Tutorial 2: Greedy Algorithms and Huffman Coding

Definition 1. *Recall that a cut in a graph $G = (V, E)$ is a partition of V into two disjoint sets S and T. We denote the cut between S and T by (S, T). We say that an edge $e = (u, v) \in E$ is cut by (S, T) if one endpoint of e lies in S and the other lies in T (that is, if either $u \in S$ and $v \in T$ or $u \in T$ and $v \in S$). The size of (S, T) is the number of edges cut by (S, T).*

Problem 1. *Design an algorithm that given a graph $G = (V, E)$ finds a cut (S, T) of size at least $|E|/2$ in G. (Note that there may be many cuts of size at least $|E|/2$; the algorithm needs to find just one of them.) What is the running time of your algorithm?*

<u>**Solution**</u>**.** Consider the following algorithm:

$S := \emptyset$; $T := \emptyset$
for each $v \in V$
    if $|N(v) \cap S| > |N(v) \cap T|$ (i.e. v has more neighbors in S than in T)
        $T := T \cup \{v\}$
    else
        $S := S \cup \{v\}$

Consider the sets S and T when the algorithm terminates. Clearly, (S, T) forms a partition of V. Now, consider an edge $e = (u, v) \in E$. We say that e is *activated* by u if u was examined after v in the execution of the algorithm; otherwise, e is activated by v. That is, e is activated by that vertex whose addition to $S \cup T$ during the execution of the algorithm made e to be entirely in $S \cup T$. Let A_v be the set of edges activated by v, and let C_v be the set of those edges in A_v being cut by (S, T). Clearly, $|C_v| \geq |A_v|/2$. Summing over all $v \in V$, we get

$$\sum_{v \in V} |C_v| \geq \sum_{v \in V} |A_v|/2.$$

Notice that the left hand side is the number of edges cut by (S, T) and the right hand side is $|E|/2$. The running time of the algorithm is $O(|V| + |E|)$ encoding G as an adjacency list.

Problem 2. *Let $G = (V, E)$ be a graph with n vertices and m edges. Note that the average degree of G is*

$$\frac{1}{n} \sum_{u \in V} \deg u = \frac{2m}{n}.$$

Denote it by Δ_{avg}. Prove that the following greedy algorithm finds a non-empty subgraph with minimum degree at least $\Delta_{avg}/2$.

```
function FindSubgraph
   Input: a graph G = (V, E);
   Output: a subgraph H such that the minimum degree of H is at least Δ_avg/2.
begin
   let H = G
   while there is a vertex u of degree less than Δ_avg/2 in H
      remove u from H
   return H
end
```

Hint: *What is the average degree of graph H throughout the execution of the algorithm. Suggest an efficient implementation of the algorithm? What is its running time?*

Solution. Clearly, every vertex of the graph H returned by the algorithm will have degree at least $\Delta_{avg}/2$. We need to argue that H will be non-empty. Notice that if G has no edges, then its average degree is zero, and the algorithm will return G, which is a subgraph of G with minimum degree at least $\Delta_{avg}/2$. So let us assume that G has at least one edge, hence its average degree is positive. We will show, by induction, that in every iteration, the average degree of H can only increase. We will then have that in every iteration

$$\Delta_{avg}(H) \geq \Delta_{avg}(G) > 0,$$

therefore the returned graph will be non-empty.

Initially, $H = G$, hence obviously $\Delta_{avg}(H) \geq \Delta_{avg}(G)$. Now consider the graph H at an arbitrary iteration of the algorithm and the graph $H' = H - v$ at the next iteration, and assume $\Delta_{avg}(H) \geq \Delta_{avg}(G)$. If n is the number of vertices of H, m its number of edges, and $\Delta = 2m/n$ its average degree, then the number of vertices of H' is $n' = n - 1$, the number of its edges is $m' = m - \deg v$ and its average degree is $\Delta' = 2m'/n'$. Notice that since $\deg v \leq \Delta_{avg}(G)/2 \leq \Delta/2$, we have

$$m' = m - \deg v \geq m - \Delta/2 = \Delta n/2 - \Delta/2 = \Delta n'/2,$$

and therefore

$$\Delta' = 2m'/n' \geq \Delta.$$

For an efficient implementation, an idea is to store the vertices of G in a priority queue, with their degrees being the keys. At every iteration of the while loop, we may extract the vertex v with the minimum key from the priority queue (if its degree is less than $\Delta_{avg}/2$), and decrease the priorities of v's neighbors by one, which will give a running time of $O(|V| \log |V| + |E|)$.

Problem 3. *We are given an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and frequencies p_i. Assume that $p_i < 1/3$ for every i. Consider a Huffman code for σ. Prove all codewords are of length at least 2.*

Solution. We will show the contrapositive statement, which is: if there is a codeword of length at most 1, then there must be an i for which $p_i \geq 1/3$. First of all, a codeword of length zero (i.e. a tree with only one node) would mean that Σ has only one character and the frequency of that character must be 1. If there is a codeword of length one, then the code viewed as a tree must look like as either the code on the left or right of Figure 1, where $\mathbf{T}_1$ and $\mathbf{T}_2$ are meant to represent arbitrary trees. In the first case, one of the characters α

Figure 1: Two codes with a codeword of length 1

or β must have frequency at least 1/2. In the second case, let p_1 be the frequency of α, p_2 be the frequency of β and p_3 the frequency of γ. Since α appears at depth one whereas β and γ appear at depth two, we have that p_1 is the biggest among p_1, p_2, p_3, so we get

$$3p_1 \geq p_1 + p_2 + p_3 = 1,$$

hence $p_1 \geq 1/3$.