add edges from s to each of the row vertices, and edges from each of the column vertices to t. The capacities of the (s, row_i) edge is the desired row sum r_i for row i. You can think of each unit of flow as a token, and this edge allows r_i tokens to be placed onto row i. For each row i, we add edges from the row i node to each of the column nodes. These edges have capacity k. If the flow pushes r_i "tokens" onto the row i node, these edges allow it to be distributed among the columns in that row. Finally, we add edges from each column node to t. The edge (col_j, t) has capacity c_j, the desired column sum for column j.

If the maximum flow saturates the edges leaving s and entering t, then the flow on each of the (row_i, col_j) edges gives the integer than should be placed into cell (i, j). Schematically:



(25 pts) 3. **(You Think You've Got Problems?)** You just realize that the semester is almost half-over, and you need to get your act together. Your textbook has n chapters, each

quite technical, where chapter i costs C_i dollars to read. ("Time is money," as they say.)

The homework consists of a set of m problems $p_1, p_2, \ldots, p_m$. For each problem p_j, there is a subset $P_j \subseteq \{1, \ldots, n\}$ of the chapters that it depends on. If you have read *all* the chapters in P_j you can solve the problem p_j, but if you've missed even one of the chapters in P_j you cannot solve the problem. Solving p_j gives you V_j dollars of value. The *net utility* is the value of the problems solved minus the cost of the chapters read. The goal is to find a subset $R \subseteq \{1, \ldots, n\}$ of the chapters to read, to maximize the net utility you get.

> E.g., if $P_1 = \{2, 3, 5\}$, $P_2 = \{1, 2, 3\}$ and $P_3 = \{2, 3, 4\}$. Suppose the costs for the $n = 5$ chapters are $1, 4, 3, 8, 1$ respectively, and values for the three problems are $V_1 = 14, V_2 = 4, V_3 = 7$. If you have read chapters $\{2, 3, 4, 5\}$ then your cost is $4+3+8+1 = 16$ and the value you get from having solved P_1 and P_3 is $14+7 = 21$. So the net utility is $21-16 = 5$. On the other hand, having read just $2, 3, 5$ you'd have net utility $14 - (4 + 3 + 1) = 6$. And having read just $1, 2$ you'd get net utility $0 - (1 + 4) = -5$.

Show how to use an s-t-min-cut algorithm to solve this problem in polynomial time.

Hint: Can you solve the problem of *minimizing* the cost of the chapters you read *plus* the sum of the values of problems you *did not* solve. Why is solving this problem this useful? Think about how you could solve this problem using an s-t min-cut algorithm.

Solution: First, the x that maximizes a function $f(x)$ is the same as the x that minimizes the function $-f(x)$, so maximizing the values of problems solved minus the cost of chapters read is the same as minimizing the cost of chapters read minus the values of problems solved. Next, adding a constant offset doesn't change the location of the maximum, so by adding the sum of values of all problems we get to the equivalent question of finding the chapters to read that minimizes the cost of chapters read plus the values of problems not solved. Let's call this the *penalty* of a solution. So, our goal is to find a solution of minimum penalty.

Now, we construct the following flow graph. We have a source s (call this "level 0"), a set of m vertices labeled $p_1, \ldots, p_m$ at level 1, a set of n vertices labeled $ch_1, \ldots, ch_n$ at level 2, and a sink t at level 3. The source is connected to each vertex p_j by an edge of capacity V_j. Vertices p_j are connected by an edge of infinite capacity to each vertex ch_i for $i \in P_j$, and each vertex ch_i is connected by an edge of capacity C_i to the sink. All edges are directed left-to-right. We solve for the minimum s-t cut, and output as our solution to read all chapters and solve all problems whose associated vertices are on the s-side of the cut. (An equally-good construction is to connect the source to the chapters, connect the chapters to the problems, and then connect the problems to the sink, in which case we would output the chapters and problems on the t-side of the cut).

To analyze this we need to prove two directions: (1) the solution proposed is legal (we never say to solve a problem whose chapters haven't all been read) and has penalty equal to the capacity of the cut, and (2) vice-versa: for any proposed solution Y, there exists a cut of capacity equal to the penalty of Y. Note that we must prove (2) in order to conclude that the *minimum* cut gives the solution of *minimum* penalty (otherwise, perhaps there is a better solution that doesn't correspond to any cut at all).

Proof of (1): the minimum cut will not cut any edge of infinite capacity, which means that if a problem p_j is on the s-side of the cut, all chapters it depends on will be on the s-side as well. This shows that the solution is *legal*. Moreover, the capacity of the cut is the capacity of all edges between the source and problems p_j on the t-side of the cut plus the capacity of all edges from chapters ch_i on the s-side of the cut to t. This is precisely the penalty of the proposed solution.

Proof of (2): Given a proposed solution Y, we cut edges from s to problems not solved, and from t to chapters read. We cut edges of total capacity equal to the penalty of the solution, so what remains is to prove this is an s-t cut. Suppose for contradiction there was a path remaining from s to t. Since all edges are directed left-to-right, this path must be of the form "s-p_j-ch_i-t". But this means that we solved problem p_j without reading the chapter ch_i that it depended on, contradicting the legality of Y. Note that it is crucial for this argument that the edges in the graph be directed: otherwise there could be a path from s to a problem solved, to a chapter read, back to a problem not solved, forward to a chapter not read, and then to t.