COT5405 Analysis of Algorithms Homework 4

The rules for this HW are same as the previous ones. Due date: April 10 Thursday 4 PM (not 5 PM).

1. (Page Limit: 1 page)

   Using dynamic programming solve the following problem:

   Given n integers $(a_1, \ldots, a_n)$ between 0 and K, partition them into two subsets such that the difference between the sums of each subset is minimized. In other words, if the sum of the first partition is A and the sum of the second partition is B, we want to have $|A - B|$ to be as small as possible.

   Solution:

   Solution using dynamic programming: Let S be the sum of all the n integers. The problem is equivalent to minimizing $S/2 - A$, where A is the sum of the first partition, as we can assume $A \leq B$.

   Let $P(i, j) = 1$ if there is a partition of $a_0, ..., a_i$ which has sum j, and 0 otherwise.

   So $P(i) = max\{P(i-1, j), P(i-1, j - A_i))\}$. The first part comes from the fact that if you have a partition of $a_0, ..., a_i$ with sum j, you also have a partition of $a_0, ..., a_{i+1}$ with sum j. The second part is obvious.

   To solve the problem using this recurrence, just work up till $P(S/2)$ and find the largest m such that $P(m) = 1$. The difference between the partitions would be $2m$. Of course one also needs to use the $P(i)$ table until the partitions are recovered - which is straightforward.

   The worst case running time would be $O(n^2K)$

   Grading criteria:

   If you gave a heuristic-based algorithm, you will be penalized heavily since such algorithms always fail to give the correct result for some deliberately crafted input set, i.e., such algorithms are not deterministic about their accuracy.

   If you have an algo that is exponential, you would be penalized heavily.

   Many of you contended that the problem is NP-complete and thus unsolvable in polynomial time. That would be true, if we did not know about the size of the individual integers. In this specific problem, we gave that each integer is bounded by K. That makes the algorithm $O(n^2K)$, or polynomial.

   Those who did not give any complexity analysis lost 5 points (and gained a "C ?" annotation). If you did the complexity analysis as $O(nS)$ where you stated S is the sum of the n numbers, you would still lose couple of points since $S = O(nK)$, and thus the complexity $= O(nS) = O(n^2K)$. I understand this is more of a formality, but the truth is that we had provided all the necessary parameters with which you can express the complexity as. If we did not supply K, then you had a point of using S, as without using that it would have been impossible to express the complexity. Besides, the penalty is not that harsh.

2. (Page Limit: 1 page)

Give a dynamic programming algorithm that computes the length of the Longest Common Subsequence of two strings X, Y ($LCS(X, Y)$) and uses $\Theta(\min\{|X|, |Y|\})$ space and $O(|X||Y|)$ time.

Solution:

The stock LCS algorithm requires $O(|X||Y|)$ memory and run time $O(|X||Y|)$. We wish to modify it to use less memory, without changing the time complexity. Notice that only two rows the current row i and previous row $i-1$ are used in the computation of an element $c[i, j]$ in the memoization table. Thus we can discard data in rows $0 \ldots i-2$. However, when we say "discard", we do not mean using the space. Basically, all you do it to maintain two rows for current and previous. Once the current row is computed, you can either copy it back to the previous or simply treat the current as previous and previous as current for the next round. To further reduce memory requirements, we should associate the smaller of the two strings (i.e. $\min |X|, |Y|$) with the table rows. Without loss of generality, lets assume X is the shortest string; we can always swap X and Y as necessary to achieve this.

Grading criteria:

You cannot just state the stock LCS algorithm and expect to gain "some" points. You need to state clearly exactly how you are reducing the memory consumption. Nevertheless, we have been very lenient in this question.

3. (Page Limit: 1 Page)

Let $G(V, E)$ be a graph with real weighted edges. For each vertex $v \in V$ you are given a value $f(v)$ with the property, that for every edge $(u, v) \in E$, $w(u, v) + f(v) - f(u) \geq 0$. The values $f(v)$ are given as input. You are also given sets of vertices $S_1, S_2, ..., S_k$, such that $S_i \cap S_j = \phi$ (that is, no vertex $v \in V$ belongs to more than one set S_i). Now define

$$g(S_i, S_j) = min\ \{g(u, v) | u \in S_i, v \in S_j\}$$

Here $g(u, v)$ is the minimum weighted path from u to v. Note that you have to consider both vertex and edge weights here when calculating path weight. The value $f(u)$ is the weight of vertex u. Give an algorithm that computes $g(S_i, S_j)$, for every pair S_i, S_j, and takes $O(k|E|log|E| + |V|^2)$.

Solution:

First we transform the graph such that the weights on vertices are removed. So new edge weight $w'(u, v) = w(u, v) + f(u) - f(v)$.

The main idea behind this algorithm is avoiding computing all pair wise distances for vertices in S_i and S_j. We do this by adding a representative vertex x_i to each S_i. Connect x_i to all vertices in S_i with edges of weight zero. Note that this vertex and edge addition is done to the transformed graph, so we need not worry about vertex weights. Now the minimum distance between two sets is the distance between its two representatives.

Now the algorithm is as follows:

Transform the Graph with $w'(u, v) = w(u, v) + f(u) - f(v)$.

1 FOR i from 1 to k

2 Add x_i to S_i with $w'(x_i, v) = 0, v \in S_i$
3 Find shortest path lengths from x_i to $v, v \in S_i$.
4 Also note the length and the vertex to which x_i has shortest path.
5 Call it $g(x_i, S_i)$.
6 FOR i from 1 to k
7 FOR j from 1 to k and $j \neq i$
8 $g(S_i, S_j) = g(x_i, S_i) + g(x_j, S_j)$.

Transformation takes $O(|V|^2)$, since you have it for all pairs of vertices. Lines 1-5 calculate a dijkstras over graph with $|E|$ edges and is done k times $O(k|E|log|E|)$. Lines 6-8 calculate shortest paths between k sets is $O(k^2)$. So total time is $O(k|E|log|E| + |V|^2)$, since $k \leq |V|$.

4. (Page Limit: 1 Page)

Let $G(V, E)$ be a graph where each edge capacity is an integer. Assume f is a flow on G, with the condition that flow on each edge is again an integer. E_f denotes the edges in the residual graph. Now show that

(a) $|E| \leq |E_f| \leq 2 * |E|$. Here $|E|$ and $|E_f|$ are the number of edges in E and E_f respectively.

(b) If the capacity of each edge in G takes only 0 or 1 then $|E_f| = |E|$.

Solution:

(a) For any edge (u, v) in the original graph there will be atmost two and atleast one edge in the residula graph. If the flow along that edge is zero then the edge remains as it is. If the flow is equal to capacity of edge then a new edge will appear (which is equal is capacity but from (v, u), but the original edge disappears. When the flow is between zero and capacity of original edge, then a new edge will be created, but the old edge remains with reduced capacity (sum of their capacities is equal to the original capacity of the edge). So the number of edges never decreases. The number of edges in the residual graph can become atmost twice the number of edges already present.

(b) This is similar to first part of previous proof. Here the flow is either equal to zero or equal to one, which is the capacity of original edge. So in both cases there is only one edge left. Either the original edge or the new edge. So the number of edges does not change.

Grading criteria:

Almost all of you got you right. Again, we have been very lenient in grading this question.

5. (Page Limit: 1/2 page)

Prove or disprove: If $(N, \bar{N})$ is a cut in network $G(V, E)$, define $E(N, \bar{N})$ to be the set of edges leading from vertices in N to vertices in $\bar{N}$. Suppose E' is a minimal subset of E such that there is no path from s (source) to t (sink) in $(V, E - E')$. Is there necessarily a cut $(N, \bar{N})$ such that $E' = E(N, \bar{N})$?

Solution:

Yes, there is always such a cut. We will prove this by constructing the cut.

We will first form two sets of vertices, A and B. Let A contain all the vertices that are reachable from s in $(V, E - E')$.

Let B contain all the vertices from which t is reachable in $(V, E - E')$. (For simplicity we ignore all the vertices neither reachable from s nor from which t is reachable, but adding all these edges into any of the two sets would not change the proof.)

First we need to show that A and B are disjoint, which follows easily from the fact that there is no path from s to t in $(V, E - E')$.

If A and B were not disjoint, there would be a vertex v common to both sets, meaning that it is reachable from s and t is reachable from v - leading to a contradiction.

Now we claim that (A, B) is actually the cut we were looking for. First of all, it is a cut, as it partitions the vertices into two disjoint sets, where s and t lie in different partitions. It is also impossible to have an edge in $E(A, B)$ which is not in E', as that would imply an edge from a vertex in A to a vertex in B in $(V, E - E')$, leading to the same contradiction as before. This means that $E' \supseteq E(A, B)$. On the other hand we also have $E' \subseteq E(A, B)$. To see this consider all the cases for an edge e in E':

- e lies completely in A. In this case we can just remove it from E', and there will still be no path from s to t in $(V, E - E')$, contradicting our definition of E'
- e lies completely in B. The same idea in the previous case holds for B too.
- e is an edge from B to A. Again, one can just remove this edge and still have no path from s to t, as e is pointing to the wrong direction.
- e is an edge from A to B. This is the only possible case, but now it also lies in $E(A, B)$ by definition.

Since we have $E' \supseteq E(A, B)$ and $E' \subseteq E(A, B)$, it follows that $E' = E(A, B)$. Now setting $N = A$ and $\bar{N} = B$ completes the proof.

Grading criteria:
Almost nobody got this problem right. Most of the students misunderstood the problem - they interpreted the sentence

> define $E(N, \bar{N})$ to be the set of edges leading **from** vertices in N **to** vertices in $\bar{N}$.

as

> define $E(N, \bar{N})$ to be the set of edges **between** vertices in N **and** vertices in $\bar{N}$.

The correct interpretation excludes all the edges that are in the reverse direction of the flow (i.e, which start from $\bar{N}$ and end in N).

Many of you gave counterexamples where the E' was a proper subset of $E(N, \bar{N})$ according to the cut whose cut-set was closest to E'. However, if you look carefully, all those "extra" edges (which were in $E(N, \bar{N})$ but not in E') would be in the reverse direction; i.e., they would start from $\bar{N}$ and end in N - not the other way round!

The grading on this problem has been totally subjective, i.e., there is no fixed set of rules according to which I graded. Basically, I read you solution, and if you misinterpreted the problem and gave

a counterexample, probably you would not get more than 40% of marks. Worse, if you tried to give a counterexample where there actually *is* a cut consisting of the same edges as E' and yet you deliberately chose another cut, then that means your basic understanding is wrong and you would be penalized heavily.

Basically, your answer **must** bring up the issue of the reverse edges in the cut and why they would not feature in $E(N, \bar{N})$. You cannot give a proof without tackling this problem – you must grab the bull by the horn. If it appears that you did not address (and resolve) the issue of reverse-direction edges in the cut, then again you would be penalized heavily.

Lastly, I admit that I have probably been a little bit harsh in grading this question. But, this question is the very reason I have been very lenient in other questions (especially in Q2 and Q4).

6. (Page Limit: 1/2 page)

   Show that a maximum flow in a network $G = (V, E)$ can always be found by a sequence of at most $|E|$ augmenting paths. (Hint: Determine the paths after finding the maximum flow.)
   Solution:
   The solution to this question is fairly straight forward. First you find the maximum flow. Then you could get the minimum cut of the network. We also know that you saturate one edge on the minimum cut each time. Thus, the upper bound is simply $|E|$.

7. (Page Limit: 1/2 page) Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cap\!\!\cup R$, and let G be its corresponding flow network. Give a good upper bound on the length of any augmenting path found in G during the execution of Ford-Fulkerson.
   Solution:
   By definition, an augmenting path is a simple path $s \rightarrow t$ in the residual graph G_f . Since G has no edges between vertices in L and no edges between vertices in R, neither does the flow network G and hence neither does G_f . Also, the only edges involving s or t connect s to L and R to t. Note that although edges in G can go only from L to R, edges in G_f can also go from R to L. Thus any augmenting path must go $sL \rightarrow R \rightarrow \ldots L \rightarrow R \rightarrow t$ crossing back and forth between L and R at most as many times as it can do so without using a vertex twice. It contains s; t, and equal numbers of distinct vertices from L and R, at most $2 + 2min(|L|, |R|)$ vertices in all. The length of an augmenting path (i.e., its number of edges) is thus bounded above by $2min(|L|, |R|) + 1$.