CS 530 Homework 1 Solutions

1. i. Yes
   ii. No
   iii. No
   iv. No

2. (e) This is false. Let \( f(n) = \frac{1}{n^2} \). We need \( \frac{1}{n^2} \leq c(\frac{1}{n})^2 = \frac{c}{n^2} \). This implies that \( n \leq c \) which can’t happen since \( c \) is a constant.

   (g) This is false. Let \( f(n) = 2^{2n} \). This is saying that \( 2^{2n} = \Theta(2^n) \) which is not true by question 1 (\( 2^{2n} \neq O(2^n) \) implies that \( 2^{2n} \neq \Theta(2^n) \)).

3. The flow graph corresponding to the maximum flow is given below. The value of the maximum flow is 12.

   ![Flow Graph](image)

   The minimum cut is \((\{s, d, e\}, \{a, b, c, f, t\})\).

4. Create a new graph \( G' \) where the capacity of each edge \((u, v)\) is \( f_{\text{max}}(u, v) \). Run Ford-Fulkerson, with the modification that we do not insert back edges in the residual graph. This will still produce correct output in our case because we never exceed the actual maximum flow through an edge, so it is never advantageous to cancel flow. The augmenting paths chosen in this modified version of Ford-Fulkerson are precisely the ones we want. There are at most \(|E|\) because every augmenting path produces at least one edge whose flow is equal to its capacity.

5. i.
Every vertex in this graph has at least degree two, so any cut with a single vertex on one side and the rest of the graph on the other has size at least 2. The min cut, however, has size 1 and is \((\{1, 2, 3\}, \{4, 5, 6\})\).

ii. Suppose there were fewer than 25 edges. Since each edge \((u, v)\) contributes 1 to \(u\)'s degree and 1 to \(v\)'s degree, there must be a vertex \(a\) of degree less than 5, since if every vertex had degree 5 there would have to be \(\frac{1}{2}(10 \times 5) = 25\) edges. Consider the cut \((\{a\}, V - \{a\})\). The cost of this cut is \(a\)'s degree, which is less than 5. This is a contradiction since the min cut was assumed to have cost 5.

iii. Assuming the cut \((A, B)\) is equal to the cut \((B, A)\), \(H\) has \(n(n-1) = 10(9) = 90\) min cuts. Choose from \(n\) nodes where to start set \(A\), and choose from the leftover \(n - 1\) nodes where to end it (\(n - 1\) choices because you can’t have \(A = V\)) and then divide by 2 since \((A, B) = (B, A)\). These are the min cuts since if you start at node \(a\) and end at node \(c\) and \(b\) is between \(a\) and \(c\) in the cycle but is not included in \(A\), you increase the size of the cut since the edges from \(b\) will cross the cut in addition to the edges from \(a\) and from \(c\).

Assuming \((A, B)\) is different than \((B, A)\), you get just \(n(n - 1) = 10(9) = 90\) min cuts.

6. Answer to centroid problem

A centroid of a tree with \(n\) vertices is a vertex such that its deletion leaves no subtree with more than \(\frac{n}{2}\) vertices.

Write an algorithm to find a centroid of a tree. What is the complexity of your algorithm? You should be able to get a linear solution although partial credit will be given for a nonlinear algorithm.

Answer to HW 4, problem 6 (in brief):

The idea for an algorithm to find the centroid is: We will attach a weight of 1 to every vertex in the input tree. Now we repeatedly delete leaves from the tree and as we do so we add the weight of the deleted leaf to the weight of its adjacent vertex. When we reach a vertex with a weight at least \(\frac{n}{2}\), we output that vertex and halt.

The algorithm has only 2 steps.

1. Initially all vertices are given weight 1.

2. We now repeat this step until a vertex is output as the centroid.

   Choose a vertex \(u\) which is a leaf, i.e. has degree 1. (A leaf is a vertex in the graph with degree 1.)

   Let \(v\) be the unique vertex connected to \(u\). Now add the weight at vertex \(u\) to that of vertex \(v\), and delete vertex \(u\) and the edge \((u, v)\) from the tree. (Note: This reduces the degree of \(v\) by 1 in the resulting tree.)

   If the weight of \(v\) is now at least \(\frac{n}{2}\) then output \(v\) as the centroid, and the algorithm now halts.
Now to carry out this algorithm in linear time we need to do some work and in particular define the right data structure to keep track of the algorithm in an efficient way. One data structure we can use is a small table. The table has a row for each vertex $u$ and stored in that row as the algorithm proceeds is the (current) degree of $u$, the sum of the weights of all vertices connected to $u$, a flag indicating which vertices are leaves - this flag changes as the algorithm proceeds, and the weight of $u$. If we update this table carefully as the algorithms goes along we can carry out the whole algorithm in $O(n)$ steps.